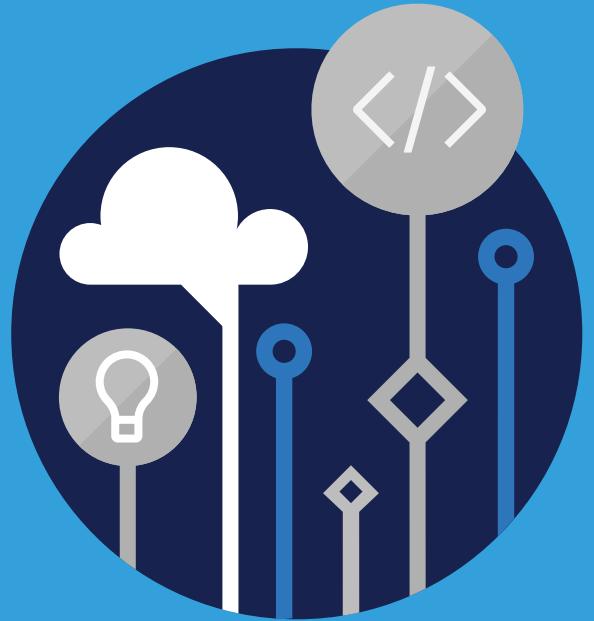


Microsoft  
Official  
Course



**PL-400T00**

Microsoft Power Platform  
Developer

**PL-400T00**  
**Microsoft Power Platform**  
**Developer**

---

## II Disclaimer

Information in this document, including URL and other Internet Web site references, is subject to change without notice. Unless otherwise noted, the example companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted herein are fictitious, and no association with any real company, organization, product, domain name, e-mail address, logo, person, place or event is intended or should be inferred. Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation.

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

The names of manufacturers, products, or URLs are provided for informational purposes only and Microsoft makes no representations and warranties, either expressed, implied, or statutory, regarding these manufacturers or the use of the products with any Microsoft technologies. The inclusion of a manufacturer or product does not imply endorsement of Microsoft of the manufacturer or product. Links may be provided to third party sites. Such sites are not under the control of Microsoft and Microsoft is not responsible for the contents of any linked site or any link contained in a linked site, or any changes or updates to such sites. Microsoft is not responsible for webcasting or any other form of transmission received from any linked site. Microsoft is providing these links to you only as a convenience, and the inclusion of any link does not imply endorsement of Microsoft of the site or the products contained therein.

© 2019 Microsoft Corporation. All rights reserved.

Microsoft and the trademarks listed at <http://www.microsoft.com/trademarks><sup>1</sup> are trademarks of the Microsoft group of companies. All other trademarks are property of their respective owners.

---

<sup>1</sup> <http://www.microsoft.com/trademarks>

## MICROSOFT LICENSE TERMS

### MICROSOFT INSTRUCTOR-LED COURSEWARE

These license terms are an agreement between Microsoft Corporation (or based on where you live, one of its affiliates) and you. Please read them. They apply to your use of the content accompanying this agreement which includes the media on which you received it, if any. These license terms also apply to Trainer Content and any updates and supplements for the Licensed Content unless other terms accompany those items. If so, those terms apply.

**BY ACCESSING, DOWNLOADING OR USING THE LICENSED CONTENT, YOU ACCEPT THESE TERMS.  
IF YOU DO NOT ACCEPT THEM, DO NOT ACCESS, DOWNLOAD OR USE THE LICENSED CONTENT.**

**If you comply with these license terms, you have the rights below for each license you acquire.**

#### 1. DEFINITIONS.

1. "Authorized Learning Center" means a Microsoft Imagine Academy (MSIA) Program Member, Microsoft Learning Competency Member, or such other entity as Microsoft may designate from time to time.
2. "Authorized Training Session" means the instructor-led training class using Microsoft Instructor-Led Courseware conducted by a Trainer at or through an Authorized Learning Center.
3. "Classroom Device" means one (1) dedicated, secure computer that an Authorized Learning Center owns or controls that is located at an Authorized Learning Center's training facilities that meets or exceeds the hardware level specified for the particular Microsoft Instructor-Led Courseware.
4. "End User" means an individual who is (i) duly enrolled in and attending an Authorized Training Session or Private Training Session, (ii) an employee of an MPN Member (defined below), or (iii) a Microsoft full-time employee, a Microsoft Imagine Academy (MSIA) Program Member, or a Microsoft Learn for Educators – Validated Educator.
5. "Licensed Content" means the content accompanying this agreement which may include the Microsoft Instructor-Led Courseware or Trainer Content.
6. "Microsoft Certified Trainer" or "MCT" means an individual who is (i) engaged to teach a training session to End Users on behalf of an Authorized Learning Center or MPN Member, and (ii) currently certified as a Microsoft Certified Trainer under the Microsoft Certification Program.
7. "Microsoft Instructor-Led Courseware" means the Microsoft-branded instructor-led training course that educates IT professionals, developers, students at an academic institution, and other learners on Microsoft technologies. A Microsoft Instructor-Led Courseware title may be branded as MOC, Microsoft Dynamics, or Microsoft Business Group courseware.
8. "Microsoft Imagine Academy (MSIA) Program Member" means an active member of the Microsoft Imagine Academy Program.
9. "Microsoft Learn for Educators – Validated Educator" means an educator who has been validated through the Microsoft Learn for Educators program as an active educator at a college, university, community college, polytechnic or K-12 institution.
10. "Microsoft Learning Competency Member" means an active member of the Microsoft Partner Network program in good standing that currently holds the Learning Competency status.
11. "MOC" means the "Official Microsoft Learning Product" instructor-led courseware known as Microsoft Official Course that educates IT professionals, developers, students at an academic institution, and other learners on Microsoft technologies.
12. "MPN Member" means an active Microsoft Partner Network program member in good standing.

13. "Personal Device" means one (1) personal computer, device, workstation or other digital electronic device that you personally own or control that meets or exceeds the hardware level specified for the particular Microsoft Instructor-Led Courseware.
  14. "Private Training Session" means the instructor-led training classes provided by MPN Members for corporate customers to teach a predefined learning objective using Microsoft Instructor-Led Courseware. These classes are not advertised or promoted to the general public and class attendance is restricted to individuals employed by or contracted by the corporate customer.
  15. "Trainer" means (i) an academically accredited educator engaged by a Microsoft Imagine Academy Program Member to teach an Authorized Training Session, (ii) an academically accredited educator validated as a Microsoft Learn for Educators – Validated Educator, and/or (iii) a MCT.
  16. "Trainer Content" means the trainer version of the Microsoft Instructor-Led Courseware and additional supplemental content designated solely for Trainers' use to teach a training session using the Microsoft Instructor-Led Courseware. Trainer Content may include Microsoft PowerPoint presentations, trainer preparation guide, train the trainer materials, Microsoft One Note packs, classroom setup guide and Pre-release course feedback form. To clarify, Trainer Content does not include any software, virtual hard disks or virtual machines.
2. **USE RIGHTS.** The Licensed Content is licensed, not sold. The Licensed Content is licensed on a **one copy per user basis**, such that you must acquire a license for each individual that accesses or uses the Licensed Content.
- 2.1 Below are five separate sets of use rights. Only one set of rights apply to you.
    1. **If you are a Microsoft Imagine Academy (MSIA) Program Member:**
      1. Each license acquired on behalf of yourself may only be used to review one (1) copy of the Microsoft Instructor-Led Courseware in the form provided to you. If the Microsoft Instructor-Led Courseware is in digital format, you may install one (1) copy on up to three (3) Personal Devices. You may not install the Microsoft Instructor-Led Courseware on a device you do not own or control.
      2. For each license you acquire on behalf of an End User or Trainer, you may either:
        1. distribute one (1) hard copy version of the Microsoft Instructor-Led Courseware to one (1) End User who is enrolled in the Authorized Training Session, and only immediately prior to the commencement of the Authorized Training Session that is the subject matter of the Microsoft Instructor-Led Courseware being provided, **or**
        2. provide one (1) End User with the unique redemption code and instructions on how they can access one (1) digital version of the Microsoft Instructor-Led Courseware, **or**
        3. provide one (1) Trainer with the unique redemption code and instructions on how they can access one (1) Trainer Content.
      3. For each license you acquire, you must comply with the following:
        1. you will only provide access to the Licensed Content to those individuals who have acquired a valid license to the Licensed Content,
        2. you will ensure each End User attending an Authorized Training Session has their own valid licensed copy of the Microsoft Instructor-Led Courseware that is the subject of the Authorized Training Session,
        3. you will ensure that each End User provided with the hard-copy version of the Microsoft Instructor-Led Courseware will be presented with a copy of this agreement and each End

User will agree that their use of the Microsoft Instructor-Led Courseware will be subject to the terms in this agreement prior to providing them with the Microsoft Instructor-Led Courseware. Each individual will be required to denote their acceptance of this agreement in a manner that is enforceable under local law prior to their accessing the Microsoft Instructor-Led Courseware,

4. you will ensure that each Trainer teaching an Authorized Training Session has their own valid licensed copy of the Trainer Content that is the subject of the Authorized Training Session,
5. you will only use qualified Trainers who have in-depth knowledge of and experience with the Microsoft technology that is the subject of the Microsoft Instructor-Led Courseware being taught for all your Authorized Training Sessions,
6. you will only deliver a maximum of 15 hours of training per week for each Authorized Training Session that uses a MOC title, and
7. you acknowledge that Trainers that are not MCTs will not have access to all of the trainer resources for the Microsoft Instructor-Led Courseware.

**2. If you are a Microsoft Learning Competency Member:**

1. Each license acquire may only be used to review one (1) copy of the Microsoft Instructor-Led Courseware in the form provided to you. If the Microsoft Instructor-Led Courseware is in digital format, you may install one (1) copy on up to three (3) Personal Devices. You may not install the Microsoft Instructor-Led Courseware on a device you do not own or control.
2. For each license you acquire on behalf of an End User or MCT, you may either:
  1. distribute one (1) hard copy version of the Microsoft Instructor-Led Courseware to one (1) End User attending the Authorized Training Session and only immediately prior to the commencement of the Authorized Training Session that is the subject matter of the Microsoft Instructor-Led Courseware provided, **or**
  2. provide one (1) End User attending the Authorized Training Session with the unique redemption code and instructions on how they can access one (1) digital version of the Microsoft Instructor-Led Courseware, **or**
  3. you will provide one (1) MCT with the unique redemption code and instructions on how they can access one (1) Trainer Content.
3. For each license you acquire, you must comply with the following:
  1. you will only provide access to the Licensed Content to those individuals who have acquired a valid license to the Licensed Content,
  2. you will ensure that each End User attending an Authorized Training Session has their own valid licensed copy of the Microsoft Instructor-Led Courseware that is the subject of the Authorized Training Session,
  3. you will ensure that each End User provided with a hard-copy version of the Microsoft Instructor-Led Courseware will be presented with a copy of this agreement and each End User will agree that their use of the Microsoft Instructor-Led Courseware will be subject to the terms in this agreement prior to providing them with the Microsoft Instructor-Led Courseware. Each individual will be required to denote their acceptance of this agreement in a manner that is enforceable under local law prior to their accessing the Microsoft Instructor-Led Courseware,

4. you will ensure that each MCT teaching an Authorized Training Session has their own valid licensed copy of the Trainer Content that is the subject of the Authorized Training Session,
5. you will only use qualified MCTs who also hold the applicable Microsoft Certification credential that is the subject of the MOC title being taught for all your Authorized Training Sessions using MOC,
6. you will only provide access to the Microsoft Instructor-Led Courseware to End Users, and
7. you will only provide access to the Trainer Content to MCTs.

**3. If you are a MPN Member:**

1. Each license acquired on behalf of yourself may only be used to review one (1) copy of the Microsoft Instructor-Led Courseware in the form provided to you. If the Microsoft Instructor-Led Courseware is in digital format, you may install one (1) copy on up to three (3) Personal Devices. You may not install the Microsoft Instructor-Led Courseware on a device you do not own or control.
2. For each license you acquire on behalf of an End User or Trainer, you may either:
  1. distribute one (1) hard copy version of the Microsoft Instructor-Led Courseware to one (1) End User attending the Private Training Session, and only immediately prior to the commencement of the Private Training Session that is the subject matter of the Microsoft Instructor-Led Courseware being provided, **or**
  2. provide one (1) End User who is attending the Private Training Session with the unique redemption code and instructions on how they can access one (1) digital version of the Microsoft Instructor-Led Courseware, **or**
  3. you will provide one (1) Trainer who is teaching the Private Training Session with the unique redemption code and instructions on how they can access one (1) Trainer Content.
3. For each license you acquire, you must comply with the following:
  1. you will only provide access to the Licensed Content to those individuals who have acquired a valid license to the Licensed Content,
  2. you will ensure that each End User attending an Private Training Session has their own valid licensed copy of the Microsoft Instructor-Led Courseware that is the subject of the Private Training Session,
  3. you will ensure that each End User provided with a hard copy version of the Microsoft Instructor-Led Courseware will be presented with a copy of this agreement and each End User will agree that their use of the Microsoft Instructor-Led Courseware will be subject to the terms in this agreement prior to providing them with the Microsoft Instructor-Led Courseware. Each individual will be required to denote their acceptance of this agreement in a manner that is enforceable under local law prior to their accessing the Microsoft Instructor-Led Courseware,
  4. you will ensure that each Trainer teaching an Private Training Session has their own valid licensed copy of the Trainer Content that is the subject of the Private Training Session,

5. you will only use qualified Trainers who hold the applicable Microsoft Certification credential that is the subject of the Microsoft Instructor-Led Courseware being taught for all your Private Training Sessions,
6. you will only use qualified MCTs who hold the applicable Microsoft Certification credential that is the subject of the MOC title being taught for all your Private Training Sessions using MOC,
7. you will only provide access to the Microsoft Instructor-Led Courseware to End Users, and
8. you will only provide access to the Trainer Content to Trainers.

**4. If you are an End User:**

For each license you acquire, you may use the Microsoft Instructor-Led Courseware solely for your personal training use. If the Microsoft Instructor-Led Courseware is in digital format, you may access the Microsoft Instructor-Led Courseware online using the unique redemption code provided to you by the training provider and install and use one (1) copy of the Microsoft Instructor-Led Courseware on up to three (3) Personal Devices. You may also print one (1) copy of the Microsoft Instructor-Led Courseware. You may not install the Microsoft Instructor-Led Courseware on a device you do not own or control.

**5. If you are a Trainer.**

1. For each license you acquire, you may install and use one (1) copy of the Trainer Content in the form provided to you on one (1) Personal Device solely to prepare and deliver an Authorized Training Session or Private Training Session, and install one (1) additional copy on another Personal Device as a backup copy, which may be used only to reinstall the Trainer Content. You may not install or use a copy of the Trainer Content on a device you do not own or control. You may also print one (1) copy of the Trainer Content solely to prepare for and deliver an Authorized Training Session or Private Training Session.

2. If you are an MCT, you may customize the written portions of the Trainer Content that are logically associated with instruction of a training session in accordance with the most recent version of the MCT agreement.
3. If you elect to exercise the foregoing rights, you agree to comply with the following: (i) customizations may only be used for teaching Authorized Training Sessions and Private Training Sessions, and (ii) all customizations will comply with this agreement. For clarity, any use of "customize" refers only to changing the order of slides and content, and/or not using all the slides or content, it does not mean changing or modifying any slide or content.

- 2.2 **Separation of Components.** The Licensed Content is licensed as a single unit and you may not separate their components and install them on different devices.
- 2.3 **Redistribution of Licensed Content.** Except as expressly provided in the use rights above, you may not distribute any Licensed Content or any portion thereof (including any permitted modifications) to any third parties without the express written permission of Microsoft.
- 2.4 **Third Party Notices.** The Licensed Content may include third party code that Microsoft, not the third party, licenses to you under this agreement. Notices, if any, for the third party code are included for your information only.
- 2.5 **Additional Terms.** Some Licensed Content may contain components with additional terms, conditions, and licenses regarding its use. Any non-conflicting terms in those conditions and licenses also apply to your use of that respective component and supplements the terms described in this agreement.

3. **LICENSED CONTENT BASED ON PRE-RELEASE TECHNOLOGY.** If the Licensed Content's subject matter is based on a pre-release version of Microsoft technology ("Pre-release"), then in addition to the other provisions in this agreement, these terms also apply:
  1. **Pre-Release Licensed Content.** This Licensed Content subject matter is on the Pre-release version of the Microsoft technology. The technology may not work the way a final version of the technology will and we may change the technology for the final version. We also may not release a final version. Licensed Content based on the final version of the technology may not contain the same information as the Licensed Content based on the Pre-release version. Microsoft is under no obligation to provide you with any further content, including any Licensed Content based on the final version of the technology.
  2. **Feedback.** If you agree to give feedback about the Licensed Content to Microsoft, either directly or through its third party designee, you give to Microsoft without charge, the right to use, share and commercialize your feedback in any way and for any purpose. You also give to third parties, without charge, any patent rights needed for their products, technologies and services to use or interface with any specific parts of a Microsoft technology, Microsoft product, or service that includes the feedback. You will not give feedback that is subject to a license that requires Microsoft to license its technology, technologies, or products to third parties because we include your feedback in them. These rights survive this agreement.
  3. **Pre-release Term.** If you are an Microsoft Imagine Academy Program Member, Microsoft Learning Competency Member, MPN Member, Microsoft Learn for Educators – Validated Educator, or Trainer, you will cease using all copies of the Licensed Content on the Pre-release technology upon (i) the date which Microsoft informs you is the end date for using the Licensed Content on the Pre-release technology, or (ii) sixty (60) days after the commercial release of the technology that is the subject of the Licensed Content, whichever is earliest ("Pre-release term"). Upon expiration or termination of the Pre-release term, you will irretrievably delete and destroy all copies of the Licensed Content in your possession or under your control.
  4. **SCOPE OF LICENSE.** The Licensed Content is licensed, not sold. This agreement only gives you some rights to use the Licensed Content. Microsoft reserves all other rights. Unless applicable law gives you more rights despite this limitation, you may use the Licensed Content only as expressly permitted in this agreement. In doing so, you must comply with any technical limitations in the Licensed Content that only allows you to use it in certain ways. Except as expressly permitted in this agreement, you may not:
    - access or allow any individual to access the Licensed Content if they have not acquired a valid license for the Licensed Content,
    - alter, remove or obscure any copyright or other protective notices (including watermarks), branding or identifications contained in the Licensed Content,
    - modify or create a derivative work of any Licensed Content,
    - publicly display, or make the Licensed Content available for others to access or use,
    - copy, print, install, sell, publish, transmit, lend, adapt, reuse, link to or post, make available or distribute the Licensed Content to any third party,
    - work around any technical limitations in the Licensed Content, or
    - reverse engineer, decompile, remove or otherwise thwart any protections or disassemble the Licensed Content except and only to the extent that applicable law expressly permits, despite this limitation.
  5. **RESERVATION OF RIGHTS AND OWNERSHIP.** Microsoft reserves all rights not expressly granted to you in this agreement. The Licensed Content is protected by copyright and other intellectual property

laws and treaties. Microsoft or its suppliers own the title, copyright, and other intellectual property rights in the Licensed Content.

6. **EXPORT RESTRICTIONS.** The Licensed Content is subject to United States export laws and regulations. You must comply with all domestic and international export laws and regulations that apply to the Licensed Content. These laws include restrictions on destinations, end users and end use. For additional information, see [www.microsoft.com/exporting](http://www.microsoft.com/exporting).
7. **SUPPORT SERVICES.** Because the Licensed Content is provided "as is", we are not obligated to provide support services for it.
8. **TERMINATION.** Without prejudice to any other rights, Microsoft may terminate this agreement if you fail to comply with the terms and conditions of this agreement. Upon termination of this agreement for any reason, you will immediately stop all use of and delete and destroy all copies of the Licensed Content in your possession or under your control.
9. **LINKS TO THIRD PARTY SITES.** You may link to third party sites through the use of the Licensed Content. The third party sites are not under the control of Microsoft, and Microsoft is not responsible for the contents of any third party sites, any links contained in third party sites, or any changes or updates to third party sites. Microsoft is not responsible for webcasting or any other form of transmission received from any third party sites. Microsoft is providing these links to third party sites to you only as a convenience, and the inclusion of any link does not imply an endorsement by Microsoft of the third party site.
10. **ENTIRE AGREEMENT.** This agreement, and any additional terms for the Trainer Content, updates and supplements are the entire agreement for the Licensed Content, updates and supplements.
11. **APPLICABLE LAW.**
  1. United States. If you acquired the Licensed Content in the United States, Washington state law governs the interpretation of this agreement and applies to claims for breach of it, regardless of conflict of laws principles. The laws of the state where you live govern all other claims, including claims under state consumer protection laws, unfair competition laws, and in tort.
  2. Outside the United States. If you acquired the Licensed Content in any other country, the laws of that country apply.
12. **LEGAL EFFECT.** This agreement describes certain legal rights. You may have other rights under the laws of your country. You may also have rights with respect to the party from whom you acquired the Licensed Content. This agreement does not change your rights under the laws of your country if the laws of your country do not permit it to do so.
13. **DISCLAIMER OF WARRANTY. THE LICENSED CONTENT IS LICENSED "AS-IS" AND "AS AVAILABLE." YOU BEAR THE RISK OF USING IT. MICROSOFT AND ITS RESPECTIVE AFFILIATES GIVES NO EXPRESS WARRANTIES, GUARANTEES, OR CONDITIONS. YOU MAY HAVE ADDITIONAL CONSUMER RIGHTS UNDER YOUR LOCAL LAWS WHICH THIS AGREEMENT CANNOT CHANGE. TO THE EXTENT PERMITTED UNDER YOUR LOCAL LAWS, MICROSOFT AND ITS RESPECTIVE AFFILIATES EXCLUDES ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT.**
14. **LIMITATION ON AND EXCLUSION OF REMEDIES AND DAMAGES. YOU CAN RECOVER FROM MICROSOFT, ITS RESPECTIVE AFFILIATES AND ITS SUPPLIERS ONLY DIRECT DAMAGES UP TO US\$5.00. YOU CANNOT RECOVER ANY OTHER DAMAGES, INCLUDING CONSEQUENTIAL, LOST PROFITS, SPECIAL, INDIRECT OR INCIDENTAL DAMAGES.**

This limitation applies to

- anything related to the Licensed Content, services, content (including code) on third party Internet sites or third-party programs; and
- claims for breach of contract, breach of warranty, guarantee or condition, strict liability, negligence, or other tort to the extent permitted by applicable law.

It also applies even if Microsoft knew or should have known about the possibility of the damages. The above limitation or exclusion may not apply to you because your country may not allow the exclusion or limitation of incidental, consequential, or other damages.

**Please note: As this Licensed Content is distributed in Quebec, Canada, some of the clauses in this agreement are provided below in French.**

**Remarque : Ce le contenu sous licence étant distribué au Québec, Canada, certaines des clauses dans ce contrat sont fournies ci-dessous en français.**

**EXONÉRATION DE GARANTIE.** Le contenu sous licence visé par une licence est offert « tel quel ». Toute utilisation de ce contenu sous licence est à votre seule risque et péril. Microsoft n'accorde aucune autre garantie expresse. Vous pouvez bénéficier de droits additionnels en vertu du droit local sur la protection dues consommateurs, que ce contrat ne peut modifier. La ou elles sont permises par le droit locale, les garanties implicites de qualité marchande, d'adéquation à un usage particulier et d'absence de contrefaçon sont exclues.

**LIMITATION DES DOMMAGES-INTÉRÊTS ET EXCLUSION DE RESPONSABILITÉ POUR LES DOMMAGES.** Vous pouvez obtenir de Microsoft et de ses fournisseurs une indemnisation en cas de dommages directs uniquement à hauteur de 5,00 \$ US. Vous ne pouvez prétendre à aucune indemnisation pour les autres dommages, y compris les dommages spéciaux, indirects ou accessoires et pertes de bénéfices.

Cette limitation concerne:

- tout ce qui est relié au le contenu sous licence, aux services ou au contenu (y compris le code) figurant sur des sites Internet tiers ou dans des programmes tiers; et.
- les réclamations au titre de violation de contrat ou de garantie, ou au titre de responsabilité stricte, de négligence ou d'une autre faute dans la limite autorisée par la loi en vigueur.

Elle s'applique également, même si Microsoft connaissait ou devrait connaître l'éventualité d'un tel dommage. Si votre pays n'autorise pas l'exclusion ou la limitation de responsabilité pour les dommages indirects, accessoires ou de quelque nature que ce soit, il se peut que la limitation ou l'exclusion ci-dessus ne s'appliquera pas à votre égard.

**EFFET JURIDIQUE.** Le présent contrat décrit certains droits juridiques. Vous pourriez avoir d'autres droits prévus par les lois de votre pays. Le présent contrat ne modifie pas les droits que vous confèrent les lois de votre pays si celles-ci ne le permettent pas.

Revised April 2019



# Contents

■	<b>Module 0 Welcome and Introduction to Power Platform</b>	1
	Start Here .....	1
	Power Platform overview .....	3
	Module summary .....	8
■	<b>Module 1 Create a model-driven app in Power Apps</b>	11
	Introduction to model-driven apps and Common Data Service .....	11
	Get started with model-driven apps in Power Apps .....	17
	Create and manage entities .....	35
	Create and manage fields within an entity in Common Data Service .....	41
	Working with option sets .....	53
	Create a relationship between entities .....	59
	Define and create business rules .....	71
	Create and define calculation or rollup fields .....	83
	Get started with security roles in Common Data Service .....	96
■	<b>Module 2 Create a canvas app in Power Apps</b>	107
	Get started with Power Apps .....	107
	Customize a canvas app in Power Apps .....	134
	Manage apps in Power Apps .....	145
	Navigation in a canvas app in Power Apps .....	152
	How to build the UI in a canvas app in Power Apps .....	158
	Use and understand Controls in a canvas app in Power Apps .....	167
	Document and test your Power Apps application .....	176
■	<b>Module 3 Master advance techniques and data options in canvas app</b>	189
	Use imperative development techniques for canvas apps in Power Apps .....	189
	Author an advanced formula in a canvas app in Power Apps .....	198
	Perform custom updates in a Power Apps canvas app .....	204
	Complete testing and performance checks in a Power Apps canvas app .....	212
	Work with relational data in a Power Apps canvas app .....	222
	Work with data source limits in a canvas app .....	228
	Connecting to other data in a Power Apps canvas app .....	235
	Use custom connectors in a Power Apps canvas app .....	241
■	<b>Module 4 Automate a business process using Power Automate</b>	259
	Get started with Power Automate .....	259

Build more complex flows .....	290
Introduction to business process flows in Power Automate .....	351
Create an immersive business process flow .....	355
Understand advanced business process flow concepts in Power Automate .....	376
Introduction to expressions in Power Automate .....	409
<b>Module 5 Introduction to developing with Power Platform .....</b>	<b>427</b>
Introduction to Power Platform developer resources .....	427
Use developer tools to extend the Power Platform .....	438
Introduction to extending the Microsoft Power Platform .....	463
<b>Module 6 Extending the Power Platform Common Data Service .....</b>	<b>471</b>
Introduction to Common Data Service for developers .....	471
Extend plug-ins .....	479
<b>Module 7 Extending the Power Platform user experience Model Driven apps .....</b>	<b>539</b>
Introduction to web resources .....	539
Performing common actions with client script .....	547
Automate business process flows with client script .....	585
<b>Module 8 Create components with Power Apps Component Framework .....</b>	<b>637</b>
Get started with Power Apps component framework .....	637
Build a Power Apps component .....	644
Use advanced features with Power Apps component framework .....	664
<b>Module 9 Extend Power Apps Portals .....</b>	<b>701</b>
Introduction to Power Apps portals .....	701
Access Common Data Service in Power Apps portals .....	719
Extend Power Apps portals .....	737
Build custom Power Apps portals web templates .....	754
<b>Module 10 Integrate with Power Platform and Common Data Service .....</b>	<b>767</b>
Work with Common Data Service Web API .....	767
Integrate Common Data Service Azure solutions .....	789

# Module 0 Welcome and Introduction to Power Platform

## Start Here

### About this course

### Course Description

The Microsoft Power Platform helps organizations optimize their operations by simplifying, automating and transforming business tasks and processes. In this course, students will learn how to build Power Apps, Automate Flows and extend the platform to complete business requirements and solve complex business problems.

### Audience Profile

Candidates for this course design, develop, secure, and troubleshoot Power Platform solutions. Candidates implement components of a solution that include application enhancements, custom user experience, system integrations, data conversions, custom process automation, and custom visualizations.

Candidates will gain applied knowledge of Power Platform services, including in-depth understanding of capabilities, boundaries, and constraints.

Candidates should have development experience that includes JavaScript, JSON, TypeScript, C#, HTML, .NET, Microsoft Azure, Microsoft 365, RESTful Web Services, ASP.NET, and Power BI.

### Course Completion

After completing this course, students will be able to:

- Create a technical design
- Configure Common Data Service

- Create and configure Power Apps
- Configure business process automation
- Extend the user experience
- Extend the platform
- Develop Integrations

## Power Platform Developer

### Certification

Upon completion of this course and practicing building solutions, we encourage you to [get certified.<sup>1</sup>](#)

### PL-400 Certification Exam

This exam measures your ability to accomplish the following technical tasks: create a technical design, configure Common Data Service, create and configure Power Apps, configure business process automation, extend the user experience, extend the platform, and develop integrations.

PL-400 Study Areas	Weights
Create a technical design	10-15%
Configure Common Data Service	15-20%
Create and configure Power Apps	15-20%
Configure business process automation	5-10%
Extend the user experience	10-15%
Extend the platform	15-20%
Develop integrations	5-10%

### PL-400 Power Platform Developer lab

This course is accompanied by a business case lab that can be accessed at [https://github.com/MicrosoftLearning/PL-400\\_Microsoft-Power-Platform-Developer/](https://github.com/MicrosoftLearning/PL-400_Microsoft-Power-Platform-Developer/)

---

<sup>1</sup> <https://docs.microsoft.com/en-us/learn/certifications/power-platform-developer-associate>

# Power Platform overview

## Introduction to the Power Platform

Modern businesses run on data. Users interact with data daily from entering their time for payroll, seeking guidance on existing processes, and analyzing data to make decisions. In our technology driven world, users can be empowered to gain insights from and interact with data all while automating those menial responsibilities that seem to be more burden than job task. Power Platform enables your business to craft solutions while empowering you to unite customized technology to help everyone, from the CEO to the front-line workers, drive the business with data.

In this lesson, you will:

- Learn the components and features of Power Platform
- Identify when to use each Power Platform component application to create business solutions
- Learn the value of using the Power Platform to create business solutions

## What is the Power Platform

Power Platform is comprised of four key products: Power Apps, Power Automate, and Power BI and Power Virtual Agents.

### Power Apps

**Power Apps** provides a rapid low code development environment for building custom apps for business needs. It has services, connectors, and a scalable data service and app platform (Common Data Service) to allow simple integration and interaction with existing data. Power Apps enables the creation of web and mobile applications that run on all devices.

People use apps for every area of their lives, and business should be no exception. Most out of the box solutions do not meet exact business needs or integrate well with other business programs. Power Apps eases users into app development with a simple interface so that every business user or pro developer can build custom apps.

### Power Automate

**Power Automate** lets users create automated workflows between applications and services. It helps automate repetitive business processes such as communication, data collections, and decision approvals. Don't waste important productive hours on drafting the same email for a weekly update or walking approvals through. Not only for the individual user, Power Automate allows for the creation of enterprise-grade process automation. Power Automate's simple interface allows every level of user to automate work tasks from beginners to seasoned developers.

### Power BI

**Power BI** (Business Intelligence) is a business analytics service that delivers insights for analyzing data. It can share those insights through data visualizations which make up reports and dashboards to enable fast, informed decisions. Power BI scales across an organization, and it has built-in governance and security allowing businesses to focus on using data more than managing it.

You can consider Power BI as the analysis and insights leg of the Power Platform. It takes business data and allows you to display it in ways that makes the most sense to users. A Power BI dashboard could

potentially replace a standing meeting to report out on company metrics such as sales data, progress against goals, or employee performance.

## Power Virtual Agents

**Power Virtual Agents** enables anyone to create powerful chatbots using a guided, no-code graphical interface, without the need for data scientists or developers.

Power virtual agents addresses many of the major issues with chatbot building. It eliminates the gap between subject matter experts and the development teams building the chatbots. It removes the complexity of exposing teams to the nuances of conversational AI and the need to write complex code. It minimizes the IT effort required to deploy and maintain a custom conversational solution by empowering subject matter experts to build and maintain their own conversational solutions.

## Features

Among the programs listed above, there are cross cutting features which enable the Power Platform to be leveraged to its full potential. Some of these are:

### AI Builder

**AI Builder** lets users and developers add AI capabilities to the workflows and PowerApps they create and use. AI Builder is a turnkey solution that allows you to easily add intelligence to your workflows and apps and predict outcomes to help improve business performance without writing code.

### The Common Data Service

**Common Data Service** is a scalable data service and app platform which lets users securely store and manage data from multiple sources and integrate that data in business applications using a common data model to ensure ease and consistency to users. Common Data Service is the common currency that enables the components of Power Platform to work together. It's the foundation that enables the consolidation, display, and manipulation of data.

### Connectors

**Connectors** enable you to connect apps, data, and devices in the cloud. Consider connectors the bridge across which information and commands travel. There are more than 275 connectors for the Power Platform, enabling all of your data and actions to connect cohesively. Examples of popular connectors include Salesforce, Office 365, Twitter, Dropbox, Google services, and more.

### Portals

**Portals** bring the power of no-code solutions to building externally facing websites. Through the Power Apps interface, you can build an anonymous or authenticated website that allows users to interact with data held in Common Data Service. The same drag and drop experience you enjoy when building apps is available to build these rich, interactive websites.

Although every feature is essential to building powerful solutions, let's dive in deeper to one of the features of Power Platform, **connectors**.

## Data connectors

Power Platform is made powerful by its ability to leverage data across many platforms. To do this, components of the Power Platform use connectors. You can think of connectors as a bridge from your data source to your app or workflow which allows information to be conveyed back and forth. Connectors allow you to extend your business solutions across platforms and add functionality for your users, and Power Platform has more than 275 connectors with the ability to build custom connectors as well.

It is important to note that premium connectors require a paid plan for use. The connector reference in the summary and resources unit lists all connectors and whether they are considered standard or premium. Custom connectors are also considered a premium feature.

## Data Sources

In order to understand the types of connectors and what you can do with them, you must first understand the types of data sources to which they connect. The two types of data sources are tabular and function-based.

**Tabular data** - A tabular data source is one that returns data in a structured table format. Power Apps can directly read and display these tables through galleries, forms, and other controls. Additionally, if the data source supports it, Power Apps can create, edit, and delete data from these data sources. Examples include Common Data Service, SharePoint, and SQL Server.

**Function-based data** - A function-based data source is one that uses functions to interact with the data source. These functions can be used to return a table of data, but offer more extensive action such as the ability to send an email, update permissions, or create a calendar event. Examples include Office 365 Users, Project Online, and Azure Blob Storage.

Both of these data source types are commonly used to bring data and additional functionality to your solutions.

As you can see, connecting to data sources allows you to integrate disparate parts of your business solutions to build them out cohesively.

## Connectors

Now that you understand more about data sources and delegation, you are ready to learn about connectors.

**Connectors** are the bridges from your data source to your app, workflow, or dashboard. The Power Platform has more than 275 connectors available to common data sources. Connectors are divided into standard and premium. Some popular standard connectors are SharePoint, Outlook, and YouTube. Premium connectors require additional licensing for your app and/or users. A few premium connectors are SQL Server, Survey Monkey, and Mail Chimp.

## Triggers and Actions

Once you have established a data source and configured your connector, there are two types of operations you can use, triggers or actions.

**Triggers** are only used in Power Automate and prompt a flow to begin. Triggers can be time based, such as a flow which begins every day at 8:00 am, or they could be based off of an action like creating a new record in a table or receiving an email. You will always need a trigger to tell your workflow when to run.

**Actions** are used in Power Automate and Power Apps. Actions are prompted by the user or a trigger and allow interaction with your data source by some function. For example, an action would be sending an email in your workflow or app or writing a new line to a data source.

Now that you understand what connectors are and how to use them, let's look at what to do when there isn't a connector already built for your data source.

## Custom Connectors

While the Power Platform offers more than 200 connectors, you also have the option to build a custom connector. This will allow you to extend your app by calling a publicly available API, or a custom API you're hosting in a cloud provider, such as Azure. API stands for Application Programming Interface and holds a series of functions available for developers. Connectors work by sending information back and forth across these APIs and gathering available functions into Power Apps or Power Automate. Because these connectors are function-based, they will call specific functions in the underlying service of the API to return the corresponding data.

An advantage of building custom connectors is that they can be used in different platforms, such as PowerApps, Power Automate, and Azure Logic Apps.

## Creating Custom Connectors

You can create custom connectors using 3 different approaches:

- **Using a blank custom connector<sup>2</sup>**
- **From an OpenAPI definition<sup>3</sup>**
- **From a Postman collection<sup>4</sup>**

While the requirements for each approach will vary, they all require a Power Apps per app or per user plan. Each link above points to the instructions for each approach.

## Pulling it together

Although we live in a data driven world, your business can find it difficult to take advantage of the data you have access to. Sales, customer, and employee data should drive our business decisions, but where do we even start? The Power Platform can add value to any business by helping you to analyze, act, and automate. Act by building custom apps in Power Apps, automate processes based on the data you collect in Power Automation, and analyze the data you have collected in Power BI.

Consider a business that has IT equipment for general use. Currently equipment check-out is conducted by visiting the IT office and if the product is available, writing your name and the equipment name in a notebook. Employees may have to visit IT several times before equipment becomes available, and IT personnel must drop their tasks to check on equipment status or go to collect it for the employee. Sometimes employees hold onto the equipment longer than they intend and an IT personnel spends time tracking it down. In addition, important equipment information such as serial number, warranty details, and instructions for use are kept somewhere in the IT office. How can the Power Platform improve this process?

Power Apps allows us to build an app that has all equipment listed, the status of that equipment, and even important details such as use instructions. This way employees can check out available equipment,

---

<sup>2</sup> <https://docs.microsoft.com/connectors/custom-connectors/define-blank>

<sup>3</sup> <https://docs.microsoft.com/connectors/custom-connectors/define-openapi-definition>

<sup>4</sup> <https://docs.microsoft.com/connectors/custom-connectors/define-postman-collection>

walk to IT at a specified pick up time where the equipment will be ready, and even access the use instructions or flag an equipment malfunction from their phone or tablet. Power Automate can read when equipment needs to be returned and send out reminder emails, or even a warning that the equipment is late being checked in. Users can see when equipment is booked through the app and request check out for a future date at which time Power Automate can send them a reminder to pick up the equipment and IT a reminder to have it ready. Power BI can take all the data generated from the app and analyze it to help you understand what equipment is used most often and by whom. This way you can decide if you need additional equipment, if some users or departments need dedicated equipment, and when your equipment has reached the end of its usefulness.

This is only one common scenario in which the Power Platform can transform the way businesses work. Consider your own business and what processes take up valuable time and are a burden to customers or employees. How can you leverage the Power Platform to improve them?

## Module summary

### Questions

#### Multiple choice

*Your social media engagement officer has requested your help in boosting followers and retweets on Twitter. How could you help her get more information to better understand and subsequently increase engagement?<<*

- Power Apps portals can create a new customer site for our followers
- Power Automate can handle our content approvals for us, reducing the time it takes to produce new content and ensure our quality procedure is followed
- Configure a Power BI report to capture and analyze data from Twitter, allowing you to better understand why certain posts elicit more responses

#### Multiple choice

*Your team has become frustrated with number of times they have to perform basic data entry on project startup. There are many divisions who need the information and sometimes human error results in mistakes, making it more difficult to make sense of your information. Which program would be the most help in this situation?<<*

- Power Apps
- Power Automate
- Power BI

#### Multiple choice

*Someone has added an item in SharePoint which prompts a workflow to run in Power Automate. What type of operation have you used to start your workflow?<<*

- Trigger
- Action
- Function-based

#### Multiple choice

*A client likes the idea of implementing a Power Platform solution, but is concerned about the ability to interact with a custom API. How should you respond?<<*

- The Power Platform offers the ability to create custom connectors for this purpose, which allow you to connect to Power Apps and Power Automate.
- The Power Platform has over 270 connectors to use in these situations.
- The Power Platform uses connectors that hold a series of functions available for developers.

# Summary

Microsoft Power Platform offers a point-and-click approach to building custom applications, data visualizations, and automated workflows. This approach makes it easy for anyone familiar with Microsoft Office to create custom business solutions.

Now that you have reviewed this module, you should be able to:

- Describe the components and features of the Power Platform
- Identify when to use each Power Platform component application to create business solutions
- Understand and explain the value of using the Power Platform to create business solutions

## Key takeaways

Here are the five key takeaways:

1. Power Platform is a system that enables users to do three key actions on data that help them drive business: gain insights from data (Analyze), drive intelligent business processes via apps they build (Act), and automate business processes (Automate).
2. Power BI helps you analyze and visualize data on a unified platform with data from internal and external sources.
3. Power Apps helps you build and deploy customized apps that work across web and mobile, embedded or standalone, on any device.
4. Connectors are bridges that allow you to send information from your data source to your app or workflow and back.
5. Power Automate helps you create automation workflows, from simple to advanced scenarios.

# Answers

## Multiple choice

Your social media engagement officer has requested your help in boosting followers and retweets on Twitter. How could you help her get more information to better understand and subsequently increase engagement? <<

- Power Apps portals can create a new customer site for our followers
- Power Automate can handle our content approvals for us, reducing the time it takes to produce new content and ensure our quality procedure is followed
- Configure a Power BI report to capture and analyze data from Twitter, allowing you to better understand why certain posts elicit more responses

### Explanation

*Power BI allows you to create visuals and better understand your data. Once you understand trends in what followers like, you can post more of that content and increase engagement.*

## Multiple choice

Your team has become frustrated with number of times they have to perform basic data entry on project startup. There are many divisions who need the information and sometimes human error results in mistakes, making it more difficult to make sense of your information. Which program would be the most help in this situation? <<

- Power Apps
- Power Automate
- Power BI

### Explanation

*Power Automate can create automated information workflows so that data entry only has to occur once.*

## Multiple choice

Someone has added an item in SharePoint which prompts a workflow to run in Power Automate. What type of operation have you used to start your workflow? <<

- Trigger
- Action
- Function-based

### Explanation

*A trigger is an operation that tells a workflow to begin or prompts some type of action.*

## Multiple choice

A client likes the idea of implementing a Power Platform solution, but is concerned about the ability to interact with a custom API. How should you respond? <<

- The Power Platform offers the ability to create custom connectors for this purpose, which allow you to connect to Power Apps and Power Automate.
- The Power Platform has over 270 connectors to use in these situations.
- The Power Platform uses connectors that hold a series of functions available for developers.

### Explanation

*You can build out a custom connector to bridge your app or workflow to the API.*

# Module 1 Create a model-driven app in Power Apps

## Introduction to model-driven apps and Common Data Service

### Introduction to Common Data Service

Whether you are building a canvas app or a model-driven app, Common Data Service is the ideal data source because it is the foundational data source of Power Platform. As a result, you will experience the most functionality, the deepest integrations, the most features, and the best ease-of-use of any available data sources. From simple web-based data design to robust, role-based security, Common Data Service is a straightforward platform that you can use to begin designing your data structures and helping to keep them safe. Then, with your data in place, you have rich integration capabilities from Power Apps and the rest of Power Platform. Additionally, by applying business rules, you can trust that your business integrity will be maintained no matter what tool you use to interact with the data.

Another benefit of using Common Data Service is that all of your data is stored in entities. An entity is a set of records that is used to store data, similar to how a table stores data within a database. Common Data Service includes a base set of standard entities that cover typical scenarios, but you can also create custom entities that are specific to your organization. Standard and custom entities within Common Data Service help provide a secure and cloud-based storage option for your data. Entities allow you to create a business-focused definition of your organization's data for use within apps.

Some benefits of using Common Data Service and its entities include:

- **Simple to manage** - Both the metadata and data are stored in the cloud so that you're confident about the details of how they're stored.
- **Helps secure your data** - Data is stored so that users can see it only if you grant them access. Role-based security allows you to control access to entities for different users within your organization.
- **Access your Dynamics 365 data** - If you use Dynamics 365, data from your Dynamics 365 application is also stored within Common Data Service, allowing you to quickly build apps that use your Dynamics 365 data and extend your apps by using Power Apps.
- **Rich metadata** - Data types and relationships are used directly within Power Apps.
- **Logic and validation** - Define calculated fields, business rules, workflows, and business process flows to ensure data quality and drive business processes.
- **Productivity tools** - Entities are available within the add-ins for Microsoft Excel to increase productivity and ensure data accessibility.

Now that you have a better understanding of how Common Data Service works and are aware of some of the benefits, you can now explore how these benefits can be applied in a model-driven app.

## Introduction to model-driven apps

Unlike canvas apps, where you build out an app screen-by-screen by adding logic and code as you go, model-driven apps can be created with a few simple steps. Model-driven apps use a component-focused approach to develop the app. When developing canvas apps, you have complete control over the appearance and behavior of your app, whereas with model-driven apps, the layout is mainly based on the components that you add to the app.

With model-driven apps, a number of different components and component properties are available for you to add and modify when designing an app.

Model-driven app design provides the following benefits:

- Rich component-focused, no-code design environments
- An ability to create complex responsive apps with a similar UI across a variety of devices from desktop to mobile
- Robust design capability
- Apps can be distributed as a solution

# Common Data Service and model-driven apps working together

When creating a model-driven app, you can use entities from Common Data Service as your building blocks. Model-driven apps start with your data model, building up from the shape of your core business data and processes in Common Data Service to model forms, views, and other components.

It's important to ensure that your business data and business processes at the data level are structured properly before you compose your app. Model-driven apps will automatically generate a UI that is responsive across devices; however, this outcome relies heavily on how your data is modeled in Common Data Service.

## Approach to model-driven app making

When creating model-driven apps, it's important to focus on three areas:

- Modeling business data
- Defining business processes
- Composing the app.

More information on creating model-driven apps is included in the Create relationships, business rules, calculations, and rollups in Common Data Service Learning Path and in **Overview of building model-driven apps**<sup>1</sup>.

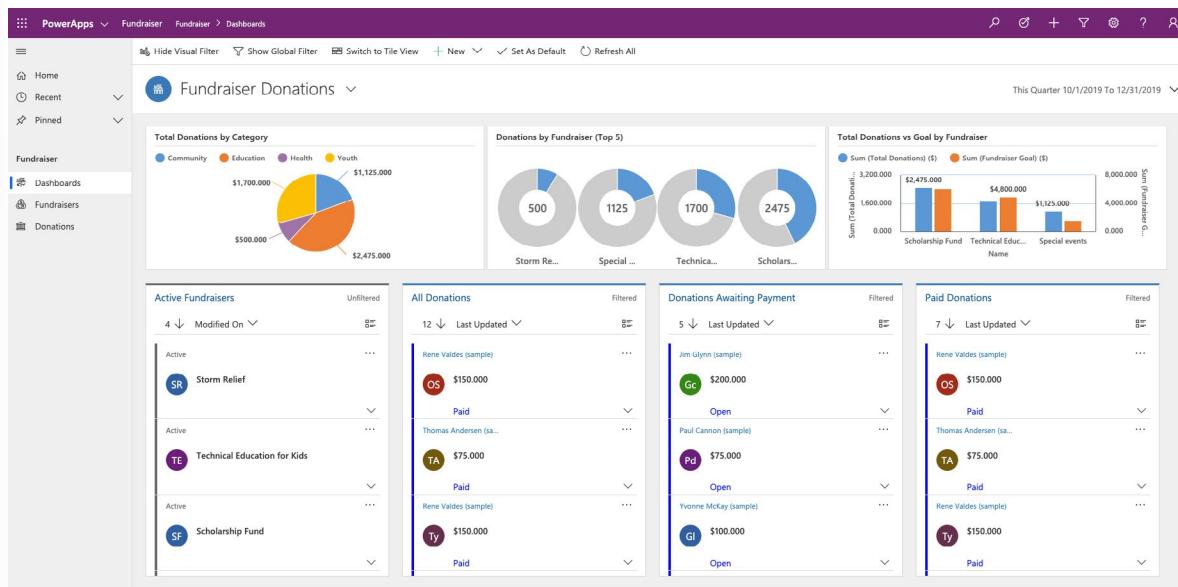
## Exercise - Explore sample template apps

On **Power Apps**<sup>2</sup>, you can use a sample app to explore design possibilities. You'll also discover concepts that you can apply as you develop your own apps. Every sample app uses fictitious data to showcase a real-world scenario.

For more details, be sure to check out the documentation that's specific to each sample app.

<sup>1</sup> <https://docs.microsoft.com/powerapps/maker/model-driven-apps/model-driven-app-overview>

<sup>2</sup> <https://make.powerapps.com/?azure-portal=true>



## Get sample apps

Before you can experiment with or edit the model-driven sample apps, you must set them up in a Common Data Service database. First, create a trial environment and a database, and then select the **Include sample apps and data** check box.

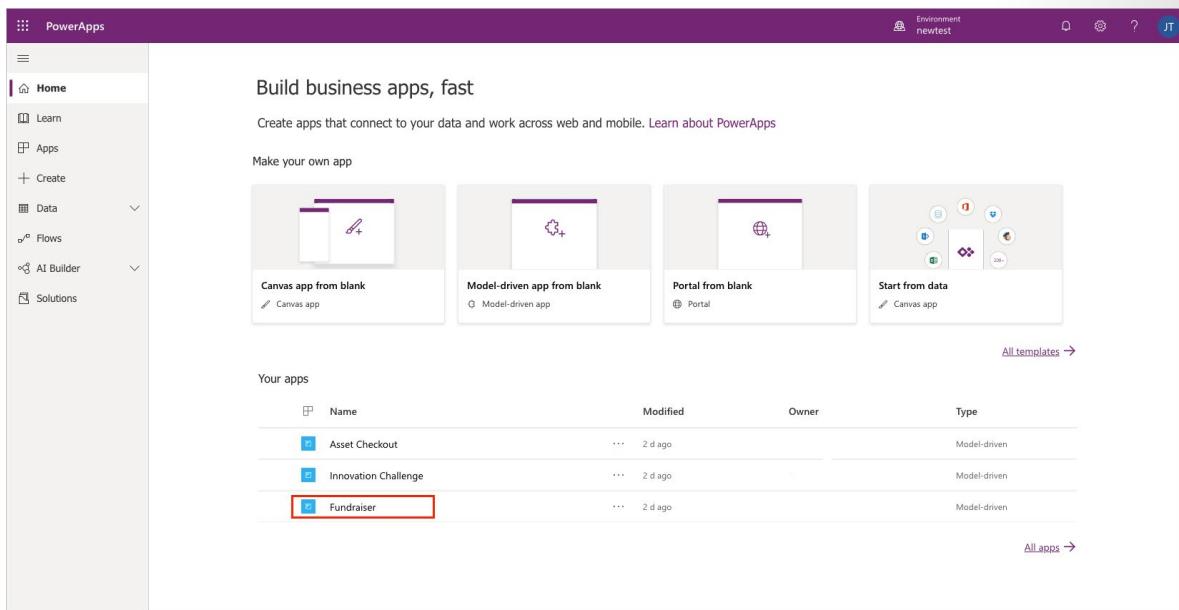
**Important:** By selecting the "Include sample apps and data" check box, you will install all available sample apps in your database. Sample apps are for educational and demonstration purposes. We don't recommend installing them in production databases.

## Run a sample app

To run a sample app, follow these steps:

1. Sign in to **Power Apps**<sup>3</sup> to see a list of available sample apps. For this example, select **Fundraiser**.

<sup>3</sup> <https://make.powerapps.com/?azure-portal=true>



2. At the top of the page, select **Show Visual Filter** to show graphs and charts where you can see how donations to fundraisers are performing. In the next steps, you'll create a new fundraiser and submit a donation to that fundraiser.
3. On the left pane, under **Fundraiser**, select **Fundraisers**.
4. Select **+ New** to add a new fundraiser.
5. Under the **General** tab, enter the following information:
  - Name** - My Fundraiser
6. In the upper-right corner, select the drop-down arrow next to **Total Donations**.
7. From the drop-down menu, enter the following information:
  - Fundraiser Goal** - 500
8. On your keyboard, press **Enter**.
9. Select **Save & Close**.
10. On the left pane, under **Fundraiser**, select **Donations**.
11. Select the drop-down arrow to the right of **Other Activities**.
12. From the drop-down menu, select **Donation**.
13. Enter the following information:
  - **Subject** - My First Donation
  - **Donation Amount** - 100
  - **From** - Nancy Anderson (sample)
  - **Regarding** - My fundraiser
14. Select **Save & Close**.
15. On the left, select **Dashboards**.
16. Select **Show Visual Filter**.

17. Notice **Total Donations vs Goal by Fundraiser** shows your donation.

You have successfully run a sample model-driven app, added a new fundraiser, and added a donation to the fundraiser. While this is a quick look at an app and the related pieces, there is much more to learn about creating an app.

## Summary

The goal of this module was to help you become familiar with the basics of Common Data Service and creating model-driven apps. Common Data Service lets you store and manage data that is used by business applications. Entities within Common Data Service are used to store data.

To review, this module explained the following concepts:

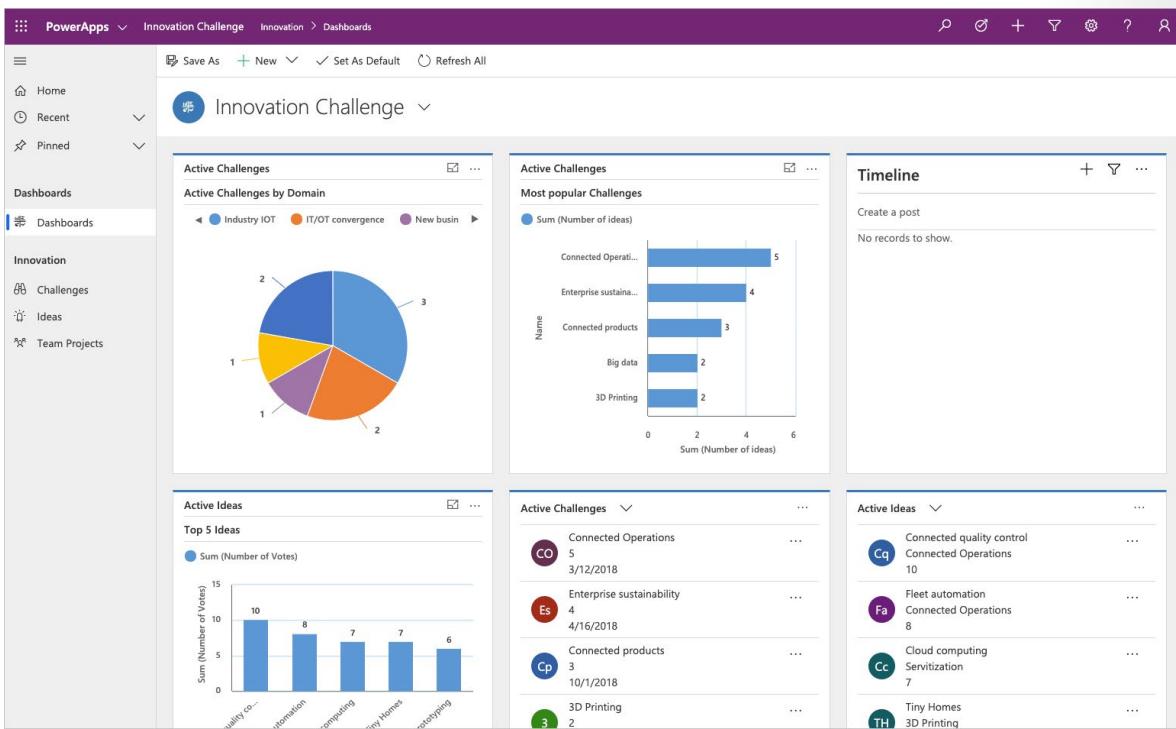
- Common Data Service data is stored in entities.
- You can use as many default entities as possible to quickly build apps.
- You can create new entities.
- Data is stored in a way that users can see it only if you grant them access.
- Sample apps and templates are available to help you learn and create your own apps.

# Get started with model-driven apps in Power Apps

## Introducing model-driven apps

Model-driven app design is an approach that focuses on adding dashboards, forms, views, and charts to your apps. With little or no code, you can build apps that are simple or very complex.

In canvas apps, the app maker has total control over the app layout. In model-driven apps, on the other hand, much of the layout is determined by the components you add. The emphasis is more on quickly viewing your business data and making decisions instead of on intricate app design.



## The approach to making model-driven apps

Model-driven apps have three design phases:

1. Model your business data
2. Define your business processes
3. Build the app

## Model your business data

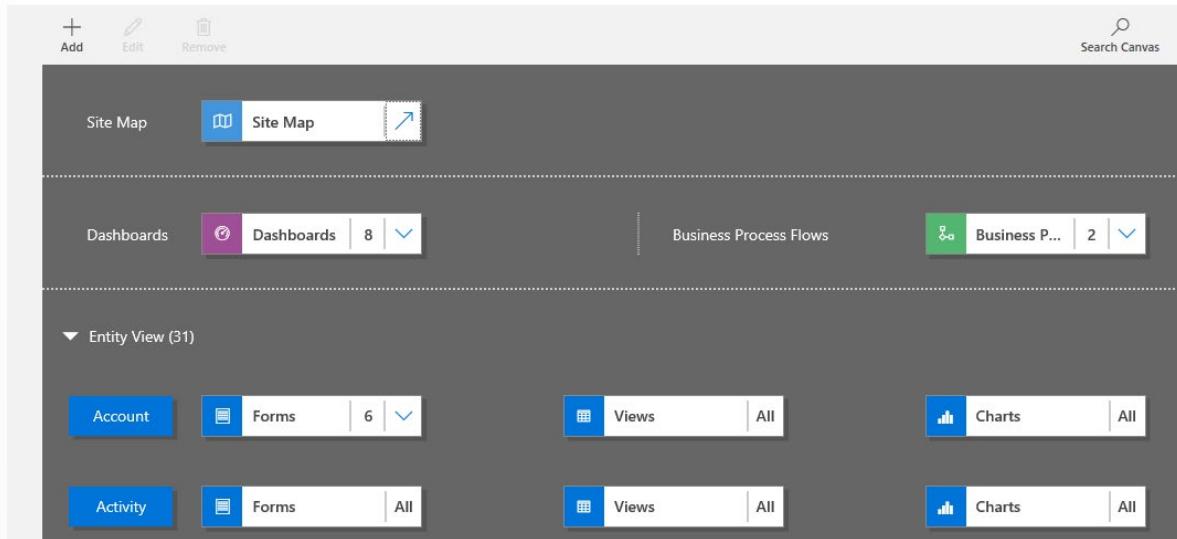
Model-driven design uses metadata-driven architecture so that designers can customize apps without writing code. To model business data, you determine what data the app will need and how that data will relate to other data. Metadata means *data about data* and it defines the structure of the data stored in Common Data Service.

## Define your business processes

Defining and enforcing consistent business processes is a key aspect of model-driven app design. Consistent processes help ensure that your app users can focus on their work and not worry about having to remember to perform a set of manual steps. Processes can be simple or complex, and they often change over time.

## Build the app

After modeling data and defining processes, you build your app by selecting and setting up the components you need in the App Designer.



## Building blocks of model-driven app

A model-driven app consists of several components that you select by using the App Designer. The components and component properties become the metadata. Let's look more closely at these components.

### Data

The data components determine what data the app will be based upon.

Component	Description	Designer
Entity	Entities are items with properties that you track. Examples include contacts and accounts. Many standard entities are available. You can customize a non-system standard entity (or production entity). You can also create a custom entity from scratch.	Entity designer

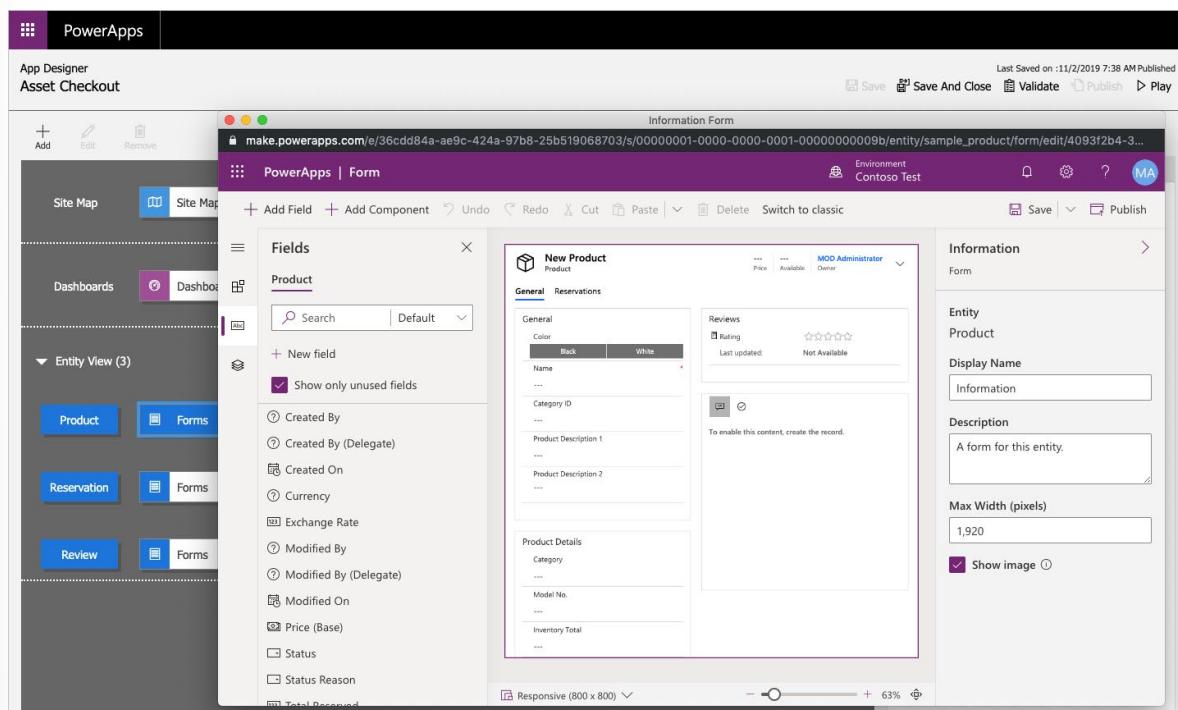
Component	Description	Designer
Field	Fields are properties that are associated with an entity and help define that entity. A field is defined by a data type, which determines the type of data that can be entered or selected. Examples of data types include text, number, date and time, currency, and lookup (which creates a relationship with another entity). Fields are typically used with forms, views, and searches.	Entity designer
Relationship	Relationships define how entities can be related to each other. There are 1:N (one-to-many), N:1 (many-to-one), and N:N (many-to-many) relationships. For example, adding a lookup field to an entity creates a new 1:N relationship between the two entities and lets you add that lookup field to a form.	Entity designer
Option set field	This type of field shows a control that lets the user select among predefined options. Each option has a number value and a label. Option set fields can require either a single value or multiple values.	Entity designer

## User interface

The user interface components determine how users will interact with the app.

Component	Description	Designer
App	Apps determine the app fundamentals, like components, properties, the client type, and the URL.	App designer
Site map	A site map specifies the navigation for your app.	Site map designer

Component	Description	Designer
Form	Forms include a set of data entry fields for a given entity. This set of data entry fields matches the items that your organization tracks for the entity. One example is a set of data entry fields where users enter relevant information to track a customer's previous orders together with specific requested reorder dates.	Form designer
View	Views define how a list of records for a specific entity appears in your app. A view defines the columns shown, the width of each column, the sort behavior, and the default filters.	View designer



## Logic

The logic components determine what business processes, rules, and automation the app will have. Microsoft Power Apps makers use a designer that's specific to the type of process or rule.

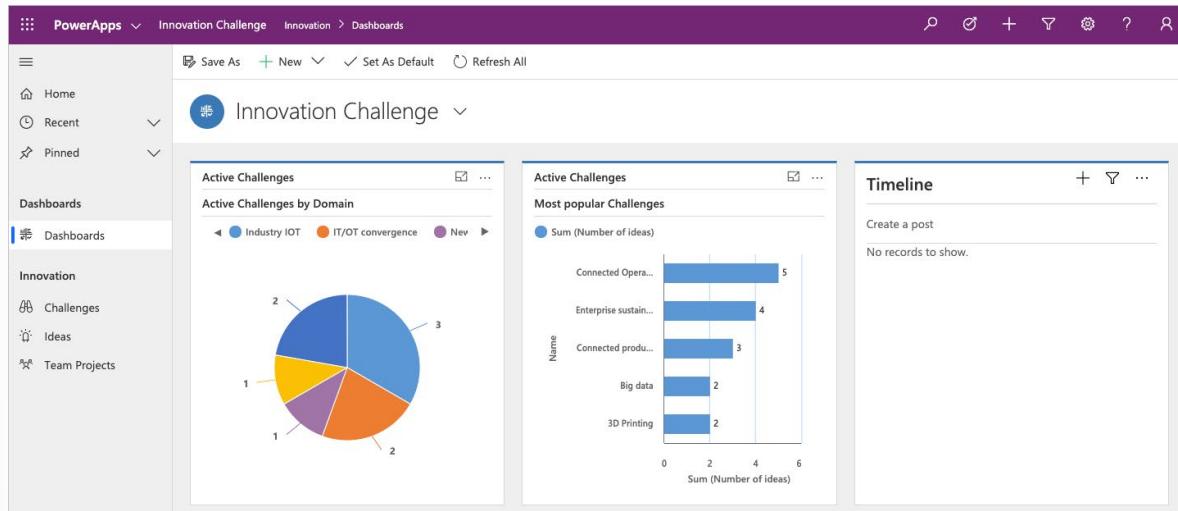
Type of logic	Description	Designer
Business process flow	Business process flows walk users through a standard business process. Use a business process flow if you want everyone to handle customer service requests the same way. Or you can use a business process flow to require staff to gain approval for an invoice before submitting an order.	Business process flow designer
Workflow	Workflows automate business processes without a user interface. Designers use workflows to initiate automation that doesn't require any user interaction.	Workflow designer
Actions	Actions are a type of process that lets you manually invoke actions, including custom actions, directly from a workflow.	Process designer
Business rule	Business rules apply rules or recommendation logic to a form to set field requirements, hide fields, validate data, and more. App designers use a simple interface to implement and maintain fast-changing and commonly used rules.	Business rule designer
Flows	Power Automate is a cloud-based service that lets you create automated workflows between apps and services to get notifications, sync files, collect data, and more.	Power Automate

## Visualization

The visualization components determine what type of data and reporting the app will show.

Component	Description	Designer
Chart	Charts are individual graphical visualizations that can appear in a view or a form or that can be added to a dashboard.	Chart designer
Dashboard	Dashboards show one or more graphical visualizations that provide an overview of actionable business data.	Dashboard designer

Component	Description	Designer
Embedded Microsoft Power BI	Power BI adds embedded Power BI tiles and dashboards to your app. Power BI is a cloud-based service that provides business intelligence (BI) insight.	A combination of chart designer, dashboard designer, and Power BI



## Advanced model-driven apps

Solution Explorer is used to make advanced model-driven apps. By using the navigation pane on the left side of the tool, you can navigate a hierarchy that consists of all app components.

To access the classic Solution Explorer, you must first select a Solution then select **Switch to classic**.

The screenshot shows the Power Apps studio interface with the 'Solutions' section selected in the left sidebar. The main area displays a table of solutions in the 'Default Solution' environment. The table columns are: Display name, Name, Type, Managed..., Modified, Owner, and Status. The data is as follows:

Display name	Name	Type	Managed...	Modified	Owner	Status
Application Ribbons	...	Client Extension	⋮	-	-	-
Site Map	...	Client Extension	⋮	1 wk ago	-	-
Account	... account	Entity	⋮	-	-	-
Account Manager	... Account Manager	Connection Role	⋮	1 wk ago	-	-
Account Reconnect	... Account Reconnect	Email Template	⋮	1 wk ago	-	-
Account Summary	... Account Summary	Report	⋮	4 d ago	-	-

The screenshot shows the PowerApps Entity list screen. On the left, there's a navigation pane titled "Solution: Default Solution" with categories like Information, Components, Entities, Option Sets, Client Extensions, Web Resources, Processes, Plug-in Assemblies, Soft Message Processing Steps, Service Endpoints, Dashboards, Dialog Boxes, Reports, Connection Roles, Email Templates, Mail Merge Templates, Security Roles, Field Security Profiles, Model-driven Apps, Custom Controls, Virtual Entity Data Providers, and Virtual Entity Data Sources. The "Entities" category is selected. The main area displays a table of entities with columns: Component Type (Entity), Display Name, Name, Schema Name, State, Customizable, Audit Status, and Description. The table lists various entities such as Account, Action Card, Activity, Address, Appointment, Attachment, Attribute, Business Unit, Challenge, Challenge Management Process, Channel Access Profile, Channel Access Profile Rule, and Channel Access Profile Rule Item. Each entity has a detailed description in the "Description" column.

## Design model-driven apps

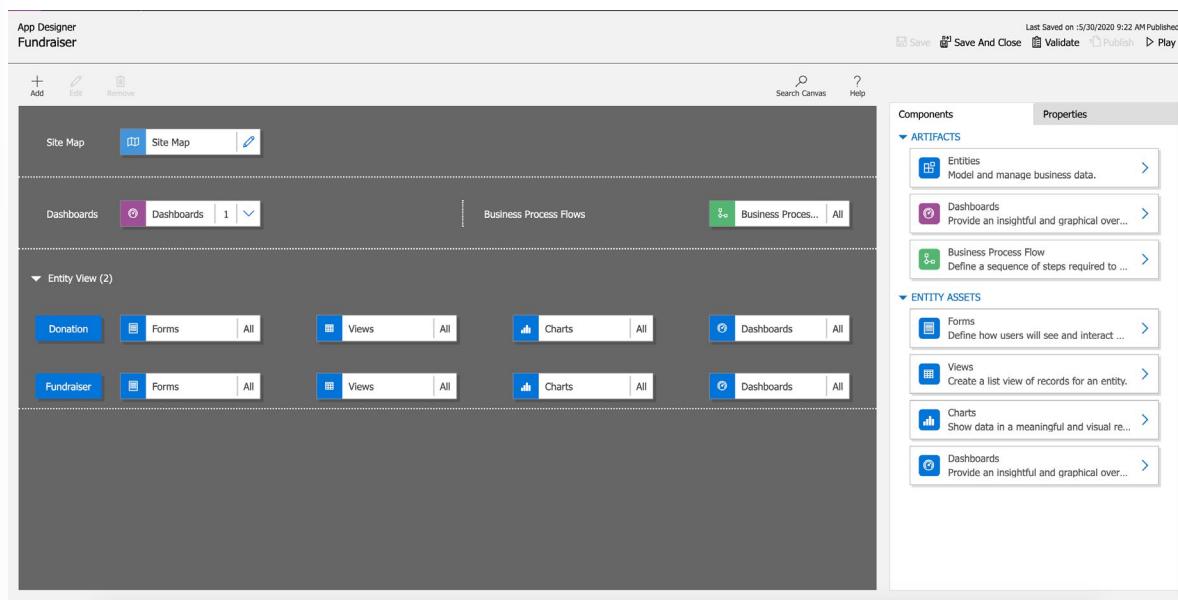
As an App Maker, before you begin building your Power Apps solution, it's recommended to go through a design process.

When designing your Power Apps solution, there are several different factors to consider:

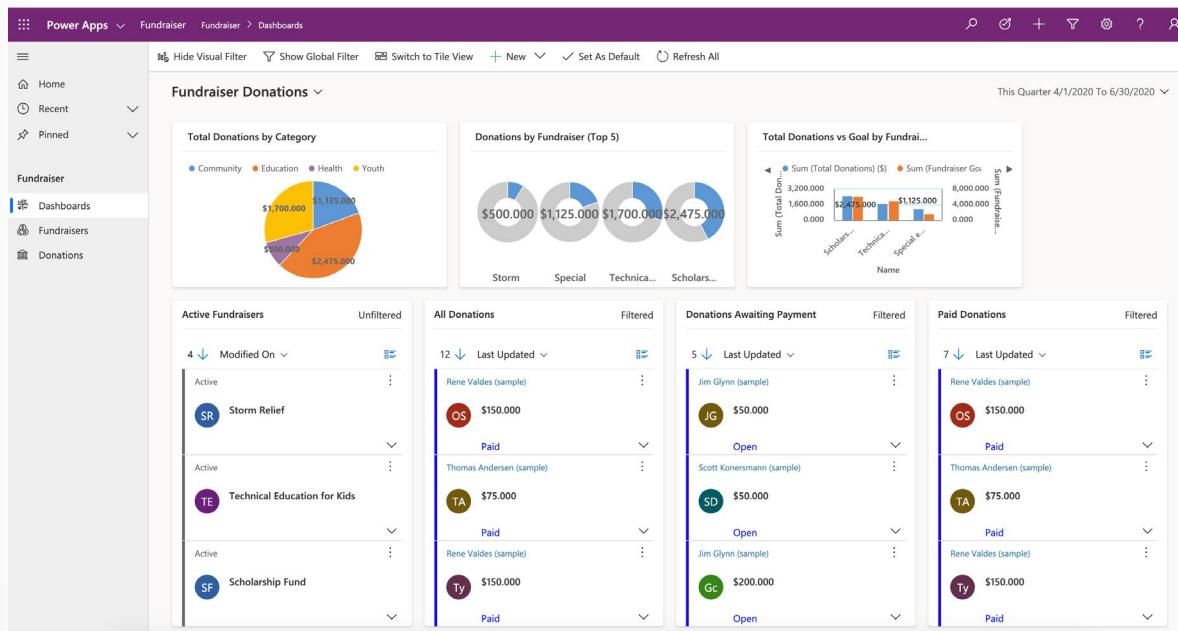
- Business requirements
- Data Model
- Business Logic
- Output

By going through a simple design process, you can flush out any minor issues before they become a larger problem once the app is in production.

Here is a quick look at the App Designer for an example Model-driven app called "Fundraiser."



When the app is put into Play Mode, it looks completely different.



## Understand the needs of the user

With Model-driven apps, the name says it all. Your primary design goal is to get your Common Data Service data model in order. With that in place, you can connect Power Apps, and a Model-driven app will be created for you from that model.

Model-driven apps are created using the App Designer. You will choose the entities, dashboards, Business Process flows, forms, and other components that you want to make available in your app, and then the app will be created for you. This means you will need to spend more time understanding what your user needs than how it is going to look.

## Business Requirements

The first step in the process is to understand your business requirements. Work with the app stakeholders to consider your security, accessibility, data, and design needs.

For security, the Common Data Service has a robust security model. You will want to consider how securing your app's data affects your app and what security model best supports your business needs. There are lots of options available, including hierarchy security, row-level security, to name a few. You will need to confirm your data is secured to meet your needs, and then your app will honor that security. For more information, see **Security in Common Data Service**<sup>4</sup>.

During this process, you will also want to identify any government regulations or authentication/authorization requirements (if applicable). You may want to implement multi-factor authentication but will need to think about how this will affect users connecting to your application. You don't necessarily have to have all the answers to your questions here; you just want to flush out all of the requirements.

Finally, does your app need to be available when the user is disconnected from the internet? This is called Offline Mode and is supported by the Common Data Service and Model-driven apps when using iOS or Android clients. It does require additional design considerations. For more information, see **Set up mobile offline synchronization**<sup>5</sup>.

## Data Model

As you begin the data modeling process, there are a couple of important questions to ask yourself:

- What type of data will your solution be storing and or collecting?
- How will this data relate or coincide with the other data you are working with?

These questions are important when designing a model-driven application because of how model-driven applications function. Remember, model-driven applications use a metadata-driven architecture. This means a large portion of the model-driven app is based on how your data is modeled, and there is no need to write custom code to alter the app design. To expand on this a little further, when thinking about Metadata this simply means "data about data" and this data defines the structure stored in the system.

You can view the app metadata by reviewing the Entity in the Common Data Service.

Name	Fundraiser Goal	Owner	Total Donations	Created On
Scholarship Fund	\$2,000,000	Jill Taylor	\$825,000	5/30/2020 5:27 AM
Special events	\$500,000	Jill Taylor	\$375,000	5/30/2020 5:27 AM
Storm Relief	\$1,200,000	Jill Taylor	\$250,000	5/30/2020 5:27 AM
Technical Education for Kids	\$1,200,000	Jill Taylor	\$425,000	5/30/2020 5:27 AM

You can also view the app metadata by putting the app in Play mode.

<sup>4</sup> <https://docs.microsoft.com/power-platform/admin/wp-security?azureportal=true>

<sup>5</sup> <https://docs.microsoft.com/dynamics365/mobile-app/setup-mobile-offline-for-admin?azureportal=true>

The screenshot shows the Microsoft Power Apps interface. On the left, there's a navigation sidebar with options like Home, Recent, Pinned, Fundraiser, Dashboards, Fundraisers (which is selected), and Donations. The main area is titled 'Active Fundraisers' and lists several entries with columns for Name, Fundraiser Goal, Owner, Total Donations, and Created On. The entries include 'Scholarship Fund', 'Special events', 'Storm Relief', and 'Technical Education for Kids'. A search bar at the top right says 'Search this view'.

In the example above, for the Fundraiser Entity, there are several pieces of metadata being collected, such as:

- Name
- Fundraiser Goal
- Owner
- Total Donations
- Created On

Each solution you develop and deploy will have its own set of metadata to collect. This basic understanding of metadata is important as you continue the design process and modeling your app data.

As you think about your data model also think about field types. When adding fields to your entity in the Common Data Service, the field type you choose will determine how users enter and view that in your Model-driven app. Option sets show as dropdowns, currency shows with currency symbols, while decimal numbers don't. These little changes in the entity can have a profound effect on how your user experiences your app.

**Note:** If a field type needs to be changed to a different field type, (i.e. text field to an option set), then you will need to delete that field and recreate with the correct field type. This will cause you to lose any data associated with that field.

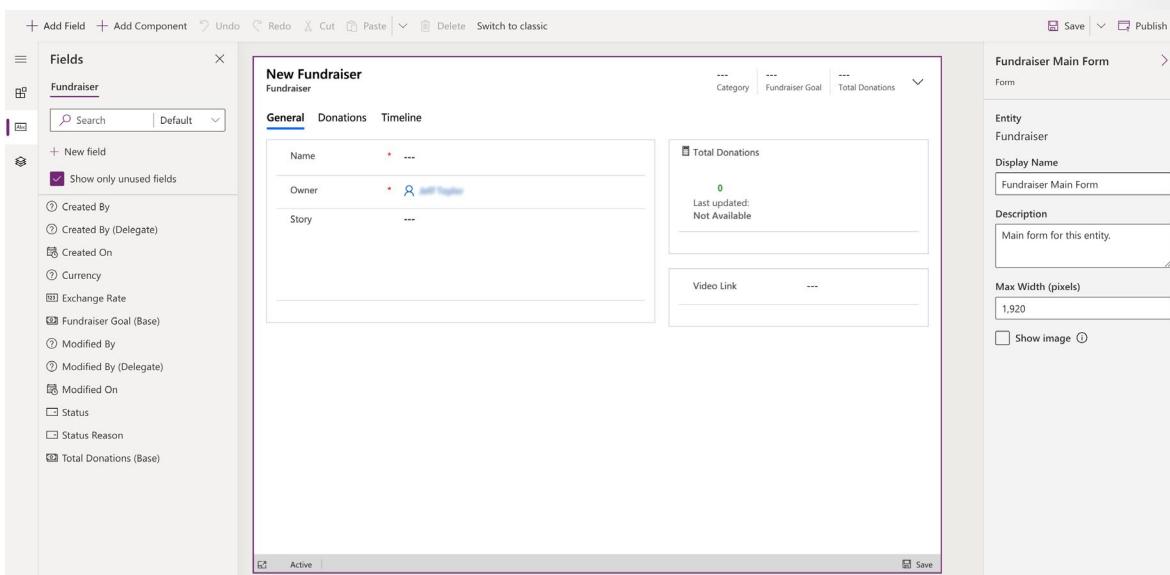
The screenshot shows the Microsoft Common Data Service Entity Fields page for the 'Fundraiser' entity. The left sidebar has sections for Home, Learn, Apps, Create, Data (which is expanded), Entities (selected), Option Sets, Dataflows, Export to data lake, Connections, Custom Connectors, Gateways, Flows, AI Builder (expanded), and Solutions. The main area shows the 'Fields' tab for the 'Fundraiser' entity. It lists various fields with columns for Display name, Name, Data type, Type, Customizable, Required, and Searchable. Fields include Category (Option Set, Managed, Optional, Searchable), Created By (Lookup, Standard, Optional, Searchable), Created By (Delegate) (Lookup, Standard, Optional, Searchable), Created On (Date and Time, Standard, Optional, Searchable), Currency (Lookup, Custom, Optional, Searchable), Exchange Rate (Decimal Number, Custom, Optional, Searchable), Fundraiser (Unique Identifier, Standard, Required, Searchable), Fundraiser Goal (Currency, Managed, Optional, Searchable), Fundraiser Goal (Base) (Currency, Managed, Optional, Searchable), Import Sequence Number (Whole Number, Standard, Optional, Searchable), Modified By (Lookup, Standard, Optional, Searchable), Modified By (Delegate) (Lookup, Standard, Optional, Searchable), Modified On (Date and Time, Standard, Optional, Searchable), Name (Text, Managed, Required, Searchable), Owner (Owner, Standard, Required, Searchable), Owning Business Unit (Lookup, Standard, Optional, Searchable), and Owning Team (Lookup, Standard, Optional, Searchable). There are also tabs for Relationships, Business rules, Views, Forms, Dashboards, Charts, Keys, and Data.

## User Interface (UI) and User Experience (UX)

When building a Model-driven app, most of the UI and UX are predetermined for you. You define the data model to build from, and then Power Apps determines the controls used in the app. You can influence these controls by determining what entity assets you include. You define in the App Designer What Forms, Views, Charts, and Dashboards are used in the app. You also control the navigation options via the Site Map. As you are planning your app, determine which components are needed in the app design, and create them before building your app.

To continue building off of the example we've been using throughout this module, below is a simple Model-driven Form, which captures various pieces of information for creating a New Fundraiser.

Here is an example of what the New Fundraiser form looks like when editing from the App Designer.



## Business Logic

When incorporating business logic in your app, there are two primary options available. You can set **Business Rules** on your Common Data Service entities or you can build **Business Process Flows**.

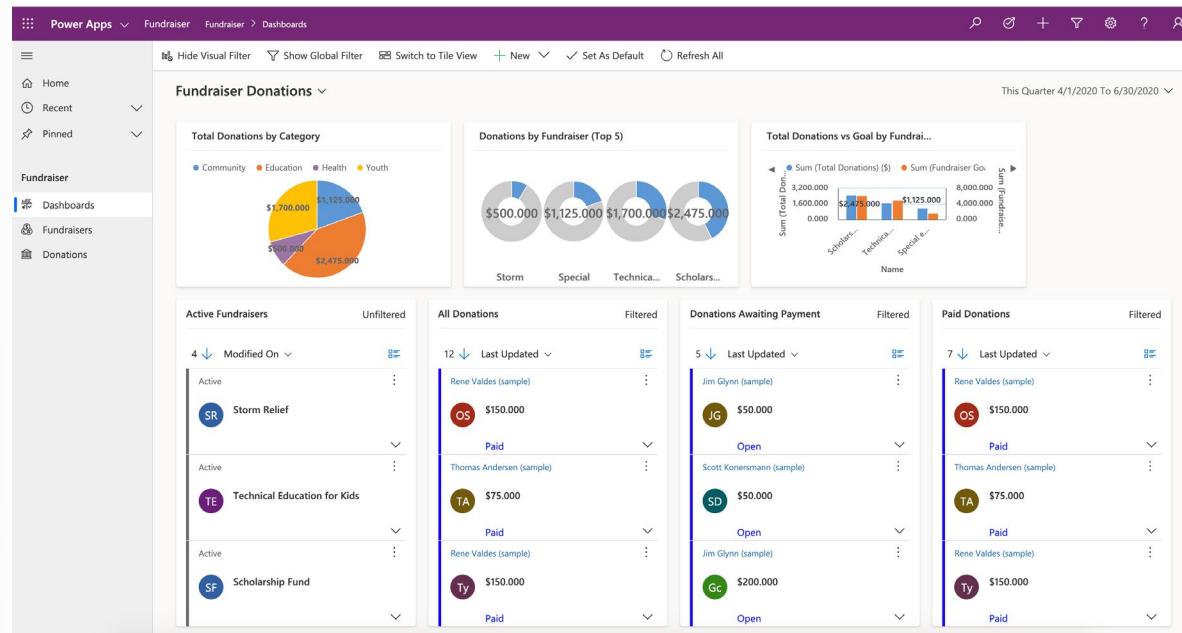
With **Business Rules**, you will define behaviors at the data layer. This is great for changing when a field is required, setting a default value, or even showing or hiding a field based on a criteria. An example could be an entity for tracking expenses. You could have a field for type of travel and then build a business rule that says if they choose automobile then the mileage field is required, else it is optional. This gives you great power to make sure you maintain data consistency in all scenarios.

**Business Process Flows** are used to guide users through using your app. These workflows can provide visuals on next steps based on the status of the data and facilitate other actions that you want to occur as the user uses the app. Business Process Flows let you bring automation to your app and make it more of a guided experience than just a place to enter data.

## Output

A common output need for apps is to visualize the data. For this requirement, you can implement Dashboards with custom filters and visual graphics to tie all of this data together right in your app. When creating your Dashboards, make sure it's simple for your users to consume without overwhelming them

with all the data. Provide high-level snapshots of your data and allow them to use filters to dive deeper into the data if needed.



## Additional third-party solutions and app accelerators

It is also important to know about the different App accelerators and third-party solutions available to you. Depending on the industry you are in, Health, Financial, Banking, Education, Non- Profit, Automotive, or Media, Microsoft has released a number of accelerators or foundational components to assist you with quickly standing up your solution. For more information, see [Industry accelerators overview<sup>6</sup>](#).

For more information, see [Planning a Power Apps project<sup>7</sup>](#).

## Exercise - Create a model-driven app

In this unit, you'll create a model-driven app by using one of the standard entities that's available in your Microsoft Power Apps environment.

### Create a model-driven app

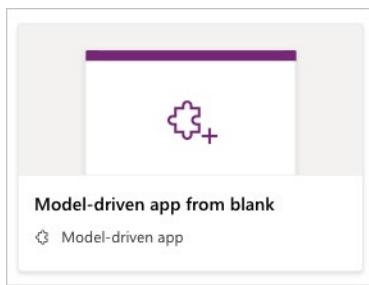
1. Sign in to [Power Apps<sup>8</sup>](#) by using your organizational account.
2. Select the environment you want, or go to the [Power Apps admin center<sup>9</sup>](#) to create a new one.
3. On the **Home** page, select the **Model-driven app from blank**.
4. Click **Create**.

<sup>6</sup> <https://docs.microsoft.com/common-data-model/industry-accelerators?azureportal=true>

<sup>7</sup> <https://docs.microsoft.com/powerapps/guidance/planning/introduction>

<sup>8</sup> <https://make.powerapps.com/>

<sup>9</sup> <https://admin.powerapps.com/>

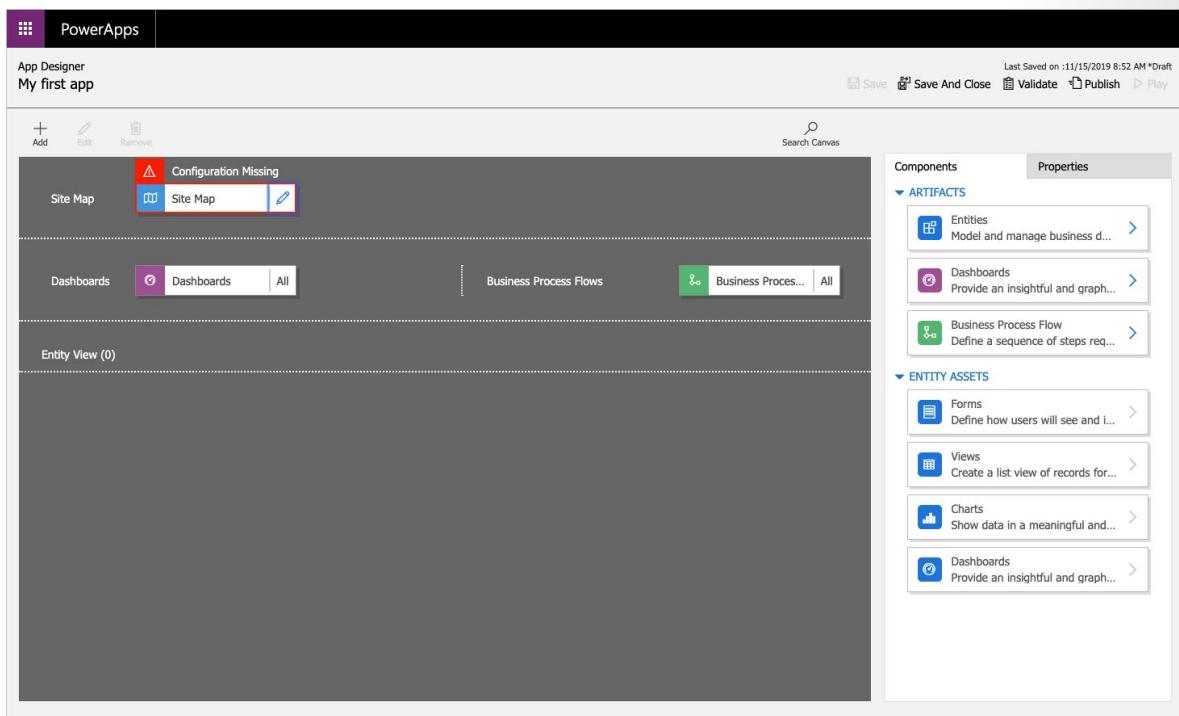


5. On the **Create a New App** page, enter a name and description for the app.
6. Select **Done**. Your new app appears in the App Designer, and you can now add components to it.

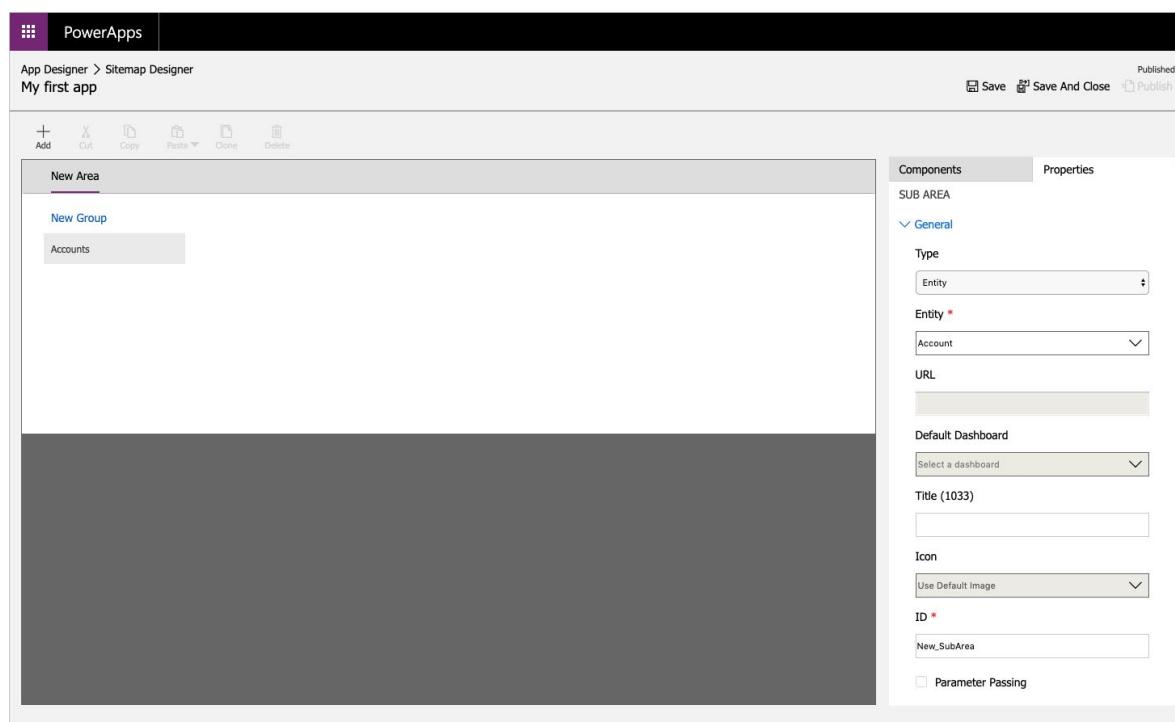
## Add components to your app

You add components to your app by using the App Designer.

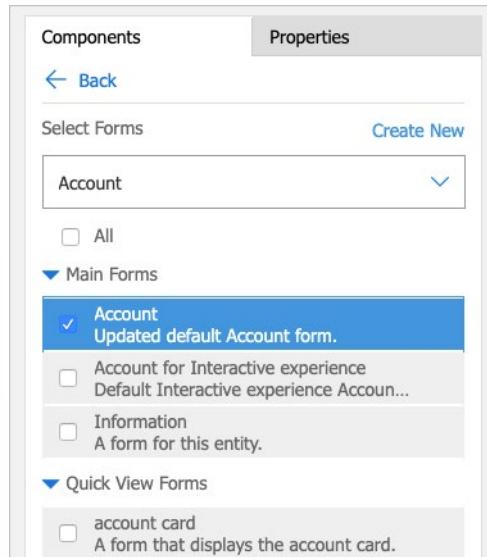
1. Select the **Open the Site Map Designer** pencil icon to open the site map designer.



2. In the site map designer, select **New Subarea**, and then, in the right pane on the **Properties** tab, select the following properties:
  - **Type:** Entity
  - **Entity:** Account



3. Select **Save And Close**.
4. In the App Designer, select **Forms**, and then, in the right pane under **Main Forms**, select the **Account** form.



5. In the App Designer, select **Views**, then select the following properties:
  - Active Accounts
  - All Accounts
  - My Active Accounts
6. In the App Designer, select **Charts**, then select the **Accounts by Industry** chart.

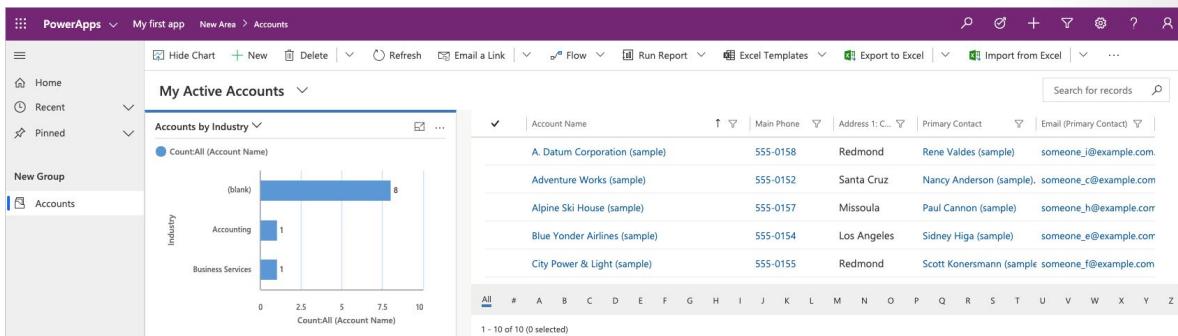
7. On the App Designer toolbar, select **Save**.

## Publish your app

On the App Designer toolbar, select **Publish**. After you publish the app, it's ready for you to run or share with others.

Above My Active Accounts, select **Show Chart**.

If the sample data for your accounts does not have an Industry populated, go into a few accounts and add an Industry. Once you have updated a few accounts with an industry the chart will update as well.



## Incorporate business process flows

You can help ensure that people enter data consistently and follow the same steps every time they work with a customer by creating a business process flow.

For example, you might want to create a business process flow to have everyone handle customer service requests the same way, or to require that people get approval for an invoice before submitting an order. Business process flows use the same underlying technology as other processes, but the capabilities that they provide are very different from other features that use processes. To learn how to create or edit a business process flow, see [Create a business process flow<sup>10</sup>](#)

## Use business process flows

Why should you use business process flows? Business process flows provide a guide for people to get work done. They provide a streamlined user experience that leads people through the processes their organization has defined for interactions that need to be advanced to a conclusion of some kind. This user experience can be tailored so that people with different security roles can have an experience that best suits the work they do.

You should use business process flows to define a set of steps for people to follow to take them to a desired outcome. These steps provide a visual indicator that tells people where they are in the business process.

Business process flows reduce the need for training because new users don't have to focus on which entity they should be using. They can let the process guide them.

<sup>10</sup> <https://docs.microsoft.com/power-automate/create-business-process-flow>

You can configure business process flows to support common sales methodologies that can help your sales groups achieve better results.

For service groups, business process flows can help new staff get up to speed more quickly and avoid mistakes that could result in unsatisfied customers.

## System business process flows

The following business process flows can be found in Power Automate. To understand how business process flows work, review these system business process flows:

- Lead to Opportunity Sales Process
- Opportunity Sales Process
- Phone to Case Process

## Multiple entities in business process flows

You can use a business process flow for a single entity or to span multiple entities. For example, you may have a process that begins with an opportunity, then continues to a quote, an order, and then an invoice, before finally returning to close the opportunity.

You can design business process flows that tie together the records for up to five different entities into a single process so that people using the app can focus on the flow of their process rather than on which entity they are working in. This way, they can easily navigate between related entity records.

## Multiple business process flows are available per entity

Not every user in an organization may follow the same process and different conditions may require that a different process be applied. You can have up to 10 active business process flows per entity to provide appropriate processes for different situations.

## Scenario

In the exercise in the previous module of this learning path, you created the Prospects entity in Common Data Service and imported the existing leads, now in this exercise you will use this data to create a model-driven app. This app will allow the sales team to enter and edit leads on the go and keep the managers up to date on the current leads and forecasted revenue.

## Create the model-driven app for the prospects entity

1. Sign in to **Power Apps**<sup>11</sup> by using your organizational account.
2. Select the environment you want, or go to the **Power Apps admin center**<sup>12</sup> to create a new one.
3. On the **Home** page, select the **Model-driven app from blank**.
4. Select **Create**.

---

<sup>11</sup> <https://make.powerapps.com/?azure-portal=true>

<sup>12</sup> <https://admin.powerapps.com/?azure-portal=true>

5. Enter the following and Select **done**.
  - Name: Prospect Entry
6. Select the **Open the Site Map Designer** pencil icon to open the site map designer.
7. Select **New Subarea**.
8. On the right, for **Title** enter **Prospects Area**.
9. Click the drop down for **select a type**, then choose **Entity**.
10. For Entity, select **Prospects**.
11. Select **New Group**.
12. On the right, for **Title** enter **Prospects Group**.
13. Click **Save**.
14. Click **Publish**.

## Creating a chart

1. Select **Charts**.
2. Select **Create New**.
3. For the Chart Name, enter **Forecasted Revenue by Stage**.
4. For Legend Entries (Series), check the box, and select the **Forecasted Revenue** field.
5. For Horizontal (Category) Axis Labels, select the **Stage** Field.
6. Select **Save and Close**.
7. Back on the App Designer, select the checkbox next to **Forecasted Revenue by Stage**.
8. Select **Save**.
9. Select **Publish**.

## Creating the form

You can attempt to create a new form in the App Designer but if you run into any issues, you can follow the steps below to create a new main form.

1. Go to the Power Apps Home Page, and on the left, Select **Data**.
2. Select **Entities**.
3. Locate and select the **Prospects** entity.
4. Select **Forms**.
5. Select the drop-down arrow next to Add form, and then select **Main Form** from the drop down. A new window will open.
6. Drag the **Stage** field from the right and place it below the **Owner** field in the center.
7. Drag the **Contract Amount** field and place it below the **Stage** field.
8. Drag the **Probability** field and place it below the **Contract Amount** field.

9. Drag the **Forecasted Revenue** field and place it below the **Probability** field.
10. Now select **Forecasted Revenue** and then select **Change Properties** on the ribbon.
11. Select the checkbox for **Read-only field**.
12. Select **Save**.
13. Select **Publish**.
14. Close the window.
15. Select **Done**.

## Summary

Congratulations on creating your first model-driven app!

Defining and enforcing consistent business processes is a key aspect of model-driven app design. Consistent processes help ensure that your app users can focus on their work and not have to remember to perform a set of manual steps.

Let's review what you've learned:

- Model-driven app design is an approach that focuses on quickly adding components to apps. These components include dashboards, forms, views, and charts.
- Little or no code is required to build model-driven apps.
- Model-driven design uses metadata-driven architecture so that designers can customize their apps.
- The best way to get started building model-driven apps is to use sample apps and data; then customize the apps.

# Create and manage entities

## Identify entities and entity types

An entity is similar to a table in a database. It is a logical structure containing records that are made up of fields. Standard entities differ from a simple database table because they have nearly 100 properties that are used to define relationships between other standard entities and how the entity is used within Dynamics solutions from Microsoft.

### Types of entities

The four types of entities are:

- **Standard** - The base set of entities that are created for every instance of a Common Data Service database. You can add more fields to any entity, but you cannot delete any field from a standard, premium, or restricted entity.
- **Complex** - Entities that contain complex, server-side business logic, including real-time workflows or plug-ins. Some of the entities that are used in Dynamics 365 applications are complex. To use complex entities, users are required to have a P2 or Dynamics 365 license. Care must be taken if you add server-side logic to ensure that users have the proper license to use the complex entity. Additional information about complex entities can be accessed by following the link within the summary unit of this module.
- **Restricted** - Certain entities that are tied to Dynamics 365 application functionalities require each user to have the corresponding license for that Dynamics 365 application if they want to create, update, or delete records within the restricted entities. Additional information about restricted entities can be accessed by following the link within the summary unit of this module.
- **Custom** - Are created for a specific business application. All licensed users of Common Data Service can access custom entities if they are assigned proper security permissions to do so.

## Create a custom entity

Adding a new entity to a Common Data Service database is a simple process.

[!TIP]

Make sure that you always verify that the functionality of the entity that you want to create already exists in the standard entities listed in the summary unit of this module.

After you have verified existing entities and have decided that you need a new custom entity, follow these steps:

1. Sign in to Power Apps.
2. On the navigation pane, select or tap **Data** to expand it and then select or tap **Entities**.

3. Select **New Entity** in the command bar.
4. In the New entity panel, in the **Display name** box, enter the name of your new custom entity, and then optionally enter a description (descriptions are helpful if other people will use this entity). Other fields in the panel are auto populated, such as the following. When you are finished, select **Next**.

**Plural display name** - This field is auto populated when you enter a display name, but you can change it if needed. The plural display name is the name of the entity in the Common Data Service WebAPI and is used when you are interacting with this entity from Power Apps or Power Automate.

**Name** - This field is also auto populated when you enter a display name. The prefix was set up when the environment was created and ensures that the entities you create can be exported and imported into other environments without conflicting with other entity names. You can change this prefix by updating the prefix on your Publisher for the Common Data Service Default Solution. To keep existing apps from breaking, you cannot change the name after saving the entity.

5. On the **Entity details** page, select or tap the **Primary Name** field to open the Primary Name panel, and then in the **Display name** box, enter a name of **Primary Name**. In the **Name** box, replace **Primary Name** with a name of your choosing for the new primary field, and then select or tap **Done**.

By default, every entity contains a **Primary Name** field that is used by lookup fields when establishing relationships with other entities. Typically, the **Primary Name** field stores the name description of the data that is stored in an entity's record. You can update the name and display name of the **Primary Name** at any time.

6. To **add a field to the entity**, use the following procedure:
  - a. In the command bar, select or tap **Add field** to open the Field properties panel.
  - b. In the **Display name** box, enter the name of the field that you want to add.
  - c. From the **Data type** drop-down list, select the type of data that you want to add.
  - d. Select or tap the **Required** check box if desired.
  - e. Select or tap **Done**.

You will learn much more about fields in a later module.

[!NOTE]

All entities have read-only system fields. By default, system fields are not shown in the list of fields, even though they exist within the entity. To view all fields, change the filter on the command bar from **Default** to **All**. For more information on the metadata that is related to an entity, refer to Entity metadata.

7. Select **Save entity** to save your entity and make it available for use in apps.

[TIP]

You will need to assign permissions to the new custom entity in an existing or new user security role. This way, users can add, view, edit, or delete data in the entity. Refer to unit four in module two if you need additional information on user security roles.

## Enable attachments within an entity

You might want to allow users the ability to upload files and save with a record in an entity. Common Data Service entities allow users to add an attachment by turning on the attachment property that is associated with that entity. If you want to let users upload one or more files into a record within an entity, use the following procedure.

[NOTE]

Attachments will be enabled if you are working with a standard entity.

1. Sign in to Power Apps
2. Select **Data** and then **Entities** on the menus on the left-hand pane.
3. Select an **Entity** and then select **Settings**.
4. Select the **Enable Attachments** check box.
5. Select **Save**.

[NOTE]

You cannot undo the addition of attachments to an entity.

## Licensing requirements for each entity type

As Unit 1 explained, complex and restricted entities require additional licensing. The following table summarizes license requirements for each type of entity.

Standard and Custom Entities	Microsoft 365, P1, P2, Dynamics 365
Complex Entities	P2, Dynamics 365
Restricted Entities	Specific Dynamics 365 application licenses

For more information on complex and restricted entity license requirements, see the link to licensing that is provided in the summary unit in this module.

# Exercise - Create a new custom entity and enable attachments

## Create a new custom entity

This hands-on lab shows you how to create a new custom entity in Common Data Service. Follow along to create a custom entity in your environment.

1. Sign in to Power Apps.
2. Select the **Environment** where you want to add a new custom entity within the menu at the top of the page.
3. Select **Data** in the panel on the left-hand side of the Power Apps Portal to expand the available choices.
4. Select **Entities** under the Data options on the left-hand side of the portal.
5. Select **New Entity** in the menu at the top of the list of entities.
6. Enter the following values:
  1. Display name: **PC**
  2. Description: **Custom entity to track PCs**
7. Select **Next** and then, after the default fields are shown, select the **Save Entity** button at the lower right of the screen.

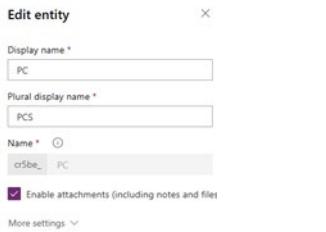
The new entity called PC will be shown in the list of entities in this environment. In the next hands-on labs, you will add the ability to enable attachments and some custom fields.

The screenshot shows the Power Apps Entities screen. On the left, there's a navigation sidebar with Home, Learn, Apps, Create, Data, and Entities. Under Entities, there are Option Sets, Data Integration, Connections, Custom Connectors, Gateways, Flows, and AI Builder (preview). The main area shows 'Entities > PC'. The 'Fields' tab is selected, showing a table with columns: Display name, Name, Data type, Type, Required, and Searchable. Two fields are listed: 'Name' (crbac\_name, Text, Custom, required, searchable) and 'New Field' (crbac\_newfield, Text, Custom, required, searchable). At the bottom right, there are 'Discard' and 'Save Entity' buttons, with 'Save Entity' being highlighted by a red box.

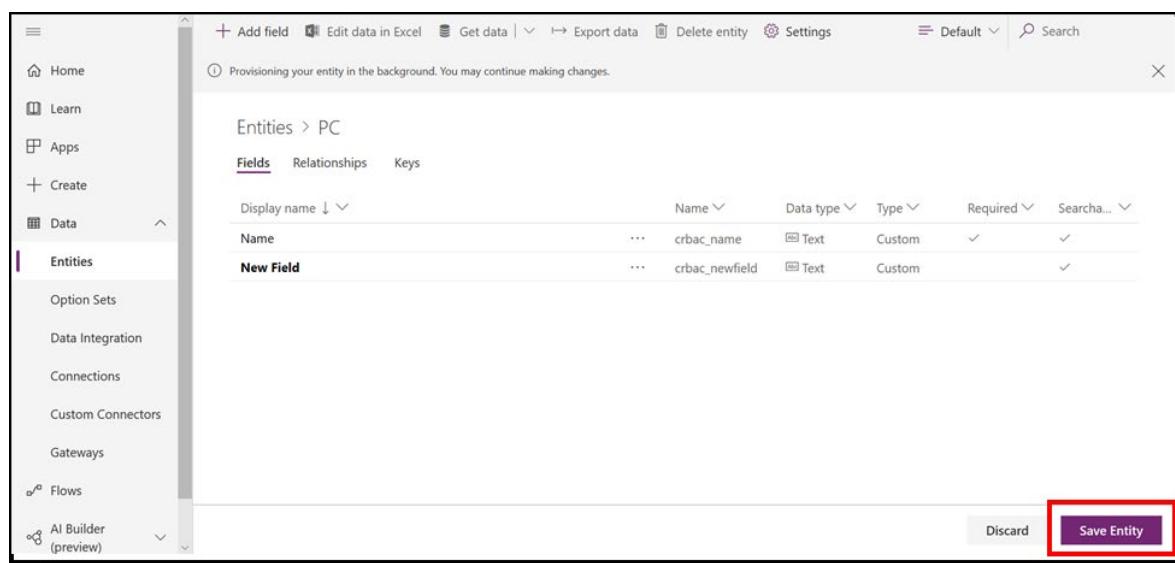
## Enable attachments

In this lab, you will enable attachments to be added to a record in the new custom entity PC Manufacturers, which was created in the last hands-on lab.

1. Sign in to Power Apps.
2. Select the environment where you added the new custom entity PC Manufacturers.
3. Select **Data** on the panel on the left-hand side of the Power Apps Portal to expand the available choices.
4. Select **Entities** under the Data option on the left-hand side of the portal.
5. Select the new custom entity PC Manufacturers in the list of entities. This will open the list of fields in the entity.
6. Select **Settings** on the menu above the entity fields.
7. Select the **Enable Attachments** check box.
8. Select **Done**.



9. Select the **Confirm Changes** button when the dialog appears.
10. Select the **Save Entity** button on the bottom of the screen to save the change to the entity settings that allow attachments.



## Summary

In this module, you learned:

- What an entity is and how it's used in Common Data Service.
- What types of entities are available in Common Data Service.
- How to view standard entities and create a custom entity.
- How to enable attachments within an entity.
- What licensing requirements to apply to use each type of entity.

The following links were referenced within this module. Check them frequently because Common Data Service is a rapidly-changing service from Microsoft.

**Licensing and Pricing<sup>13</sup>** - Find the right Power Apps plan for your business needs. Note that Common Data Service is licensed with Power Apps and Dynamics 365.

**Complex entities and licensing requirements<sup>14</sup>** - Entities that include complex server-side logic may have additional licensing requirements.

**Restricted entities requiring Dynamics 365 licenses<sup>15</sup>** - App makers can use most of the entities available within Common Data Service to create apps and flows for users, but some are restricted.

<sup>13</sup> <https://powerapps.microsoft.com/pricing/>

<sup>14</sup> <https://docs.microsoft.com/powerapps/maker/common-data-service/data-platform-complex-entities>

<sup>15</sup> <https://docs.microsoft.com/powerapps/maker/common-data-service/data-platform-restricted-entities>

# Create and manage fields within an entity in Common Data Service

## Define fields in Common Data Service

Fields are a way to store a discrete piece of information within a record in an entity. Fields have types, meaning that you can store data of a certain type in a field that matches that data type. For example, if you have a solution that requires dates, then you would store the date in a field with the type of Date. Similarly, if you want to store a number, then you store the number in a field with the type of Number.

The number of fields within an entity varies from a few fields to a hundred or more. If you need more than a few hundred fields in an entity, you might want to reconsider how you are structuring data storage for your solution because, likely, there is a better way.

Every database in Common Data Service starts with a standard set of entities and each standard entity has a standard set of fields.

**Tip:** Always use standard entities and fields when possible. You can rename an entity if that makes the entity more understandable in the context of your solution. Always review the list of standard entities and make sure a standard entity will not meet your needs before you create a new entity.

## Field types in Common Data Service

The field type determines the values that can be stored within that field. All fields have one and only one field type. The field types that are available in Common Data Service include:

**Text** - A text value intended to be displayed in a single-line text box.

**Text Area** - A text value intended to be displayed in a multi-line text box. If you require more than 4,000 characters, use a Multiline Text data type.

**Email** - A text value that is validated as an email address and rendered as a mailto link in the field.

**URL** - A text value that is validated as a URL and rendered as a link to open the URL.

**Ticker Symbol** - A text value for a ticker symbol that will display a link that will open to show a quote for the stock ticker symbol.

**Phone** - A text value that is validated as a phone number rendered as a link to initiate a phone call by using Skype.

**Whole Number** - A number value presented in a text box.

**Duration** - A number value presented as a drop-down list that contains time intervals. A user can select a value from the list or type an integer value that represents the number of minutes.

**Timezone** - A number value presented as a drop-down list that contains a list of time zones.

**Language** - A number value presented as a drop-down list that contains a list of languages that have been enabled for the environment. If no other languages have been enabled, the base language will be the only option. The saved value is the Locale Identifier (LCID) value for the language.

**Currency** - A money value for any currencies that are configured for the environment. You can set a level of precision, or you can choose to base the precision on a specific currency or a single standard precision that is used by the organization.

**Decimal Number** - A decimal value with up to 10 points of precision. See [Using the right type of number<sup>16</sup>](#) for more information.

**Floating Point Number** - A floating point number with up to 5 points of precision. See [Using the right type of number<sup>17</sup>](#) for more information.

**Image** - Displays a single image for each record in the application. Each entity can have one image field. The name that you enter when creating an image field will be ignored. Image fields are always named **EntityImage**. You can only have one image field for each entity.

**Multi Select Option Set** - Displays a list of options where more than one option can be selected.

**Multiline Text** - A text value intended to be displayed in a multi-line text box. This value is limited to a maximum of 1,048,576 characters. You can also set a lower Max Length.

**Option Set** - Displays a list of options where only one can be selected.

**Two Options** - Displays two options where only one can be selected. You can choose which labels are displayed for each option. The default values are Yes and No.

**Tip:** You can add any combination of fields to a custom or standard entity to meet your needs, but you can't delete a standard field from a standard entity.

## Add a field to an entity

You can add fields when you create a new custom entity, or you can add fields to an existing entity at any time. Adding a new field is the same whether you are creating a new entity or adding to an existing entity.

**TIP:** Before you create a custom field, evaluate whether an existing field meets your requirement.

1. Sign in to Power Apps.
2. Select the **Data** option on the left-hand menu.
3. Open an existing entity or create a new entity.
4. Select **Add field** on the menu at the top of the page.

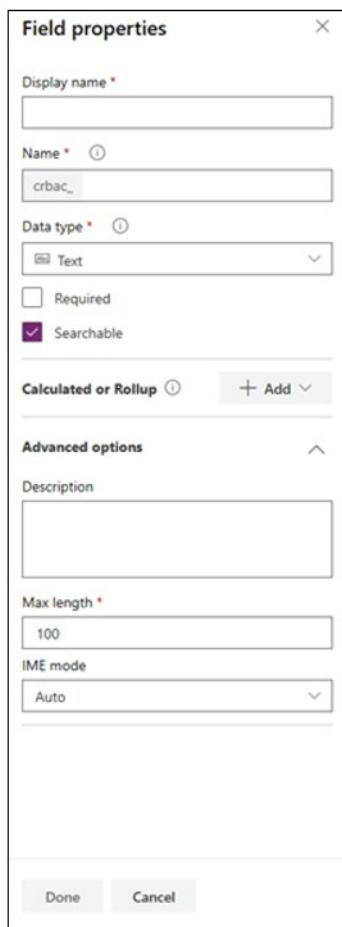
---

<sup>16</sup> <https://docs.microsoft.com/powerapps/maker/common-data-service/types-of-fields#using-the-right-type-of-number>

<sup>17</sup> <https://docs.microsoft.com/powerapps/maker/common-data-service/types-of-fields#using-the-right-type-of-number>

5. Enter information in the following fields:

- a. **Display name** - The name that is shown to users.
- b. **Name** - This is the internal name that is used by your application.
- c. **Data type** - This is the type of data that you want to store in the field.
- d. **Required** - Select this check box if you want to ensure that this field always has a value when a user tries to add a record to this entity.
- e. **Searchable** - Clear this check box for fields for the entity that you do not use. When a field is searchable, it appears in **Advanced Find** and is available when you are customizing views. Clearing this check box reduces the number of options that are shown to people who are using **Advanced Find**, and helps make it easier for users to create custom views without seeing unused fields. Clear the **Searchable** check box when you are working with fields in a standard or complex entity that you do not use.
- f. **Calculated or Rollup** - Use to define a calculation or a rollup in this field.
- g. **Description** - This is a description of the field.
- h. **Max Length** - Use to define the maximum length of the data that a user can enter in this field. Note that this is used with text fields.
- i. **Minimum and Maximum Values** - This field is available and used with number fields.



## Create a primary name field

A **Primary Name** field is always created by Common Data Service when you create a new custom entity. This is the first field that is listed and available so you have a way to identify a record in the new custom entity by using a business value or an auto-generated whole number.

The **Primary Name** field is not the same as the internal key field that is also auto-generated when you create a new custom entity. The **Primary Key** field is a GUID. You can change the name of the **Primary Name** field to make it friendlier for business users. You can also choose between a text field or auto-generated whole number.

*Tip:* If you want to make the primary field unique, then make a key and assign the **Primary Name** field to the new key.

*Note:* If you have duplicated data in the **Primary Name** field in multiple records, the key will not be created. You can only create a key if data in the field's existing record is unique across all records in the entity.

## Restrictions that apply to fields in an entity

When using Common Data Service for your business solutions, you should keep a few restrictions in mind.

Maximum Number of fields in an entity:

- There is no hard upper limit on the number of fields that you can have in an entity, but there is an upper boundary due to limits in how much data you can store in a single record. It is difficult to provide a specific number because each type of field can use a different amount of space. The upper limit depends on the total space that is used by all the fields for the entity.
- As a rule, you should have less than a few hundred fields in an entity. If you have more than a few hundred fields in an entity, then you should look at restructuring how you have designed the entities in your solutions and try to split the entity with an excessive number of fields into more than one entity.

**Rollup** fields:

- Max of 100 for each organization
- Max of 10 for each entity
- Can't trigger workflows

**Option set** fields:

- Option set fields provide a set of options that will be displayed in a drop-down control on a form or in picklist control when you are using Advanced Find. Dynamics 365 can support thousands of options within an Option set, but you should not consider this as the upper limit. Usability studies have shown that people have trouble using a system where a drop-down control provides a large number of options.
- Use an Option set field to define categories for data. Don't use Option set fields to select categories that actually represent separate items of data. For example, rather than maintaining an Option set field that stores each of hundreds of possible manufacturers of a type of equipment, consider creating an entity that stores references to each manufacturer and use a lookup field instead of an Option set.

**Primary Name** field

- You can only have one primary field for each custom entity.

**Searchable** fields

- You can only have five searchable fields for each entity.

## Create an auto numbering field

Autonumber fields automatically generate alphanumeric strings whenever they are created. You can customize the format of these fields and then rely on the system to generate matching values that automatically fill them in.

Autonumber fields appear in the Text category of fields when you create a new field. You can also activate autonumber functionality in an existing text field by opening the field and selecting **Autonumber** in the datatype dropdown. Similarly, auto number functionality can also be

disabled at any time by opening the field and selecting text as the type in the **Data type** drop-down list.

Autonumber fields offer many options for the type of auto-generated number that you want to generate. You can create any of the following autonumber type fields.

**String prefixed number** - The most common auto number format is a simple string prefixed number. When this type is selected, the autonumber will consist of an automatically incrementing number with an optional string constant prefix. For example, a string prefixed number with the prefix of Contoso would generate records such as Contoso-1000, Contoso-1001, Contoso-1002, and so on.

**Date prefixed number** - Another common auto number format is a date prefixed number. When this type is selected, the auto number will consist of an automatically incrementing number with a formatted date prefix. The date portion of the record will reflect the current date and time at which the record was created in UTC time. For example, a date prefixed number would generate records such as 2019-26-02-1000, 2019-27-02-1000, 2019-28-02-1000, and so on, depending on the current date and selected date format.

**Custom** - For more advanced makers with specific use cases, you have the option to fully customize the desired format of an autonumber field. The format might consist of string constants, automatically incrementing numbers, formatted dates, or random alphanumeric sequences. For detailed information about how to define custom formats, see **AutoNumberFormat options<sup>18</sup>**.

## Seeding a starting value

You can also set a *seed* value so you can start autonumbering at any value. The seed value of an autonumber field is the starting number that is used for the number portion of the format. For example, if you want an autonumber field to generate records such as Contoso-1000, Contoso-1001, Contoso-1002, and so on, then the desired seed value is 1000 because that is the value that your number sequence starts with. Autonumber fields have a default seed value of 1000, but you can set a custom seed value if you want.

## Create an alternate key

It is common to need a way to uniquely identify a record in an entity. By default, Common Data Service entities have a GUID as their only unique field. This GUID is called the **Primary Field** and it consists of a long string of numbers and letters that are not useful to a regular user regarding meaning or significance.

Defining a new key for an entity allows you to identify a record in a more meaningful way by using a field that is familiar to users. When you define a field as a key, Common Data Model makes sure that every entry in that key field is required and unique so you can use the key field to distinctively identify a specific record. This can be especially

---

<sup>18</sup> <https://docs.microsoft.com/dynamics365/customer-engagement/developer/create-auto-number-attributes#autonumberformat-options>

helpful if you are integrating your data with an external system that uses that ID or number to identify a record (and not the Common Data Service GUID). It also improves the search and filtering on the particular field because alternate key fields are always indexed.

Finally, keys can be based on a single field (Order ID) or a combination of fields, such as Financial year and Order ID. If you set a key on a field and try to enter duplicate data, the record will fail to save.

Note: You are limited to five alternate keys for each entity.

Set up an alternate key for an entity with the following steps:

1. Sign in to Power Apps and select the **Data** menu option in the left-hand column.
2. Select the entity that you want to add a new key to.
3. Select **Keys** from the options above the list of fields.
4. Select **Add key** in the upper left of the screen.
5. Select one or more fields that will make up the new key.
6. Give the key a name.
7. Select the **Done** button.

It will take a few minutes for Common Data Service to create the new key and indexes. Then, you can start using it in your business solution.

Tip: If you have duplicated data in a field that is used by the key in multiple records, then the key will not be created. You can only create a key if the data in the field's existing record is unique across all records in the entity.

## Add fields to a custom entity

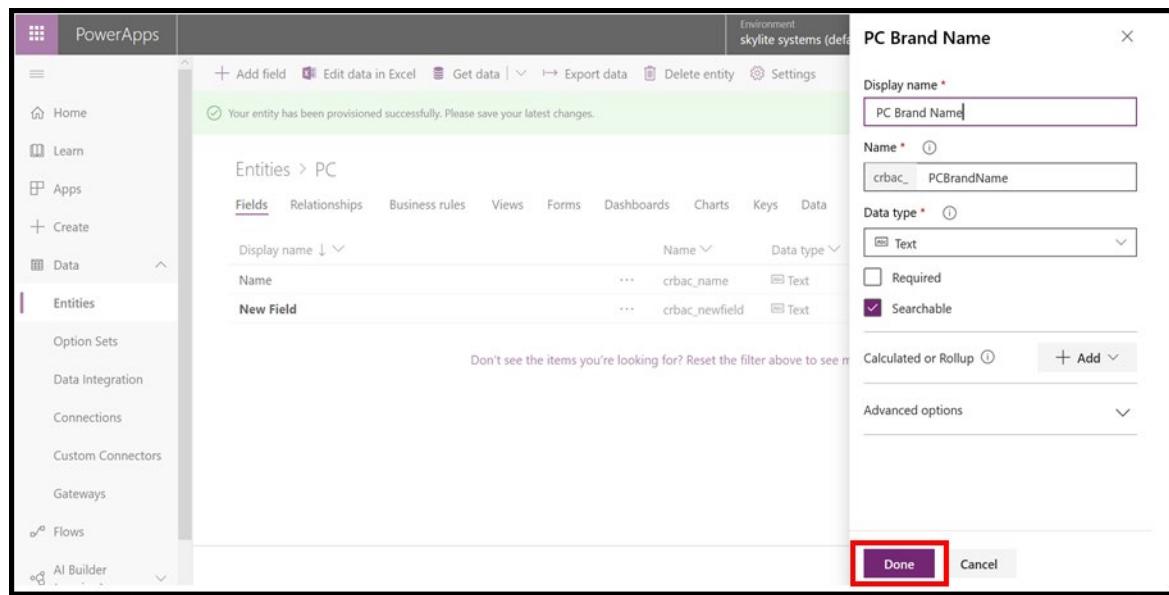
In this lab, you will add a few fields to the custom entity PC Manufacturers.

If you have not created the custom entity as a part of this learning path, you can complete the steps [here<sup>19</sup>](#) to complete these exercises.

1. Sign in to Power Apps.
2. Select the environment where you added the new custom entity PC Manufacturers.
3. Select **Data** on the panel on the left-hand side of the Power Apps Portal to expand the available choices.
4. Select **Entities** under **Data Options** on the left-hand side of the portal.
5. Select the new custom entity **PC** in the list of entities.  
This will open the list of fields in the entity.
6. Select **+ Add field** on the menu above the list of fields in the entity.

<sup>19</sup> <https://docs.microsoft.com/learn/modules/create-manage-entities/5-exercise>

7. Type **PC Brand Name** in the **Display name** field.
8. Keep the **Searchable** check box selected.
9. Select **Done**.



10. Select **+ Add field** on the menu above the list of fields in the entity.
11. Type **Description** in the **Display name** field.
12. Select **Text area** in the Data type.
13. Select the **Done** button.
14. Select the **Save entity** button to save changes.

Display name	Name	Data type	Type	Required	Searchable
Description	crsbe_description	Text	Custom	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Name	crsbe_name	Text	Custom	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
PC Brand Name	crsbe_pcbrandname	Text	Custom	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>

After you select the **Save entity** button, the custom PC Manufacturers entity should have three fields, as shown in the previous figure.

## Rename a primary field

In this lab, you will rename the primary field that was created by default when you created the custom entity PC Manufacturers.

1. Sign in to Power Apps.
2. Select the environment where you added the new custom entity PC Manufacturers.

3. Select **Data** in the panel on the left-hand side of the Power Apps Portal to expand the available choices.
4. Select **Entities** under the **Data** option on the left-hand side of the portal.
5. Select the new custom entity **PC Manufacturers** in the list of entities. This will open the list of fields in the entity.
6. Recall that the **Name** field was auto-created when you created the custom entity. Notice that it is selected as both **Searchable** and **Required**, as shown in the following figure.

Display name	Name	Data type	Type	Required	Searchable
Description	cr5be_description	Text	Custom	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Name	cr5be_name	Text	Custom	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
PC Brand Name	cr5be_pcbrandname	Text	Custom	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>

7. Select the **Display Name** field and update it to **PC Model Name**, as shown in the following figure. You cannot change the internal name (name\*) of the field.

PC Model Name

Display name \*  
PC Model Name

Name \*  
cr5be\_name

Data type \*  
Text  
Required  
Searchable

Advanced options

Done Cancel

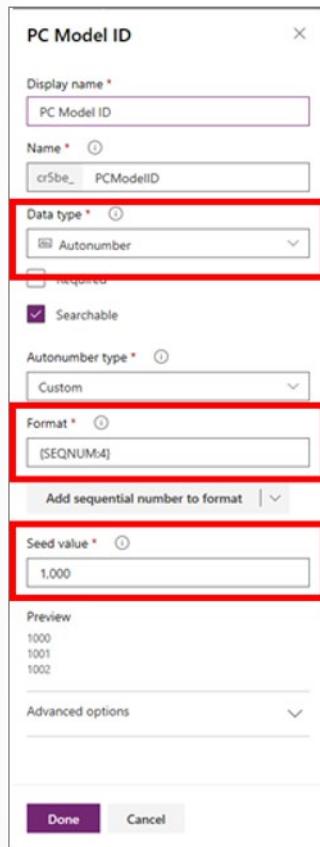
8. Select **Done**.
9. Select **Save entity**.

## Add an autonumber field

In this lab, you will add an autonumber field to help identify the manufacturer record in the custom entity.

1. Sign in to the Power Apps portal.

2. On the left pane, expand **Data** and select **Entities**.
3. Select the custom entity **PC**.
4. On the toolbar, select **Add field**.
5. On the right pane, enter a **PC Model ID** for the **Display name** and select **Autonumber** for the **Data type**.
6. Under **Autonumber type**, select **Custom**.
7. Leave the **Format** as the default of **{SEQNUM:4}**.
8. Customize a **Seed value** or keep the default value of **1,000**, as shown in the following figure.



9. Select **Done**.
10. Select the **Save entity** button.

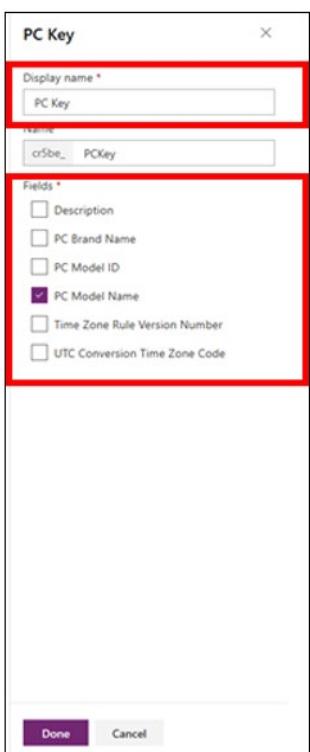
## Create a key

Create a key to ensure that the value is unique and indexed.

1. Sign in to Power Apps.
2. Select the environment where you added the new custom entity PC Manufacturers.
3. Select **Data** in the panel on the left-hand side of the Power Apps Portal to expand the available choices.

4. Select **Entities** under the **Data** option on the left-hand side of the portal.
5. Select the new custom entity **PC** in the list of entities. This will open the list of fields in the entity.
6. Select **Keys** on the menu at the top of the list of fields in your custom entity PC Manufacturers.
7. Select **New Key** on the menu at the top of the list of keys.
8. Enter **PC Key** in the **Display name** field.
9. Select the check box next to the **PC Model Name** field.

Note: A key can be made up of one or many fields. If you select multiple fields, then the key is called a compound key.



10. Select the **Done** button.
11. Select the **Save entity** button.

After selecting **Done**, you will see the new key added to the list of all keys that are created for this entity. You can create up to five keys for each entity. All values in the key will be unique.

## Summary

In this module, you learned about:

- The fields in Common Data Service.
- Types of fields that are available in Common Data Service.
- Adding a field to an entity.

- The **Primary Name** field and what it's used for.
- Restrictions that are associated with fields in Common Data Service.
- Creating an autonumbering field.
- Creating an alternate key.

# Working with option sets

## Define option sets

Option sets are a list of choices that are displayed by using a drop-down menu during data entry. Option sets help streamline new record creation and standardize data entry, and they are best suited for a small set of standard short terms rather than long list of complex entries. You should not use option sets when the field usually contains unique entries that are not easily standardized.

For example, consider the use of option sets on a sales tracking page. An option set is well-suited to capture the stage of a sales lead if you have a short list of possible sales stages. On the other hand, an option set is not a good choice when you are entering a street name because, likely, there could be hundreds or thousands of street name choices in nearly any city.

[!TIP]

Make sure that you monitor the quality of data when using an option set. As a best practice, check the data that you collect with an option set to ensure that users are not disproportionately using the default value rather than selecting the proper option within the option set. You might find that users accept the default so that they can enter data in the page without much thought. Determine what incentive users have to enter high quality data rather than the first entry or default in an option set, and avoid terms like "other" because options like these do not provide meaningful data.

Option sets are managed as a separate list and then associated with a drop-down field. When used properly and closely managed, option sets help ensure consistent data entry and improves the quality and usefulness of the data that you collect. Option sets can help create meaningful reports and identify trends and clusters of data. Finally, you can make fields of type Option sets required when they are created or edited.

## Standard option sets

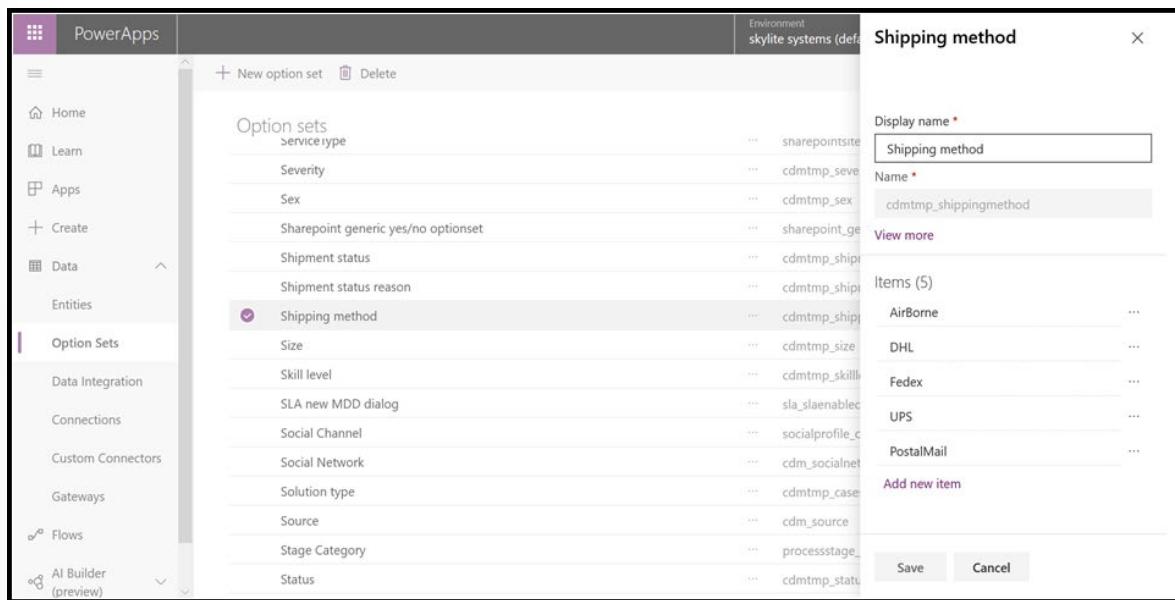
Common Data Service includes over one hundred standard option sets that are created each time that you instantiate a new instance of a Common Data Service database. You can add to a standard option set, but you cannot delete an entry from a standard option set.

Explore and learn what standard option sets are available in Common Data Service by following these steps:

1. Sign in to the Power Apps portal.
2. On the left pane, expand **Data** and select **Option sets**.
3. Scroll through the list of over 100 standard option sets. Select one of the standard option sets and examine the default list of entries for that option set.

By default, option sets are created as global option sets, allowing them to be reused across multiple entities. When creating a new option set, you can choose to make it **Local** under the **View more** option. This option is only available when you are creating an option set while adding a field, and not through the option set list.

Local option sets can only be used by the entity and field that they are created against and cannot be reused on other entities. This approach is only recommended for advanced users that have a specific need for a local option set.



## Exercise - Create a new option set or modify an existing option set

You might want to make a new custom option set to use across many entities in Common Data Service. You can create an option set in two ways:  
by creating an option set when you define a new field within an entity,  
or by creating an option set by using the option set functionality within the Power Apps portal.

### Create a new custom option set

To create a new custom option set by using the option set functionality in the Power Apps Portal, follow these steps:

1. Sign in to the Power Apps portal.
2. On the left pane, expand **Data** and select **Option sets**.
3. Select **New option set** on the menu at the top of the list.
4. Enter a **PC Type** for the **Display name**.
5. Add the following items to the option set:

**Laptop**

**Desktop**

**Tablet**

**Server**

[!NOTE]

You can assign a unique numeric value by selecting the ellipsis (...) beside the option set entry.

6. Select the **Save** button.



The option set **PC Type** is now available to use with any field in any entity within the environment. To use the new option set, follow the next steps. If you have not created the custom entity PC Manufacturers earlier in this learning path, complete the steps to [create a new custom entity](#)<sup>20</sup> and then complete these exercises.

[!TIP]

It might take a few minutes for the new option set to appear in the list of available option sets.

1. Sign in to Power Apps again.
2. Select the environment where you added the new custom entity PC Manufacturers.
3. Select **Data** on the panel on the left-hand side of the Power Apps Portal to expand the available choices.
4. Select **Entities** under the **Data** option on the left-hand side of the portal.
5. Select the new custom entity **PC** in the list of entities. This will open the list of fields in the entity.
6. Select the **Add field** button on the menu above the list of fields.
7. Enter **Type of PC** as the name of the new field.
8. Select **Option set** for the data type.
9. Select **PC Type** from the list of option sets, as shown in the following figure.

<sup>20</sup> <https://docs.microsoft.com/learn/modules/create-manage-entities/5-exercise>

10. Select the **Save** button.

You can also define an option set as you add a new field, without creating an option set in the portal. The option set that you add while defining a field can also be used with any field of type **Option set** in any entity within the environment.

## Define an option set when adding new fields

If you would rather define an option set as you add new fields to an entity, follow these steps:

1. Sign in to Power Apps.
2. Select the environment where you added the new custom entity PC Manufacturers.
3. Select **Data** on the panel on the left-hand side of the Power Apps Portal to expand the available choices.
4. Select **Entities** under the **Data** option on the left-hand side of the portal.
5. Select the new custom entity **PC** in the list of entities. This will open the list of fields in the entity.
6. Select the **Add field** button on the menu above the list of fields.
7. Enter **Screen Size Field** in the **Display name** field.
8. Enter **Option set** in the **Data type** field and select **New**.
9. Type **Screen Size** in the **Display name** field.
10. Type **14 Inch, 17 Inch, 21 Inch, and None** in the items and then select the **Save** button.

Screen Size ⊕

Display name \*

Name \*

[View more](#)

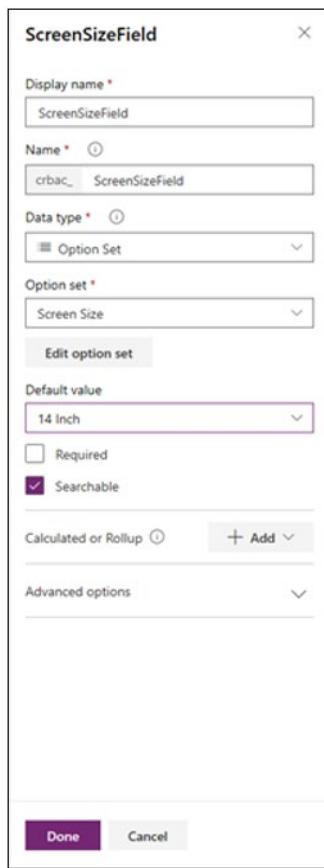
Items (4)

14 Inch	---
17 Inch	---
21 Inch	---
None	---

[Add new item](#)

Save Cancel

11. Type **14 inch** in the **default value** field and select the **Save** button.
12. Select the **Save entity** button.



If you need to modify an existing option set, follow these steps:

1. Sign in to the Power Apps portal.
2. On the left pane, expand **Data** and select **Option sets**.
3. Select the **Option set** that you want to modify.
  - a. Add a new entry by selecting the **Add new item** hyperlink.
  - b. You can delete an item by selecting the ellipsis (...).
  - c. You can edit an item by selecting the ellipsis (...) next to the entry and then selecting the **View more** option.
4. Select the **Done** button to save changes.

## summary

In this module, you learned:

- What option sets are and what types of option sets are available.
- What standard option sets are in Common Data Service.
- How to create a new option set or modify an existing option set.

# Create a relationship between entities

## Relate one or more entities - Introduction

To make an efficient and scalable solution for most of the solutions that you build, you will need to split up data into different containers (entities).

Trying to store everything into a single container would likely be inefficient and difficult to work with and understand.

The following example helps illustrate this concept.

Imagine that you need to create a system to manage sales orders. You will need a master list of items that you sell along with the inventory on hand, cost of the item, and the selling price. You also need a master list of customers with their addresses and credit ratings. Finally, you will need to manage invoices of sales that you make so you will want a way to store invoice data. The invoice should include invoice information (such as date, invoice number, and salesperson), customer information including address and credit rating, and a line item for each item on the invoice. Line items should include a reference to the item that you sold and be able to provide the proper cost and price for each item and decrease the quantity on hand based upon the quantity that you sold in that line item.

Trying to create a single entity to support the functionality that was previously described would be ill advised and inefficient. The correct way to approach this business scenario is to create the following four entities:

- Customers
- Products
- Invoices
- Line Items

Creating an entity for each of these items and relating them to one another will allow you to build an efficient solution that can scale while maintaining high performance. Splitting the data into multiple entities also means that you will not have to store repetitive data or support huge records with large amounts of blank data. Additionally, reporting will be much easier if you split the data into separate entities.

Entities that relate to one another have a relational connection. The technology that underlies Common Data Service is a relational database that is managed in the cloud by Microsoft. Relationships between entities exist in many forms, but the two most common are one-to-many and many-to-many, both of which are supported by Common Data Service.

One-to-many relationships are also known as parent-child relationships. In the previous invoice example, the invoice entity would be the parent and the line items would be a child entity. An invoice can have zero, one, or many line items (child records), but the line item will always be related to just one invoice (parent record). Typically, the child records will not exist without a parent record.

A field that only allows unique values is used to identify the parent record. This unique field is called a key. The same value (the parent key) is stored in the related child records. This field is called a foreign key when the child record is used to store the parent key value. Ingeniously, filtering is used to display child records with a value in the foreign key that matches the key value in the parent record. This allows applications to display the child records (line items in the previous example) that belong to a particular parent record (invoice in the previous example). This concept underlies many business software applications.

[!TIP]

Splitting data into different entities makes for an efficient solution design that can scale, but knowing how to split up the data into entities can be difficult. Common Data Service contains many of the entities that most organizations will need. Using standard entities and extending them will ensure that you are building solutions around a proven, scalable way of storing the data that is used by your solutions.

## Relationship types that are available in Common Data Service

In unit one of this module, you learned that Common Data Service supports two kinds of relationships: one-to-many and many-to-many.

### One-to-many relationship

The one-to-many relationship (which is also called 1:N or parent-child) includes a primary (parent) entity that can be associated to many other related (child) entity records by using a lookup field on the related (child) entity. The primary record is the parent and the related entity records are called child records.

When viewing a primary entity record in a 1:N relationship, you can view a list of the related child entity records by filtering all child records that contain the same key value as the key value in the primary record.

### Many-to-many relationship

The many-to-many relationship (which is also called N:N) includes a special third entity called a relationship entity, sometimes called an intersect entity, which maps how the many records of one entity can be related to the many records of another entity.

When viewing records of either entity in an N:N relationship, you can view a list of any records of the other entity that are related to it.

One-to-many relationships are simplistic and are universal.

An example of a one-to-many relationship includes an invoice (the one) with line items (the many), as previously discussed. Another example is a classroom (the one) and students in the classroom (the many).

Many-to-many relationships are a bit more complex. Entities that have this type of relationship require a special entity in between them to map how they are related to one another. Examples include authors and books. It is possible that a book could have many authors and an author could write many books. A new intersection entity between the book

entity and the author entity is needed to map (resolve) the books and authors with an entry in the intersection entity that contains the name of the book and author name in each record. You can create a report by using this intersection entity to show all the books that are written by an author, even if that writer was one of many or the only author on the book.

## Create a one-to-many relationship between entities

This unit shows how to implement relationships in Common Data Service with the following steps.

[!TIP]

One-to-many or many-to-one relationships are the same if you are looking at the relationship from one side or the other.

1. Sign in to the Power Apps Portal.
2. Select the environment that you want to work within by using the drop-down list in the top menu bar.
3. Select **Data** on the left-hand side of the page.
4. Select **Entities**, and a list of entities will be shown.
5. Select one of the entities in the relationships that you want to create.
6. Select the **Relationship** tab.
7. Select either the **One-to-Many** or **Many-to-One** relationship option.
8. Depending on which option that you selected, choose either the related entity or the parent entity for the relationship that you want to create between the two entities.

[!TIP]

With either choice, a lookup field will be created on the primary entity.

9. Fill out additional information as needed about the relationship.

Field	Description
Lookup field display name	The localizable text for the lookup field that will be created on the related entity. This can be edited later.
Lookup field name	The name for the lookup field that will be created on the related entity.
Relationship name	The name for the relationship that will be created.
Lookup field description	The description for the lookup field. In model-driven apps, this will appear as a tooltip when people hover their mouse over the field. This can be edited later.

10. Save the relationship.

11. Save the entity.

Now, you can use this relationship and the lookup in your business solution. A lookup field will be available to use that shows all the many (child) records.

[!TIP]

There are certain instances where you must use the Solution Explorer to create a many-to-one or one-to-many relationship. You can get more information about these special situations in the Summary unit of this module.

## Create a many-to-many relationship between entities

One-to-many entity relationships establish a hierarchy between records. With many-to-many (N:N) relationships, there is no explicit hierarchy and no lookup fields or behaviors to configure. Records that are created by using many-to-many relationships can be considered peers and the relationship is reciprocal.

With many-to-many relationships, a relationship (or intersect) entity stores the data that associates the entities. This entity has a one-to-many entity relationship with both of the related entities and only stores the necessary values to define the relationship. You cannot add custom fields to a relationship entity and it is never visible in the user interface.

Creating a many-to-many relationship requires choosing the two entities that you want to participate in the relationship. These are the same options that are used for the primary entity in one-to-many entity relationships.

[!TIP]

Not all entities can be used with many-to-many relationships. If the entity is not listed in the designer, you cannot create a new many-to-many relationship with this entity.

If you need to create a many-to-many relationship, follow these steps:

1. Sign in to Power Apps.
2. Select **Data > Entities** and select one of the entities in the relationships that you want to create.
3. Select the **Relationship** tab on the menu.
4. Select the **Many-to-Many** relationship option.
5. Depending on which option you selected, choose either the related entity or the primary entity for the relationship that you want to create between the two entities.
6. On the Many-to-Many panel, choose the entity that you want to be related to the current entity.
7. Select **More Options** to view the **Relationship Name** and **Relationship entity name** fields.
8. Select **OK** to save the new many-to-many relationship.

Now, you can use the many-to-many relationship in your apps. To read an informative post on how to use a many-to-many relationship in an app in Power Apps, follow the link in the Summary section of this module.

[!TIP]

You cannot edit the entities in a many-to-many relationship after it has been created; you can only delete it.

## Edit or delete relationships

With the following steps, you can edit or delete any relationships that are created in Common Data Service at any time.

### Edit a one-to-many or many-to-one relationship

To edit a one-to-many relationship, use the following procedure:

1. Sign in to Power Apps.
2. Select **Data > Entities** and select one of the entities in the relationships that you want to create.
3. Select the **Relationship** tab on the menu.
4. Open the relationship that you want to edit.

[!TIP]

You can edit the Lookup field display name and Lookup field description as needed.

5. Select **Save**.

### Edit a many-to-many relationship

You cannot edit a many-to-many relationship in the Power Apps portal. If you want to edit a many-to-many relationship, you must use the **Solution Explorer**. A link to allow editing in the **Solution Explorer** is included within the Summary section of this module.

### Delete a relationship

If you want to delete a relationship, follow these steps:

1. Sign in to Power Apps.
2. Select **Data > Entities** and select one of the entities in the relationships that you want to create.
3. Select the **Relationship** tab on the menu.
4. Select the relationship that you want to delete.
5. Select **Delete Relationship** on the menu.

## Exercise - Create two entities and relate them by using a one-to-many relationship

In this exercise, you will create the entities needed for the Anchors Away Cruise Line to book charter cruises. You will create two entities and relate them to one another by using a one-to-many relationship.

1. Sign in to the Power Apps Portal.
2. Select the environment that you want to work within by using the drop-down list in the top menu bar.
3. Select **Data** on the left-hand side of the page.
4. Select **Entities**, and a list of entities will be shown.
5. Select the **+New Entity** button on the menu.
6. Enter the following information in the **New entity** window.

The screenshot shows the 'New entity' configuration window. It includes fields for 'Display name' (set to 'Cruises'), 'Plural display name' (set to 'Cruises'), and 'Name' (set to 'cr5be\_ Cruises'). A checkbox for 'Enable attachments (including notes and files)' is also present.

Field	Value
Display name *	Cruises
Plural display name *	Cruises
Name *	cr5be_ Cruises
<input type="checkbox"/> Enable attachments (including notes and files)	(unchecked)

7. Select the **Next** button.
8. Select the **Name** field.
9. Enter the following information in the fields (as shown in the following figure).

The screenshot shows the configuration dialog for the 'Cruise ID' field. The 'Display name' is 'Cruise ID'. The 'Name' is 'crSle\_ Name'. The 'Data type' is 'Autonumber'. The 'Format' is '{SEQNUM:3}'. The 'Seed value' is '100'. The 'Preview' section shows the generated values: 100, 101, 102. The 'Done' button is highlighted.

10. Select **Done**.

11. Add the following fields by selecting **Add New Field** on the top menu.

Field Name	Field Type
Cruise Date	Date Only
Boat Name	Option Set – Add the following items to the options set: <b>Mudskipper, Sunshine, Holiday</b>
Destination	Option Set – Add the following items to the options set: <b>Catalina, Santa Cruz, Santa Rosa</b>
Captain	Option Set – Add the following items to the options set: <b>Renny, Blye, Jones</b>

12. Select the **Save Entity** button.

13. Select the **Keys** tab on the menu.

14. Select **New Key** in the ribbon.

15. Enter **CruiseIDKey** in the **Key name** field.

16. Select **Done**.

17. Select **Save Entity**.

The screenshot shows the Microsoft Power Apps entity list interface. At the top, there are several action buttons: '+ Add field', 'Get data', 'Export data', 'Open in Excel', 'Delete entity', and 'Settings'. Below this, the title 'Entities > Cruises' is displayed. A navigation bar with tabs 'Fields' (which is selected), 'Relationships', 'Business rules', 'Views', 'Forms', 'Dashboards', 'Charts', 'Keys', and 'Data' follows. The main area displays a table of fields for the 'Cruises' entity:

Display name	Name	Data type
Boat Name	cr5be_boatname	Option Set
Captain	cr5be_captain	Option Set
Cruise Data	cr5be_cruisedata	Date Only
Cruise ID	cr5be_name	Autonumber
Destination	cr5be_destination	Option Set

Congratulations, you just made the parent entity for the cruise booking app. Now you will make the child entity called Passengers. Each cruise will have many passengers, so the Cruise entity is the parent and the Passenger entity is the child.

1. Select the word **Entities** at the top of the screen where it says **Entities > Cruises**, as shown in the previous screenshot. This will take you back to the list of entities.
2. Select the **+ New Entity** button on the menu at the top of the list of entities.
3. Enter the name **Passengers** in the **Entity Name** field and select the **Next** button at the bottom of the new entity panel. This will take you to the list of fields in the Passenger entity.
4. Select the **Name** field.
5. Rename the field to **Passenger ID** and change the type to **AutoNumber**. Adjust the format and starting number as shown in the following screenshot.

**Passenger ID**

Display name \*

Name \*

Data type \*

Required

Searchable

Autonumber type \*

Format \*

Add sequential number to format

Seed value \*

Preview  
000001  
000002  
000003

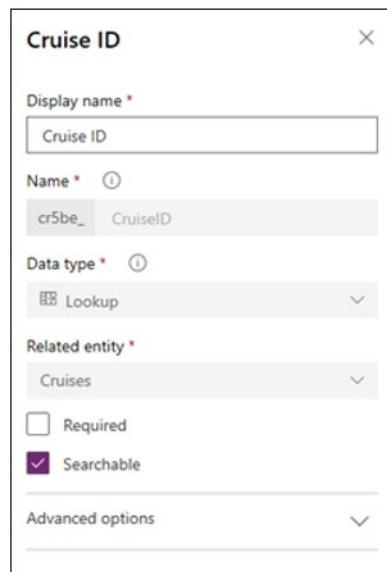
Advanced options

**Done** **Cancel**

6. Select the **Done** button.
7. Add the following fields to the Passenger entity.

Field Name	Type
Cruise ID	Look Up – Select “Cruises” in the <b>Related entity</b> field, as shown in the following figure
Passenger Name	Text
Gender	Option Set – <b>Male, Female</b>
Type of ID	Option Set - Enter the following: <b>Drivers License, Passport, Student ID, Other</b>

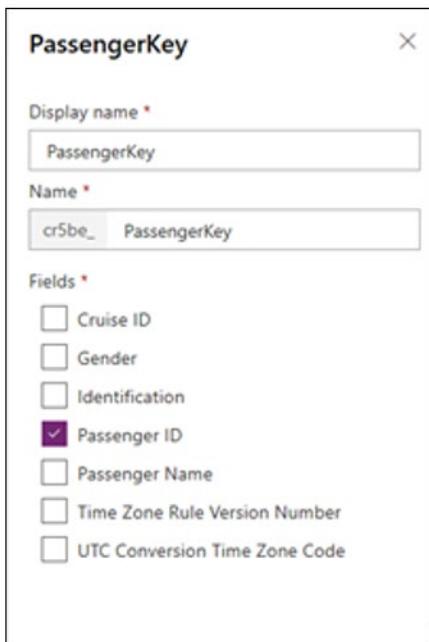
The lookup pane of the **Cruise ID** field is shown in the following figure.



After you have entered all the passenger fields, the list of fields in the Passenger entity should look like the following screenshot.

Entities > Passengers		
Fields	Relationships	Business rules
Display name ↴	Name ↴	Data type ↴
Cruise ID	... cr5be_cruiseid	Lookup
Gender	... cr5be_gender	Option Set
Identification	... cr5be_identification	Option Set
Passenger ID	... cr5be_name	Autonumber
Passenger Name	... cr5be_passengername	Text

8. Select **Keys** on the menu and then select **+Add Key**.
9. Enter **PassengerKey** for the name, select **Passenger ID**, and then select the **Done** button.

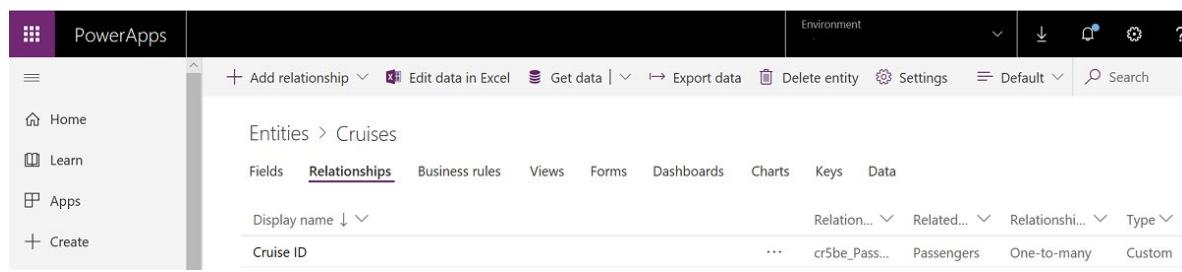


10. Select the **Save Entity** button to save the new Passenger entity.
11. Select **Relationships** on the menu in the Passenger entity and you will see the many-to-one relationship that you created in the Passengers entity by using the **CruiseID** lookup.

The screenshot shows the 'Relationships' tab for the 'Passengers' entity. It lists a single relationship named 'Cruise ID' with the following details: Relation... (dropdown), Related... (dropdown), Relationship... (dropdown), Type (Many-to-one), and Custom. The 'Cruises' entity is listed as the related entity.

Now that you have a many-to-one relationship with Passengers to Cruises, you can look at the Cruises entity.

1. Select the word **Entities** in the title of the Passenger entity to view a list of all entities.
2. Select the **Cruises** entity.
3. Select the **Relationship** menu within the Cruises entity.
4. Notice that the relationship also exists in the Cruises entity, but it is a one-to-many.



You can now use the entities and reference the relationship to build a simple booking app.

## summary

In this module, you learned how to use relationships to build powerful solutions, in addition to the following:

- Reasons why relationships are created between entities and their importance to proper business solution design
- The types of relationships that are available between entities in Common Data Service
- How to build the following relationships between entities:
  - One-to-many
  - Many-to-many
- How to manage, edit, or delete relationships

The following links and topics were referenced within this module.

There is guidance available on when to use a solution to create a many-to-one or one-to-many relationship.

You can get more information about these special situations at the following links:

### **Create one-to-many entity relationships using solution explorer<sup>21</sup>**

You can get more information at the following link about using a one-to-many and many-to-many relationship within a Power Apps app.

### **Relate records in Many-to-Many relationships<sup>22</sup>**

You cannot edit the entities in a many-to-many relationship, but you can edit certain properties in the **Solution Explorer**. Select the following link to see available editing options.

### **Create Many-to-Many entity relationships in Common Data Service using solution explorer<sup>23</sup>**

<sup>21</sup> <https://docs.microsoft.com/powerapps/maker/common-data-service/create-edit-1n-relationships-solution-explorer>

<sup>22</sup> <https://powerapps.microsoft.com/blog/relate-records-in-many-to-many-relationships/>

<sup>23</sup> <https://docs.microsoft.com/powerapps/maker/common-data-service/create-edit-nn-relationships-solution-explorer>

# Define and create business rules

## Introduction

Business rules are server-side logic that is used with canvas or model-driven apps to set or clear values in one or many fields in an entity. They can also be used to validate stored data or show error messages. Model-driven apps can use business rules to show or hide fields, enable or disable fields, and create recommendations based on business intelligence.

[!TIP]

Business rules are usually defined for an entity and apply to all forms, but you can define a business rule for a specific model-driven form. Canvas apps cannot have a business rule applied to a specific form.

Business rules give you a powerful way to enforce rules, set values, or validate data regardless of the form that is used to input data. Additionally, business rules are effective in helping to increase the accuracy of data, simplify application development, and streamline the forms presented to end users.

Business rules can be used by canvas apps or model-driven apps to do the following:

- Set field values.
- Clear field values.
- Validate data and show error messages.

Model-driven apps can also use business rules to:

- Show or hide fields (model-driven apps only).
- Enable or disable fields (model-driven apps only).
- Create business recommendations based on business intelligence (model-driven apps only).

## Define the components of a business rule

Business rules encapsulate logic in a predefined set of steps that will run each time that data is entered or modified and the data meets certain criteria to trigger the business rule. The business rule will run regardless of the way that data was added or edited in the entity.

Additionally, business rules are server-side, meaning that the logic runs on the servers that manage Common Data Service.

Business rules have the following components:

**Condition** - All business rules have a condition. It is a trigger and is used to determine if the business rule is run based on the values that are added or edited in an entity. A condition always runs as true or false, and you can have more than one condition in a business rule.

**Action** - An action is some logic, like setting a field to a certain value, that runs on either the true or false branch of a condition.

**Scope** - Business rules can have a scope. The following table outlines the available scope options for a business rule.

[!TIP]

A business rule that is used by a canvas app always has the scope set to **Entity**.

Scope of a business rule	Applies to
Entity	Model-driven forms and server
All forms	Model-driven forms
Specific form ( <b>Account</b> form, for example)	Just that model-driven form

## Create a business rule

Business rules are flexible and can be used for many purposes, so they vary considerably in complexity and scope. Follow these steps to build any business rules and adjust for your specific needs:

1. Sign in to Power Apps.
2. Select **Data > Entities** and select one of the entities in the relationships that you want to create.
3. Select the **Business Rules** tab on the menu and then select **New**.
4. Add a name and description.
5. Select the **Scope** button.
6. Select the **Condition** component and then add conditions to the business rule.

[!TIP]

To add more conditions to your business rule, drag the **Condition** component from the **Components** tab to a plus sign (+) in the designer.

7. To set properties for the condition, select the **Condition** component in the designer window, and then set the properties on the **Properties** tab on the right side of the screen. As you set properties, Common Data Service creates an expression at the bottom of the **Properties** tab.

[!TIP]

To add another clause (an **AND** or an **OR**) to the condition, select **New** on the **Properties** tab to create a new rule, and then set the properties for that rule. In the **Rule Logic** field, you can specify whether to add the new rule as an **AND** or an **OR** clause.

8. Add action(s) from the **Action** component with the following procedure:

Drag one of the action components from the **Components** tab to a plus sign (+) next to the **Condition** component. Drag the action to a plus sign (+) next to a check mark if you want the business rule to take that action when the condition is met, or drag the action to a plus sign (+) next to an **X** if you want the business rule to take that action if the condition is not met.

9. Set the property for each step.

To set properties for the action, select the **Action** component in the designer window, and then set the properties on the **Properties** tab.

When you are done setting properties, select **Apply**.

10. Select **Validate** to validate the business rule in the action bar.
11. Select **Save** to save the business rule.
12. Select **Activate** in the **Solution Explorer** window to activate the business rule and start its running process.

[!TIP]

Consider the following tips as you work on business rules in the designer window:

**Snapshot** - To take a snapshot of everything in the **Business Rule** window, select **Snapshot** on the action bar. This is useful, for example, if you want to share and get comments on the business rule from a team member.

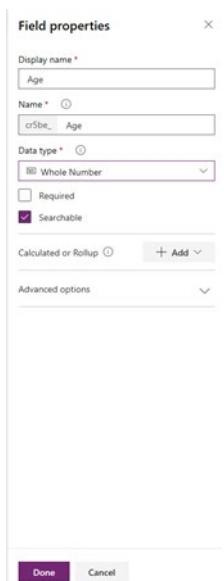
Use the mini-map to quickly browse through different parts of the process. The mini-map is useful when you have a complicated process that scrolls off the screen.

As you add conditions, actions, and business recommendations to your business rule, code for the business rule is built and appears at the bottom of the designer window. This code is read-only.

## Exercise - Create a business rule

In this exercise, you will use the Passenger entity set up in this learning path. If you did not create the Cruises and Passenger entity as a part of this learning path, complete the steps [here<sup>24</sup>](#) to complete these exercises.

1. Sign in to Power Apps.
2. Open **Data > Entities**.
3. Select the **Passengers** entity.
4. In the **Field Properties** window, enter **Age** in the **Display Name** field. In the **Data Type** drop-down list, select **Whole Number**, and then select **Add** to add the new field.

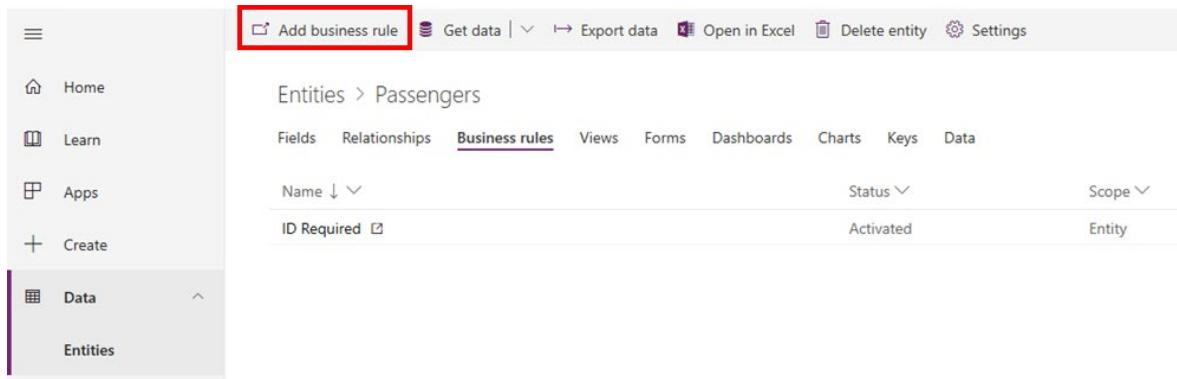


<sup>24</sup> <https://docs.microsoft.com/learn/modules/create-relationship-between-cds-entities/6-exercise>

5. In the **Field Properties** window, add another field. Enter **ID Required** in the **Display Name** field, select **Two Options** in the **Data Type** drop-down list, and then select **Add** to add the new field.

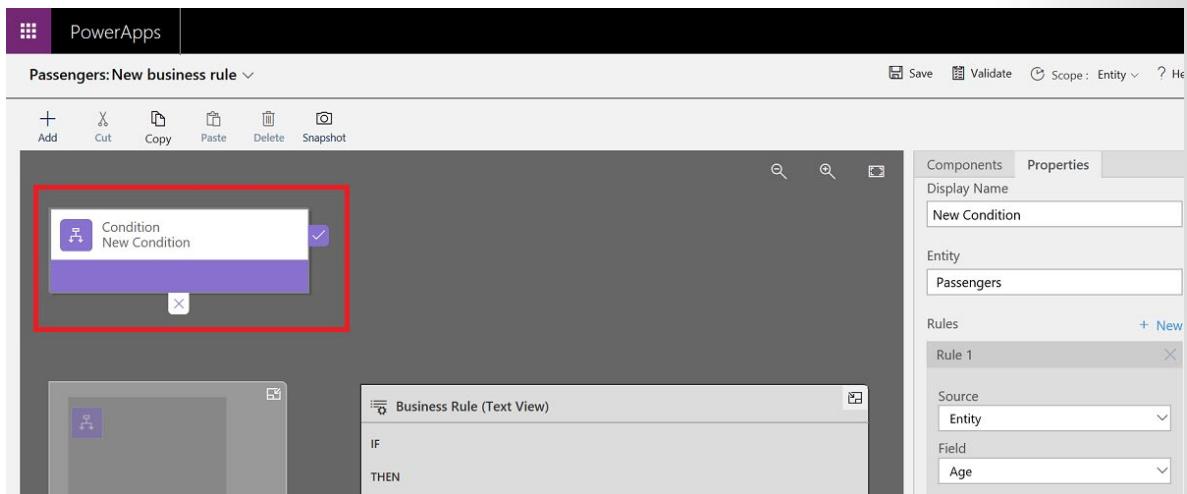


6. Select **Save** to save the entity.
7. Select the **Business rules** tab and then select **Add business rule** on the menu. Be patient; it will take time to load the business rule designer.



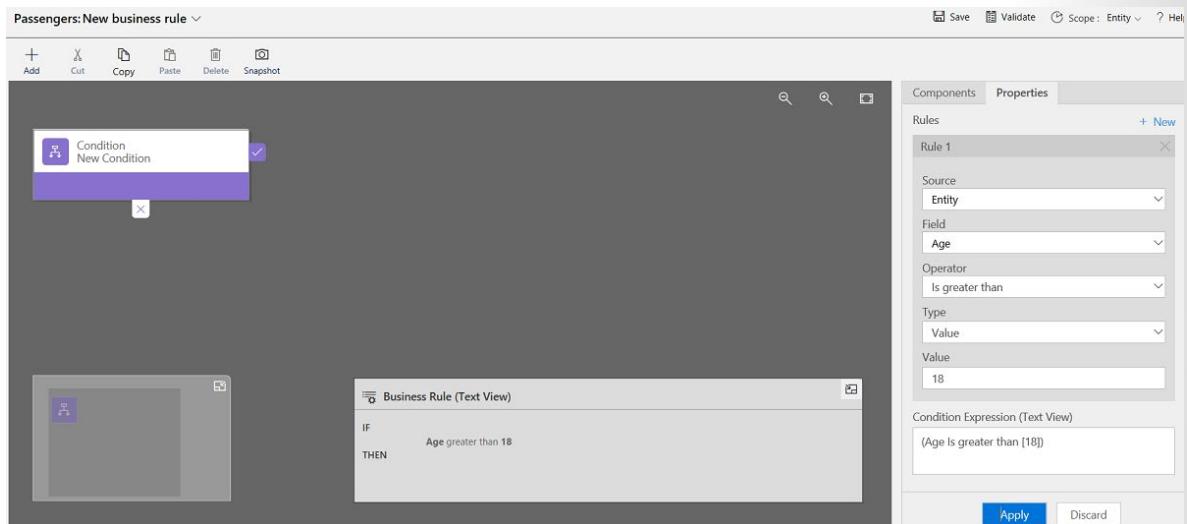
The screenshot shows the Power Apps Studio interface for the 'Passengers' entity. The left sidebar has 'Data' selected under 'Entities'. The top navigation bar includes 'Add business rule' (which is highlighted with a red box), 'Get data', 'Export data', 'Open in Excel', 'Delete entity', and 'Settings'. The main area shows the 'Business rules' tab is active. Below it, there's a table with columns for 'Name', 'Status', and 'Scope'. One row is visible: 'ID Required' is listed with 'Activated' status and 'Entity' scope.

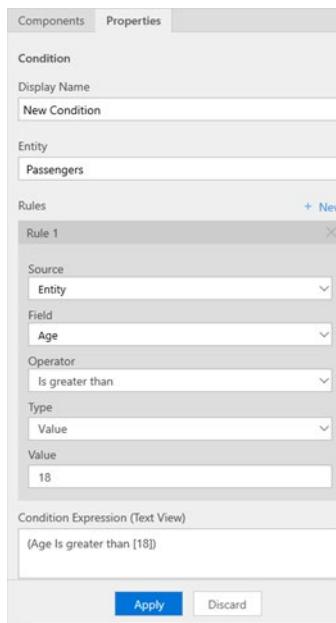
8. In the main design area, select **Condition**. This selection opens a panel to let you set the condition for the business rule.



9. Add the following information into the fields on the Condition panel.

Condition panel field	Value
Entity	Passengers
Source	Entity
Field	Age
Operator	Is Greater Than
Type	Value
Value	18

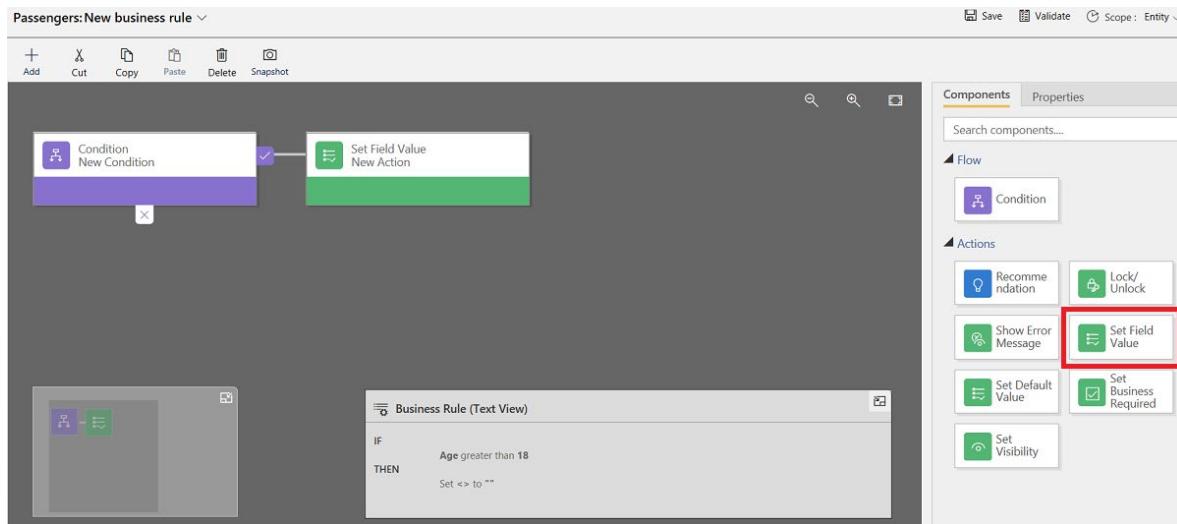




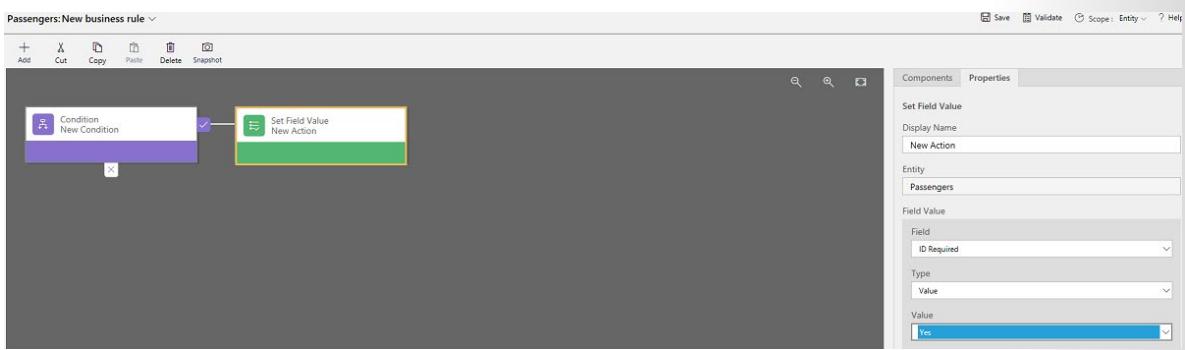
10. Select the **Apply** button at the bottom of the panel.

Now, you will define action for the **Yes** or **No** branch, meaning that you will indicate what you want to happen if the age of the person who is entered is *18 or under* or *over 18*.

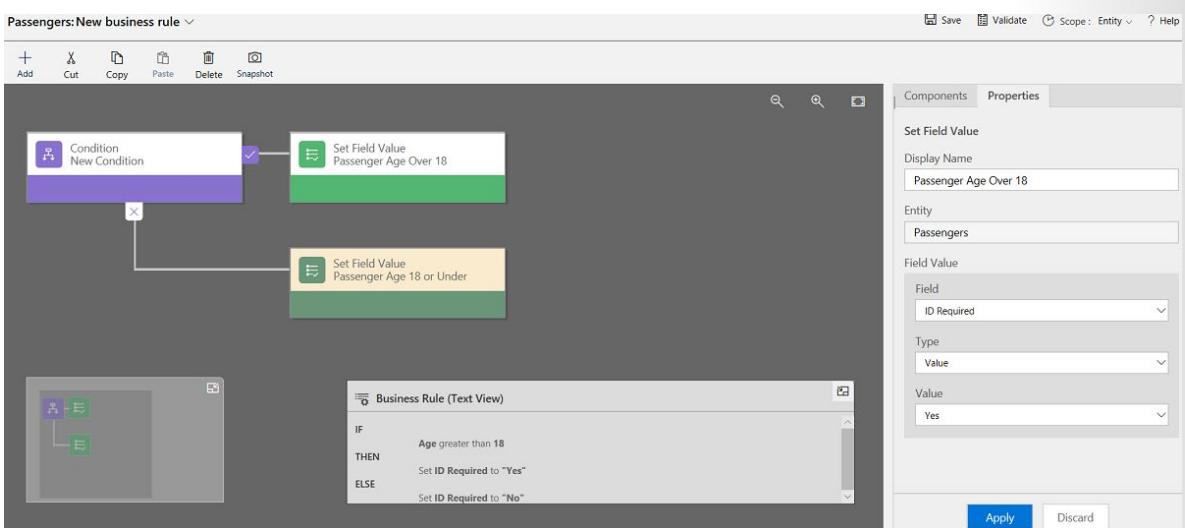
11. Select the **Components** tab and then select the **Set Field Value** new action and drag it to the right-hand side of the **Conditions** panel.

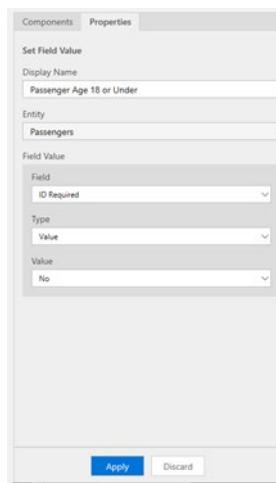


12. On the **Set Field Value** component, enter **Passenger Over Age 18** in the **Display Name** field to indicate that the passenger's age is greater than 18. In the **Entity** field, enter **Passengers**. Select **ID Required** in the **Field** drop-down menu, and then select **Yes** in the **Value** drop-down menu, and select **Apply**.

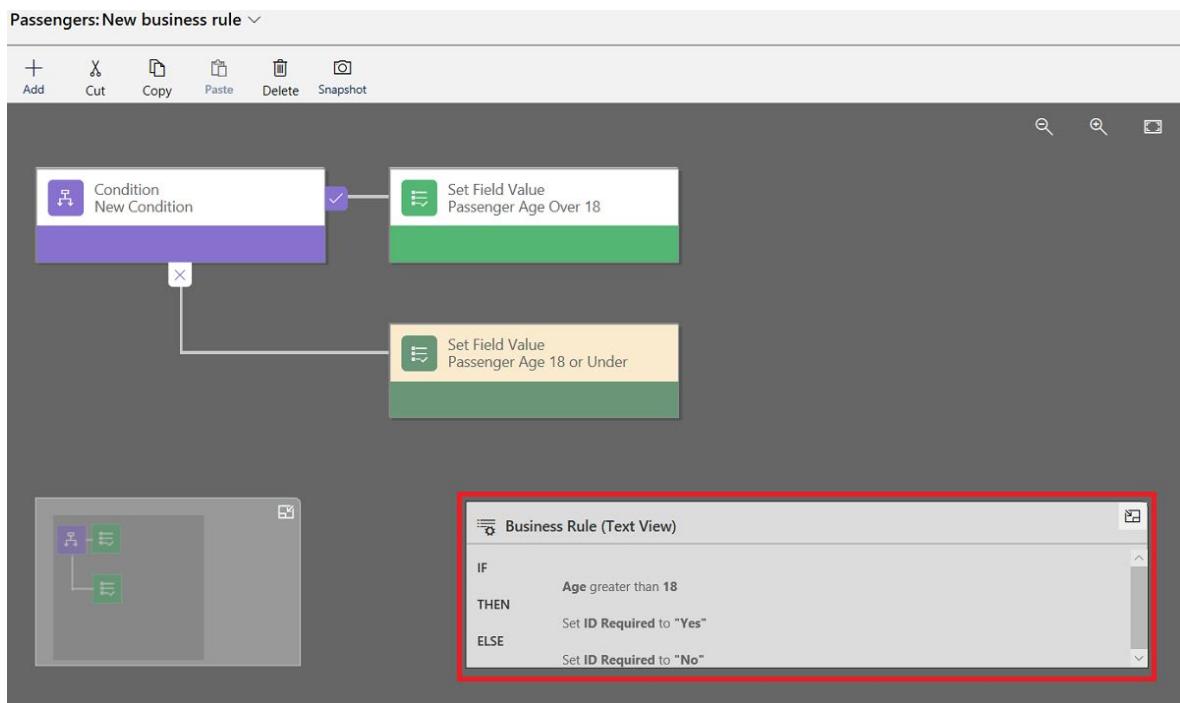


13. Select the **Set Field Value** component and drag it under the **Condition**, as shown in the following screenshot. Then, on the **Components** panel on the right, in the new **Set Field Value** component, enter **Passenger Age 18 or Under** in the **Display Name** field, and then set the **Value** option to **No**.

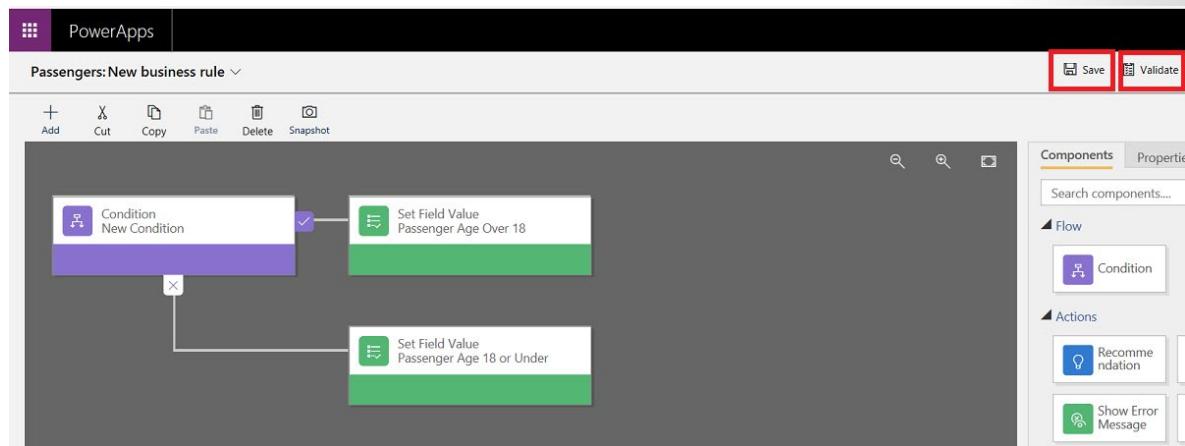




14. Now that the rule is set, notice that the business rule formula is shown in the window.



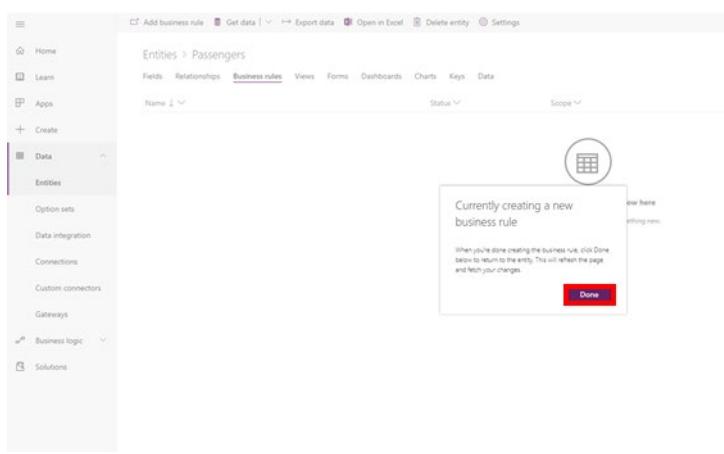
15. If you receive no errors, select **Validate** and then select **Save**.



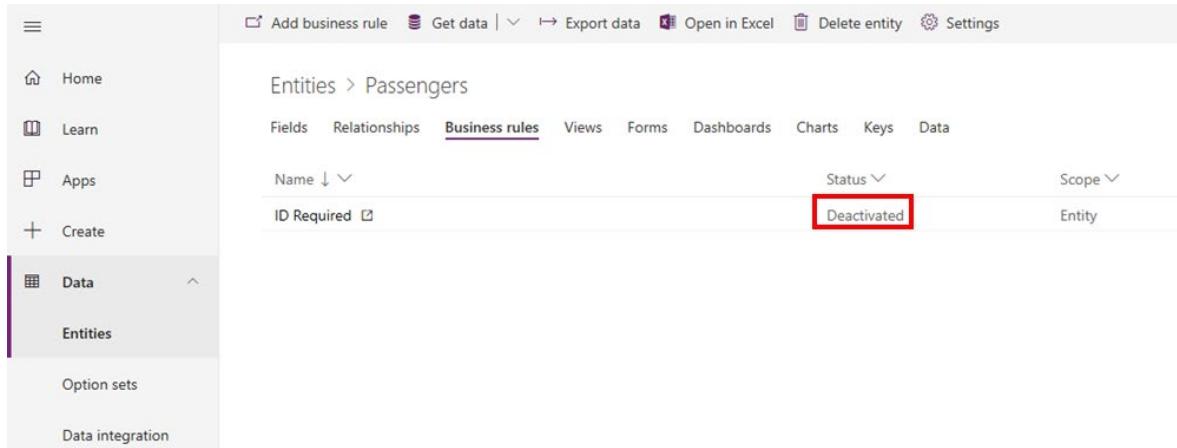
16. Select the arrow next to **Passengers: New business rule** in the top left of the screen. Enter a name in the **Business rule name** field and a description for the business rule in the **Description** field. Select **Save**.

This screenshot shows the configuration details for the business rule. At the top, the title 'Passengers: New business rule' is followed by a red box around the 'Save' button. Below, the 'Business rule name' is set to 'ID Required' and the 'Description' is 'Checks Passenger Age. ID required if over 18.'. The bottom part shows the same flow diagram as the previous screenshot, with the 'Save' button again highlighted with a red box.

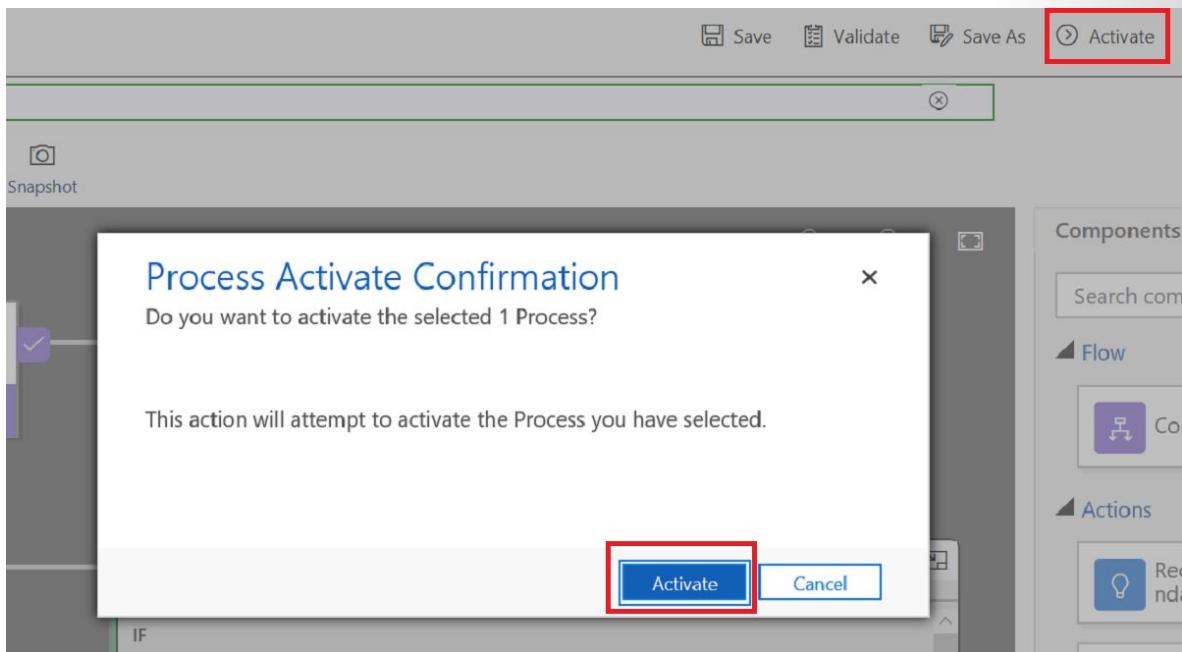
17. Close the browser window. Return to browser, select **Entities > Passengers**, and then select the **Business rules** tab. In the **Currently creating a new business rule** window, select the **Done** button.



18. The business rule that you created will appear on the list as **Deactivated**. To activate it, you will need to reset the **ID Required** field value. To start the process, select the new business rule that you created.



19. Select the **Activate** button on the menu bar at the top right. This will cause a **Process Activate Confirmation** dialog box to appear. Confirm activation by selecting the **Activate** button at the bottom of the dialog box.



20. Close the **Browser** tab and select **Entities > Passengers > Business rules** tab, which shows that the business rule that you created now has the status of **Activated**.

Name	Status	Scope
ID Required	Activated	Entity

Congratulations, you have just created a business rule. You can test it by selecting the **Data** tab and entering data by using Excel. If you are unfamiliar with using Excel to enter data in Common Data Service, see the link in the Summary section of this module for more information on this process.

## summary

In this module, you learned how to build business rules for an entity to enforce business logic regardless of how data is entered into the entity in Common Data Service. You also learned how to:

- Define and create business rules.
- Identify the different components of a business rule.
- Make a business rule.

For more information on how to use Excel with Common Data Service for data editing and data entry, select the following link.

**Open entity data in Excel<sup>25</sup>**

For information about licensing requirements, select the following link.

**License requirements for entities<sup>26</sup>**

---

<sup>25</sup> <https://docs.microsoft.com/powerapps/maker/common-data-service/data-platform-excel-addin>

<sup>26</sup> <https://docs.microsoft.com/powerapps/maker/common-data-service/data-platform-entity-licenses>

# Create and define calculation or rollup fields

## Introduction

Rollup fields perform calculations on values that are stored in a field in a related entity across a one-to-many relationship. Rollup fields can calculate sum, min, max, and count values in a fashion that is similar to how these same functions are used within Microsoft Excel. Rollup fields are read-only and can calculate values based on a system account with full access to all records in a related entity, so they might calculate values based on records that a current user cannot access due to security settings that prevent the user from access.

Rollup fields are calculated by Common Data Service and the value of the rollup calculation is stored within the field in the entity, meaning that rollup fields can be used for sorting, searching, and filtering like any other field.

Rollup calculations are calculated asynchronously (meaning not in real time) by using a scheduled system job in Common Data Service. Rollup calculations run once an hour by default, but you can schedule them to run as often as needed by accessing **Settings > Customizations > Customize the System > Components > Entities** in the **Solution Explorer** window and updating the settings for when the rollup occurs.

## Create a rollup field

Creating a rollup field is a simple process, but you should consider the following points before you begin.

You must create a supported field type in an entity and then define the rollup that you want to calculate. The following field types support rollups:

- Whole Number
- Decimal Number
- Currency
- Date Time

The key considerations about rollups are as follows:

- When you specify the field and select a supported field type, you can define the rollup calculation.
- Rollups calculate a value from a field in one or many child records in a related entity.
- You are limited to a maximum of ten (10) rollups for each parent entity.
- Rollup fields are read-only.
- Because rollup attributes persist in the database, they can be used for filtering or sorting just like regular attributes.
- Remember that rollups are asynchronous and are calculated by Common Data Service on a set schedule, so you might not see the expected value in a rollup until that job runs.

Follow these steps to create a rollup calculation:

1. Open PowerApps.
2. Expand **Data > Entities**.
3. Select the entity that you want, select **Fields**, and then select **Add Field**.
4. Provide the required information for the field, including the **Display name**, **Name**, and **Data type**.
5. Select **Add > Rollup**.

[!NOTE]

You must define a relationship of *one* from this entity to *many* for another entity or the ability to define a rollup will not be available.

6. Define the rollup calculation by using the following guidance.

In the Source entity section, specify the entity for which the rollup field is defined and whether you will aggregate over a hierarchy or not. You can add filters with multiple conditions to specify the records in the hierarchy that you want to use for rollup.

In the Related entity section, you can specify that you want to calculate a rollup value from records in the related entity that are directly related to the current parent record by selecting **Yes** for the **Hierarchy** option. This specification means that the calculation is performed by using values in the child records, or you can create a rollup of all records in a related entity by selecting **No** in the hierarchy.

You can add filters with multiple conditions to specify which related records to use in the calculation. For example, you can include the revenue from the open opportunities with an annual revenue that is greater than \$1000 US dollars.

In the Aggregate section, specify the metric that you want to compute. You can choose available aggregate functions, such as SUM, COUNT, MIN, MAX, or AVG.

7. Save the new field and entity that is holding the rollup.

The rollup is now ready to use. It will run and recalculate based on the schedule that you set regardless of which form that you use to access the data that is associated with this rollup.

## Introduction to calculation fields

A calculation field lets you define a calculation formula that is run by Common Data Service regardless of the form that is used to edit or add data. Calculations allow you to improve data integrity and simplify form development. Unlike asynchronous rollups that calculate based on a scheduled job, calculations run in real time when the record is saved.

Calculated fields are powerful, and have the following key considerations:

- Calculated fields use the fields from the current entity or related parent entities from a many-to-one relationship.
- Calculated fields are read-only.

- The expression support is available on the current entity and the related parent entity fields in the Condition sections and the Action sections. The built-in functions include:

ADDHOURS, ADDDAYS, ADDWEEKS, ADDMONTHS, ADDYEARS, SUBTRACTHOURS, SUBTRACTDAYS, SUBTRACTWEEKS, SUBTRACTMONTHS, SUBTRACTYEARS, DIFFINDAYS, DIFFINHOURS, DIFFINMINUTES, DIFFINMONTHS, DIFFINWEEKS, DIFFINYEARS, CONCAT, TRIMLEFT, and TRIMRIGHT.

- You can define branching and multiple conditions. The logical operations that are available include **AND** and **OR** operators.
- The following field types support calculations:
  - Text
  - Option Set
  - Two Options
  - Whole Number
  - Decimal Number
  - Currency
  - Date Time
- The visual editing capabilities include intellisense when you define the calculation in the Action portion of the field.
- You can configure calculated fields to use custom controls.

## Calculated field limitations

- If the calculated field depends on another value, it will not be recalculated until the record is retrieved.
- You cannot use values in calculated attributes that reference a related entity, another calculated attribute, or a logical value in the same entity to sort data that is returned by a query. Though your query can specify that the results should be ordered by using a calculated attribute, the sort direction will be ignored and will not throw an error. If the calculated attribute references only simple values in the same record, sorting works normally.
- Only attributes from an immediate parent entity can be used in a calculated attribute.
- Calculated attributes can reference other calculated attributes in their formula, but they cannot reference themselves.

## Create a calculation field

Creating a calculated field is a simple process; however, you should consider the following points before you begin:

- You must create a supported field type and then define the calculation. The following fields support calculations:
  - Text

- Option Set
- Two Options
- Whole Number
- Decimal Number
- Currency
- Date Time
- Sorting is disabled on:
  - A calculated field that contains a field of a parent record.
  - A calculated field that contains a logical field (for example, address field).
  - A calculated field that contains another calculated field.
- Calculated fields can span two entities only.
- A calculated field can contain a field from another entity (spanning two entities: current entity and parent record).
- A calculated field can't contain a calculated field from another entity that also contains another field from a different entity (spanning three entities).

Follow these steps to create a calculation field:

1. Open Power Apps.
2. Expand **Data > Entities**.
3. Select the entity that you want, select **Fields**, and then select **Add Field**.
4. Provide the required information for the field, including the **Display name**, **Name**, and **Data type**.
5. Add a calculation by selecting **Add > Calculation**.

[!NOTE]

Selecting **Calculation** will force you to save changes to the entity. Go ahead and save.

6. When you have saved the entity, the calculated field definition editor will open in a separate tab in the browser. This is where the new calculated field has been created, but no formula has been set. The calculated field definition consists of two sections: Condition and Action.
7. Define your calculation by using Condition and Action.

In the Condition section, you can specify an entity, field, operator, type, and value. In the **Entity** drop-down box, you can choose a current entity or a related entity. In the **Field** drop-down box is a selection of all available fields for the entity. Depending on the operator that you choose, you might need to provide type and value. You can specify multiple conditions by using the **AND** or **OR** operators.

In the Action section, provide the formula for the calculated field.

8. Select **Save** to save your work or **Save and Close** to close the tab.

## Exercise - Create a rollup field

In this exercise, you will create a count of the total number of passengers that have booked a seat on a cruise by using a rollup field. If you have not created the custom entity as a part of this learning path, you can complete the steps [here<sup>27</sup>](#)

to create the related entities, and

[here<sup>28</sup>](#)

to create the business rules to facilitate the completion of these exercises.

1. Sign in to Power Apps.
2. Select **Entities > Cruises**.
3. Select **Add Field** in the menu and enter the following into the new field panel:

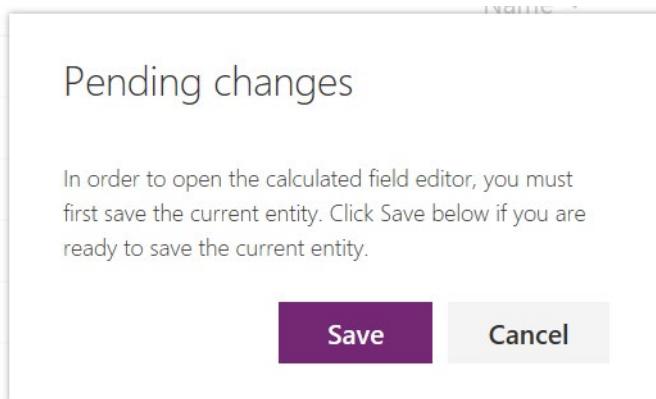
New Field Entry	Value
Display Name	Passenger Count
Data Type	Whole Number

The screenshot shows the Microsoft Power Apps interface. On the left, there's a list of fields for the 'Cruises' entity, including 'Boat Name', 'Captain', 'Cruise Data', 'Cruise ID', 'Currency', 'Destination', etc. On the right, a modal dialog is open for creating a new field named 'Passenger Count'. The dialog has sections for 'Display name' (set to 'Passenger Count'), 'Name' (set to 'cr5be\_PassengerCount'), 'Data type' (set to 'Text'), and 'Calculated or Rollup'. The 'Calculated or Rollup' section contains a '+ Add' button, which is highlighted with a red box.

4. Select the **+ Add** button and select **Rollup** in the drop-down menu that is associated with the **+ Add** button in the new field panel.
5. Select the **Save** button in the **Pending changes** window when prompted.

<sup>27</sup> <https://docs.microsoft.com/learn/modules/create-relationship-between-cds-entities/6-exercise>

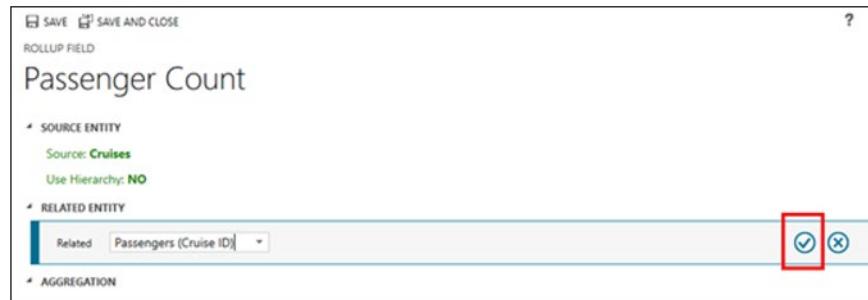
<sup>28</sup> <https://docs.microsoft.com/learn/modules/define-create-business-rules/4-exercise>



6. After you have saved the changes, a new tab will open in the browser. This is the rollup definition screen.

[!TIP]

Drag and expand the rollup definition screen to make sure that you can see the check mark button, as shown in the following screenshot. Select the Passengers entity, and then select the check mark button.



7. After you have selected the check mark button next to **Passengers (Cruise ID)**, select the **Add Aggregation** button (you do not need to set a filter).

SAVE SAVE AND CLOSE

ROLLUP FIELD

## Passenger Count

▲ SOURCE ENTITY

Source: **Cruises**

Use Hierarchy: **NO**

▲ RELATED ENTITY

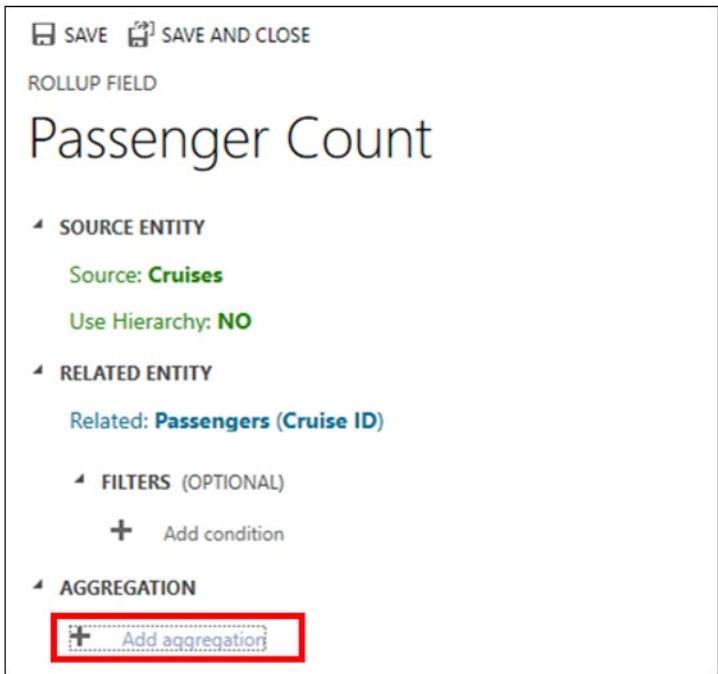
Related: **Passengers (Cruise ID)**

▲ FILTERS (OPTIONAL)

+ Add condition

▲ AGGREGATION

+ **Add aggregation**



- Define the aggregation by selecting **Count** in the **Aggregate function** drop-down menu and selecting **Passengers** in the **Aggregated Related Entity** field drop-down menu. Select the check mark button on the bottom-right of the screen.

SAVE SAVE AND CLOSE ?

ROLLUP FIELD

## Passenger Count

▲ SOURCE ENTITY

Source: **Cruises**

Use Hierarchy: **NO**

▲ RELATED ENTITY

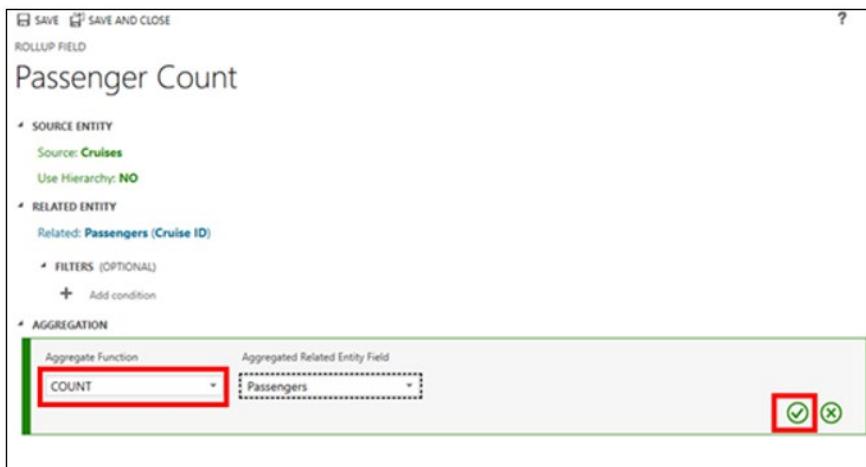
Related: **Passengers (Cruise ID)**

▲ FILTERS (OPTIONAL)

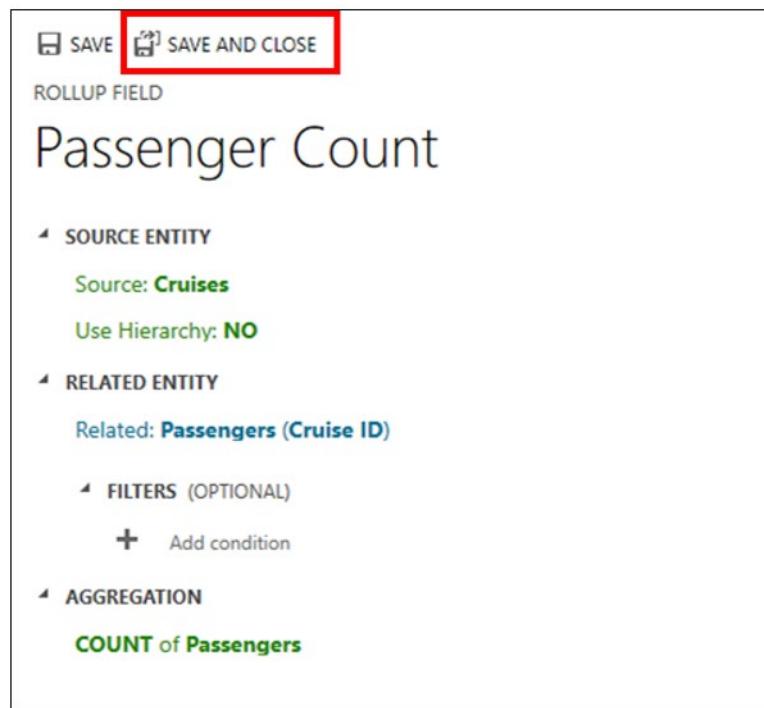
+ Add condition

▲ AGGREGATION

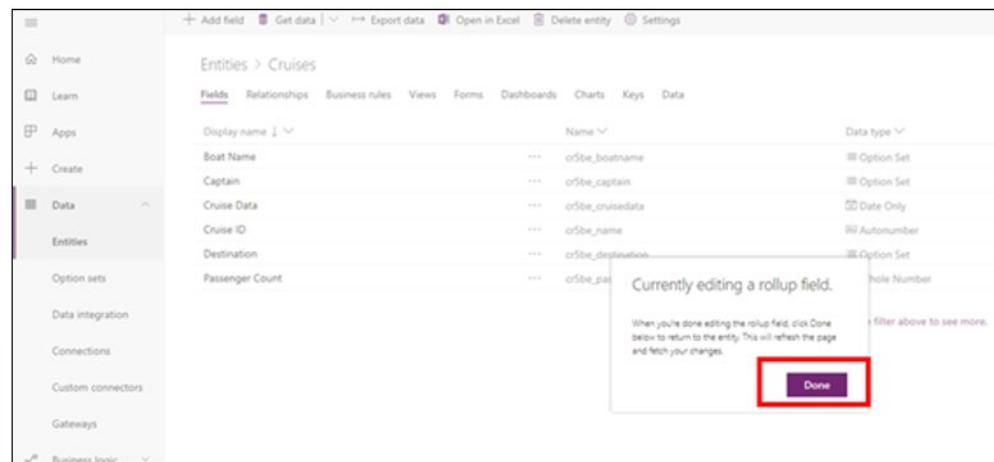
Aggregate Function	Aggregated Related Entity Field
<b>COUNT</b>	<b>Passengers</b>



- Select the **Save and Close** button to save this new rollup. This will close the rollup design screen and return you to the entity detail screen.



10. Select the **Done** button in the **Currently editing a rollup field** of the Cruises entity screen.



Congratulations! You have created a rollup field called **Passenger Count** that can be used in solutions that you create by using the Cruises entity.

[!NOTE]

Rollup values are calculated by using a scheduled job on the server and are not real-time. The default setting is a calculation every twelve hours after the rollup is defined and saved or edited. The frequency of when the rollup job runs can be adjusted as needed. See the link in the Summary unit of this module for more information on how to adjust when the rollup calculation job runs.

## Exercise - Create a calculation field

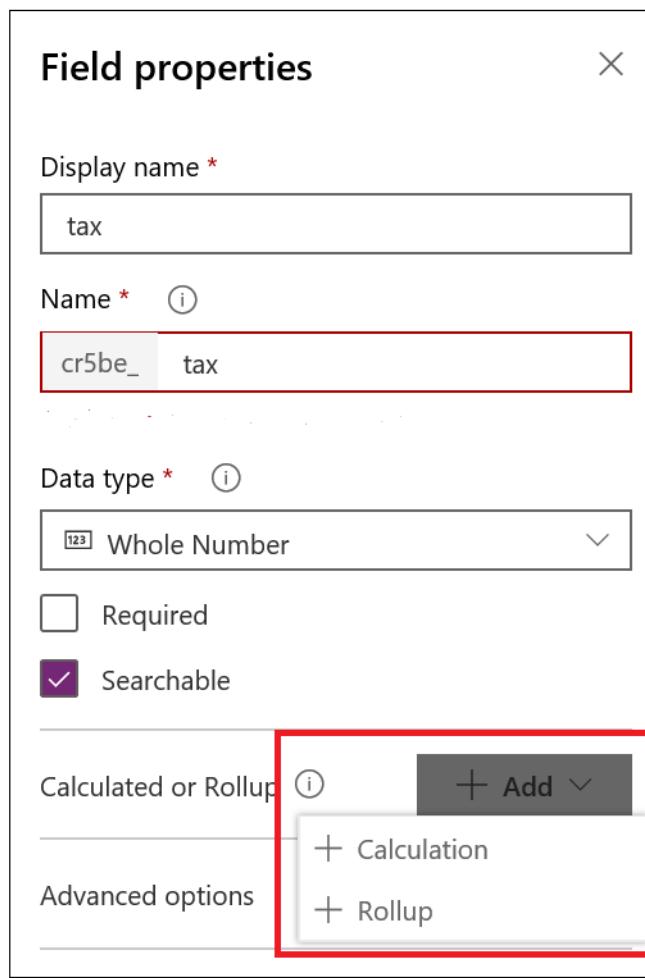
In this exercise, you will add a field called **Price** to hold the price for a passenger on the cruise. Then, you will add a second field called **Tax** and add a calculation to determine the tax to charge for each ticket.

1. Sign in to Power Apps.
2. Select **Entities > Cruises**.
3. Select **Add Field** in the menu, enter the following into the new Field Panel fields, and then save.

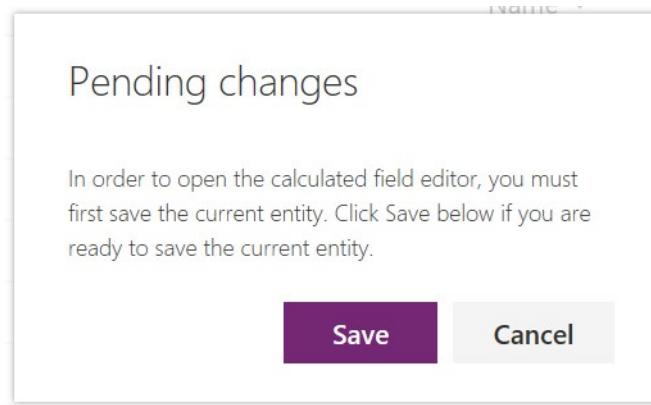
New Field Entry	Value
Display Name	Price
Data Type	Currency

4. Select **Add Field** in the menu, add a second field, enter the following into the new Field Panel fields, select the **+ Add** button next to **Calculated** or **Rollup**, and then select **+ Calculation**.

New Field Entry	Value
Display Name	Tax
Data Type	Currency



5. Select the **Save** button in the **Pending changes** screen so you can define the calculation.



6. The Cruises entity will save, and a new screen in the browser will open to let you define the Tax calculation. The process might take 30 seconds to open the calculation screen, so be patient.
7. Select the action option and then select the **Tax** field in the Cruises entity that you just added.  
[!TIP]  
Widen the screen so you can scroll through all your options.

The screenshot shows the PowerApps Studio interface with the 'Cruises' entity selected. The 'Fields' tab is active. A red box highlights the 'Tax' field, which has the formula `= cr5be_price * 0.085`.

Display name	Name
Boat Name	cr5be_boatname
Captain	cr5be_captain
Cruise Data	cr5be_cruisedata
Cruise ID	cr5be_name
Currency	transactioncurrencyid
Destination	cr5be_destination
Exchange Rate	exchangerate
Passenger Count	cr5be_passengercount
Price	cr5be_price
Price (Base)	cr5be_price_base
<b>Tax</b>	<b>cr5be_tax</b>
Tax (Base)	cr5be_tax_base

8. Complete the tax calculation by adding **.085**, as shown in the following screenshot, and then select the check mark button to finish creating the calculation.

The screenshot shows the configuration of the 'Set Tax' calculated field. The formula is set to `= cr5be_price * 0.085`. The 'Save and Close' button is visible at the bottom right.

**Saved** **Save and Close** ?

**CALculated FIELD**

**Set Tax**

**IF...THEN**

**CONDITION (OPTIONAL)**

+ Add condition

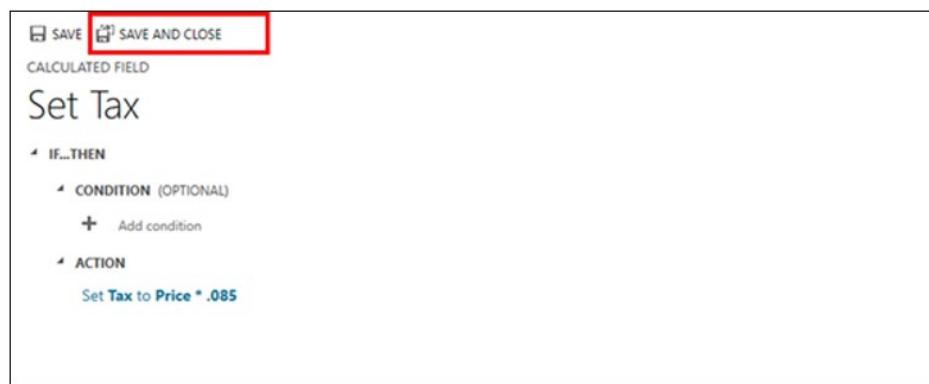
**ACTION**

**Set Tax (currency)**

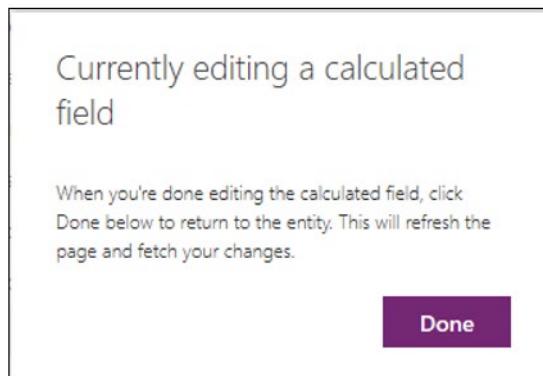
= `cr5be_price * 0.085`

✓ ✎

9. Select the **Save and Close** button to save the tax calculation.



10. Select the **Done** button, and the new **Price** and **Tax** fields are shown in the Cruises entity.



You might notice in the **Data type** drop-down list that **Currency** has two entries, one named **Price** and the other named **Price (Base)**, as shown in the following screenshot. It also has two other entries named **Tax** and **Tax (Base)**.

Display name	Name	Data type
Boat Name	cr5be_boatname	Option Set
Captain	cr5be_captain	Option Set
Cruise ID	cr5be_name	Date Only
Currency	transactioncurrencyid	Autonumber
Destination	cr5be_destination	Lookup
Exchange Rate	exchangerate	Decimal Number
Passenger Count	cr5be_passengercount	Whole Number
Price	cr5be_price	Currency
Price (Base)	cr5be_price_base	Currency
Tax	cr5be_tax	Currency
Tax (Base)	cr5be_tax_base	Currency

The reason for the two entries for **Currency** and **Tax** is because Common Data Service supports many different currencies. You can deploy a solution that is globally built with Common Data Service and convert local currency to a common base currency. While local values might be dollars or

euros in the **Price** and **Tax** fields, a common currency is used in the **Currency base** and **Tax base** fields. Common Data Service calculates the conversion rate so that you can collect information in a standard currency (base fields) regardless of the local currency that is used in your solution. You can read more about currency conversion by following a link provided in the Summary unit of this module.

Try adding some data into the Cruises entity to test out the **Tax** field by selecting **Data** in the menu above the entity and then selecting the **Excel** option. A link is included within the Summary unit of this module if you need additional information on how to enter and edit data in Common Data Service with Excel.

## summary

In this module, you learned what rollup fields are and how to create them. You also learned that rollup values are asynchronous and calculated in a scheduled job that can be adjusted by the administrator. Additionally, you learned about calculated fields and how to create them, and how they are updated each time you save a record with a calculated field.

The following links were referenced within this module:

**Currency Conversion<sup>29</sup>** - This is a good overview of how the **Currency** field works in Common Data Service.

**Open entity data in Excel<sup>30</sup>** - This provides information on using Excel to add and edit entity data.

**Rollup calculations<sup>31</sup>** - This provides some information about rollup calculations and rollup jobs.

<sup>29</sup> <https://community.dynamics.com/crm/b/henryjammespowerblog/archive/2018/08/12/currency-management-in-dynamics-365-cds-for-apps>

<sup>30</sup> <https://docs.microsoft.com/PowerApps/maker/common-data-service/data-platform-excel-addin>

<sup>31</sup> <https://docs.microsoft.com/PowerApps/maker/common-data-service/define-rollup-fields#rollup-calculations>

# Get started with security roles in Common Data Service

## Introduction to environment roles

Each environment has zero or one instance of a Common Data Service database, and your organization can have many environments that are available to many different groups of users at a time. It is common practice to set up an environment so you can limit who can access the data, apps, and workflows within it.

Classic software lifecycle management provides a good use case of why you might want to set up different environments in Common Data Service. Consider the following example. It is common to segment environments for development, test, and production. Common Data Service allows you to set up a development environment and limit access so only developers and a few managers or test users have permission to access the data and apps in that development environment. You can then set up a test environment and set up permissions so that a few test users and developers have access to it and the data within the instance of Common Data Service within that environment. Finally, you can set up production environment permissions so that a wide audience of users has access to the production environment and the data in the instance of a Common Data Service database, Power Apps, and Power Automates within that production environment.

*Important:* Access to an environment does not give a user access to any data, apps, or workflow within that environment. Users must be given explicit access to data by an administrator in Common Data Service while the maker who creates an app, connector, or workflow must grant access to their work products.

## Understand environment roles

You can manage environment security by using roles and then adding users to the environment and assigning roles to users. A role has certain permissions that are associated with it, and you can associate a user with one or many roles. Environments have two built-in roles that provide access to permissions within an environment, and you'll assign users to one of these two roles when considering what permissions you want to give to a user in an environment.

The built-in environment roles are:

- System Administrator
- Environment Maker

This unit examines each role to help you understand how it works within an environment.

*Important:* A user is automatically associated with the Environment Maker role when they are added to an environment.

## System Administrator role

The System Administrator role can perform all administrative actions on an environment, including the following tasks:

- Add or remove a user or group from either the Environment Admin or Environment Maker role.
- Provision a Common Data Service database for the environment.
- View and manage all resources that are created within an environment.
- Set data loss prevention policies.

## Environment Maker role

The Environment Maker role can create resources within an environment such as apps, connections, custom connectors, gateways, apps that use Power Apps, and flows that use Power Automate. The following applies to members of the Environment Maker role:

- Environment Makers can distribute the apps that they build in an environment to other users within an organization by sharing the app with individual users, security groups, or to all users in the organization.
- Users or groups that are assigned to these environment roles are not automatically given access to the environment's database (if it exists) and must be given access separately by a Database owner.
- Whenever a new user signs up for Power Apps, they are automatically added to the Maker role of the default environment.
- When you add a user to an environment, they are assigned two roles by default.
  - Common Data Service User (this role is created when you instantiate an instance of a Common Data Service database and all users in the environment are assigned this role)
  - Environment Maker

Users or security groups can be assigned to either of these two roles by a System Administrator from the Power Apps Administration Center.

## Exercise-Adding or disabling an environment user

You can add users to any environment that you create in Common Data Service.

The following steps will help you add users to an environment.

1. Sign in to Power Apps.
2. Select the gear icon in the ribbon and select the **Admin Center** option.
3. Select an environment.
4. Select the **Security** option, **Users + Permissions**, and then **Users**.

5. Add one or more users by entering their name and email address.
6. Select the **Add Users** button in the ribbon.

If you want to disable a user within an environment, remove a license from the user or remove the user from the security group that's associated with an environment. Removing a user from the security group doesn't remove the user's license. If you want to make the license available to another user, you have to remove the license from the user account that was disabled.

Removing a license, disabling a user, and removing a user from a security group is done with the **Microsoft 365 admin center**<sup>32</sup>. For more information, see **Disable a user account in an environment**<sup>33</sup>.

## Understand user security roles and security role defaults

Roles are groups of permissions that you can assign to a user to grant them access and various capabilities and functionality like read, delete, or edit of records in an entity within an environment. Roles are granular and can be assigned to one or many entities in an environment. Roles can also control certain actions like the ability to create a custom entity or option sets. Additionally, users are associated with one or many roles, and associating a user with a role gives them access to data and functionality that is specified within that role.

User security roles are either:

- Standard and created with every instance of Common Data Service.
- Custom and created by an administrator.

This unit examines each type of security role.

## Default user security roles

When you create a new instance of Common Data Service in an environment, a database is created with standard entities and several default security roles are created. The following predefined roles are available every time you create a Common Data Service environment by using the Power Apps portal. Unless otherwise noted, all the privileges have global scope.

Common Data Service includes several default roles with different access levels to standard entities and actions. The default standard roles are listed in the following table.

---

<sup>32</sup> <https://admin.microsoft.com/>

<sup>33</sup> <https://docs.microsoft.com/power-platform/admin/create-users-assign-online-security-roles#disable-a-user-account-in-an-environment/?azureportal=true>

Security role	Privileges	Description	
System Customizer	Create (self), Read (self), Write (self), Delete (self), Customizations	This role has full permission to customize the environment. However, users who have this role can view records only for environment entities that they create. To learn more, see <b>Privileges required for customization</b> ( <a href="https://docs.microsoft.com/dynamics365/customer-engagement/customize/privileges-required-customization">https://docs.microsoft.com/dynamics365/customer-engagement/customize/privileges-required-customization</a> ).	
Common Data Service User	Read, Create (self), Write (self), Delete (self)	Users who have this role can run an app in the environment and perform common tasks for the records that they own.	
Delegate	Act on behalf of another user	This role lets code run as a user or to impersonate another user. This role is typically used with another security role to provide access to records. To learn more, see <b>Impersonate another user</b> ( <a href="https://docs.microsoft.com/dynamics365/customer-engagement/developer/org-service/impersonate-another-user">https://docs.microsoft.com/dynamics365/customer-engagement/developer/org-service/impersonate-another-user</a> ).	

*Tip:* Add the System Customizer role to a user if you want them to be able to create new entities.

When you add a user to an environment in Common Data Service, the user is automatically assigned to the following:

- Security user roles - Common Data Service User
- Environment roles - Environment Maker

## Exercise-Create a custom role

Common Data Service has many standard default roles, but there might be times when you want to define a custom security role. Common Data Service supports the following eight different record-level privileges that can be used to define how a user interacts with data for

one or more entities for use in building a custom role. The available record-level privileges for custom roles include:

**Create** - Required to make a new record. The records that can be created depends on the access level of the permission that is defined in your security role.

**Read** - Required to open a record to view the contents. The records that can be read depends on the access level of the permission that is defined in your security role.

**Write** - Required to make changes to a record. The records that can be changed depends on the access level of the permission that is defined in your security role.

**Delete** - Required to permanently remove a record. The records that can be deleted depends on the access level of the permission that is defined in your security role.

**Append** - Required to associate a record with the current record. For example, if a user has Append rights on an opportunity, the user can add a note to an opportunity. The records that can be appended depends on the access level of the permission that is defined in your security role.

**Append To** - Required to associate the current record with another record. For example, a note can be attached to an opportunity if the user has Append To rights on the note. The records that can be appended to depends on the access level of the permission that is defined in your security role.

**Assign** - Required to give ownership of a record to another user. The records that can be assigned depends on the access level of the permission that is defined in your security role.

**Share** - Required to give another user access to a record while keeping your own access. The records that can be shared depends on the access level of the permission that is defined in your security role.

These record-level privileges can be grouped as needed and associated with a custom role. That custom role can then be applied to one or many entities as needed.

**TIP:** Roles can be copied so you can quickly create similar roles that might be slightly different.

## Create a custom security role and assign to entities and users

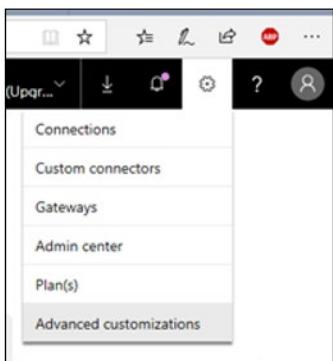
This lab will show you how to create a new role and associate that role with a custom entity. Then, you can associate users to the new role so they can access the data in the custom entities as needed.

To grant access, you will need to do the following:

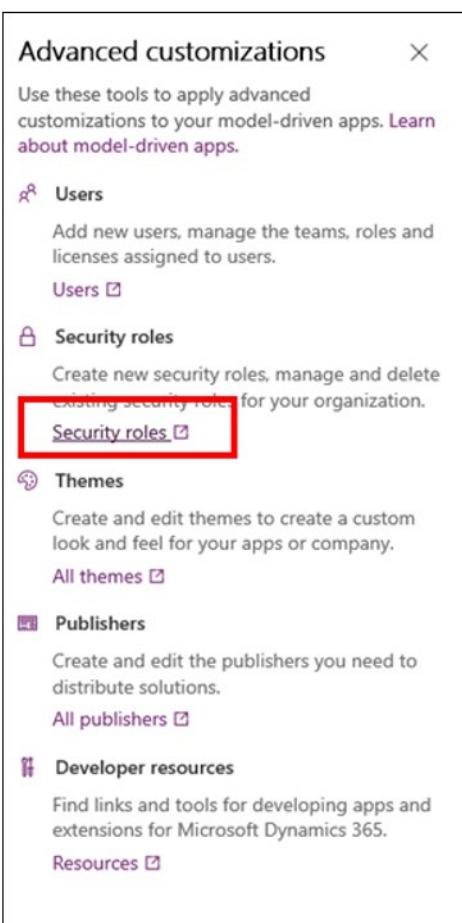
- Create a new user security role or amend an existing user security role to include settings for the custom entity.
- Assign users to the security role.

To get started, use the following steps to create a new security role.

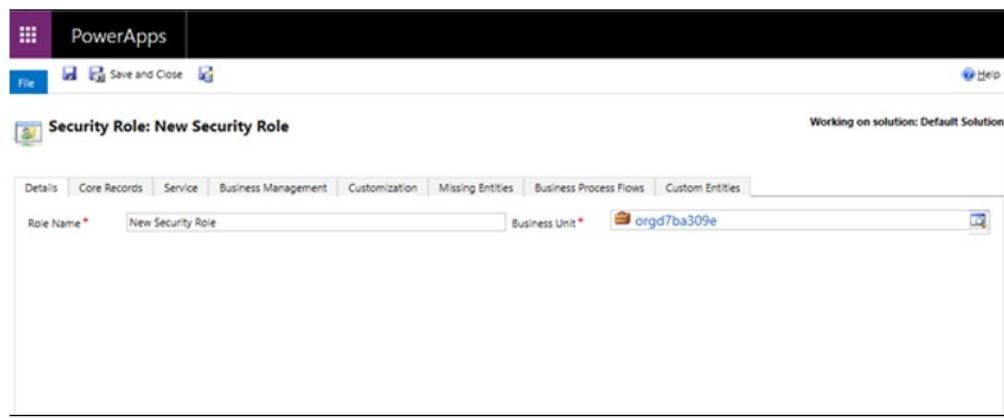
1. Sign in to Power Apps as an administrator.
2. Select the gear icon in the menu and select **Advanced Customizations**.



3. Select **Security Roles**. Note that this step can take a little time, depending on the number of instances in your tenant.



4. Select **New** in the menu bar, which will open the security role designer.
5. Enter a name for your security role in the **Role Name** field.



6. Locate the entities that your app uses by selecting each tab in the security role designer. If your entities are custom, they will be under the **Custom Entities** tab.
7. When you have located your entities, select the privileges that you want to grant your users, such as Read, Write, Delete, and so on. Select the scope for performing that action by selecting the name of the entity. Scope determines how deep or high within the environment's hierarchy that the user can perform a particular action.

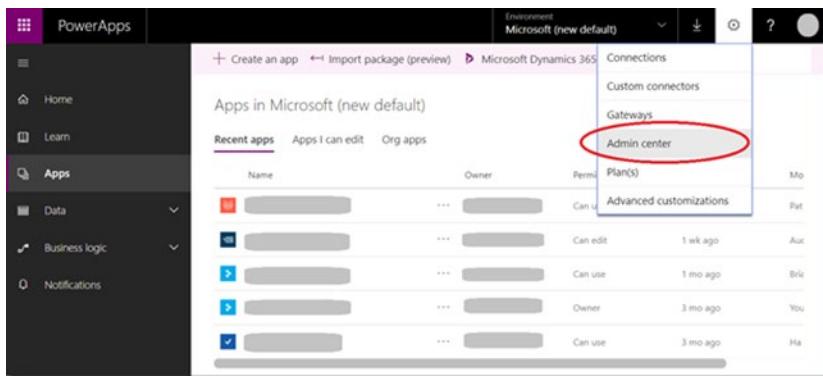


8. Select **Save and Close**.

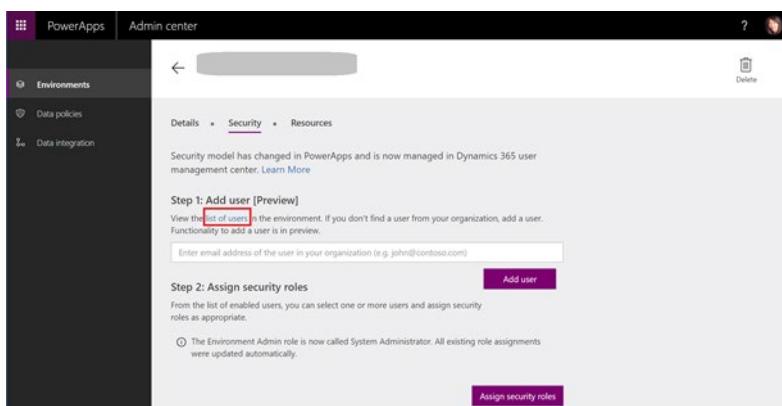
Congratulations, you have created a new custom security role. Next, you will assign users to this role.

To assign a user to a security role, you need to be a member of the System Administrator role in the current environment and then follow these steps:

1. Sign in to Power Apps as an admin, select the settings gear, and then select **Admin Center**.



2. In Power Apps admin center, select the environment where you want to update a security role.
3. Select the **Security** tab.
4. Verify if the user(s) already exists in the environment by selecting **View the list of users** in the environment. If the user is not on the list, go to step 5. Otherwise, you can skip to step 6.

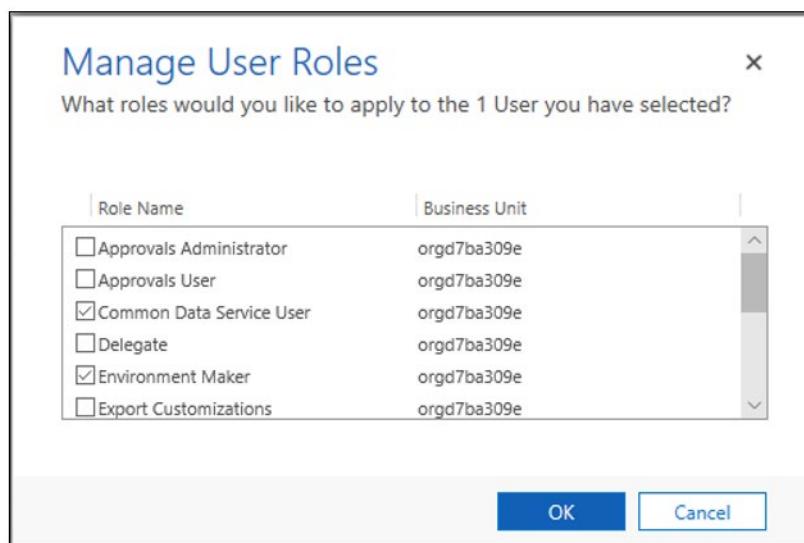


5. In case a user does not exist in the environment, you can add the user by entering the user's email address in your organization and then selecting the **Add user** button.
6. After you know the user(s) whom you want to assign a security role to exists in your environment, select the **Assign security roles** button.
7. Select the check boxes next to the user(s) that you want to assign and then select **Manage Roles**.

A screenshot of the 'Enabled Users' list screen in the Admin center. At the top, there are several buttons: NEW, APPROVE EMAIL, REJECT EMAIL, PROMOTE TO ADMIN, MANAGE ROLES, CHANGE BUSINESS UNIT, CHANGE MANAGER, and CHANGE POSITION. Below this is a search bar and a dropdown menu for 'Enabled Users'. The main area shows a table with columns: Full Name, Business Unit, Title, Position, and Main Phone. There are checkboxes next to each user's name. The table contains the following data:

	Full Name	Business Unit	Title	Position	Main Phone
<input type="checkbox"/>	admin admin	orgd7ba309e			
<input checked="" type="checkbox"/>	Alan Steiner	orgd7ba309e			
<input checked="" type="checkbox"/>	Amy Adams	orgd7ba309e			
<input checked="" type="checkbox"/>	Brad Sutton	orgd7ba309e			

8. In the **Manage User Roles** dialog box, in the **Role Name** section, select the check boxes next to the role(s) that you created in the previous section and make sure to also select the **Common Data Service User** role (if it wasn't already). The Common Data Service User role must be assigned to any user who wants to use your app or access Common Data Service.

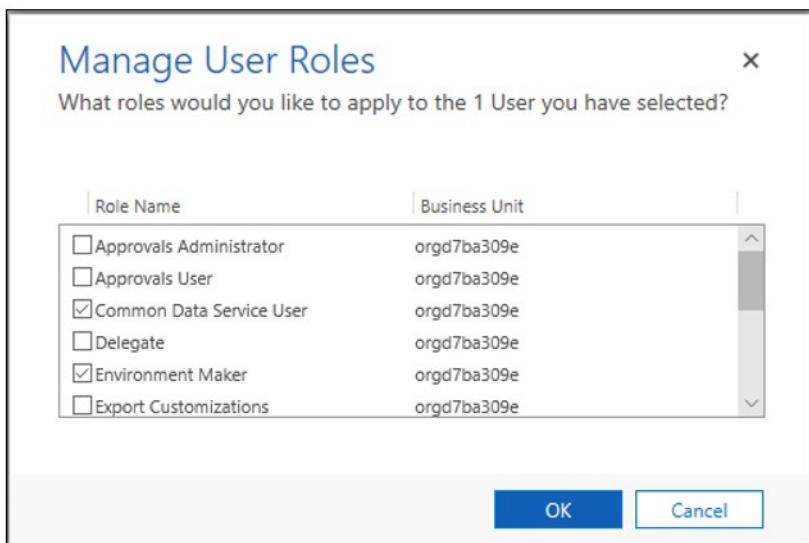


9. Select **OK** to assign the role(s) to the user that you selected.

## Exercise-Check the roles that a user belongs to

Checking the roles that a user belongs to is simple and you can do it from within Power Apps with the following steps:

1. Sign in to Power Apps as an admin.
2. Open the **Environments** option on the left-hand side of the page.
3. Select the environment where you want to check the user's permission settings.
4. Select the **Security** tab.
5. Select the **list of users** hyperlink under **Step One - Add User** option on the screen.
6. Select the check box next to the name of the user to view the security roles that the user has been assigned to.
7. In the **Manage User Roles** dialog box, select **Manage Roles** in the top menu to view user roles that are assigned to the user.



## Summary

In this module, you learned about security and the user in Common Data Service. In addition, you learned the following:

- What Environment Security roles are in Common Data Service
- How to add or disable a user within an environment
- What User Security roles are, and which are assigned by default when you add a new user to an environment
- How to create a custom security role
- How you can check the roles that are assigned to a user

The following links were referenced within this module:

**Power Apps Admin Center<sup>34</sup>**

**Power Platform Admin Portal<sup>35</sup>**

**Power Apps<sup>36</sup>**

<sup>34</sup> <https://admin.powerapps.com>

<sup>35</sup> <https://admin.powerplatform.microsoft.com>

<sup>36</sup> <https://www.powerapps.com>



## Module 2 Create a canvas app in Power Apps

### Get started with Power Apps

#### Introducing Power Apps

Welcome to Microsoft Power Apps. This self-paced, online module helps you build apps from the ground up.

In this module, you will:

- Explore how Power Apps can make your business more efficient.
- Learn which technologies to use to perform tasks in Power Apps.
- Learn about the different ways to build an app in Power Apps.
- Create your first app from data in a Microsoft Excel workbook.

In this introductory module, you'll learn how to create an app from data in an Excel workbook. As a prerequisite, you'll download a workbook that contains sample data. Next, you'll upload the workbook to Microsoft OneDrive for Business, where you can share the data with others. Then, you'll build the app without using code.

Power Apps is a suite of apps, services, connectors, and a data platform that provides you with an opportunity to build custom apps for your business needs. By using Power Apps, you can quickly build custom business apps that connect to your business data that is stored either in the underlying data platform (Common Data Service) or in various online and on-premises data sources (SharePoint, Excel, Office 365, Dynamics 365, SQL Server, and so on).

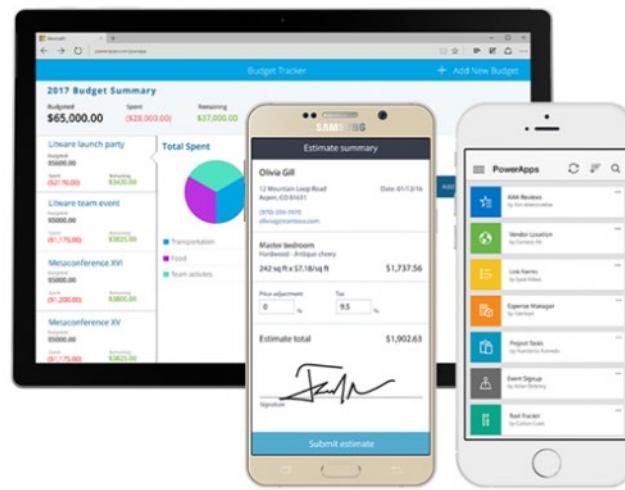
Apps that are built by using Power Apps provide rich business logic and workflow capabilities to transform your manual business processes to digital, automated processes. Power Apps simplifies the custom business app building experience by enabling users to build feature-rich apps without writing code.

Power Apps also provides an extensible platform that lets pro developers programmatically interact with data and metadata, apply business logic, create custom connectors, and integrate with external data.

With Power Apps, you can:

- Build an app quickly by using the skills that you already have.

- Connect to the cloud services and data sources that you're already using.
- Share your apps instantly so that coworkers can use them on their phones and tablets.



When it comes to using Power Apps to get things done and keep people informed, your options are nearly limitless. The following examples can help you think about how to use an app, instead of traditional paper notes, to run your business:

- **Equipment in the field** - Often, company representatives who visit customers in the field carry clipboards to help guarantee a paper trail of parts with scheduled replacement dates. By running an app on a tablet, reps can look up the customer's equipment, see a picture of a part, test and analyze the part, and then order new parts. Reps can perform these tasks on-site instead of leaving the customer's warehouse.
- **Restaurant employee management** - Employees of a large restaurant might fill out work schedules and vacation requests on a piece of paper that's affixed to a wall. With Power Apps running on everyone's smartphone, employees can open the app to record the same information, anywhere, anytime. The app can even send reminders for the start of the next day's shift.

If you're a beginner with Power Apps, this module gets you going quickly; if you're familiar with Power Apps, it ties concepts together and fills in the gaps.

## Power Apps building blocks

Power Apps is a collection of services, apps, and connectors that work together to let you do much more than just view your data. You can act on your data and update it anywhere and from any device.

To create, share, and administer apps, you'll use the following sites:

1. **Make a Power App<sup>1</sup>** - On this site, you can open apps, specify the type of app that you want to create, share your app, and create data connections and flows. To use this site, you'll need to log in by using your organizational account.
2. **Power Apps Studio<sup>2</sup>** - On this site, you build apps by configuring user interface (UI) elements and Excel-like formulas.

<sup>1</sup> <https://make.powerapps.com>

<sup>2</sup> <https://create.powerapps.com/studio/>

### 3. Power Apps admin center<sup>3</sup>

- On this site, you'll define environments and data policies.

Note: To use these sites, you'll need to sign in by using your organizational account.

When you've completed your tasks, you can run your apps in a browser or in Power Apps Mobile (available for Windows tablets, iOS devices, and Android devices).

## Power Apps building blocks

This unit explores each part of the following Power Apps components:

- **Power Apps Home Page** - Apps start here, whether you build them from data, a sample app, or a blank screen.
- **Power Apps Studio** - Develop your apps further by connecting to data, adding and arranging user interface (UI) elements (known as controls), and building formulas.
- **Power Apps Mobile** - Run your apps on Microsoft Windows, Apple iOS, and Google Android devices.
- **Power Apps Admin Center** - Manage Power Apps environments and other components.

## Power Apps Home Page

If you are building an app, you'll start with the **Power Apps Home Page**<sup>4</sup>. You can build apps from sample apps, templates, or a blank screen. All the apps that you've built appear here, along with any apps that others have created and shared with you.

The screenshot shows the Power Apps Home Page with a purple header bar. The left sidebar includes links for Home, Learn, Apps, Create, Data, Flows, AI Builder, and Solutions. The main content area features a heading 'Build business apps, fast' and a sub-section 'Create apps that connect to your data and work across web and mobile. Learn about Power Apps'. Below this are sections for 'Make your own app' (with four cards for 'PowerApps Training', 'PowerApps Training for Office', 'Leave Request', and 'Meeting Capture'), 'Start from data' (with icons for SharePoint, Excel Online, SQL Server, Common Data Service, and Other data sources), 'Learning for every level' (with four cards for 'Getting started with canvas apps', 'Get started with canvas app formulas', 'Show a list of items in Power Apps', and 'Share a canvas app in Power Apps'), and a 'Your apps' section at the bottom.

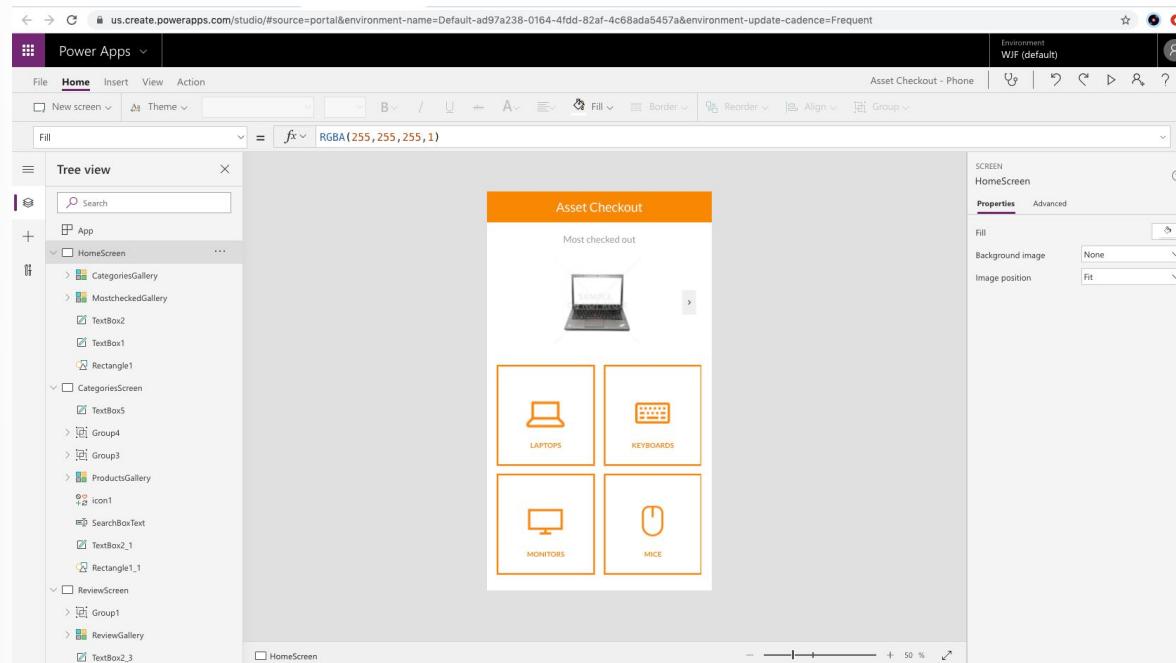
<sup>3</sup> <https://admin.powerplatform.microsoft.com/>

<sup>4</sup> <https://make.powerapps.com>

## Power Apps Studio

Power Apps Studio is where you can fully develop your apps to make them more effective as a business tool and to make them more attractive. Power Apps Studio has three panes that make creating apps seem more like building a slide deck in Microsoft PowerPoint:

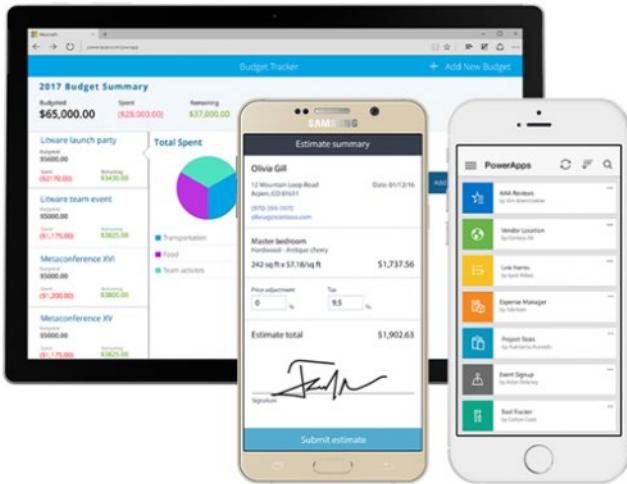
- **Left pane** - Shows a hierarchical view of all the controls on each screen or a thumbnail for each screen in your app.
- **Middle pane** - Shows the canvas app that you're working on.
- **Right pane** - Where you set options such as the layout, properties, and data sources for certain controls.



## Power Apps Mobile

Power Apps Mobile for Windows, iOS, and Android devices allows you to use all the apps that you've created, and those others have shared with you, on your mobile device. You or your users can download the Microsoft Power Apps app from the appropriate app store. When users log in with their credentials, they will see all apps that have been shared with them. The Power Apps Mobile app only needs to be downloaded once.

When you use apps in Power Apps Mobile, you get the most out of your device's capabilities: camera controls, GPS location, and more.



## Power Apps Admin center

The Power Apps admin center is the centralized place for managing Power Apps for an organization. On this site, you can define and manage different environments to house the apps. For example, you might have separate environments for development and production apps. Additionally, you can define data connections and manage environment roles and data policies.

## Licensing

Most users get their initial start with Power Apps by utilizing one of the licenses that comes with their Microsoft 365 Plan or Microsoft Dynamics 365 license. These licenses allow you to extend the functionality of the app that is licensed. This means if you purchased a Microsoft 365 plan that included a Power Apps license then you can build apps that extend and use SharePoint as a data source. But Power Apps doesn't have to stop at just extending that platform.

Power Apps has over 300 available data source connectors available including Common Data Service. To incorporate Common Data Service or any of those additional connectors all users of the app will need a premium license. There are two different ways to acquire a Premium license:

- Per App model
- Per User model.

The Per App license allows users to access premium connectors for a specific business solution. The Per User license allows users to run unlimited premium licensed apps. This gives you the ability to grow with Power Apps and control cost by purchasing the license that best matches your business goals.

In addition, Power Apps also has the capability to use Power Apps portals to build externally or internally facing websites using Common Data Service and Power Apps controls. Power Apps portals have their own licensing model and are not included in any of the licenses discussed previously above. With Power Portals you will purchase a capacity based license to meet your business needs.

Review the following links about licensing.

### **Microsoft Power Apps pricing<sup>5</sup>**

<sup>5</sup> <https://powerapps.microsoft.com/pricing/?azureportal=true>

### Microsoft Power Automate pricing<sup>6</sup>

### Microsoft Power Apps portals pricing<sup>7</sup>.

## Power Apps related technologies

Microsoft Power Apps works with other technologies to help you build powerful apps for your organization. Some of these technologies include:

- **Data sources** - Without data, you don't have a business. Data sources bring cloud and on-premises data into your apps. You access data through built-in connections, custom connectors, and gateways.
- **Common Data Service** - A compliant and scalable data service that's integrated into Power Apps.
- **Power Automate** - Allows you to build automated workflows to receive notifications, run processes, collect data, and more.

## Data sources, connections, and gateways

In Power Apps, most canvas apps use external information that is stored in Data Sources. A common example is a table in an Excel file that is stored in OneDrive for Business or SharePoint. Apps access these data sources by using connections. Some connections allow Power Apps to read and write stored data. In Power Apps, you can add many data sources to your apps through built-in or custom connectors. Some of the most popular data sources are shown in the following figure.

	Common Data Service		Office 365 Outlook
	SharePoint		Excel
	SQL Server		OneDrive for Business
	Dynamics 365		OneDrive
	Office 365 Users		Dropbox

Many data sources are cloud services, like Salesforce. Even Twitter can be a data source if, for example, you're tracking your company's hashtags. Connectors might not seem like the most exciting part of app development; however, they're essential when you work with data that you, your colleagues, and your customers care about. When an app shows up with your data source for the first time, you might suddenly find that they are, in fact, exciting.

For data that's stored on-premises instead of in the cloud, you can use a gateway to provide a reliable connection between Power Apps and your data source. The gateway sits on an on-premises computer and communicates with Power Apps.

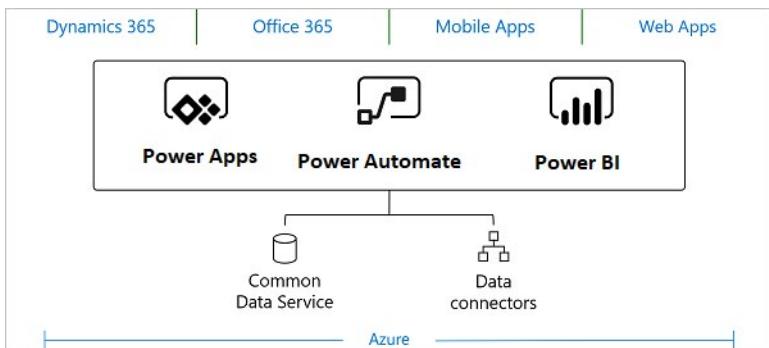
An advantage of building your business apps in Power Apps is being able to connect to many data sources in a single app. With the connectors in Power Apps, you can connect to where your data lives. To learn more about data sources in Power Apps, refer to the Working With Data learning path.

<sup>6</sup> <https://us.flow.microsoft.com/pricing/?azureportal=true>

<sup>7</sup> <https://powerapps.microsoft.com/portals/?azureportal=true>

## Common Data Service

An important data source option to explore further is the Common Data Service. Common Data Service lets you store and manage data that's used by business applications. Data within Common Data Service is stored within a set of entities. An entity is a set of records that are used to store data, similar to how a table stores data within a database. Common Data Service includes a base set of standard entities that cover typical scenarios, but you can also create custom entities that are specific to your organization and then populate them with data by using Power Query. App makers can then use Power Apps to build rich applications by using this data.



For information on purchasing a plan to use Common Data Service, refer to the [License<sup>8</sup>](#) and [Pricing<sup>9</sup>](#) information pages.

## Reasons to use Common Data Service

Standard and custom entities within Common Data Service provide a cloud-based storage option for your data. Entities let you create a business-focused definition of your organization's data for use within apps. If you're unsure if entities are your best option, consider the following benefits:

- **Simple to manage** - Both the metadata and data are stored in the cloud. You don't need to worry about the details of how they're stored.
- **Helps to secure data** - Data is stored so that users can see it only if you grant them access. Role-based security allows you to control access to entities for different users within your organization.
- **Access your Dynamics 365 Data** - Data from your Dynamics 365 applications is also stored within the Common Data Service, which allows you to quickly build apps that use your Dynamics 365 data and extend your apps by using Power Apps.
- **Rich metadata** - Data types and relationships are used directly within Power Apps.
- **Logic and validation** - Define calculated fields, business rules, workflows, and business process flows to ensure data quality and drive business processes.
- **Productivity tools** - Entities are available within the add-ins for Microsoft Excel to increase productivity and ensure data accessibility.

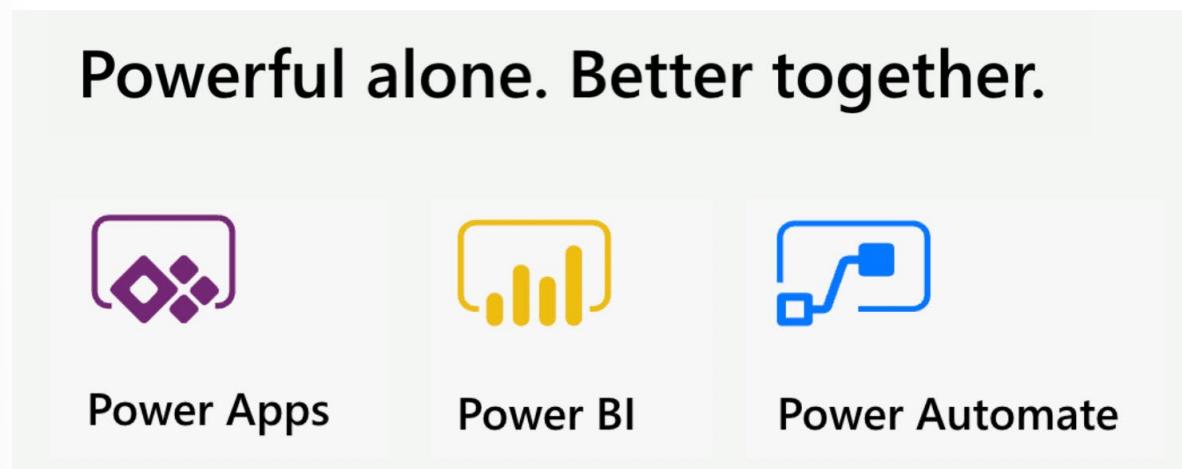
## Related Power Platform technologies

As you continue developing your application, you may want to consider implementing additional Power Apps related technologies such as Power Automate and or Power BI. For example, you may have a simple

<sup>8</sup> <https://docs.microsoft.com/power-platform/admin/pricing-billing-skus>

<sup>9</sup> <https://powerapps.microsoft.com/pricing/>

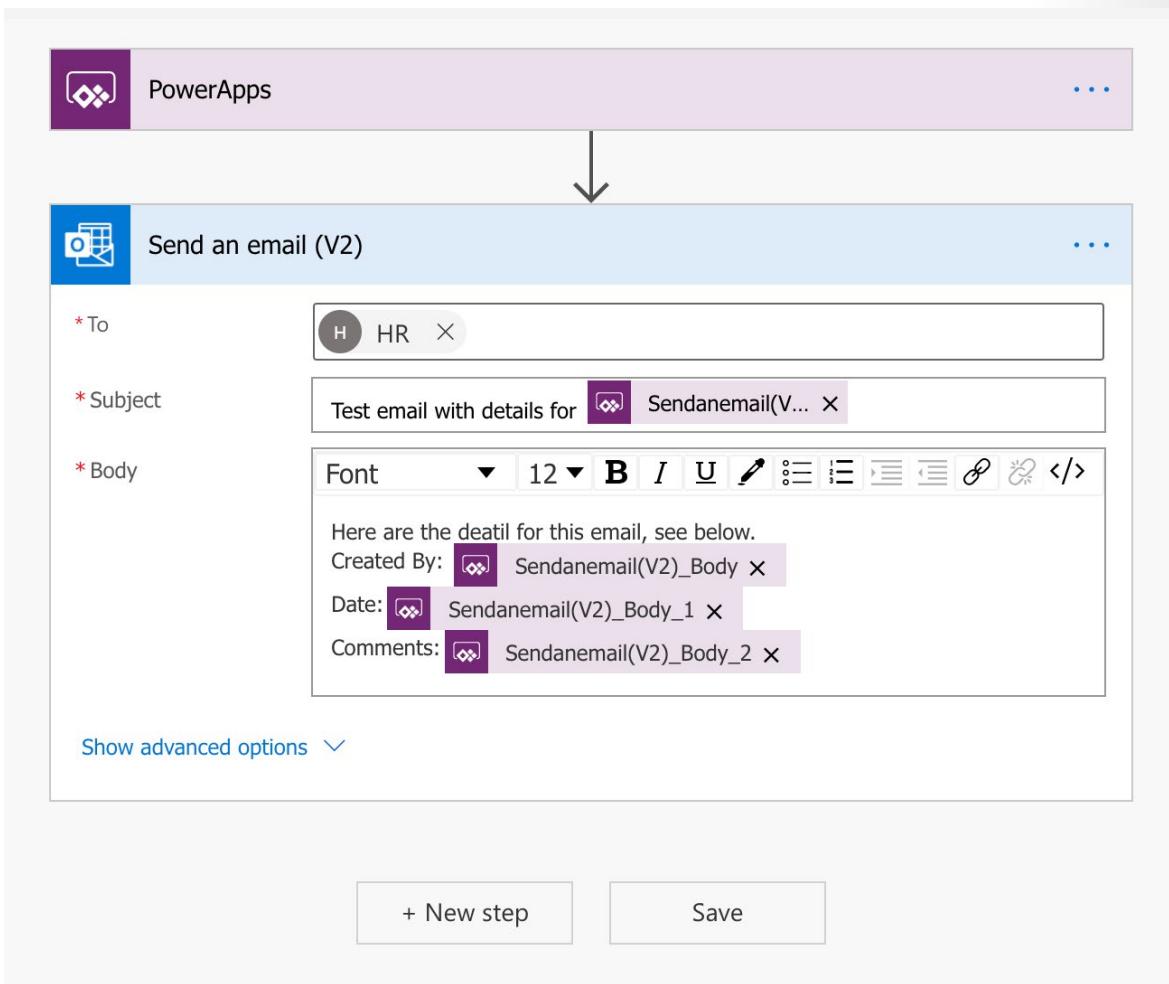
Expense Report App that requires an approval before an item can be purchased. With Power Automate, you can create a simple Flow to make this happen. Or maybe you want to display your data with custom charts and graphs giving your users a more visual look into the data, which can often be useful. In this section, you will learn more about some of the other Power Platform technologies and how you could apply them in your own Power Apps solution. Keep in mind, if you decide to implement these Power Apps related technologies you should also review their licensing structure and associated costs.



## Power Automate

Power Automate brings automation to your business. This can be traditional workflows via flow, Robotic Process Automation (RPA) for automating legacy systems via UI Flows, or business process automation via Business Process Flows. Each of these capabilities increases your productivity to connect disjointed systems to build the business solution you need and make your app more powerful.

You can use Power Automate to create logic that performs one or more tasks when an event occurs in a canvas app. For example, configure a button to execute a flow to do one of the following: create an item in a SharePoint list, send an email or meeting request, or add a file to OneDrive. The button could be configured to do all of those in a single Power Automate flow. You can configure any control in the app to start the flow, which continues to run even if you close Power Apps. Below is an example using Power Automate to send a flow:



## Identify Flows in your Solution

Now that you have a general overview of Power Automate, how do you determine if the solution you're building requires a Flow? There are a number of simple functions Power Apps can do, like sending an email when a button is pressed in your application. This email generated from Power Apps can also contain dynamic/specific information and be sent to any email address you would like. Often, customers will use Power Automate to create this same functionality even though Power Apps can do this out of the box. Power Automate should be used for more complex solutions, such as the approval workflows. With Power Automate you can run an approval when a button is pressed, on a schedule, when an item is created or modified, and so on.

For many Power Apps solutions Power Automate is used to handle complex business logic. Do you need a way to make sure someone actioned on the incident report that was generated by your app? Or, do you need a process to kick off every time new data is created in another system so Power Apps will have the data it needs? Do you need to check each morning to see if an inspection is due that day and then send an email with a link to your Power Apps inspection form? These are great uses of Power Automate to transform your app from a point solution to a fully featured business solution.

## Power BI

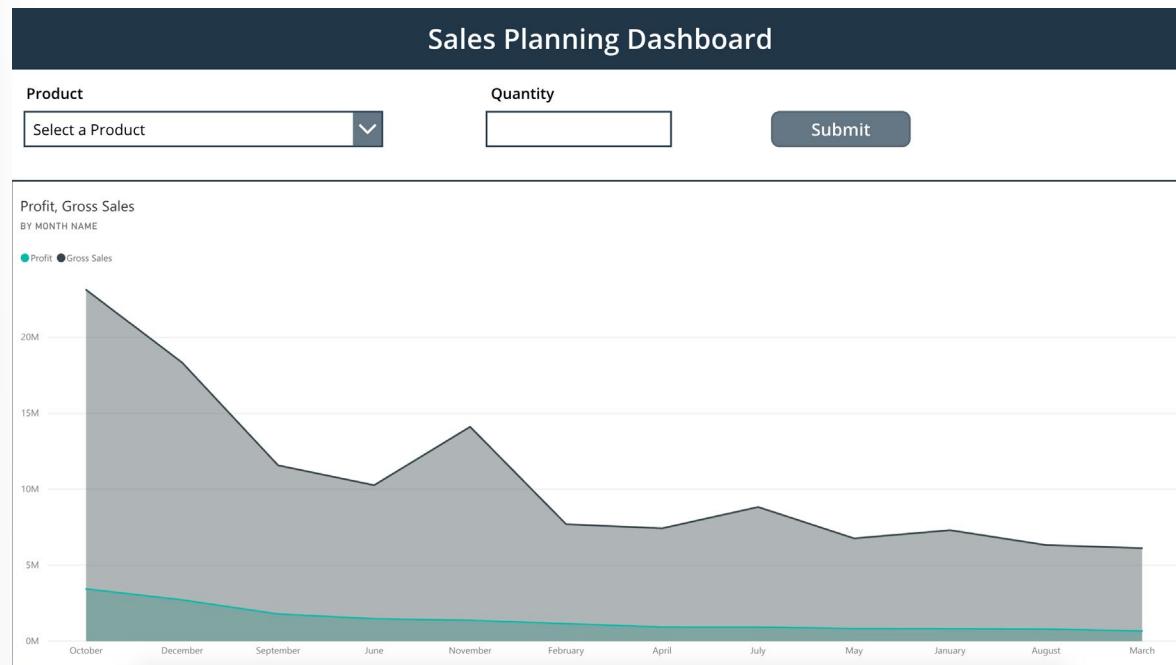
Power BI is an analytics tool within the Power Platform suite. Power BI connects data from multiple sources and transforms the data into graphical visualizations to gain insights. It allows business users to utilize a number of different visualizations to build comprehensive reports and dashboards. When creating Power BI reports to view and analyze your app data, you have the ability to customize them for personal use and will only be accessible by you, providing you with a more unique and custom experience. If you need to share the report with others, you and each of the report consumer will need a Power BI Pro license. This license allows you to not only share the content but also control what others are able to do with the shared report or dashboard.

While Power Apps has capabilities to include simple graphs or tables, many solutions would be better served with a visualization provided by Power BI. Power Apps and Power BI have two options for seamless integration:

### Embed a Power BI tile in a Power Apps app

By embedding a Power BI tile in a Power Apps solution, you are able to bring valuable visualizations into the app to allow the user to consume that data within the context of the app.

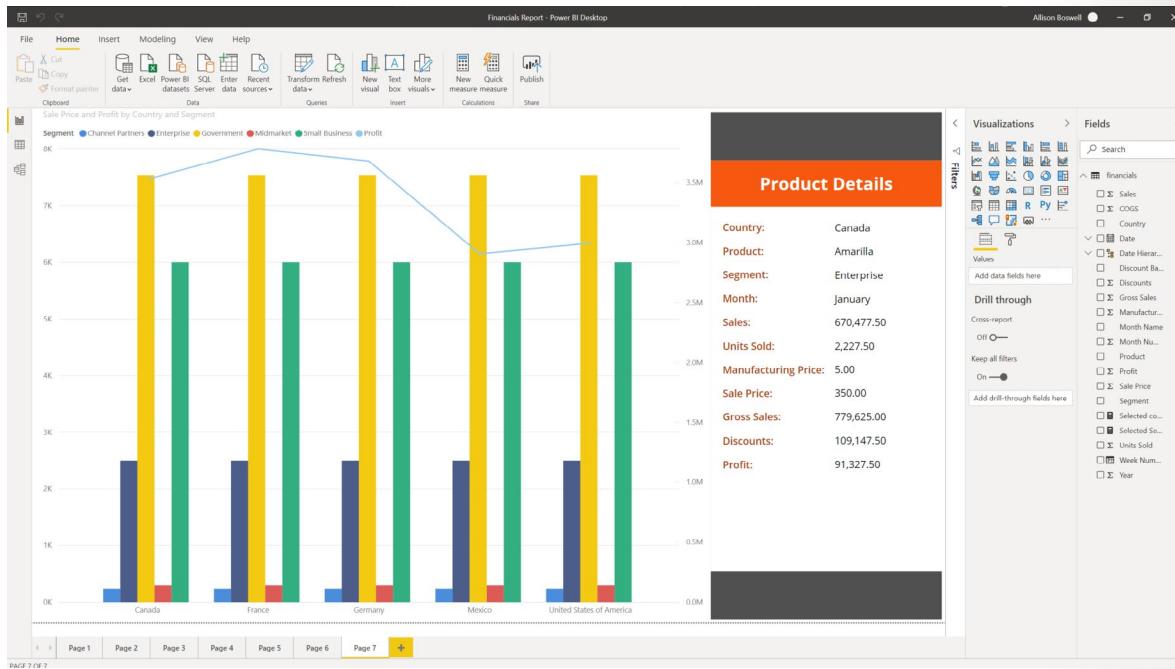
In the example below, you will see a simple Power BI Tile embedded in a Sales Planning app built in Power Apps. The visual is displaying the Profit and Gross Sales and the Power Apps form allows the user to enter sales predictions.



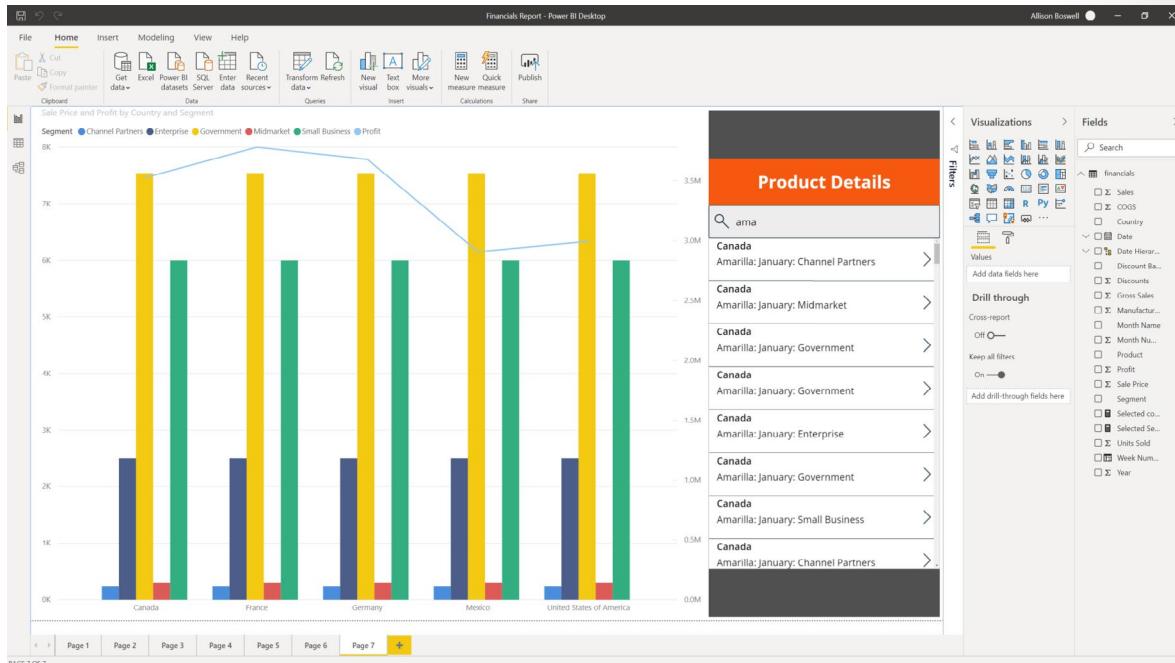
### Embed a Power Apps app in a Power BI Dashboard

Another integration between these two applications, is to embed a Power Apps app in your Power BI report. This allows the user to action on data while never leaving the dashboard resulting in a better user experience. Consider an inventory management dashboard for a manufacturing facility. Without leaving the dashboard, the user can submit to purchasing an order for additional material. While the solution may have been utilizing both the Power Apps and Power BI platforms, the user simply experiences a complete end to end solution in one window on their desktop.

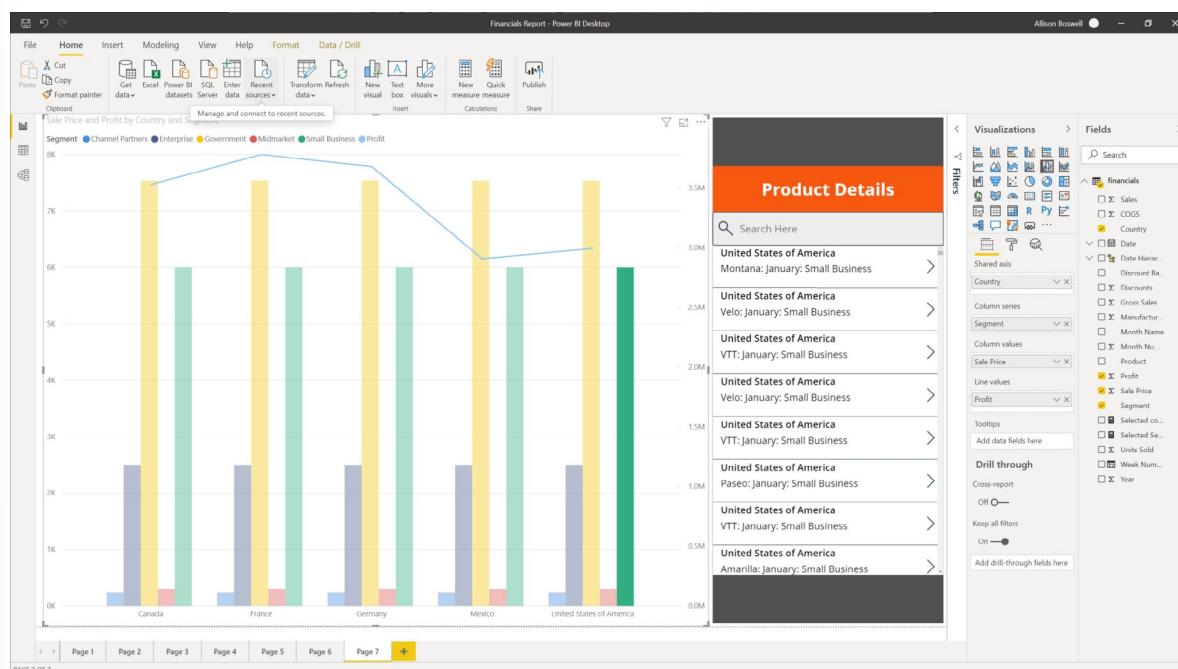
In the example below, we are analyzing the Sale Price and Profit by Country and Segment. Notice once you have embedded your Power App in a Power BI Dashboard you can navigate between screens.



In this next screenshot, still working with the same data as the previous example, you can utilize the native Power Apps features like Search with Power BI data.



In this last screenshot, for this example, you will see the embedded Power App is filtered by the Power BI selection.



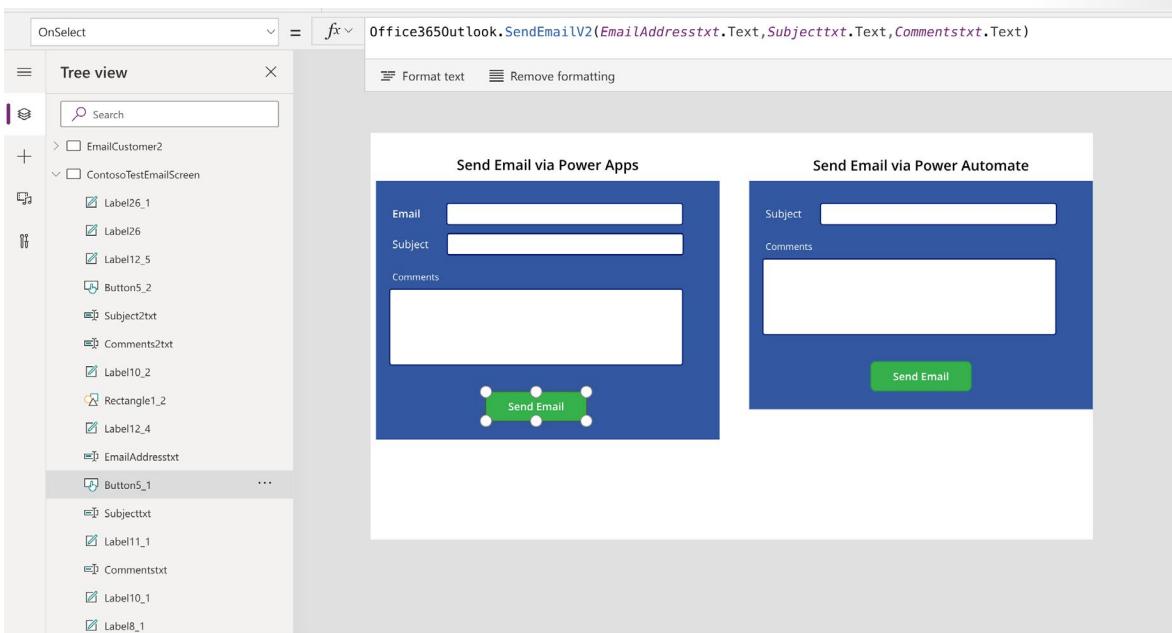
## Translating needs to the appropriate technology

To build the best solution, think through the use cases and determine how you want to collect the data, use the data, and analyze the data. Once you have determined how the solution will be used in each one of those cases, you can begin to select the right technology to execute each function.

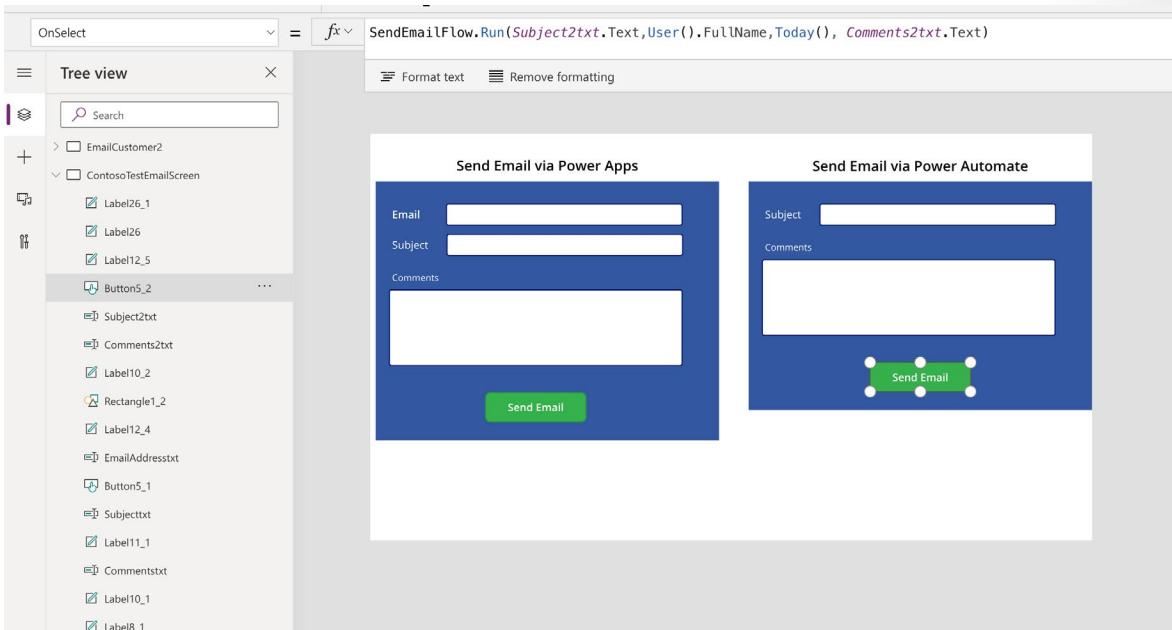
It would be difficult to cover every use case and decision point, but to help you understand the decision-making process let's explore sending an email via Power Automate versus sending an email via Power Apps. First consider the look and feel of the email, does your solution require special formatting of the email? To format the text of your email in Power Apps, like adding italics or bold text, you would need to write HTML. In Power Automate though, this functionality can be implemented by using the simple Design Interface that is provided out of the box.

Below are examples of the formulas to execute sending an email via Power Apps versus via Power Automate.

### Send an Email via Power Apps



### Send an Email via Power Automate



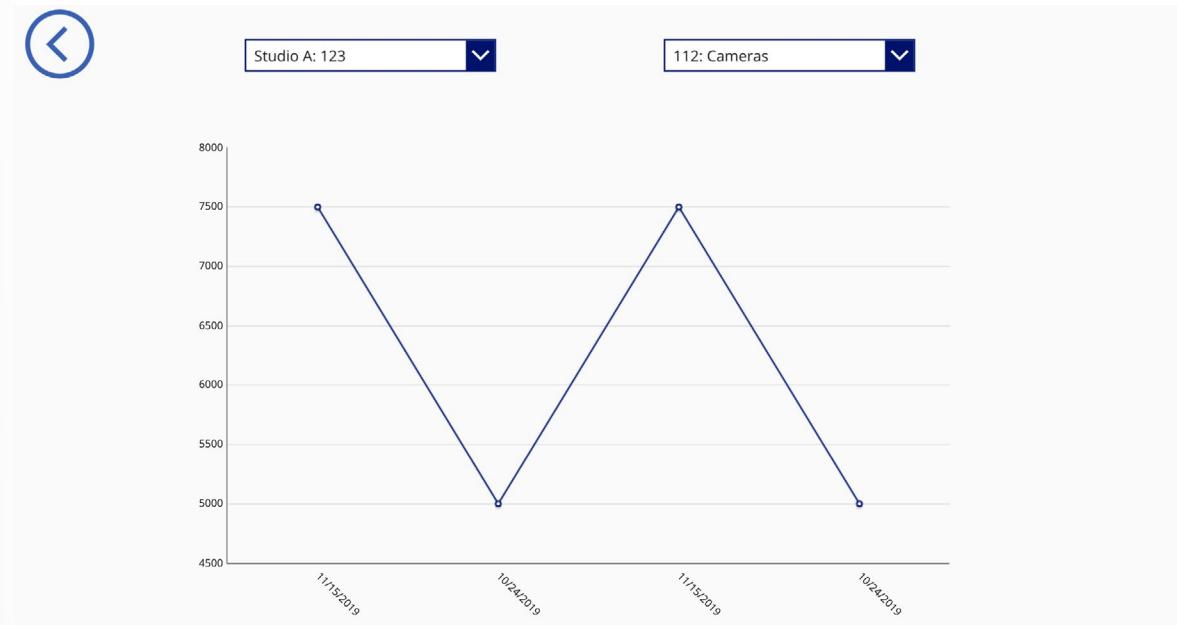
Also, the number of steps in your solution/process will aid you in determining which technology best suits your needs. Power Apps is great for performing simple solutions with minimal steps but as your solutions become more complex and requires multiple steps, Power Automate would be the better solution.

Again for this particular example, both technologies can provide the same solution, but there are little nuances that should be considered and thoroughly discussed during the design process to determine your requirements and help you choose the best product for your solution.

Let's not forget about discussing Power Apps and Power BI, and when to use one vs. the other. When deciding whether to use the basic charts, graphs, and visuals that come with Power Apps out of the box

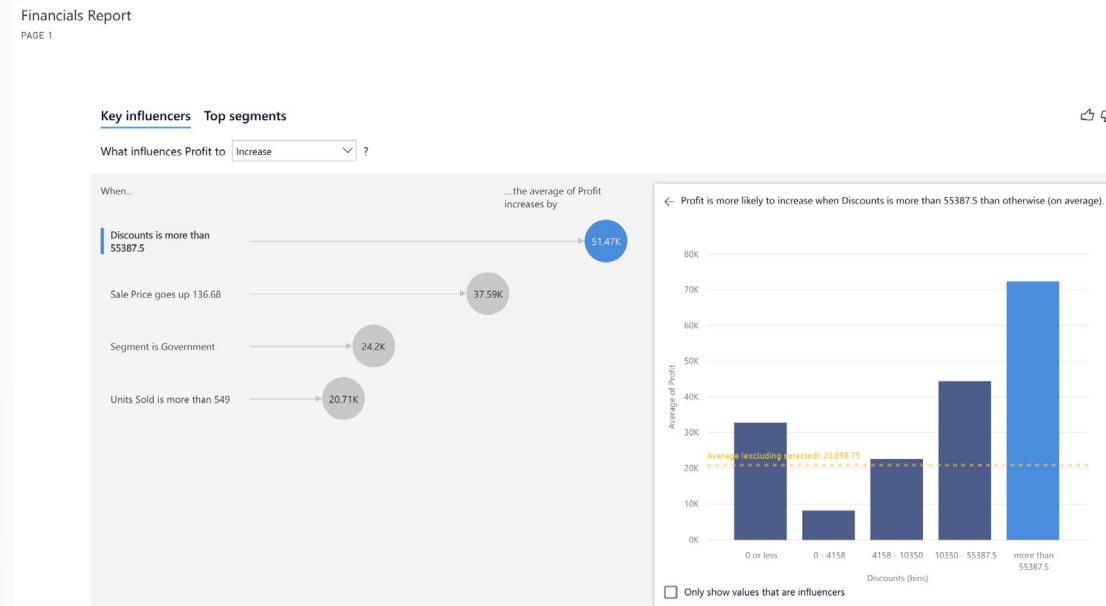
or to utilize a more powerful software like Power BI it really depends on your business solution and requirements. For example, if in your solution, you are wanting to add some basic graphs and charts to improve the apps overall look and feel while adding some visual flair for your users, Power Apps has you covered.

Here is a quick look at one of the simple, out of the box Power Apps charts.



Simple and minimal design above, nothing crazy but it gets the job done.

On the other hand, if your solution requires in-depth analysis of your data, and robust visuals, Power BI will be the best product for your solution. Keep in mind, with Power BI, each app user will need an additional license on top of the Power Apps license. This is a small price to pay though if our solution relies on intuitive dashboards, charts, graphs, and several other features to help you get the most out of your solution.



By identifying the needs of related Power Apps technologies in your solution and strategically implementing them, you will be able to provide your users with a better overall experience when using the solution.

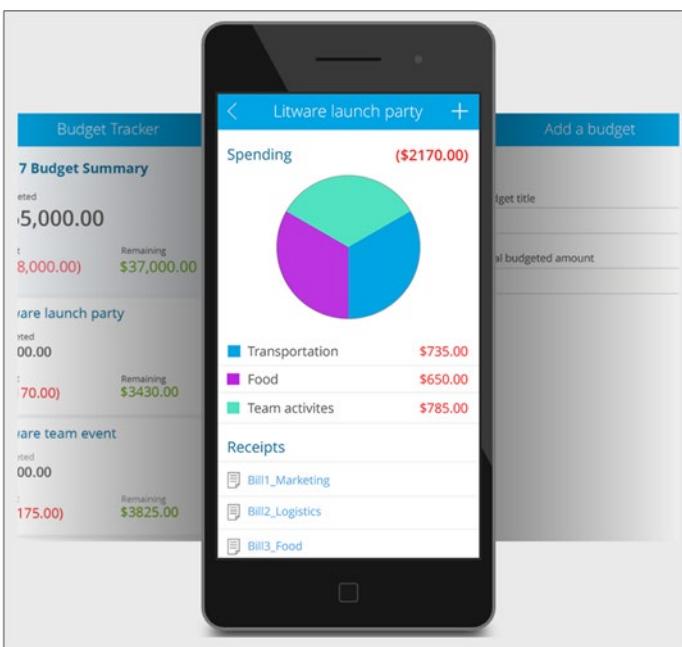
## Ways to build Power Apps

This unit describes how to create an app from a template, a blank canvas, and a data source. In the next unit, you'll be able to get more hands-on with app creation. This learning path is focused on canvas apps, which give you the flexibility to arrange the user experience and interface the way that you want it. You can get started in many different ways; however, for all of the options, you will use the Power Apps Studio features and functionality to build your app.

### Create an app from a template

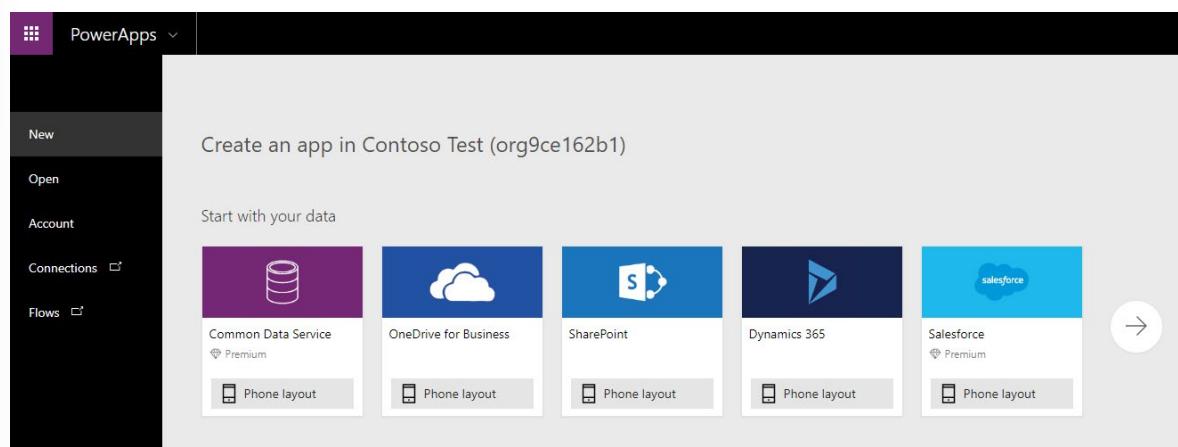
A good way to create an app is to start from a template. Templates use sample data to help you determine what's possible. By opening templates in Power Apps Studio, you can learn, hands-on, how an app is built.

For example, you can use the Budget Tracker template to create an app that helps you track the budget for projects and events with custom categories, simple data entry, and visuals that highlight expenditures for an effortless inspection.



### Create an app from a data source

Another great way to get started is to generate an app from your own data. Simply point Power Apps at the data source of your choice (for example, a list in Microsoft SharePoint or Common Data Service), and watch as Power Apps automatically builds a three-screen app. This three-screen app lets you display, edit, delete, and create records.



A special data source is SharePoint. Modern lists in SharePoint and Power Apps have a tight integration. You can either build an app from within a SharePoint site or you can use Power Apps to customize your modern list forms.

The screenshot shows a SharePoint list titled "Clients". The top navigation bar includes "Sales - Clients - All Items", a back/forward button, a lock icon, and the URL "https://shanescows.sharepoint.com/sites/TestSales/Lists/Clients/AllItems.aspx". Below the navigation is a "Apps" bar with a message to import bookmarks. The main content area shows a table with columns "Name", "Category", and "Image". The "Name" column lists items like "Ageless Beauty Clay", "Bianco Carrara Marble", "Bolivian Rosewood", "Brazilian Koa", "Caserta Sky Grey", "Caserta Stone Beige", "Edimax Slaty Porcelain", and "Golden Teak". The "Category" column shows values like "Carpet", "Tile", and "Hardwood". The "Image" column displays small thumbnail images of the floorings. At the top of the list view is a ribbon with "Search", "+ New", "Quick edit", "Export to Excel", "Flow", "PowerApps", and a three-dot ellipsis. A context menu is open over the first item in the list, showing "Create an app" and "Customize forms".

The following app was created from a SharePoint list and lets you browse items in the list, view item details, and create and edit items. After Power Apps generates an app, you can customize it to make it look and behave exactly the way you want.

The screenshot shows a Power Apps generated app for a "FlooringEstimates" list. On the left is a list view with items: Ageless Beauty Clay (Carpet), Bianco Carrara Marble (Tile), Bolivian Rosewood (Hardwood), Brazilian Koa (Hardwood), Caserta Sky Grey (Carpet), Caserta Stone Beige (Carpet), Edimax Slaty Porcelain (Tile), and Golden Teak. In the center is a larger view of the "Brazilian Koa" item, showing its name, category, and a large image of the wood grain. Below the image is a "Overview" section with the text: "Its distinctive orange coloring and brown/black striping make it one of the most unique and exotic species available on the market today." On the right is another view of the same item, showing fields for "Name" (Brazilian Koa), "Category" (Hardwood), and "Price" (5.69).

## Build from a blank canvas

You can also build an app from the ground up and add all the pieces as you go. You can then branch out and use your imagination.

## Designing a Power Apps app

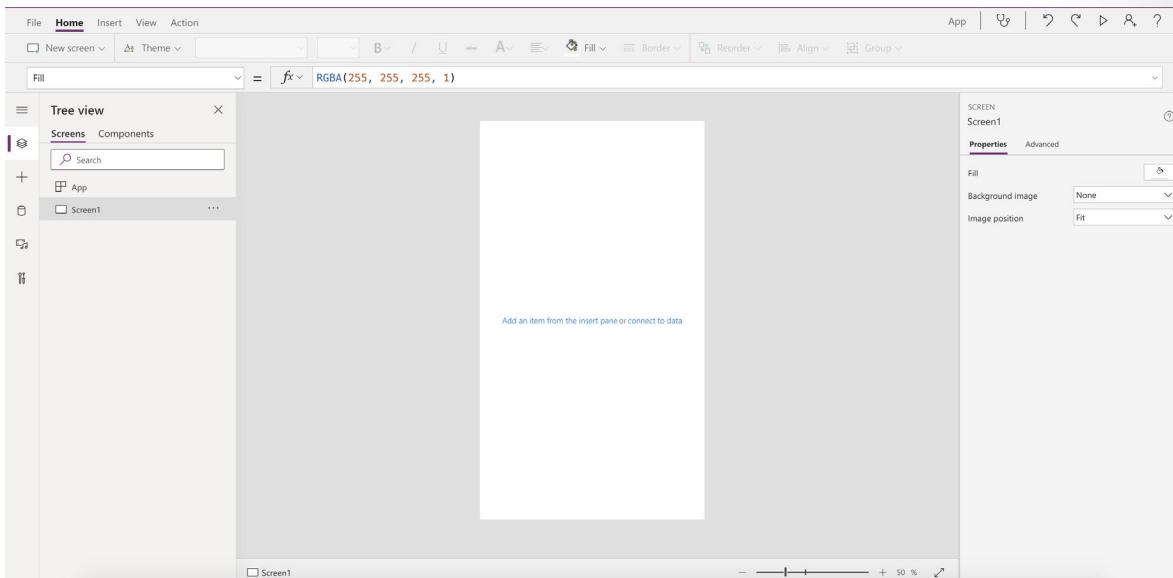
As an App Maker, before you begin building your Power Apps solution, it's recommended to go through a design process.

When designing your Power Apps solution, there are several different factors to consider, such as:

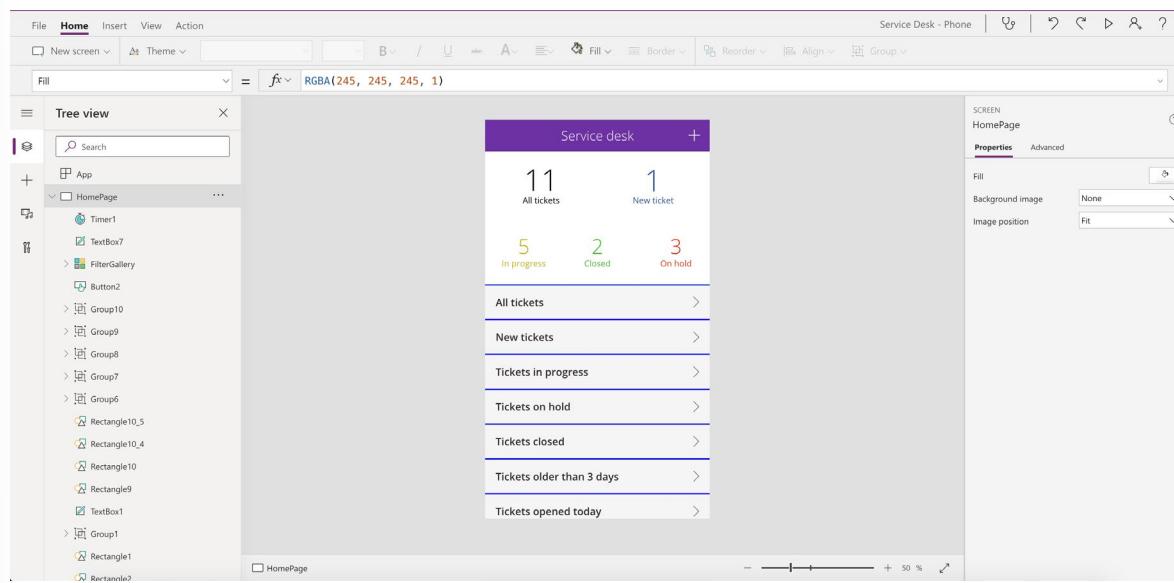
- Business requirements
- Data Model
- User Experience (UX)
- User Interface (UI)
- Business Logic
- Output

By going through a simple design process, you can flush out any minor issues before they become a larger problem once the app has been put into production. It is also important to understand that this design process is for Canvas apps.

So how do you go from a simple blank Canvas app, as seen below?



To a fully customized Canvas app solution?



## Understand the needs of the user

One of the most powerful and, at the same time, challenging parts of building a canvas app is that you start with a blank screen. This gives you the ability to build what you want, but to do that you have to know what you want.

In many cases, when purchasing software to solve and or streamline business solutions, there are many business processes that don't quite fall within the software's supported guidelines. When you run into this issue, typically, there are several internal discussions and meetings held to determine how those unsupported processes can be updated/ altered to meet the software requirements. For most organizations, this isn't ideal because of the cost or time takes to update those business processes. The great news is, by using Power Apps to build your solution, you won't have to worry about unsupported business solutions. Why? With Power Apps, you can build a custom solution tailored to the exact needs of your business requirements.

Often when building an app, you are tempted to recreate the piece of paper or legacy software-driven process exactly. This is possible but might not be the ideal solution. By challenging the existing process and asking what it is the business needs to do, not what does the piece of paper or old software allows you to do, it opens the possibility of better, more efficient processes. For example, maybe on the paper process, the user had to type notes about what they see. Would it be better instead to just take a picture? This type of thinking will lead to better apps and better outcomes.

## Business Requirements

Every app you develop will have a different set of business requirements based on the solution. Taking the time to think about all the requirements is key to rolling out a successful production app.

Depending on the solution or company policies, you may have certain security, privacy, or compliance requirements you must follow. For example, let's say you are collecting secure personal information in the app. You will want to ensure this information is securely stored and not visible to everyone.

During this process, you will also want to identify any government regulations or authentication/authorization requirements (if applicable). You don't necessarily have to have all the answers to your questions here; you just want to know all the requirements.

## Offline Mode

One of the first questions to consider when developing your application is, will the app need to function offline? If so, will the entire app or only part of the app needs to function offline? When will the data be synchronized to my data source? Are there any limitations?

This is important to consider during the planning phase because if you were to build your app without this functionality, then decide to add it later, it will be more difficult than just doing it in the first place. Why is this? You will need to make sure you are using collections and additional functions like SaveData and LoadData as you go along to allow your app to function offline. Also, if you are using Forms and trying to implement Offline mode, you will run into issues.

There is a thorough discussion that needs to take place around Offline mode, and it's best to have this early in the design process as it will affect the rest of the process.

## Data Model

In the “Power Apps related technologies” module, you learned about some of the common data sources for building apps, but with all these choices how do you actually decide which data source to use for your solution? Maybe you already have a data source implemented that users work with on a day to day basis, like SharePoint. Could you just use this as your data source to build your app? Do I need to connect to multiple data sources? These are all common questions you should ask yourself and there are number of additional factors to consider, such as:

- **Business Requirements** – Every data source and its supported functionality is slightly different. So, depending on your app requirements you need to select the data source that supports your needs or modify your business requirements to comply with the supported functionality for the selected data source.
- **Licensing/Cost** – Certain data sources like the Common Data Service or SQL are considered a “premium data source”. A premium data source will require each user who uses the app to have a Power Apps Per App Plan or a Power Apps Per User Plan. For more information about licensing, see [Power Apps pricing<sup>10</sup>](#)

## User Experience (UX)

By designing your Power Apps solution in a Canvas app, you have complete control of the end-user experience. This allows you to fully customize nearly every aspect of your app. However, just because you can doesn't necessarily mean you should. When designing your Power Apps solution your goal should be to keep it simple. When your end users open the application and begin using it, they should have no confusion about what to click on or where to go. If your app requires an extensive training program for users to understand how to use it, you may want to rethink your app.

Some of the basic designs elements you will want to consider are things like:

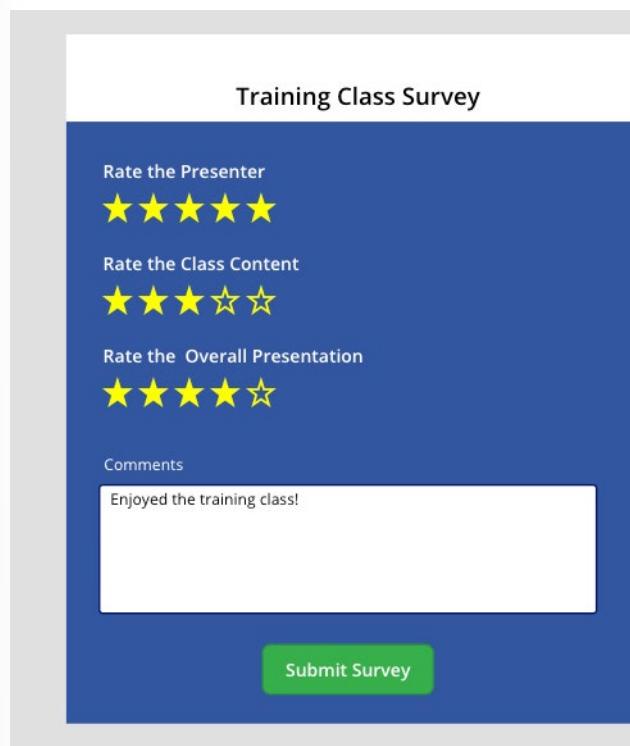
- Custom Branding (your logo and colors)
- Pop-ups
- Hide/show buttons based on users' access/permissions

One of the most common User Experience enhancements you can implement in your applications are Pop-ups. By implementing pop-ups, you can provide the users with a simple, but useful visual to confirm what the user clicked on went through or maybe your pop-up acts as a loading screen as the logic on the

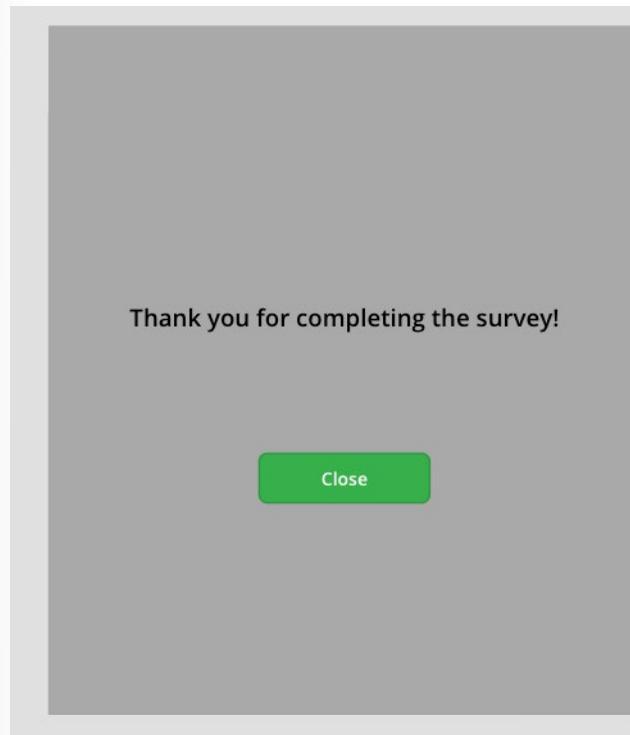
<sup>10</sup> <https://powerapps.microsoft.com/pricing/?azureportal=true>

backend is processed. For example, in the screenshot below when a user clicks on "Submit", we might have a simple pop-up display to let them know their submission was successful.

In this example app, the user completes a Survey for the training class they just attended.



Once all of the information has been written to the data source successfully, a pop-up is generated to confirm the submission was successful.



Without confirmation, a user may not be sure if their submission was accepted. They may try to click the button again, causing incorrect or inconsistent data being written to the data source.

Remember, these are not the only customizations you can make to the app, these are just some of the common ones. Another thing to keep in mind as you add different design features is the more logic you add for the customization of the app the more code your application will need to process. So, for example, if you add several different functions for hiding buttons, or showing popups on a given screen, this could cause your application performance to slow as each piece of code runs.

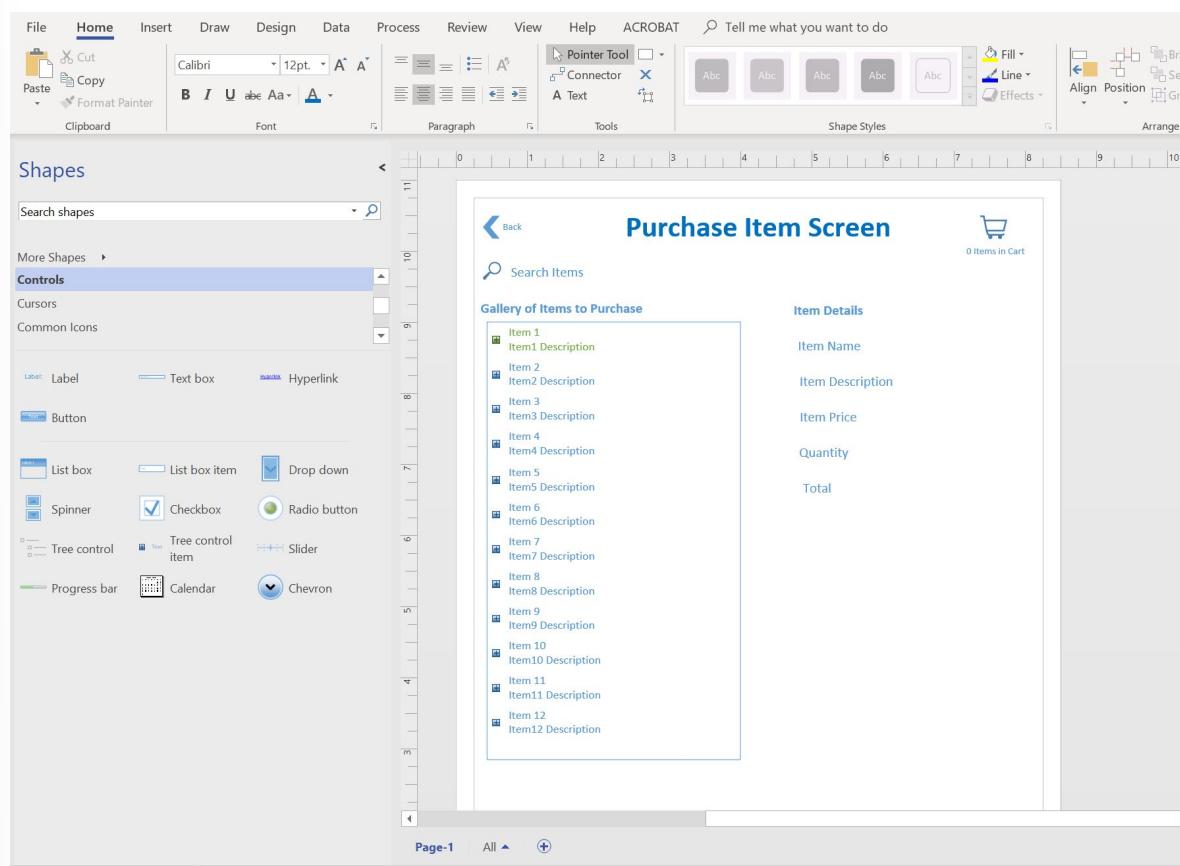
Finally, challenge yourself to do better with your user experience. Maybe today, the user records room temperatures by clicking in a box, changing the device keyboard from letters to numbers, and then typing in "70". A better option may be to replace with a slider control that defaults to 70 and ranges from 65 to 75. Then, with a swipe of their finger, they can record the temperature. Small changes like this make for happier, more productive app users.

## User Interface (UI)

To fully visualize the User Interface or UI, you may want to consider creating a mockup of your application. Two common ways to create a mockup of your application are below:

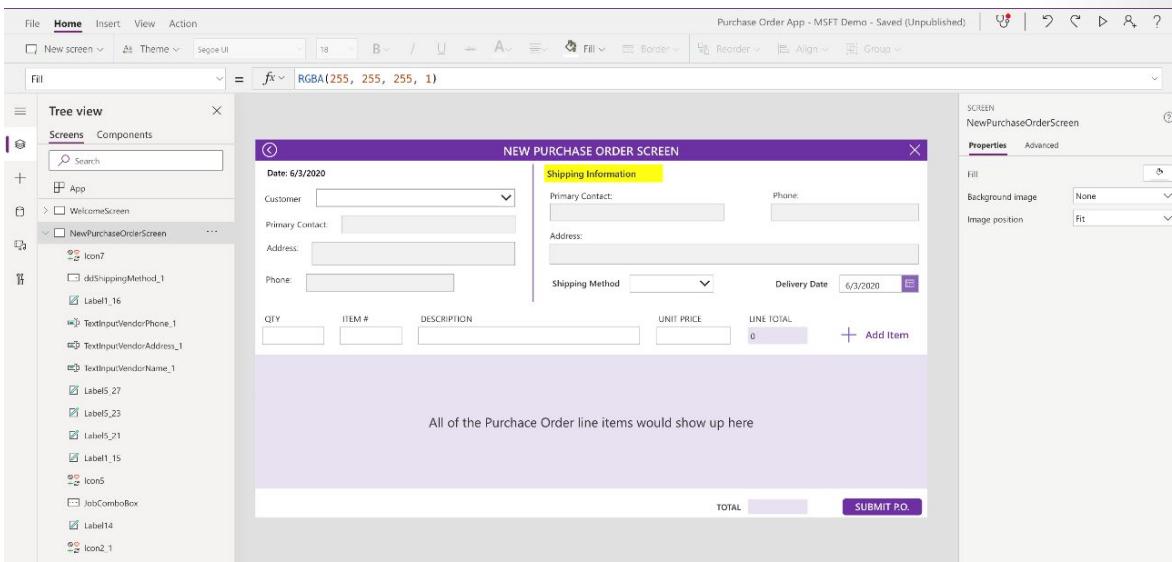
- Use Visio to create a wireframe diagram. A wireframe is a visual representation of an application's user interface. To begin, there are various website and mobile wireframe templates available, or you could start from blank template. The diagrams are a quick way to show app functionality and gain team consensus on the design.

The example below shows a simple Visio wireframe of a Purchase Items screen in an inventory app.



- Use Power Apps to create a mockup of your application. You can add most of the controls, graphics, forms, and other items to your app screens and play with the layout and size for each element as if you were building the app for real. When designing the UI you don't need to add the logic behind the various elements you placed on the screen. The goal here is to focus on what the app could look like and how it could function. This similar to what you can do with a Visio wireframe but one of the biggest pros of going this route is that you will gain more experience working with Power Apps and also learn more about the various UI elements available in the process. All of the experience and knowledge you will gain by creating your app mockup in Power Apps will only payoff later when it's time to start on the production app. Another big upside to using Power Apps for your mockup is that if you show this to your team and they like what you did, you can continue building off this app or create a new app and copy the elements you would like to keep to your other application. By not having to redo the UI or only having to redo parts of it, you could potentially save yourself hours of work.

The example below shows a simple mockup of a New Purchase Order Screen.



It really comes down to your preference and comfort with the software you are using to create the mockup. You should also consider licensing and costs when making this decision. Visio requires additional licensing to get the full functionality required for creating a wireframe diagram. Whereas with Power Apps, it doesn't matter which license you have. As long as you have Power Apps (and sufficient permissions in your environment), you can create apps and mockup apps.

As you design the User Interface, a few additional things to think about are Accessibility and Localization. It's important to ensure the app interface follows accessibility guidelines so all your users can interact with your application without any issues. To review these guidelines and additional accessibility properties, see [Create accessible canvas apps in Power Apps<sup>11</sup>](#).

Localization can be something you must consider when developing your application as well. Depending on where your app will be used, you may need to use different punctuation. For example, some regions of the world use a . (dot or period) as the decimal separator while others use a , (comma). For more information on building a globally supported application, see [Build global support into canvas apps<sup>12</sup>](#).

## Business Logic

When using the common data service, you can create business rules and recommendations to apply logic and validations without writing code or creating plug-ins. The great thing about the common data service and business rules is that they are applied at the data level. This means that you can apply rules that are enforced regardless of how the data is accessed.

Often when building apps all of the business logic is built into the app. This works great if the data is only accessed via the app. The challenge is often business data is used in many ways and from different tools. This is where Business Rules shine. You can apply logic on the data in the Common Data Service, allowing your rules to be enforced no matter which tool interacts with the data.

For example, you have built a capital project expense tracking application using Common Data Service as the data source. In your business process, the duration field is an optional field if your request is less than 10,000 but the duration field is required if the request is more than 10,000. After you set up your entity in Common Data Service, you would then apply a business rule that says if Project Amount is greater than

<sup>11</sup> <https://docs.microsoft.com/powerapps/maker/canvas-apps/accessible-apps?azureportal=true>

<sup>12</sup> <https://docs.microsoft.com/powerapps/maker/canvas-apps/global-apps?azure-portal=true>

10,000 then make Project Duration a required field. Now, regardless of how the user interacts with the data, the Business Rule will be enforced, keeping your data integrity.

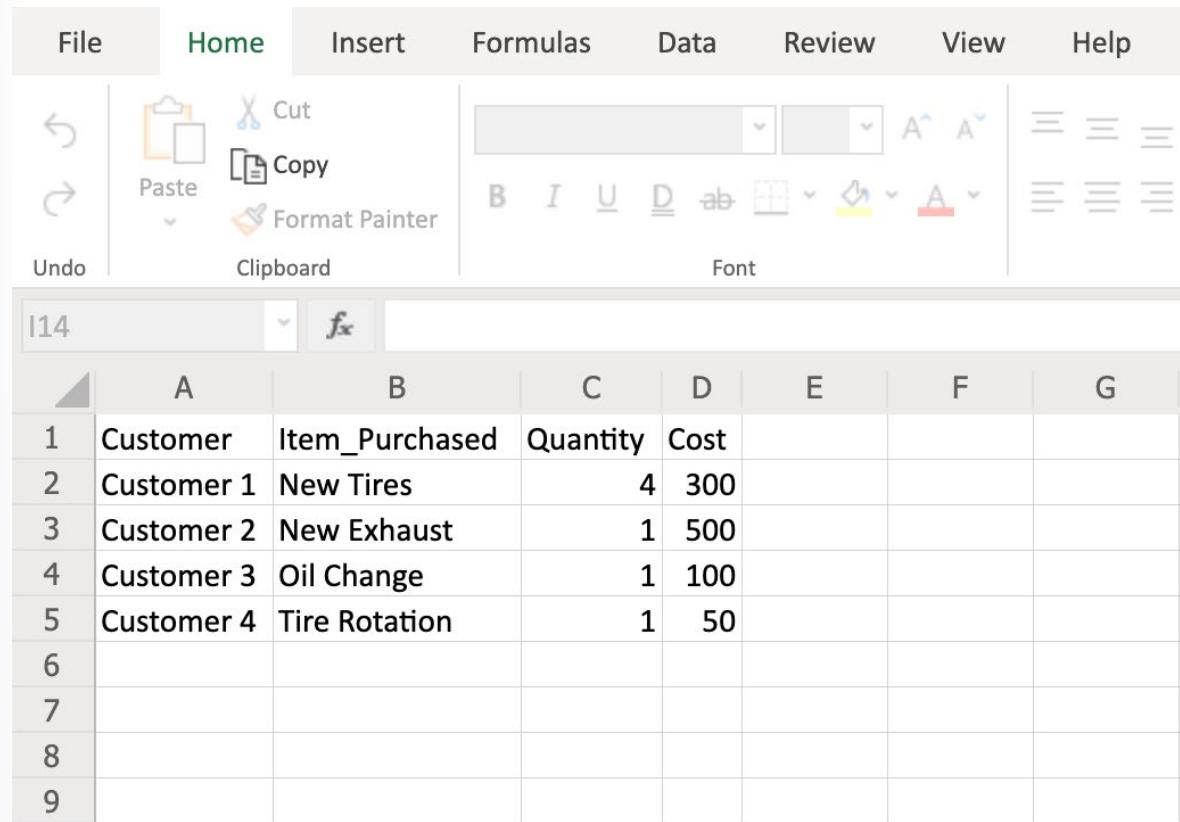
## Output

Finally, you will want to discuss your app's data output. This simply means what type of data will your app generate, and once the data is generated what will be done with it? A few questions to ask your app stakeholders:

- How does the data need to be visualized?
- What actions will be taken on the data once it is collected?
- Are there specific format or file types the data are needed?

The answers to these questions will help determine if additional functionality needs to be added to the app such as a Power BI report, email output, PDF, or CSV.

Let's look at an example. Perhaps your organization has a legacy ERP solution and the orders submitted in your Power App need to be reflected in the ERP application. While one option might be to build a custom connector to that solution, another option may be to export the data to a CSV file using Power Automate and Power Apps together, see screenshot below:



A screenshot of a Microsoft Excel spreadsheet titled 'I14'. The spreadsheet contains a table of purchase data with columns: Customer, Item\_Purchased, Quantity, and Cost. The data is as follows:

	A	B	C	D	E	F	G
1	Customer	Item_Purchased	Quantity	Cost			
2	Customer 1	New Tires	4	300			
3	Customer 2	New Exhaust	1	500			
4	Customer 3	Oil Change	1	100			
5	Customer 4	Tire Rotation	1	50			
6							
7							
8							
9							

The great thing about generating this CSV file export is that it's not linked to your data, so the changes you make to the file will not alter the app data.

For more information, see [Planning a Power Apps project<sup>13</sup>](https://docs.microsoft.com/powerapps/guidance/planning/introduction).

---

<sup>13</sup> <https://docs.microsoft.com/powerapps/guidance/planning/introduction>

## Exercise - Create your first app in Power Apps

In this unit, you'll generate a mobile app where the data source is a Microsoft Excel workbook that's stored in Microsoft OneDrive for Business. This Excel workbook lists a company's inventory of flooring samples with pictures and prices.

Keep in mind that you can use data from many other sources, including Microsoft SharePoint, cloud services like Salesforce, and on-premises sources like Microsoft SQL Server.

*Note:* Power Apps requires either an Office 365 license or a free trial. Learn more about your licensing options. **Microsoft products include Microsoft Power Apps and Power Automate.**<sup>14</sup>

Before you begin, watch this video for a brief overview of what to expect when creating your first Power App.



<https://www.microsoft.com/videoplayer/embed/RE4vls4>

### Connect to a data source

To connect to a data source, use the following procedure:

1. Download the **Flooring Estimates workbook**<sup>15</sup> and save it to OneDrive for Business.
2. Go to <https://make.powerapps.com> and sign in with your organizational account.
3. In the left pane, select **Apps**.
4. Select **+ New app** and then **Canvas** from the drop-down menu.
5. For the **OneDrive for Business** data source, select **Phone layout**.

Generated apps are always based on a single list or table, but you can add more data to the app later. The next three steps explain how to connect to the Excel workbook.

6. Under **OneDrive for Business**, select **Create**.

<sup>14</sup> <https://docs.microsoft.com/powerapps/administrator/pricing-billing-skus>

<sup>15</sup> <https://az787822.vo.msecnd.net/documentation/get-started-from-data/FlooringEstimates.xlsx>

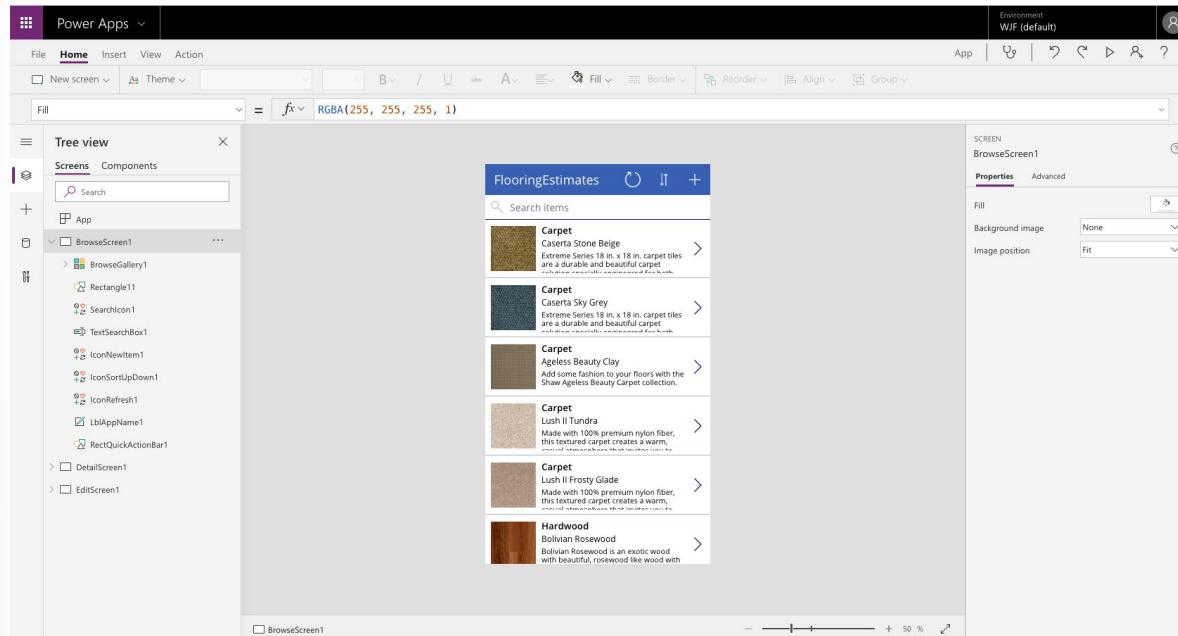
7. Under **Connections**, select **OneDrive for Business** and browse to the file location. You might need to select **New Connection** to see the **OneDrive for Business** connection.
8. Under **Choose an Excel file**, select the **FlooringEstimates.xlsx** file.
9. Under **Choose a table**, select the **FlooringEstimates** table.
10. Select **Connect** on the bottom right.

Power Apps generates the app by inspecting your data and matching it with Power Apps capabilities so that you get a working app as a starting point.

## Explore the generated app

Your new three-screen app now opens in Power Apps Studio.

The following figure shows the main development window for Power Apps Studio, which you'll learn more about in later units.



Select **Play** in the upper-right corner to practice using the app. Notice that it includes all the data from the table and provides a good default experience.

All apps that are generated from data have the same set of screens that you can view from the Screens pane:

- **Browse screen** - This screen appears by default. In it, you can browse, sort, filter, and refresh the data from the data source. In the browse screen, you can add items to the data source by selecting the plus sign (+).
- **Details screen** - The details screen shows all information about a single item. In this screen, you can open an item to edit or delete it.
- **Edit/create screen** - In this screen, you can edit an existing item or create a new one.

To make your app visible on the phone, it needs to be saved. Select **File, Save as**. Replace the current title "App" with **flooring-estimates app**, and then select **Save**. You will see a green check mark when all changes are successfully saved. You can now open the app on your phone.

## Install the app on your device

To see how the app runs on mobile, install the Power Apps Mobile app on your phone. When building an app, you should test it in the same form factor as your users.

1. Download Power Apps Mobile from the app store for the platform that you want to use.
2. Sign in by using your username and password.
3. On your phone or tablet, run the flooring-estimates app in Power Apps Mobile. If you do not want to install the app, you can run it in a browser.

## Summary

Congratulations on creating your first app with Power Apps!

In this module, you discovered what Power Apps can do for your business and the building blocks of creating your first app. You then created an app from data in a Microsoft Excel workbook.

Additionally, you learned that:

- To create, share, and administer your apps, you will use make.powerapps.com, the Power Apps Studio, and the Power Apps Admin Center.
- The power of Power Apps comes from the ability to connect to related technologies in your business. Examples of these are Common Data Service, Power Automate, Microsoft SharePoint, and other data sources.
- You can create an app by using several different methods. Some of these methods include from a template, a data source (like Microsoft SharePoint), or a blank canvas.

See **Create your first app<sup>16</sup>** if you're interested in more explanation.

---

<sup>16</sup> <https://www.youtube.com/watch?v=88F1PT7XbP0>

# Customize a canvas app in Power Apps

## Improve apps by making basic customizations in Power Apps

In the previous module, you generated the Flooring Estimates app and started to explore its default design. While the default screens make a useful app out of the box, you'll often want to customize a generated app to suit your needs.

This unit explains basic changes for each screen in the app. You can do a lot more to customize an app, but the best way to start learning is to take a generated app and make common customizations. This will allow you to become familiar with the controls, layouts, and functions.

Before you begin, watch the following video for a brief overview of what to expect when customizing your app.



<https://www.microsoft.com/videoplayer/embed/RE4vtmh>

### Browse screen

The Flooring Estimates app already shows an image and some text for each product, but the layout could be better.

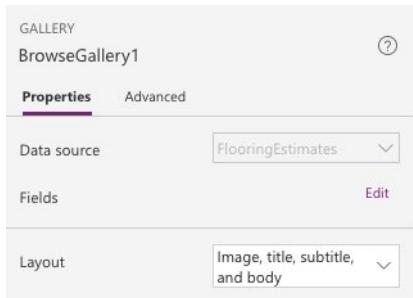
To improve the layout, use the following procedure:

1. On the Screens pane on the left, select **BrowseGallery1**.

The selection box around the gallery confirms your choice.

A screenshot of the Microsoft Power Apps studio interface. The top navigation bar shows 'FlooringEstimates' with icons for refresh, search, and add. Below is a search bar with placeholder 'Search items...'. The main area displays a 'BrowseGallery1' screen. It features a vertical list of products. Each item has a small image, a category name ('Carpet' or 'Hardwood'), a product name, and a short description. A red selection box highlights the entire list of items on the screen.

2. On the right pane, open the Data pane by selecting the drop-down menu next to **Layout**.



3. Select a different layout, such as the one that shows the image, the title, and the subtitle but not the body.

GALLERY  
BrowseGallery1 (?)

**Properties** Advanced

Data source FlooringEstimates ▼

Fields Edit

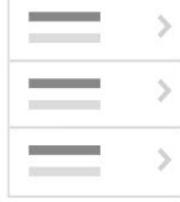
Layout Image, title, and ▼

Search All ▼

List

Blank 

Title 

Title and subtitle 

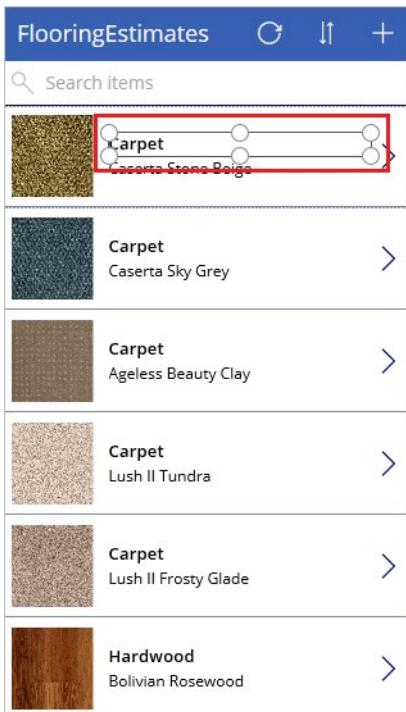
Title, subtitle, and body 

Image and title 

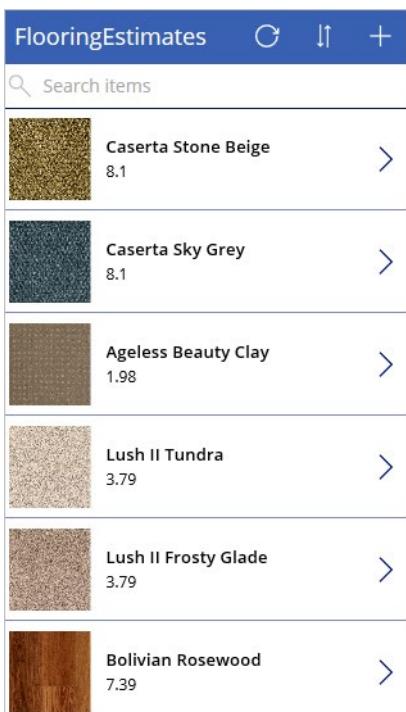
Image, title, and subtitle 

A red box highlights the "Image, title, and subtitle" item template.

4. Select the category of the item at the top of the gallery.



5. Change **ThisItem.Category** to **ThisItem.Name** in the formula bar.
6. Repeat the previous two steps but change the other **Label** control to show the price of each item.

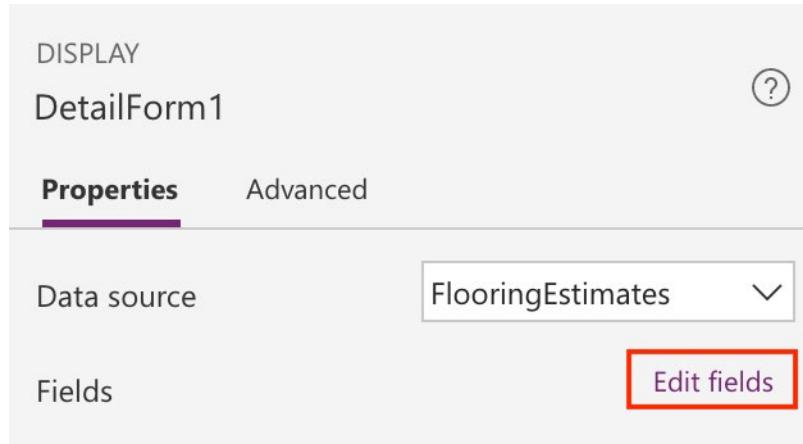


Changing the layout of a gallery and the types of data that it shows is that simple, and you might find that it's fun, too.

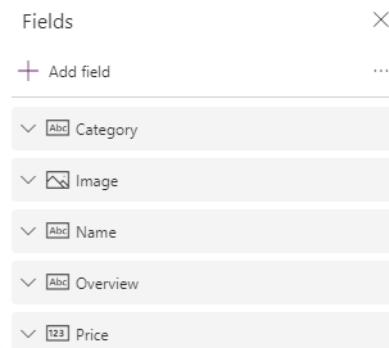
## Details screen

On the details screen, you want to change the order of the fields. The controls on this screen differ from the controls on the browse screen, so the process for changing them is also slightly different.

1. On the Screens pane on the left, select **DetailScreen1 DetailForm1**.
2. On the right pane, select **Edit fields**.



3. Drag the **Name** field to the top of the list of fields and then drag the **Image** field to the bottom.



## Edit/create screen

On the screen where your users edit and create entries, you want to make it easier for them to enter information in a text box.

1. On the Screens pane on the left, select **EditScreen1 EditForm1**.
2. On the right pane, select **Edit fields**.
3. Expand **Overview**. Select the drop-down arrow for the **Control type** and then select **Edit multi-line text**.

A multi-line edit control will simplify your user's ability to add more than a few words in this field.

A few basic steps can greatly improve the appearance and experience of using an app, and Power Apps Studio provides many options for customizing those apps.

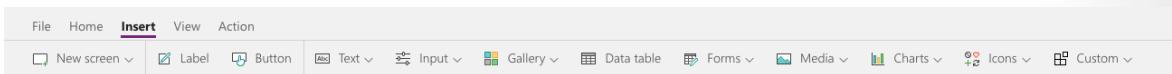
# Explore screens and controls in Power Apps

This unit examines the screens and other controls that define the behavior of apps that Microsoft Power Apps generates. All the details won't be covered; however, knowing more about how these apps work will help you build your own apps.

## Controls in Power Apps

A control is a UI element that produces an action or shows information. Many controls in Power Apps are similar to controls that you've used in other apps: labels, text-input boxes, drop-down lists, navigation elements, and so on.

In addition to these typical controls, Power Apps has more specialized controls, which you can find on the **Insert** tab.



A few controls that can add interest and impact to your apps include:

- **Galleries** - These controls are layout containers that hold a set of controls that show records from a data source.
- **Forms** - These controls show details about your data and let you create and edit records.
- **Media** - These controls let you add background images, include a camera button (so that users can take pictures from the app), a barcode reader for quickly capturing identification information, and more.
- **Charts** - These controls let you add charts so that users can perform instant analysis while they're on the road.

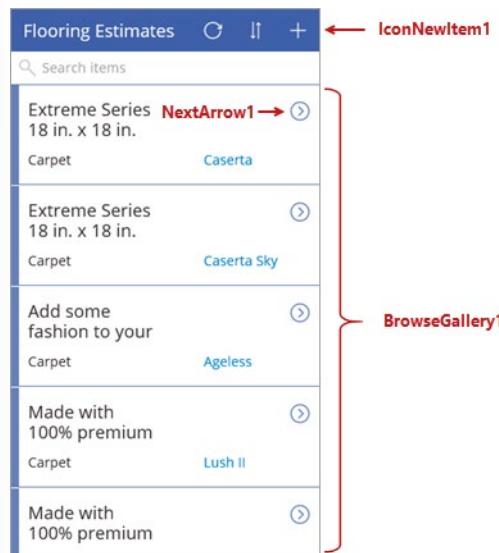
To see what controls are available, select the **Insert** tab, and then select each option in turn.

## Explore the browse screen

Each screen in the app has multiple controls, but one control takes up most of the screen space. The first screen in the app is the browse screen, which is named **BrowseScreen1** by default.

Controls in the browse screen that you'll want to become familiar with include:

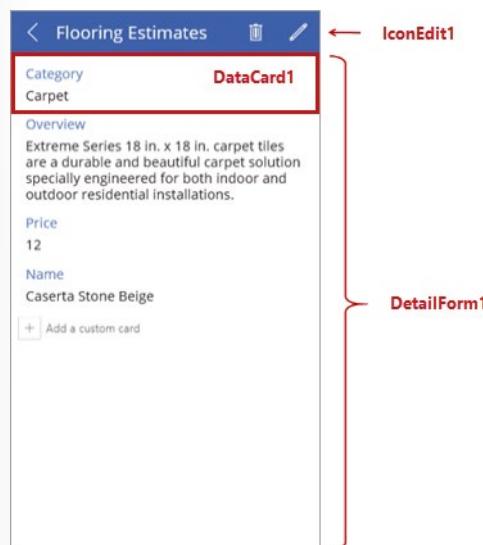
- **BrowseGallery1** - This control takes up most of the screen and shows data from your data source.
- **NextArrow1** - When this control is selected, it opens the details screen.
- **IconNewItem1** - When this control is selected, it opens the edit/create screen.



## Explore the details screen

The details screen is named **DetailScreen1** by default. Some of its controls are as follows:

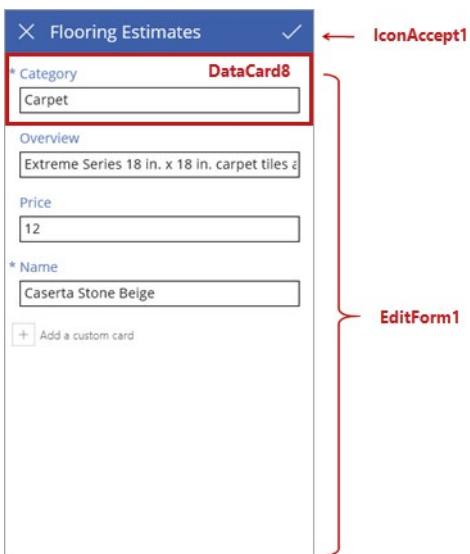
- **DetailForm1** - This control contains other controls and contains a data card for each field of the record that is being displayed.
- **DataCard1** - This is a card control. Each card represents a single field of the record. In this case, it shows a flooring category from the Flooring Estimates table, as shown in the previous unit.
- **IconEdit1** - When this control is selected, it opens the edit/create screen so that the user can edit the current item.



## Explore the edit/create screen

The third screen in the app is **EditScreen1**. Some of its controls include:

- **EditForm1** - This control contains other controls and contains a data card for each field of the record that is being edited.
- **DataCard8** - This is another card control that shows a flooring category from the Flooring Estimates table, as shown in the previous unit.
- **IconAccept1** - When this control is selected, it saves the user's changes.



## Excercise-Get started with functions in Power Apps

When using Microsoft Power Apps, you don't have to write complicated application code the way that a traditional developer does. However, you must express logic in an app and control its navigation, filtering, sorting, and other functionalities. This is where formulas come in.

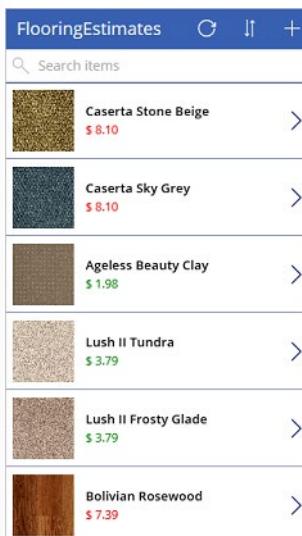
If you've used Microsoft Excel functions, you should recognize the approach that Power Apps takes. This unit shows a couple of basic formulas for text formatting and then describes three of the formulas that Power Apps includes when it generates an app. With this information, you'll have a better idea of what formulas can do, and then you can also start to write your own.

## Get started with formulas and properties

Properties determine the appearance and behavior of controls. Each type of control has a different set of properties.

The previous unit explored controls in all three screens of an app that Power Apps generated. In this section, you'll use the properties of the label control to format the price in the gallery.

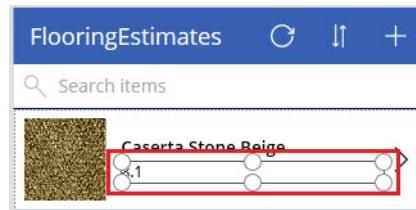
By default, the price appears as a plain number without a currency symbol. Suppose that you want to add a dollar sign and change the text color based on the item's cost (for example, red if it's more than \$5 but green otherwise). The following graphic shows the expected result.



By default, Power Apps pulls in a price value for each item. This value is set as the **Text** property of the label that shows the price.

*Note:* Be sure to complete the steps in unit 1 of this module as the description field is changed to price as reflected in the next steps.

1. In **BrowseScreen1**, select the price of the first item.

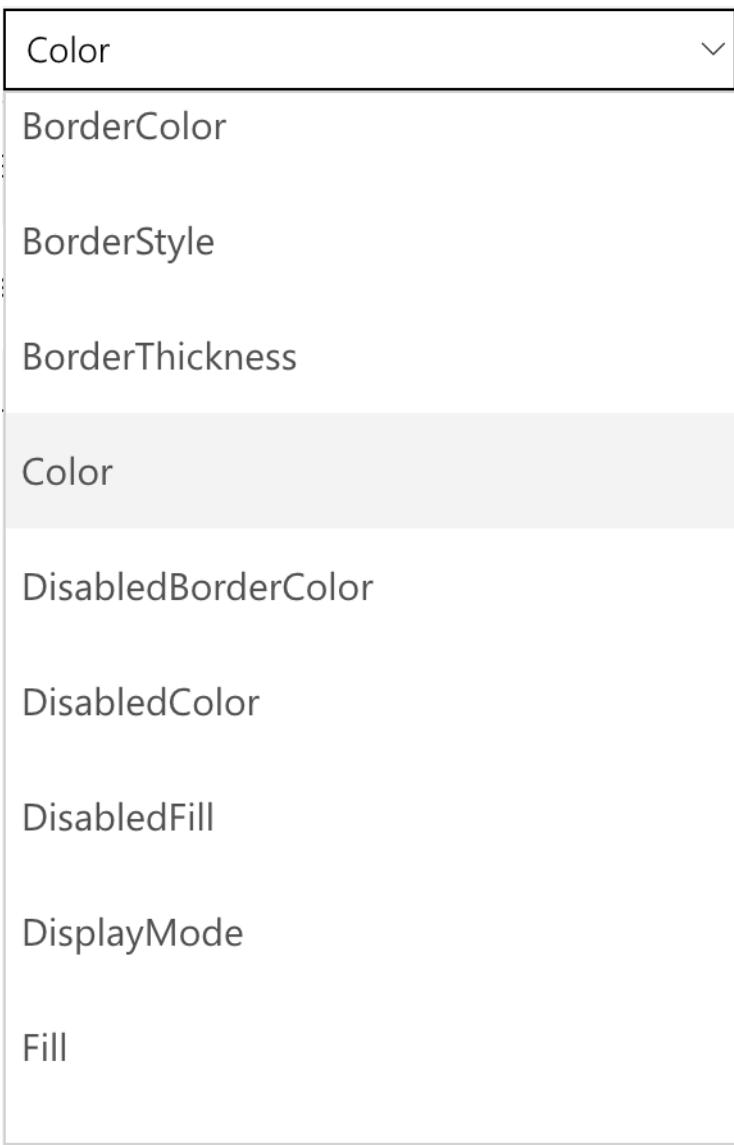


2. In the drop-down list of properties, select **Text**.
3. To add the currency symbol for US dollars, set the **Text** property to this formula:

`Text(ThisItem.Price, "$ ##.00")`

The **Text** function specifies how to format the number. The formula is like an Excel function, but Power Apps formulas refer to controls and other app elements instead of cells in a workbook.

If you select a control and then open the property drop-down list, a list of properties that are relevant to the control appears. For example, the following is a partial list of the properties for a **Label** control. Some properties are relevant across a wide range of controls, but others are relevant only for a specific control.



To conditionally format the price's color, set the **Color** property of the price's **Label** control to this formula:

```
If(ThisItem.Price < 5, Color.Red, Color.Green)
```

## Formulas included in the generated app

Power Apps uses a couple of formulas in every app that it generates. Both examples are from the browse screen and work with the **OnSelect** property. This property defines what happens when a user selects a control.

- The first formula is associated with the **IconNewItem1** control . Select this control to open the edit/  
create screen where you can create an item. To view the formula, select the **OnSelect** property and then select it in the formula bar. The formula is as follows:

```
NewForm(EditForm1);Navigate(EditScreen1, ScreenTransition.None)
```

The formula instantiates an edit page on the edit/create screen so that users can create an item. A value of `ScreenTransition.None` means that there's no transition, such as a fade, between screens.

-  The second formula is associated with the **IconSortUpDown1** control . Select this control to sort the items in the gallery. The formula is as follows:

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

The formula uses `UpdateContext` to update a variable called `SortDescending1`. The exclamation "`!`" symbol in the formula is a shortcut for the `Not` function. The value of the variable switches back and forth as you select the control. This variable tells the gallery on this screen how to sort the items.

The app contains many other formulas, so take some time to select controls and determine what formulas are set for various properties.

For more information about these and other functions, refer to [formula reference for Power Apps<sup>17</sup>](#) page.

For additional information on customizing a canvas app, refer to the Use the UI and controls in a canvas app in Power Apps learning path and the Use basic formulas to make a better canvas app in Power Apps learning path.

## Summary

This module showed you how to customize an app by adding controls, formatting, and logic in Power Apps.

Additionally, you were able to:

- Change the layout of a gallery.
- Change the data that a control shows.
- Change the order in which fields appear.
- Change the control with which a user provides information.
- Format a number as a price and coloring prices based on their values.

---

<sup>17</sup> <https://docs.microsoft.com/powerapps/maker/canvas-apps/formula-reference>

# Manage apps in Power Apps

## Exercise-Manage app versions in Power Apps

Microsoft Power Apps can help if you saved changes to an app that you shouldn't have or if something else goes wrong. For apps that you save in the cloud, Power Apps keeps a history of the changes that you make. You can view each version that you've saved and restore your app to a previous version if necessary. If you shared the app, the people whom you shared it with will also receive the restored version if you republish the app.

### View versions of your app

1. On [make.powerapps.com](https://make.powerapps.com)<sup>18</sup>, select **Apps** on the left pane.
2. In the list of apps, select the ellipsis (...) next to the app name and then select **Details**.

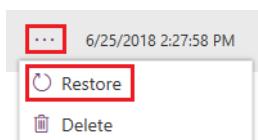
The screenshot shows the 'Your apps' section of the Power Apps portal. It lists the 'flooring-estimates app' with a 'Beginner' level and a duration of '2 mins'. To the right of the app card is a context menu with several options: Edit, Play, Share, Export package (preview), Add to Teams, Analytics (preview), Settings, Delete, and Details. The 'Details' option is highlighted with a red box.

3. Select the **Versions** tab.

The tab shows all versions of the app that you saved as you developed it.

### Restore a previous version

1. Select the ellipsis (...) next to the desired version and then select **Restore**.



2. Select **Restore** again to confirm the action.

<sup>18</sup> <https://make.powerapps.com>

A new version is added to your list.

When you restore a version of an app, the newly restored version gets a new, incremented version number and appears at the top of the list. A new version never overwrites a previous version.

*Note:* After restoring a previous version, the restored version needs to be published before users will see the new version.

If for some reason you are not able to restore a previous version, you can try the following:

- Make sure the App is not open in Power Apps Studio. If the app is open, you will not be able to restore the version.
- Verify the version you would like to restore is not older than six months. At the current time, only app versions less than six months old can be restored.

## Exercise-Share apps in Power Apps

You can share an app with specific users, groups, or your whole organization. When you share an app with other people, they can run it in a browser, from the Microsoft Dynamics 365 home page, or in Microsoft Power Apps Mobile for Microsoft Windows, Apple iOS, or Google Android.

Even better, you can give someone permission to update the app.

### Prepare to share an app

To complete the following steps, open the app that you want to share in **Edit** mode.

1. In Power Apps Studio, select the **File** menu and then select **Settings**. Give the app a meaningful name and a description so that your team knows what your app does and can easily find it in the apps list.
2. On the **File** menu, select **Save as** and then select **The cloud**.  
You must save an app to the cloud before you can share it.
3. Select **Save** and then select **Share**.
4. On the **Share** tab, specify the users or groups with whom you want to share the app. To add everyone in your organization, type **Everyone** and select **Everyone in Company Name**. If you need to share with a large group of users, a best practice is to share through an Azure Active Directory Security Group.

By default, the user receives the User permission. If you want the user to also be able to edit the app, then

select the co-owner check box. The following is a description of both permissions:

- **Co-owner** - Users can use, edit, and share the app, but can't delete or change the owner.
- **User** - Users can view and use the app, but they can't change it.

Security-group considerations

- If you share an app with a security group, existing members of that group, and anyone who joins it, will have the permission that you specify for that group. Anyone who leaves the group loses that permission unless they belong to a different group that has access or if you give them permission as an individual.
- Every member of a security group has the same permission for an app as the overall group does. However, you can specify greater permissions for one or more members of that group to allow them greater access. For example, you can give Security Group A permission to run an app, but

you can also give User B, who belongs to that group, Co-owner permission. Every member of the security group can run the app, but only User B can edit it. If you give Security Group A Co-owner permission and User B permission to run the app, that user can still edit the app.

- To notify users by email, leave the **Send an email invitation** check box selected.

If you elect to notify the users by email, everyone you shared the app with will receive an email message that has a link to the app. People whom you granted **Co-owner** permission for the app will also receive a link to Edit App in Power Apps Studio.

- Select **Share**.

If you make and save changes to a shared app, the people whom you shared it with will see your changes as soon as you publish them. This can be useful if you improve the app, but it can also negatively affect users if you remove or significantly change features. Remember to create a notification plan for alerting your users of major updates.

## Permissions and licensing

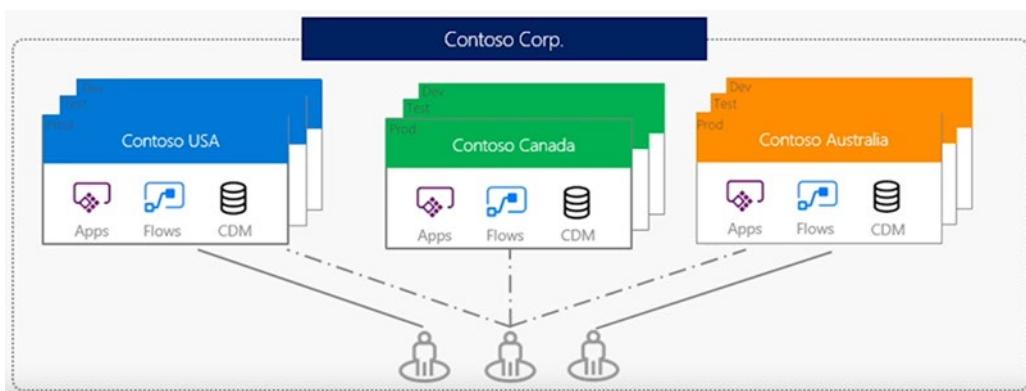
Basic information about permissions and licensing that you should be aware of are:

- Users and contributors need permissions to any data connections and gateways that a shared app uses. Some permissions come implicitly with the app, but you must explicitly grant others. If you create an app based on Common Data Service, you must also ensure that the users with whom you share the app have the appropriate permissions for the entity or entities on which the app relies. Specifically, those users must belong to a security role that can perform tasks such as creating, reading, writing, and deleting relevant records. In many cases, you'll want to create one or more custom security roles with the exact permissions that users need to run the app. You can then assign a role to each user as appropriate.
- People who have **Co-owner** permission also need a Power Apps Per app plan or Power Apps Per user plan to work directly with entities in Common Data Service.

Sharing an app is simple, and it's a great way to make an app that you find useful available to people across your organization.

## Exercise-Understand environments in Power Apps

An environment is a container for apps and other resources, such as data connections and flows from Power Automate. It's a way to group items based on business requirements.



If you've followed along with this module, you've already been working in [make.powerapps.com<sup>19</sup>](https://make.powerapps.com); therefore, you've been working in a specific environment the whole time. In the upper-right corner of the home page, you can view your current environment.



If you're new to Microsoft Power Apps, you might have only the default environment at this point. If a drop-down menu is visible next to the environment name, this indicates that other environments are available.

Note: If you want to work with Power Apps environments, you need a Power Apps Per app plan or Power Apps Per user plan. Additionally, if you want to work with Dynamics 365 restricted entities, you must have a Power Apps for Dynamics 365 license. Learn more about [licenses for Dynamics 365<sup>20</sup>](#).

## Reasons to use environments

Reasons to create environments beyond the default one include:

- **Separate app development by department** - In a large organization, each department can work in a different environment. That way, department employees see only apps and company data that are appropriate to their needs.
- **Support application lifecycle management (ALM)** - Separate environments let you separate apps that are in development stages from those that have already been shared. Alternatively, you might want to use a trial environment so that you can receive feedback from employees before publishing the final app. For some organizations, showing apps before they're completely developed and published can present security concerns.
- **Manage data access** - Each environment can have its own source of business data, called a database for Common Data Service.  
Other data connections are specific to an environment and can't be shared across environments.

*Important:* Keep in mind that environments are relevant only to app creators and Power Apps admins. When you share an app with users, those users simply run the app, providing they have the correct permissions. In other words, they don't have to worry about what environment the app came from.

## Create an environment

Only an admin can create environments. If you aren't an admin, this information can still be helpful when you talk to your admin about setting up environments.

1. On the [make.powerapps.com<sup>21</sup>](https://make.powerapps.com) home page, select the gear icon near the upper-right corner and then select **Admin center**.

You can also go directly to <https://admin.powerplatform.microsoft.com/>

2. In the Power Apps admin center, select **+ New**.

<sup>19</sup> <https://make.powerapps.com>

<sup>20</sup> <https://na01.safelinks.protection.outlook.com/?url=https%3A%2F%2Fdocs.microsoft.com%2Fen-us%2Fpowerapps%2Fadministrator%2Fpricing-billing-skus%23licenses&data=02%7C01%7Cv-tosis%40microsoft.com%7C3bb58d639d8745c27ff908d62d4c1062%7C72f988bf86f141af91ab2d7cd011db47%7C1%7C0%7C636746202048937165&sdata=8rDKLL4XBkwCLOrpZe0F2MITmkfhAmukrV3bn4a34NU%3D&reserved=0>

<sup>21</sup> <https://make.powerapps.com>

3. In the **New environment** dialog box, enter a name for the environment and then select a region and an environment type.
4. To the left of **Create a database for this environment**, select the toggle to Yes.
5. Select **Next**.
6. Select the currency and language for the data that is stored in the database. You cannot change the currency or language after the database is created.
7. Select **Save**.

It might take several minutes to create the database on Common Data Service. After the database is created, the new environment appears in the list of environments on the **Environments** page.

You now have a new environment to work in. If you go back to [make.powerapps.com](https://make.powerapps.com)<sup>22</sup>, you will see it in the environments list.

## Manage access to an environment

By default, you can access an environment in one of two ways:

- **System admin** - A system admin has full permissions to create and manage environments.
- **Environment maker** - An environment maker can view all apps in that environment, create apps, and work with Common Data Service (other permissions apply).

Environment admins can create other security roles as needed. They can also add and assign users to these roles.

1. Start by going to [make.powerapps.com](https://make.powerapps.com)<sup>23</sup>
2. On the left pane, **Environments** should be selected by default, if it is not, select **Environments**.
3. Select the test environment that you just created, and then select the **Security** tab.
4. Add the user by entering the email address of the user in your organization and then selecting **Add user**.  
Wait a few minutes for the user to be added.
5. To verify if the user is now available, select **list of users**.
6. Hover over the result that you want, select its check box, and then select **MANAGE ROLES** on the top bar.
7. In the **Manage User Roles** box, select the role(s) for the user. In this example, assign the user to the Environment Maker role.
8. Select **OK**.

## Power Apps review

Congratulations on building your first app!

To review, so far you've learned how to:

- Build an app based on data in a Microsoft Excel workbook that's stored in Microsoft OneDrive for Business. You learned that Microsoft Power Apps can connect almost as easily to data sources that

<sup>22</sup> <https://make.powerapps.com>

<sup>23</sup> <https://make.powerapps.com>

you're already using in the cloud (such as Microsoft SharePoint, Microsoft Azure, Google Drive, and Salesforce) or on-premises.

- Customize an app to make it your own by modifying the appearance and behavior and adding Excel-like functions.
- Share apps instantly with your co-workers across the web, tablets, and mobile devices.
- Set up environments so that you can separate a working environment from the one that you want to share with your team.

The best way to advance your skills is to run the samples, practice using the templates, and generate more apps from your own data.

## Next steps

One goal of this module is to provide a clearer picture of what Power Apps is and how you can start creating apps, regardless of your experience level. The following are useful resources and downloads to help enhance your further learning.

## Power Apps resources

- Explore further with the **Power Apps documentation**<sup>24</sup>.
- Stay current with the **Power Apps blog**<sup>25</sup>.
- Join the **Power Apps community**<sup>26</sup>.
- Expand your expertise with additional Power Apps learning paths:
  - **Use basic formulas to make better Power Apps canvas apps**<sup>27</sup>
  - **Work with data in a Power Apps canvas app**<sup>28</sup>
  - **Use the UI and controls in a canvas app in Power Apps**<sup>29</sup>
  - **Use advanced data options and connectors in Power Apps**<sup>30</sup>
  - **Master advanced techniques for Power Apps canvas apps**<sup>31</sup>
- Improve Power Apps by submitting an **idea**<sup>32</sup>.

## Power Apps downloads

- **Power Apps Mobile for Windows**<sup>33</sup>
- **Power Apps Mobile for iOS**<sup>34</sup>

---

<sup>24</sup> <https://docs.microsoft.com/powerapps/>

<sup>25</sup> <https://powerapps.microsoft.com/blog/>

<sup>26</sup> <https://powerusers.microsoft.com/t5/PowerApps-Community/ct-p/PowerApps1>

<sup>27</sup> <https://docs.microsoft.com/learn/patterns/use-basic-formulas-powerapps-canvas-app/>

<sup>28</sup> <https://docs.microsoft.com/learn/patterns/work-with-data-in-a-canvas-app/>

<sup>29</sup> <https://docs.microsoft.com/learn/patterns/ui-controls-canvas-app-powerapps/>

<sup>30</sup> <https://docs.microsoft.com/learn/patterns/advanced-data-options-and-connectors/>

<sup>31</sup> <https://docs.microsoft.com/learn/patterns/understand-advanced-topics/>

<sup>32</sup> <https://powerusers.microsoft.com/t5/PowerApps-Ideas/idb-p/PowerAppsIdeas>

<sup>33</sup> <https://aka.ms/powerappswin>

<sup>34</sup> <https://aka.ms/powerappsios>

- **Power Apps Mobile for Android<sup>35</sup>**

## Summary

In this module, you learned about app versions, sharing apps, and managing environments.

Additionally, you learned how to:

- View and restore app versions
- Share an app and manage its permissions
- Create environments and how to manage security access

---

<sup>35</sup> <https://aka.ms/powerappsandroid>

# Navigation in a canvas app in Power Apps

## Understand navigation

In Power Apps, because most apps have multiple screens, it's essential that you understand how to implement the **Navigate** function in your app. The **Navigate** function allows users to move or navigate from screen to screen in an app. For example, if you create an app with three screens and you want to give users a way to navigate to another screen, set the

**OnSelect** property of a **Button** control to:

**Navigate(Screen2,ScreenTransition.Cover)**

With this formula, when the button is selected, **Screen2** will automatically display. You could also use this formula on an icon, like an arrow, to provide navigation in the app.

The **Navigate** function also allows for a visual transition as the users move from screen to screen, which is set using **ScreenTransition**. In the example shown above, **ScreenTransition.Cover** was applied. There are a few different **ScreenTransitions** to choose from, and each one provides a slightly different user experience when navigating screens. **ScreenTransitions** will be covered in further detail later in this module. Similar to the **Navigate** function, you also have the option to use the **Back()** function. The **Back()** function is fairly straight-forward, it sends the user back to the previous screen.

You can use **Navigate** to set one or more context variables. You can use this approach to pass parameters to a screen. If you've used another programming tool, this is similar to passing parameters to procedures.

## Hidden screens

You can have "Hidden screens" in your app for various purposes, such as:

- Documentation screen
- Settings screen
- Special permissions screen

A "Hidden screen" allows the creator of the app to add screens but not give users to access those screens. This is accomplished by not creating any navigation for users to those screens. There's an example later in this module to demonstrate this functionality further.

## The navigation and back function

In the previous section, you learned how to implement the **Navigate** and **Back()** functions using a Control. When implementing these functions, like any other function in Power Apps, there are certain arguments that need to be defined.

## Navigate function

Here's a breakdown of the syntax for the navigation function:

**Navigate(Screen, Transition [, UpdateContextRecord ] )**

- **Screen** - Required. The name of the screen to display.
- **ScreenTransition** - Required. The visual transition to use between the current screen and the next screen.
- **UpdateContextRecord** - Optional. A record that contains the name of at least one column and a value for each column. This record updates the context variables of the new screen. For more information, see [UpdateContext function in Power Apps<sup>36</sup>](#).

In the first argument, you specify the name of the screen to display. In the second argument, you specify how the old screen changes to the new screen. In the third argument, you have the option to specify a Context variable. Taking a step back to the second argument, ScreenTransition, there are a number of different ScreenTransitions you could apply. Each ScreenTransition produces a slightly different visual experience for the user.

## Screen transitions

**ScreenTransition.Cover** - The new screen slides into view, covering the current screen.

**ScreenTransition.Fade** - The old screen fades away to reveal the new screen.

**ScreenTransition.None** - The old screen is quickly replaced with the new screen.

**ScreenTransition.UnCover** - The old screen slides out of view, uncovering the new screen.

Here are some examples using ScreenTransitions.

Formula	Description	Result
<b>Navigate(Details, ScreenTransition.None)</b>	Displays the <b>Details</b> screen with no transition or change in value for a context variable.	The <b>Details</b> screen appears quickly.
<b>Navigate(Details, ScreenTransition.Fade)</b>	Displays the <b>Details</b> screen with a <b>Fade</b> transition. No value of a context variable is changed.	The current screen fades away to show the <b>Details</b> screen.
<b>Navigate(Details, ScreenTransition.Fade, {ID: 12})</b>	Displays the <b>Details</b> screen with a <b>Fade</b> transition, and updates the value of the <b>ID</b> context variable to <b>12</b> .	The current screen fades away to show the <b>Details</b> screen, and the context variable <b>ID</b> on the screen is set to <b>12</b> .

The **Back ()** function has an optional argument for ScreenTransition.

<sup>36</sup> <https://docs.microsoft.com/powerapps/maker/canvas-apps/functions/function-updatecontext>

Here's a more detailed example using these functions with multiple controls in a common real-world scenario. In many of the apps that you develop, certain screens may have multiple controls allowing users to navigate to different screens depending on the control they select. In the following example, you will create a three screen app to demonstrate the **Navigate** and **Back()** functionality.

1. In Power Apps Studio, create 3 blank screens.
2. On Screen1, add a **Button** control and change the **Text** property to **Next**.
3. On Screen2, add two **Button** controls and change the **Text** of one button to **Next** and the other button control to **Back**.
4. On Screen3, add a **Button** control and change the **Text** property to **Back**.
5. On Screen1, set the Next button **OnSelect** property to **Navigate(Screen2,ScreenTransition.Fade)**.
6. On Screen2, set the **OnSelect** property for the Next button to **Navigate(Screen3,ScreenTransition.Cover)**.
7. Set the **OnSelect** property for the Back button to **Back()**.
8. On Screen3, set the **OnSelect** property for the Back button to **Back()**.
9. To test this, put the app in Preview or Play mode and navigate through the app as a user would.

As you select each button, notice the subtle visual transitions of each ScreenTransition. Remember, the Navigate function must include a ScreenTransition. If you tried to write your Navigate function like this, **Navigate(Screen2)**, Power Apps would display a red squiggly line in the formula box indicating that there is an issue with the function as written.

## Back () function

When using the **Back()** function it should be noted, this will take you back to the previous screen that you navigated from, the key word being "navigated". To elaborate, this does not mean the screen you or the user was just on, you need to use the **Navigate** function to navigate away from the screen to then use the **Back()** function to get back there. This can be a little confusing, so here's an example of how this works.

In the previous example there are 3 screens. Update the **OnSelect** property of the Next button on Screen1 from **Navigate(Screen2,ScreenTransition.Cover)** to **Navigate(Screen3,ScreenTransition.Cover)** and the **text** property to **Jump to Screen 3**. Now, when the user selects the button, the app will navigate to Screen3. Then when they select the Back button on Screen3, they will navigate back to Screen1 and skip Screen2 entirely. This is by design in Power Apps and is important to concept understand to ensure the navigation in your app is configured properly.

Before moving on to the next section, add one more screen to the example app and rename the screen **Documentation**. With no Navigation pointing to this screen, it is not accessi-

ble to your end users. The purpose of this screen is to give the App creator a location in the app to make notes or add documentation about certain aspects of how the app functions. The App creator can also provide instructions for other editors of the app so they can quickly and easily identify what the previous creator did.

In this simple four screen app example, you can see how easy it is to navigate screens by setting the **OnSelect** property of a control.

There are several additional ways to navigate through your app covered in the next section.

## More ways to use navigation function

As previously mentioned, one of the most common ways to move from screen to screen is by setting the OnSelect property of a control. Keep in mind that there are also several other ways to trigger the Navigate or Back() functions.

### Navigating screens by setting the **OnChange** property of a drop-down control

One option is to use a **DropDown** control and an If statement.

With this solution, depending on the choice selected in the drop-down menu the app navigates to a specific screen.

1. In Power Apps Studio, add 2 blank screens. There should be a total of 3 blank screens.
2. On Screen1, add a **dropdown** control.
3. Set the **Items** property for the **dropdown** control to

```
[" ", "Active", "Inactive"]
```

4. Set the **OnChange** property for the **dropdown** control to the following

```
If(Dropdown1.Selected.Value ="Active",Navigate(Screen2,ScreenTransition.Cover),  
If(Dropdown1.Selected.Value = "Inactive",Navigate(Screen3,ScreenTransition.Fade)) )
```

On your keyboard, hold the Alt key to test the new functionality.

When the user selects **Active** from the drop-down menu, the user will be sent to Screen2. When the user selects **Inactive** from the drop-down menu, the user will be sent to Screen3. Finally, if the blank option is selected from the drop-down menu, the app will not navigate to another screen.

### Using variables and the Timer control to navigate screens

You can use a variable and an If statement to set navigation. Depending on what the variable is set to this will send the user to a specific screen. For example, if you have the question "Do you have additional feedback to provide regarding this incident?" If the user selects "Yes", then you want to open the **Additional Information** screen. If the user selects, "No", then you want to load the **Form Completed** screen.

Another option is to use a timer control. When the time runs out, the app navigates to a different screen. For example, maybe you only want to allow a user 30 seconds to answer the questions on a screen before navigating to the next set of questions.

The following example builds off the previous example and incorporates variables and a Timer control.

1. On Screen1, add a **Timer** control.
2. Set the **Duration** property to 10000 (milliseconds).
3. Set the **Auto start** to **On**.
4. Set the **OnTimerEnd** property to

```
If(Dropdown1.Selected.Value = " ", Navigate(Screen2, ScreenTransition.None))
```

5. Select the drop-down control and change the **OnChange** property to

```
If(Dropdown1.Selected.Value =
"Active", Set(varStatus,1), If(Dropdown1.Selected.Value =
"Inactive", Set(varStatus,2), Set(varStatus,0)))
```

6. Add a **Button** control under the drop-down menu, and set the **Text** property to **Next**.

7. Set the **OnSelect** property for the button to

```
If(varStatus = 1, Navigate(Screen2, ScreenTransition.Cover),
If(varStatus = 2, Navigate(Screen3, ScreenTransition.Fade)))
```

Now, if the Timer Ends, the user is automatically sent to Screen two. If the user selects **Active** from the drop-down menu, a Variable named varStatus is set to 1. When the user selects the **Next** button, they will be sent to Screen2. If the user selects **Inactive** from the drop-down menu, a Variable named varStatus is set to 2. When the user selects the **Next** button, they will be sent to Screen3.

Test this by putting the app in Preview or Play mode, select the Timer control and wait for 10 seconds.

Now test the variables by selecting **Active** or **Inactive** from the drop-down menu and selecting the **Next** button.

As you can see there are different ways to configure navigation in your app. In the previous example, the navigation automatically changed when the user selects a value in the drop-down control. In the second example, navigation can happen in two different ways: when the timer reaches zero or when the user selects a button. The variable still allows you to control which screen they navigate to based on the drop-down value selected, but they will not automatically navigate there when the value is selected. You have the flexibility to choose what works best in your solution.

## Summary

In Power Apps, you can use a number of different controls, variables, and other functions to fully customize the end user's navigation experience.

- Use the **Navigate** function to send the user to any screen in your app.

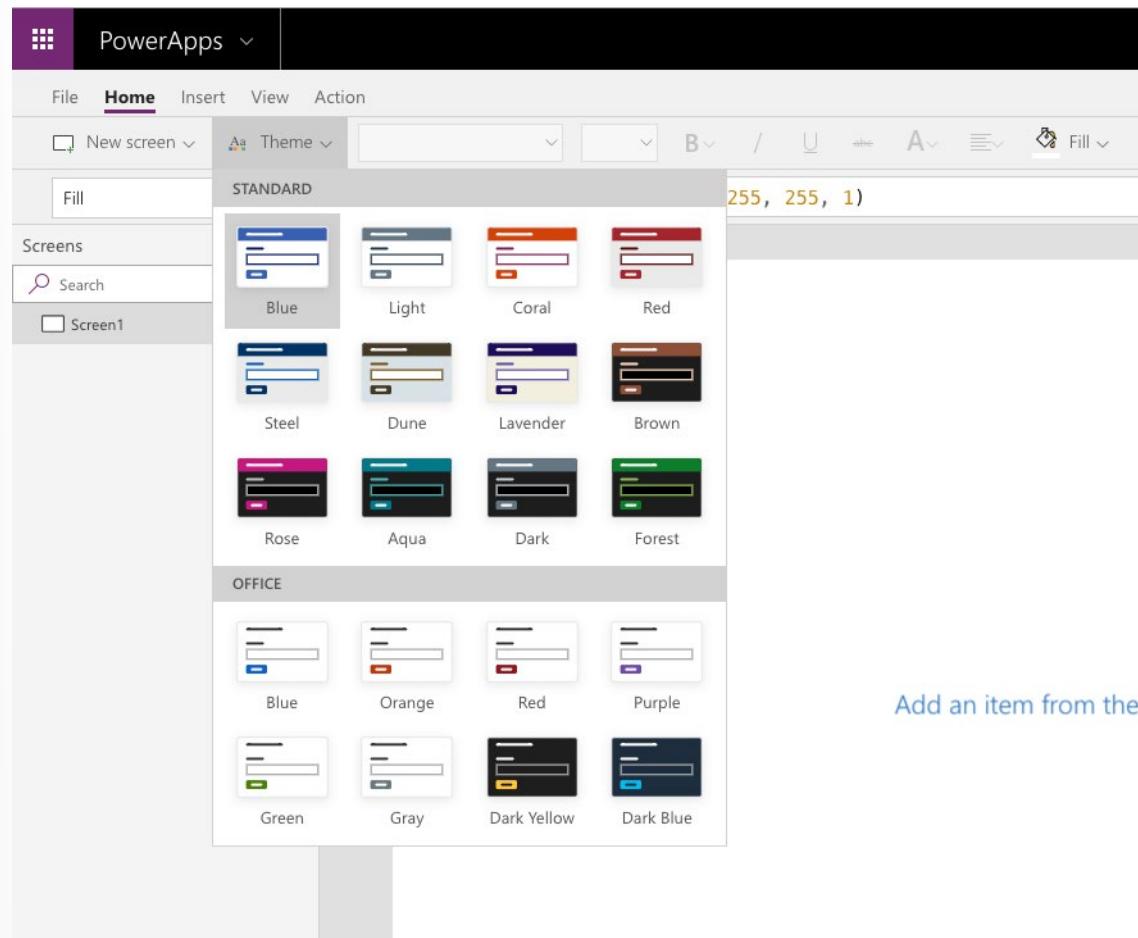
- Create additional navigational functionality by utilizing If statements, variables, and timer controls.
- Use the Back() function to send your end-users back to the screen that they previously navigated from.

By choosing to use a combination of different controls, properties, functions, and formulas you can develop and deploy an app that's both functionally sound and simple to use. It should also be noted, just because you can add navigation in many ways doesn't mean it's always the best to do so. Think about the end-user interacting with the app and keep it simple.

# How to build the UI in a canvas app in Power Apps

## Use themes to quickly change the appearance of your app

A quick and easy way to change the colors in your app is to apply a theme. In Power Apps, there are several out-of-the-box themes to choose from. The following screenshot illustrates all the themes that are available in your app.



These themes have a specific set of default colors and visual elements which will alter the look and feel of your entire app. To use any of these themes in your Canvas app, select a theme from the ribbon. The changes to your app will take effect immediately.

If none of the included themes work for your app, you can create a custom theme. For example, if you decide to select the Steel theme but want the screen background to be a lighter color, this can be easily done. To change the background of your screen, change the **Fill** property to **RGBA(250, 250, 250, 1)**. The

screen will be a slighter lighter shade of gray. Keep in mind, this only changed the fill of that specific screen, if you want to add a new blank screen it would still have the **Fill** property of **RGBA(232, 232, 232, 1)**. This is by design when working with a canvas app. Also, there is

no option to create a Custom theme for a canvas app and store it to be used for other apps.

Typically, many organizations will select the theme that best suits their needs, and then alter the various control properties to align with their corporate branding.

## Branding a control

As noted earlier, one of the built-in themes may not match your organization's desired look and feel for the app. You can customize your app by changing various properties of the app controls. By adjusting a few of the Control properties, like Fill, Hover, and Border you can completely change how the control looks. If you decide to customize your controls, it's recommended that you do this thorough testing to ensure that you don't run into any complications when users interact with the app.

For example, consider the Button control. The following are some of the properties of a Button control that you could customize to better fit your company's theme.

## Typical properties

These properties are in effect when the user is not interacting with the control.

- BorderColor - The color of a control's border.
- BorderStyle - Determines whether a control's border is solid, dashed, dotted, or none.
- Color - The color of text in a control.
- Fill - The background color of a control

## Disabled properties

These properties are in effect when the control is disabled. A control can be disabled if the **Disabled** property is set to **Disabled**.

- DisabledColor - The color of text in a control if its **DisplayMode** property is set to **Disabled**.
- DisabledFill - The background color of a control if its **DisplayMode** property is set to **Disabled**.

## Hover properties

These properties are in effect when the user hovers over the control with a mouse.

- HoverColor - The color of the text in a control when the user keeps the mouse pointer on it.
- HoverFill - The background color of a control when the user keeps the mouse pointer on it.

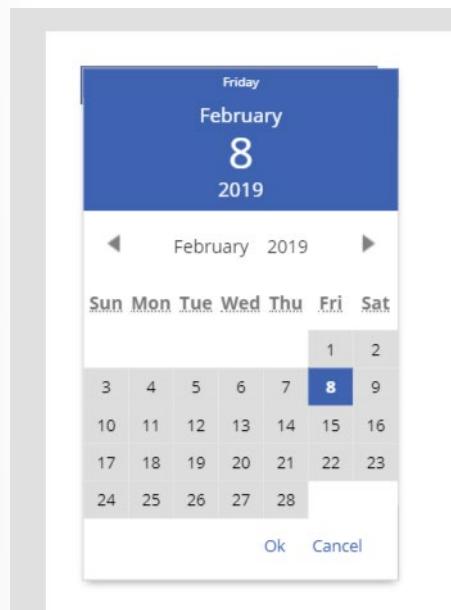
These are just some of the properties that you could modify. For more details about the properties that you can customize, see **Color and Border Properties in Power Apps<sup>37</sup>**.

Note that each control is independent. This means that if you alter the **HoverColor** property of one button control on your screen or in your app, the other buttons in the app will remain unchanged. You must edit the properties of each control that you want to appear in a different manner.

## Some color settings are only controlled by the theme

There are certain aspects of controls that cannot be altered and are specific to the theme that you select. For example, here's an example of the **Date picker** control.

1. In Power Apps Studio, add the **Date picker** control.
2. Put the app in preview mode and select the control so that it opens.
3. The color at the top of the **Date picker** control is specific to the theme, meaning there isn't a control property you can change to set the color manually. To change the color of that background you would have to select a different theme.



As you design your app, be sure to incorporate the use of icons. In the next section, you'll learn how to add an icon to a Canvas app to change the look and feel of the app.

## Icons

When designing your app, utilize Shapes and Icon controls when possible to enhance the user experience of your app. There are certain shapes and icons that are universally recognized that you will find in many of the apps that you use daily. For example, instead of adding a Button

---

<sup>37</sup> <https://docs.microsoft.com/powerapps/maker/canvas-apps/controls/properties-color-border>

control to your app and setting the **OnSelect** property to Back(), you could just as easily use the **Back** icon control and set its **OnSelect** property to Back().

Remember, icons are controls, and each control has a specific set of properties that can be modified to change how the control looks and functions. To view all of available icon controls, select **Insert** and then select the **Icons** drop-down menu.



These controls include arrows, geometric shapes, actions, and symbols. By incorporating shapes and icons, you can save some space and reduce clutter in your app, especially when working with a Phone form factor app. The Phone form factor app has a much smaller area for designing and adding controls so replacing some of the buttons with shapes and icons could really benefit the overall spacing of your app.

## Images

The Image control is a control that shows an image. The image may be from a local file or a data source. Adding an image, such as a company logo is an easy way to enhance the overall look and feel of your app. On each screen you can apply a Background image, as well as multiple Image controls. You're not limited to a certain number of images that you can display; you can have as many images as you would like. Too many images may cause issues with the app performance and load times, but you would have to add quite a few large images for this to be an issue.

There are a set of properties specific to the Image control that allow you to customize how the image is displayed. When working with the Image control, a few of the more common properties you will most likely want to modify are the Image, Image position, and Transparency properties.

**Image** - The name of the image that appears in an image, audio, or microphone control.

**Image Position** - The position (Fill, Fit, Stretch, Tile, or Center) of the image in a screen or a control if it isn't the same size as the image.

**Transparency** - The degree to which the controls behind an image remain visible.

Here's a quick example of how to change the transparency and image position of an Image control.

1. In Power Apps, create a Tablet app using the "Product Showcase" App template.
2. On the left, under **Screens**, select **Image1**.

3. In the right pane, set the **Transparency** property to **0.5**.

You will notice that the image immediately becomes lighter in color.

4. In the **Image position** property, change this to **Center**.

The image appears to be zoomed in or larger.

As you develop your app, keep these settings in mind when using the Image control to enhance the look and feel of your app.

## Personalization

In Power Apps, you can show information about the current user with the **User()** function. This includes the full name, email address, and the picture that's associated with the user who's signed into a canvas app. It will match the "Account" information that is displayed in the Power Apps players and studio, which can be found outside of any authored apps. This may not match the current user's information in Office 365 or other services.

The **User** function returns

a **record<sup>38</sup>** of

information about the current user:

Property	Description
<b>User().Email</b>	Email address of the current user.
<b>User().FullName</b>	Full name of the current user, including first and last name.
<b>User().Image</b>	Image of the current user. This will be an image URL of the form "blob: <i>identifier</i> ".

Set the **Image property<sup>39</sup>** of the **Image control<sup>40</sup>** to this value to display the image in the app.

Here's an example of how to add a user's profile picture, email, and name to your app.

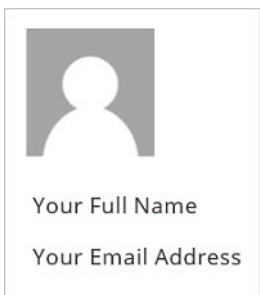
1. On the **Insert** tab, click or tap **Media**, and then click or tap **Image**.
2. Set the **Image** property to this formula: **User().Image**
3. On the **Insert** tab, click or tap **Text**, and then click or tap **Label**:
4. Set the **Text** property to this formula: **User().FullName**
5. Move the label so it's below the image control.
6. Add another label, and set its **Text** property to this formula: **User().Email**
7. Move the label so it's below the first label:

---

<sup>38</sup> <https://docs.microsoft.com/powerapps/maker/canvas-apps/working-with-tables#records>

<sup>39</sup> <https://docs.microsoft.com/powerapps/maker/canvas-apps/controls/properties-visual>

<sup>40</sup> <https://docs.microsoft.com/powerapps/maker/canvas-apps/controls/control-image>



## Using the tablet or phone form factors

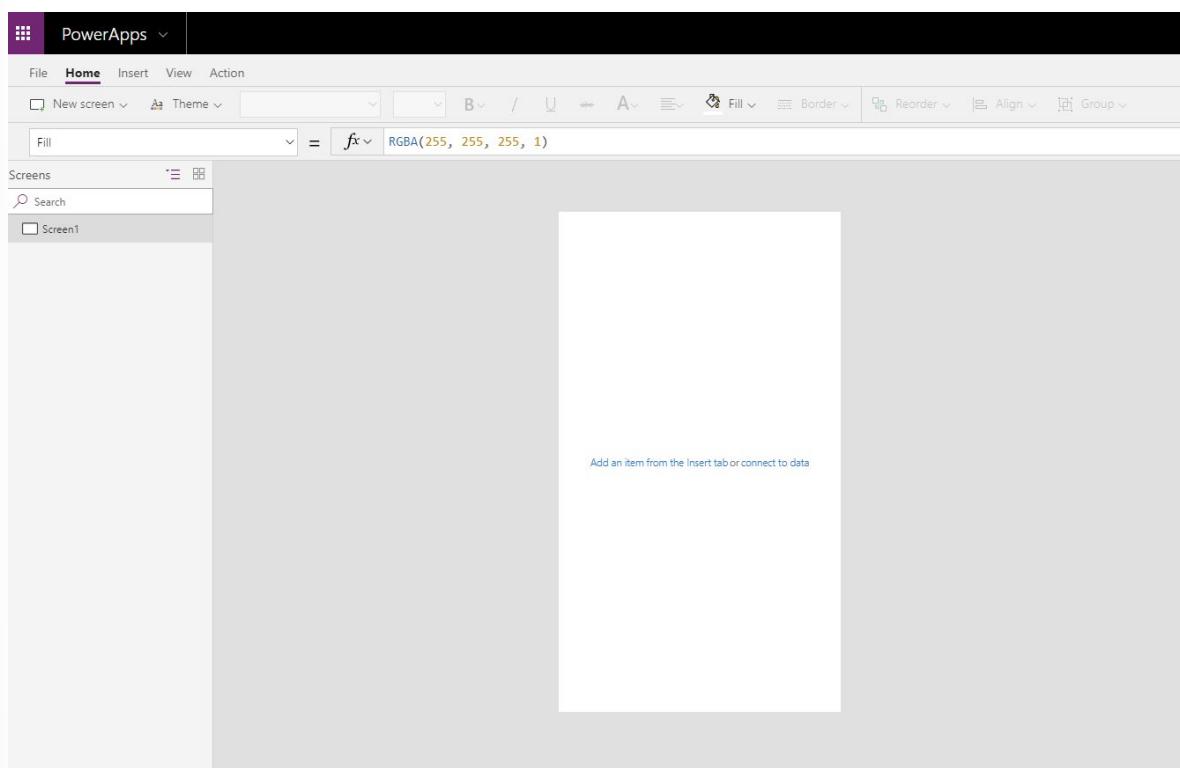
It is important to design your app for the primary device that it will be used on. There are two form factors to choose from, the Phone form factor and the Tablet form factor. The main difference between these is the Screen size. The screen size affects the amount of space available to build the app.

The Phone form factor has a significantly smaller area to build your app, but if most of your users will be accessing the app from a mobile phone then this is the best form factor for you. When building for mobile, select controls that will be easy to use on a mobile device, ensure that the text is large enough to be easily seen, and design the app in a single column vertical format.

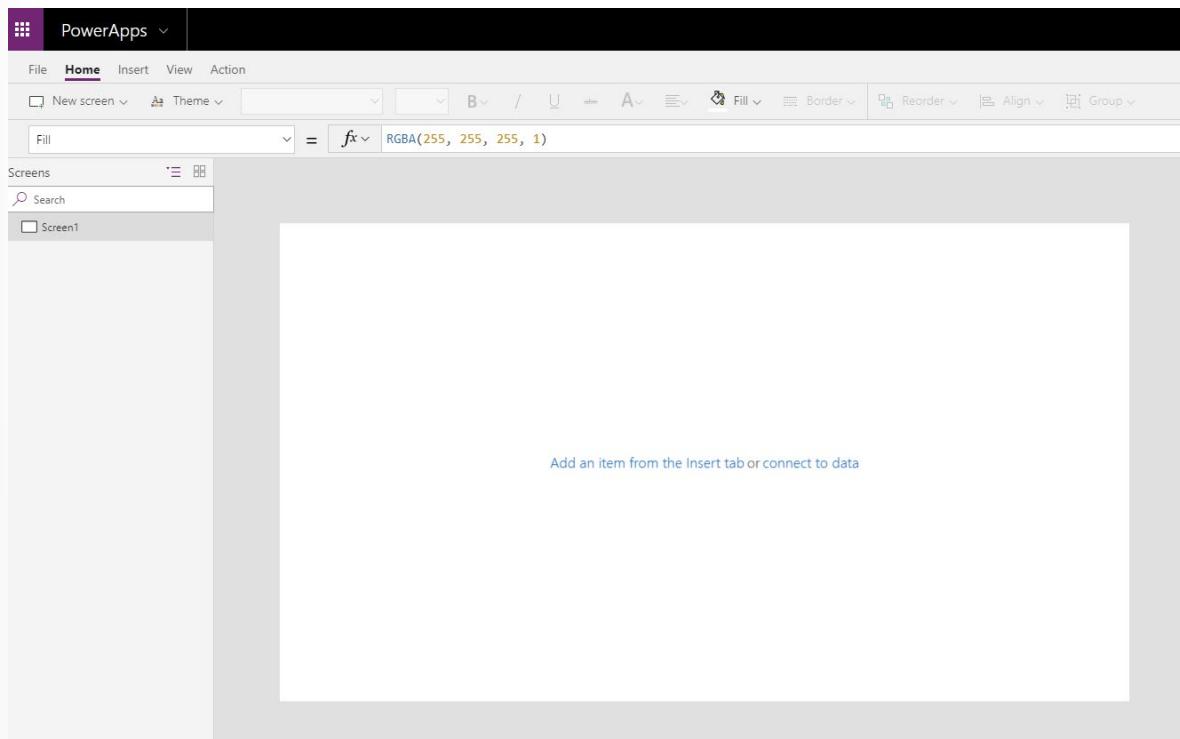
The Tablet form factor has a much larger area for designing your app and is the best option if your users will be accessing the app from a tablet or PC. Because you have more screen size to work with, you will have more flexibility in designing this app. Regardless of the form factor that you choose; the functionality available in the Power Apps Studio will be the same.

Take a moment and create two blank Canvas apps. For one of the apps use the Phone form factor and for the other app use the Tablet form factor. You will immediately notice the difference in the amount of space available on the screen to design your app.

### Phone form factor



### Tablet form factor



Depending on the form factor that you choose, you have the ability to alter the screen size. To view the current screen size and orientation, for either form factor, see following steps.

1. In Power Apps Studio, in the upper-left corner, select **File**.

## 2. On the left, select **App settings**.

When you are in the tablet app, you will notice multiple options to select from for the screen size. Be sure to select the appropriate size for the majority of your app users.

- 16:9 (Default)
- 3:2 (Surface Pro 3)
- 16:10 (Widescreen)
- 4:3 (iPad)
- Custom

The Phone form factor does not include screen size options.

## Lock aspect ratio and lock orientation

Other options that you might want to consider regarding the screen size and orientation include the lock aspect ratio and lock orientation settings.

If you lock the aspect ratio, the app will retain the appropriate aspect ratio for a phone. If the app is running on another kind of device, the app will display incorrectly and may show unwanted results. If you unlock the aspect ratio, the app will adjust to the aspect ratio of the device on which it's running.

If you lock the app's orientation, the app will retain the orientation that you specify. If the app is running on a device for which the screen is in a different orientation, the app will display incorrectly and may show unwanted results. If you unlock the app's orientation, it will adjust to the screen orientation of the device on which it's running.

You can also modify the app's orientation by enabling **Enable app embedding user experience** in **Advanced settings**. This feature aligns the app when it's embedded and changes the background color of the hosting canvas to white.

## Summary

There are many options available to enhance the usability and appearance of your app. Throughout this module, you explored themes, properties for changing a control's appearance, using icons and images, personalization, and the different form factors available. Some important items to remember are:

- There are several out-of-the box or default themes to choose from and these can be easily applied.
- By modifying various Control properties, you can completely change how the control looks and functions when users interact with the app.
- Use the User() function to personalize the app. For example, create a Welcome screen and greet the user by using the User() function.

- You must select the form factor when you begin building your app. After you have selected tablet or mobile, it cannot be changed. With the tablet form factor, you can adjust the screen size.

# Use and understand Controls in a canvas app in Power Apps

## Understand controls

In Power Apps, you can add a variety of user interface (UI) elements to your canvas app and configure aspects of their appearance and behavior directly from the toolbar, Properties tab, or formula bar. These UI elements are called **Controls**. Many of the controls in Power Apps are just like the controls that you've used in other apps, such as labels, text-input boxes, drop-down lists, and navigation elements.

You can find all the controls available in Power Apps on the **Insert** tab.



You can configure the appearance and behavior of a control by setting one or more of its properties. Each type of control has a different set of properties. Some properties, such as **Height** and **Width**, are common to almost every type of control, but other properties, such as **ChevronFill** are specific to only certain Controls.

In addition to the controls listed above, there are other types of controls that you can add to enhance your apps:

**Galleries** - These controls are layout containers that hold a set of controls that show data from a data source. For more information about galleries, see [Work with data in a Power Apps canvas app<sup>41</sup>](#).

**Data table** - The **Data table** control shows data from a data source in a format that includes column headers for each field that the control shows. As an app maker, you have full control over which fields appear and in what order. Like the **Gallery** control, the **Data table** control maintains a **Selected** property that points to the selected row. Therefore, you can link the **Data table** control to other controls.

**Forms** - These controls show details about your data and let you create and edit items. For more information about forms, see [Write data in a Power Apps canvas app<sup>42</sup>](#).

**Media** - These controls let you add background images and sounds. Controls include a camera button (so that users can take pictures from the app) and a barcode reader for quickly capturing identification information.

**Charts** - These controls let you add charts so that users can do instant analysis.

**Icons** - These controls include shapes, graphics, and symbols to enhance the user interface. They are quickly recognizable by your users to ease their interaction with the app. For more information about icon controls, see the [How to build the UI in a canvas app in Power Apps](#) module of this learning path.

<sup>41</sup> <https://docs.microsoft.com/learn/patterns/work-with-data-in-a-canvas-app/>

<sup>42</sup> <https://docs.microsoft.com/learn/modules/write-data/>

In this module, you will learn about some of these controls and their properties to see how you can incorporate them into your app to enhance the overall functionality. Before developing your app, take a moment to determine the functionality that you want to provide and then select the control that best fits those needs. The more familiar you are with controls and how to work with them, the easier it will be to design your app.

In some cases, certain controls are almost interchangeable, and in those cases, you can use the control that you prefer. For example, the **Dropdown** and **Combo box** are similar controls. One key difference between them is that a **Combo Box** allows you to search for items as well as select multiple items. The **Dropdown** control does not support this functionality.

## Core properties of controls

Because controls are designed with specific use cases in mind, the properties for each control are slightly different. Here are some important properties to be aware of:

- **Default** - The initial value of a control before it is changed by the user. For example, when working with a Drop down control you could set the default value to appear when users see the control.
- **DelayOutput** - Set to true to delay action during text input.
- **DisplayMode** - Values can be **Edit**, **View**, or **Disabled**. Configures whether the control allows user input (**Edit**), only displays data (**View**), or is disabled (**Disabled**). For more information about display modes, see Use basic formulas to make better canvas apps in Power Apps learning path.
- **Items** - The source of data that appears in a control such as a gallery, list, or chart.
- **OnChange** - How the app responds when the user changes the value of a control. For example, when a user selects a different value in a Dropdown control.
- **OnSelect** - How the app responds when the user taps or clicks a control.
- **Reset** - Whether a control reverts to its default value. For more information, see **Reset function in Power Apps**<sup>43</sup>.
- **Text** - Text that appears on a control or that the user types into a control.
- **Tooltip** - Explanatory text that appears when the user hovers over a control.
- **Visible** - Whether a control appears or is hidden.

To view a list of all the controls and their properties, see **Controls and properties in Power Apps**<sup>44</sup>.

---

<sup>43</sup> <https://docs.microsoft.com/powerapps/maker/canvas-apps/functions/function-reset>

<sup>44</sup> <https://docs.microsoft.com/powerapps/maker/canvas-apps/reference-properties>

# Entering and displaying data with text controls

In Power Apps, text controls are used for all kinds of purposes, such as displaying text, numbers, dates, and calculating currency.

For example, you could calculate the value of two Text input controls and display the results in a Label control.

There are several text controls that you could incorporate into your app.

Each of these controls have slightly different purposes and use cases.

For example, maybe you're creating an employee survey app, and you want to get employee feedback. In this scenario, you would use the **Text input** control and modify the **Mode** property to be **Multiline**

instead of single line. Most likely, every app that you work with will have text controls so familiarizing yourself with these controls will be

helpful as you develop your app. The follow list details the different text controls that are available.

- **Label** - A label shows data that you specify as a literal string of text, which appears exactly the way you type it, or as a formula that evaluates to a string of text. Labels often appear outside of any other control (such as a banner that identifies a screen), as a label that identifies another control (such as a rating or audio control), or in a gallery to show a specific type of information about an item.
- **Text input** - A box in which the user can type text, numbers, and other data. For example, a user can specify data by simply typing in a text input control. Depending on how you configure the app, that data might be added to a data source, used to calculate a temporary value, or incorporated in some other way.
- **HTML text** - An HTML text control not only shows plain text and numbers but also converts HTML tags, such as non-breaking spaces.
- **Rich text editor** - The rich text editor control provides the app user a WYSIWYG editing area for formatting text. This control should be used if you want to allow the user to provide numbered lists or bullet lists. A good example is a Power Apps app that used to collect content for an article or newsletter, where you allow the user to add formatted text that would be helpful for the person compiling the article.
- **Pen input** - A control in which the user can draw, erase, and highlight areas of an image. The user can use this control like a whiteboard, drawing diagrams and writing words that can be converted to typed text.

Here's a closer look at the Label control and a few examples so you can get a better idea as to how it works.

First, add a label control to show text.

1. In Power Apps Studio, add a **Label** control.
2. Set **Text** property for the Label to "**Hello, world**" (including the double quotation marks).

Next, create a more dynamic solution by combining a button, gallery, and multiple label controls. In this scenario, you'll create a collection called **CityPopulations** that contains data about the population of various cities in Europe. Next, you'll show that data in a gallery that contains three labels, and you'll specify the type of data that each label will show.

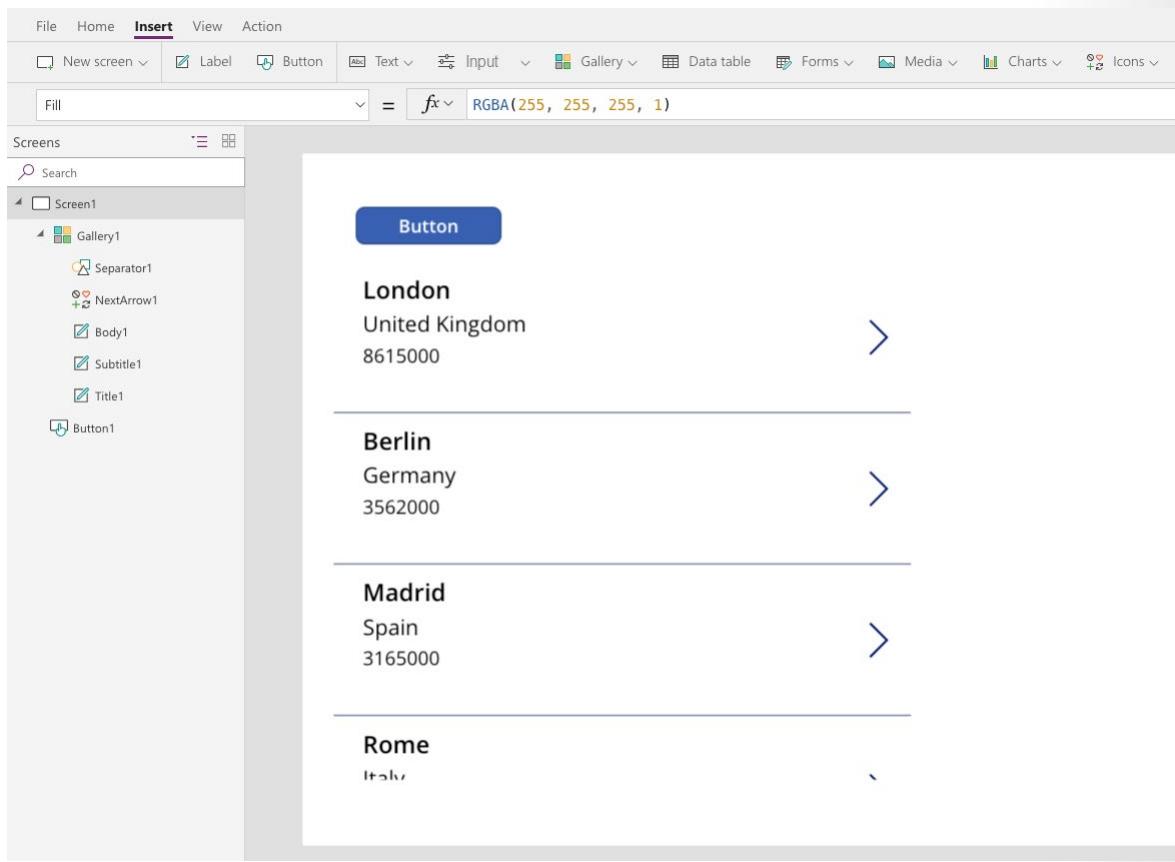
1. Add a button, and set its **OnSelect** property to this formula:

```
ClearCollect(CityPopulations, {City:"London", Country:"United Kingdom", Population:8615000}, {City:"Berlin", Country:"Germany", Population:3562000}, {City:"Madrid", Country:"Spain", Population:3165000}, {City:"Rome", Country:"Italy", Population:2874000}, {City:"Paris", Country:"France", Population:2273000}, {City:"Hamburg", Country:"Germany", Population:1760000}, {City:"Barcelona", Country:"Spain", Population:1602000}, {City:"Munich", Country:"Germany", Population:1494000}, {City:"Milan", Country:"Italy", Population:1344000})
```

2. Press and hold **Alt Key**, and select the **Button** control. (This will create your collection and store all the information.)
3. Add a **Blank vertical** gallery and set its **Items** property to **CityPopulations**.
4. With the gallery selected, in the right pane, change the layout from blank to **Title, subtitle, and body**.
5. Select the top or first Label, in the **Text** property the default for example shows **ThisItem.City**, you could change this to something else if you would like. For more information about the ThisItem operator, see **Operators and data types in Power Apps<sup>45</sup>**.
6. Select the middle or second Label, which shows as **ThisItem.Country**.
7. Select the last or third Label, change the **Text** property to **ThisItem.Population**.

---

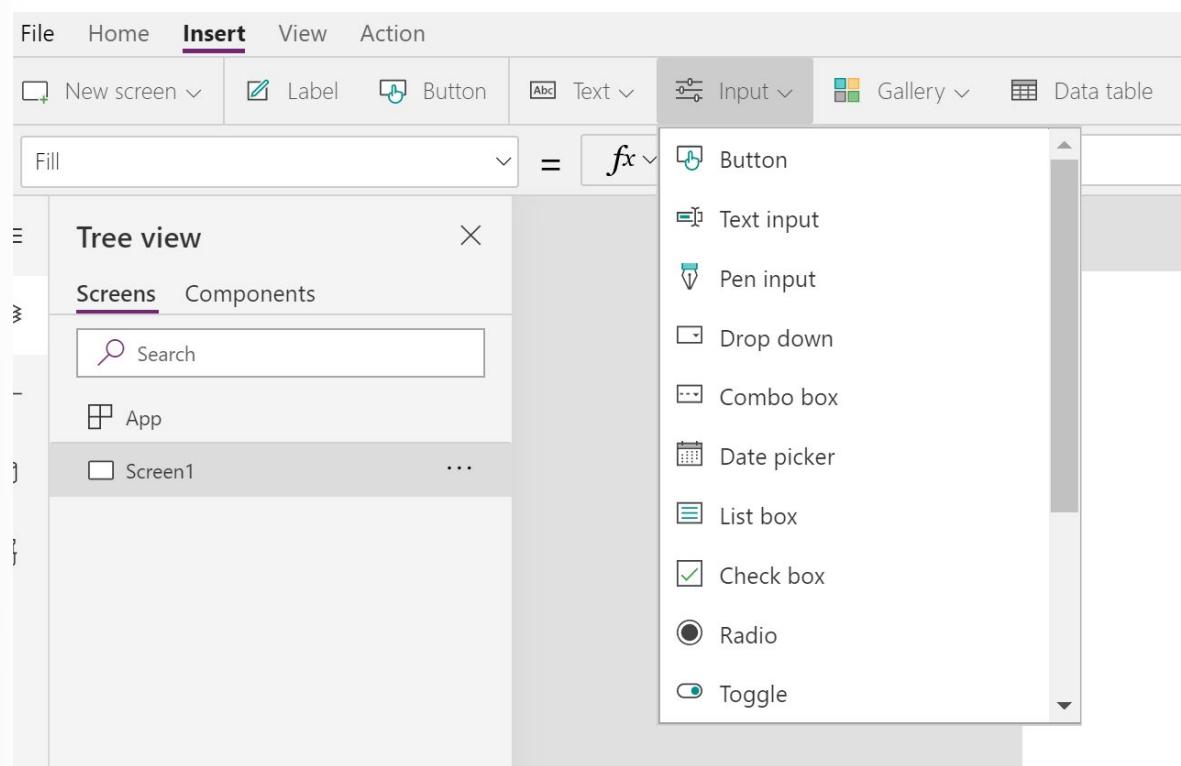
<sup>45</sup> <https://docs.microsoft.com/powerapps/maker/canvas-apps/functions/operators>



These were just two simple examples demonstrating some basic functionality of a label control. There are many other ways to utilize label controls in your app.

## Additional controls for enhancing your app's usability

At the beginning of the module, you learned what controls are, when to use them, and how to modify control properties. In this section, you get a more detailed look into some of the controls available from the Controls drop-down menu in the ribbon. On the **Insert** tab, if you select the **Controls** heading, you will get a list of several different controls.



Remember, every control was designed with different use cases in mind. The following information about a few of the controls will help you to decide when you might use each one.

## Controls with pre-populated values

Each of these controls allows you to determine the values the user can select from. Use these controls when you want to control the values for your data set. This is often an important consideration for reporting.

As a trade-off, you may miss valuable insights that come from free form answers. Here's a closer look at the differences between each of these controls.

- **Drop down** - This control conserves screen real estate, especially when the list contains a large number of choices. The control takes up only one line unless the user selects the chevron to reveal more choices. This control will show a maximum of 500 items.
- **Combo box** - This control allows you to search for items you will select. The search is performed on the server on the **SearchField** property so performance is not affected by very large data sources.
- **List box** - This control always shows all available choices (unlike a **Dropdown**<sup>46</sup> control) and in which the user can choose more than one item at a time

---

<sup>46</sup> <https://docs.microsoft.com/powerapps/maker/canvas-apps/controls/control-drop-down>

(unlike a **Radio**<sup>47</sup> control).

- **Check box** - A control that the user can select or clear to set its value to **true** or **false**. The user can specify a Boolean value by using this familiar control.
- **Radio** - A **Radio** control, a standard HTML input control, is best used with only a few, mutually-exclusive options.

## Controls for ratings

When you have a specific need for the app users to rate items, the following two controls provide a better experience than free form text or drop-down controls.

- **Slider** - The user can indicate a value, between a minimum and a maximum value that you specify, by dragging the handle of a slider left-right or up-down, depending on the direction that you choose.
- **Rating** - In this control, the user can indicate, for example, how much they liked something by selecting a certain number of stars.

## Other available controls

- **Toggle** - Use this control to enhance the UI of the app. It functions in the same manner as the check box control.
- **Timer** - A control that can determine how your app responds after a certain amount of time passes. For example, determine how long a control appears or navigate to another screen after a certain amount of time has passed.
- **Button** - Configure the **OnSelect** property of a button control to run one or more formulas when the user clicks or taps the control. The button control is frequently used to submit data to the data source.

## Media

Media controls give the app designer a way to display and share an audio file or video file easily, take a picture, scan bar codes, etc.

For example, you need to build an employee safety training app. For the solution, you want to share a video with your employees and then have them acknowledge that they have completed the training. With the Video control, the safety video can be shown in the app and then the next screen would be a training acknowledgment form. Or maybe you have an expense report app and want to let users upload pictures of their receipts. By using the Camera control, the user can take a photo and attach the picture without leaving the app.

<sup>47</sup> <https://docs.microsoft.com/powerapps/maker/canvas-apps/controls/control-radio>

Here are some details about the Media controls that are available:

- **Video** - This control plays a video clip from a file or from YouTube or Azure Media Services. Closed captions can optionally be shown when specified.
- **Audio** - This control plays a sound clip from a file, a recording from a Microphone control, or the audio track from a video file.
- **Camera** - A control which allows the user to take photos by using the camera on the device.
- **Bar code scanner** - This control opens a native scanner on an Android or iOS device. The scanner automatically detects a barcode, QR code, or data-matrix code when in view. The control doesn't support scanning in a web browser.
- **Microphone** - A control that allows app users to record sounds from their device as long as the device has a microphone accessible. Audio is stored in 3gp format in Android, AAC format in iOS, and OGG format in web browsers.
- **Add picture** - With this control users can take photos or upload image files from their device and update the data source with this content. On a mobile device, the user is presented with the device's choice dialog to choose between taking a photo or selecting one that is already available.

Here's how you can add a video to your app. In this example, you will have two screens. When a user selects the button on the first screen they will navigate to the second screen where a video will start playing automatically.

1. In Power Apps studio, add a new blank screen. You should now have two screens.
2. Make sure that you are on the first screen and add a **Button** control.
3. On the left, under **Screens**, select **Screen 2**.
4. On the ribbon, add a **Video** control.
5. For the **Media** property, in the formula box, enter a video URL from YouTube. The URL must be inside quotations.
6. Now set the **AutoStart** property to **true**.
7. Go back to Screen 1, set the **OnSelect** property for the button to: **Navigate(Screen2,ScreenTransition.Cover)**.

Test out your app. Put the app in preview or play mode and press the button.

## Summary

In this module, you learned more about controls and how you can incorporate them into your app. You also learned about the properties of controls and how you can modify them to enhance the functionality

of your app. Depending on the app that you're developing, you can use several different controls, or you can only use a few specific ones. Here are a few important points to remember:

- There are many control options for placing text on a screen or allowing a user to enter text.
- Use the pre-populated value controls to provide choices for the user.
- By adding media controls directly to your app, you can enhance the functionality and reduce the number of steps for your user. For example, users can enter their expenses and upload their receipts without leaving the app.

# Document and test your Power Apps application

## Create test plans

There are many types of testing that you should consider in your deployment planning. Each of these can be done either casually, or very formally with a well-documented process to follow. Often in different deployments these will have slightly different expectations. Some testing options are manual (user adoption testing), and some are automated (load testing).

**Unit testing-** this is the smallest component of testing, a single unit of measure. Often this is thought to be a developer's job. However, every person on a project should include this in their work. An example of unit testing could be: You are given a requirement to add a calculated field to a form. Before you advance the task to the testing team you would confirm the field was on the form as expected and test that the calculation was as expected.

**Quality assurance testing-** sometimes called QA, this is when someone independently tests the task by reviewing the expectation against the work performed. Often there will be testing of the smallest components, such as the units mentioned above and also testing of a full end-to-end user story.

**Performance testing** (sometimes also referred to as load testing)- there are many things that can be configured that will have an impact on system performance. Sometimes the number of concurrent users affects expected performance (load); sometimes it's the combined actions of these users, and sometimes an individual user could experience problems such as a slow-loading form that has too many sub-grids. Integrations with other systems might also affect performance. Performance or load testing is most often done using external tools.

**Integration testing-** did the systems we integrated with perform as expected? Did we introduce unnecessary dependencies?

**User acceptance testing-** this is an opportunity to have direct feedback from the actual users of the system being built. User acceptance testing (hopefully) confirms the system built is the system users need.

Regardless of the exact type or plan you are using; the following considerations should be included.

## Use of Testing Plans

A testing plan is an important scoping document in your project plan to communicate what will and will not be tested and necessary considerations for success. The testing plan is designed to be carried out by a technical resource to catch bugs before user acceptance testing.

At a minimum, a test plan should provide a means to test each business requirement for the project. This gives the opportunity to show that each requirement was accommodated and accomplished. When there are issues in the future, a testing plan is a helpful reference of what was tested based on understandings at the time and what may be outside the original scope. The plan needs to define the level of detail to be tested, and the process for solving and retesting any issues found.

## Environment

Requirements of the test plan outline the environment needed to test accurately. The environment for testing should be as close to the technical set up of the production environment as possible. If the

application will be available in multiple formats such as desktop, mobile, and offline testing should be completed in all of these environments.

There may be times when the needed environment is not available at the time of testing. It is important that stakeholders are aware of these needs and dependencies and how they influence deadlines.

## Security

In creating a testing plan, it is important to consider all of the user personas that will use the application and test from the correct security roles. Common considerations on security include determining relevant security roles, field security profiles, business unit membership, and team membership. The testers means of authenticating should be the same as the production users. For example, if users will access the application through a VPN, testing needs to be completed over VPN.

## Dependencies

Consider data dependencies such as child records needed to populate lookups. If needed data is uncovered during testing be sure to note it so you can remember to make the data available to users during user acceptance testing and the rollout to production.

Be mindful of customizations that reference specific records such as a specific account or custom child record. If a customization such as a workflow or business rule relies on a specific record that record will need to be migrated, so the GUID is the same in both environments. Using the data import wizard will replicate the data in the new environment with a different GUID.

## Solution Awareness

Some configurations, such as Word and Excel templates, cannot be moved between environments and will need to be configured in each environment. Any configuration that must be complete in each environment should receive extra attention in testing.

## Resources

It is best practice to have components tested by a resource that did not create or configure the component.

## Test Plan Components

The following components are commonly found in testing plans.

- Environmental needs
- Features tested
- Features not tested
- Testing Items
  - Related Business Requirement
  - Steps to test
  - Expected result
  - Item pass or fail
  - Time and date tested

- Tester name
- Testing Notes
- Testing Schedule
  - Dependencies
  - Timelines
  - Schedule risks
- Process for fixing and retesting failed test items

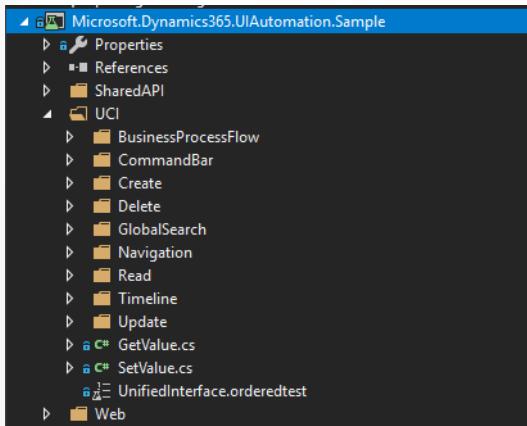
A good test plan describes the intention and scope of testing, guides the technical review process and supports a smooth rollout of functionality. A test plan should proceed user acceptance testing and have a means for tracking needed changes before rollout.

## User interface testing

EasyRepro is a library of tools from Microsoft to provide Dynamics customers the ability to facilitate automated UI testing for projects. The functionality provided covers the core commands that end users would perform on a typical workday and working to extend that coverage to more functionality. EasyRepro supports testing for Dynamics 365 versions 8.1 and higher.

## Sample Tests

Over 100 standard tests for the Web Client and over 50 tests for the Unified Interface are available. The sample test bank covers create, retrieve, update, and delete scenarios for multiple entities including contacts, opportunities, accounts, leads, and cases. This provides you the ability to easily run and start creating customized tests using the test bank as your initial code base. You can easily access the pre-defined tests within each of the specific functional areas you are testing.



## Coverage

### Functionality Covered

- Activity Record Wall
- Business Process Flow
- Charts

- Command bar
- Dashboards
- Entities (Create, Update, Delete, Duplicate Detection, Assign)
- Global Search
- Guided Help
- Grids
- Sub-Grids
- Navigation
- Notifications
- Performance Center
- Quick Create
- Run Workflows
- Run Reports

## Functionality Not Covered

- Settings Area (Administration)
- Customization
- Advanced Find Query execution (Open Advanced Find is available)
- Export to Excel
- Templates
- Other Add-on Applications

Although there are not specific commands to cover the above functionality, you do have generic commands that will allow you to still interact with those features.

Though EasyRepro is available to all, using it requires the use of Visual Studio and will likely require developer assistance.

## Performance optimization

When analyzing performance of Common Data Service, it is important to understand areas affecting the performance of an app.

## Environment

By carefully examining various environmental factors you can improve your overall experience with Common Data Service apps, even before your business begins to use it.

## Network

Network capacity can greatly affect performance of the apps from the end-user point of view. That includes web apps, Dynamics 365 for Outlook app, and custom applications using Web API.

There are two primary network characteristics affecting the performance: network bandwidth and latency. Collaboration with your network team is essential to making sure that network configuration is adequate and does not affect app performance.

## Client Configuration

Common Data Service apps are web-based applications and use web browsers as the user interface to view, add, or edit information that you've stored in the apps database. You may need to work with your system administrators to make sure that the client machines and software settings are optimized for performance.

## App Performance

Design of the app itself can greatly affect performance. One of the strengths of the Power platform is ability to quickly build robust small apps targeting specific users and specific functionality. Instead of a monolithic app that is more likely to suffer from performance issues you now have the ability to design several separate apps. This separation ensures that each app can be analyzed and tuned independently without too much impact on the others.

Design techniques for individual components may also affect performance of an app.

## Model-driven Apps

### Forms

To improve form performance, design them so that only the information essential for the users is included. If information required depends on the user's role, consider creating multiple forms instead of one monolithic form that includes every single field. Hiding tabs and sections that are not required to be immediately visible can also improve form responsiveness and performance.

One of the most common sources of performance issues on the forms is custom scripts added by developers. Good collaboration within the team is a key to identify and resolve any performance bottlenecks introduced during the development.

### Views

To improve performance of the entity views, limit the number of columns displayed only to those required by the business. As often the case, it's a balance between the responsiveness and bringing enough information to improve usability of the view.

Customize your views so that they bring back an actionable number of records. For example, Active Contacts view may contain hundreds of thousands of records while My Active Contacts will limit that view only to the contacts that are owned by the user. At the same time consider limiting the number of records per page so that the views return much more quickly.

### Quick Search

Optimizing the quick search view only to include columns that makes sense to search will make a noticeable performance impact on searches within the views.

My Active Contacts

Full Name	Email	Company Name	Business Ph...
Nancy Davolio	ndavolio@fabrikam.com	---	555-1234
Nancy Davolio	nancy@fabrikam.com	---	555
Rendall Guidden		Contoso	---
William Contoso	william@contoso.com	---	555-1233

When a user searches for records within a view, Common Data Service performs a query that searches only predefined fields. Which fields are searched is defined by the Find Columns settings in the system Quick View.

## Add Find Columns

Select the columns to be searched on for this Find view.

<input type="checkbox"/> Display Name ▲	Name	Type
<input type="checkbox"/> Employee	crf3f_employee	Lookup
<input type="checkbox"/> Fax	fax	Single Line of Text
<input type="checkbox"/> Finance Amount	crf3f_financeamount	Currency
<input type="checkbox"/> Finance Amount (Base)	crf3f_financeamount_base	Currency
<input checked="" type="checkbox"/> First Name	firstname	Single Line of Text
<input type="checkbox"/> Flight departure	crf3f_flightdeparture	Date and Time
<input type="checkbox"/> Follow Email Activity	followemail	Two Options
<input type="checkbox"/> FTP Site	ftpsiteurl	Single Line of Text
<input checked="" type="checkbox"/> Full Name	fullname	Single Line of Text
<input type="checkbox"/> Gender	gendercode	Option Set

OK Cancel

The fewer number of columns that are selected, the faster quick search will be. Balance the number of find columns with the effectiveness of the search.

## Canvas Apps

The primary source of performance issues in canvas apps are poorly designed data access, excessive or suboptimal formula use, and overly complex user interface.

## Loading Data

Apps will perform poorly when most of the heavy lifting is done on the client and not on the server. Erroneous choice of how to query your data may lead to performance issues. Understanding **delegation**<sup>48</sup> in canvas apps is the key to improving your app performance when it comes to data access.

There are some canvas apps features that you would want to consider when optimizing your app start performance by using Concurrent function to load data from multiple sources concurrently.

## Controls Optimization

Another aspect that can affect performance is the number of screens and controls used in your app. Minimizing the number of controls and reducing complexity of the controls used can help boost your app performance. It will improve the performance while authoring the app as well. There are strategies to optimizing the number of controls used in your app. For example, you can use a gallery control instead of a Canvas/Data Cards when the data displayed is uniform or vary only slightly. Gallery can be powerful in reducing complexity, making your app easier to maintain.

There are other optimizations techniques available, make sure that canvas apps developers familiarize themselves with **detailed documentation**<sup>49</sup>.

## Workflows

Real-time workflows can be very effective but, if designed poorly, will affect the performance of the most common operations such as creating or updating a record. Consider converting poorly performing or long running workflows to background ones where the impact on the system will be smaller.

Background workflows create a log record when an instance of the workflow is executed. The default setting is to delete log entries for successfully executed workflows. If this setting is changed to keep the entries, jobs log will grow and may reach the point where it affects performance of the asynchronous services. Only use this flag for troubleshooting and avoid leaving it on in production.

## Plugins

Custom extensions such as plugins are the common source of the performance issues. Work with developers to identify and resolve those.

## Performance Testing

Designing for performance is an important part of the process and so is performance testing. It is important to perform performance testing on the system where the volume of data is close to your production system. That will give you a good understanding for app performance in general and for the areas to optimize. Use the instance copy function to create a copy of the production instance including the data.

The other important aspect of adequate performance testing is to reproduce workloads expected in the production environment. There is often a disconnect between testing environments where the system is accessed by a handful of the testers and the production instance where hundreds and thousands of users may access the system at the same time. There are load testing tools available from Microsoft and

---

<sup>48</sup> <https://docs.microsoft.com/powerapps/maker/canvas-apps/delegation-overview>

<sup>49</sup> <https://docs.microsoft.com/powerapps/maker/canvas-apps/performance-tips>

third-party vendors and you will need to collaborate with your development team to ensure that the tools are installed, configured, and used appropriately within the development lifecycle.

## Instance Statistics

CDS for Apps analytics are available via Power platform admin center <https://aka.ms/ppac>. It contains various statistics that can help identify potential areas for deeper performance analysis, for example most used entities, storage consumed, failing system jobs and plugins, etc.

## Query Performance

Common Data Service is a service and, as any service, may not cater for your specific app design. Specific queries in your app may cause some performance issues. While you don't have direct access to the components of the Common Data Service such as database, there are tools available within Common Data Service to help in tuning the environment to your specific needs. You can analyze and optimize query performance using the Data Performance view discussed in detail in the next topic.

There are a number of tools that can assist in performance tuning various aspects of Power Apps.

## Dynamics 365 Diagnostics

There are two primary network characteristics affecting the performance: network bandwidth and latency. Identify if these may impact your app performance, by using the Dynamics 365 Diagnostics tool that is available at <https://<myorg>.crm.dynamics.com/tools/diagnostics/diag.aspx> where <myorg>.crm.dynamics.com is the URL of your Dynamics 365 organization.

**Dynamics 365 Diagnostics**

Diagnostic tests:

Data Point	Action	Status	Results Summary
Latency Test		complete	66 ms
Bandwidth Test		complete	200 KB/sec
Browser Info		complete	
IP Address		complete	
JavaScript Array Benchmark		complete	21 ms
JavaScript Morph Benchmark		complete	25 ms
JavaScript Base64 Benchmark		complete	5 ms
JavaScript Dom Benchmark		complete	46 ms
Organization Info		complete	orga0fe228b
All Tests	Run	complete	

Results:

```
Client Time: Mon, 21 Jan 2019 05:46:24 GMT
== DOM Benchmark ==
Total Time: 46 ms
Breakdown:
Append: 10ms
Prepend: 12ms
Index: 2ms
Insert: 4ms
Remove: 18ms
Client Time: Mon, 21 Jan 2019 05:46:24 GMT
== Organization Info ==
Organization name: orga0fe228b
Is Live: True
Server time: 1/21/2019 5:46:18 AM UTC
Url: https://practice102.crm6.dynamics.com/tools/diagnostics/diag.aspx
Client Time: Mon, 21 Jan 2019 05:46:24 GMT
```

**Clear** **E-Mail Results**

After opening the page and pressing Run, the report will become available that will assist your network team in detecting and resolving the issues. The statistic includes some browser benchmarks that may be useful in identifying if browser performance is an issue.

## Data Performance Logs

You can analyze and optimize query performance using the Data Performance view, which provides an aggregated list of entities with long-running queries. A long running query is defined as a query that takes three seconds or longer to complete. Typical examples of a component that can have a long running query is a plug-in with custom FetchXML or a sub-grid or view.

You can access Data Performance Logs via Settings - Administration - Data Performance.

Entity	Count	Optimization Status	Optimization Impact (%)
systemuser	480	Optimization Available	0.00

If one or more long running entity queries are detected, log items are displayed in the view. You can use the Optimize command to apply optimizations to the selected query.

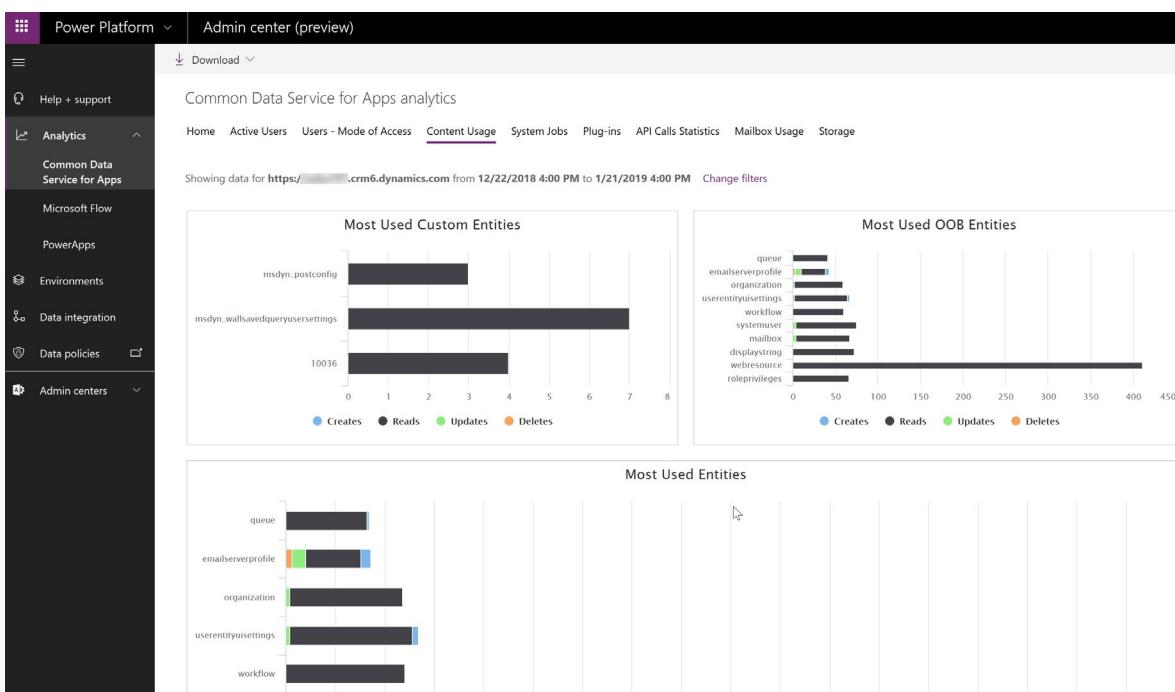
Note: With the most recent advances in Azure SQL database self-tuning capabilities you may no longer see any long running queries reported in the Data Performance view. Slow queries are automatically detected and optimized at the database level.

## Analytics

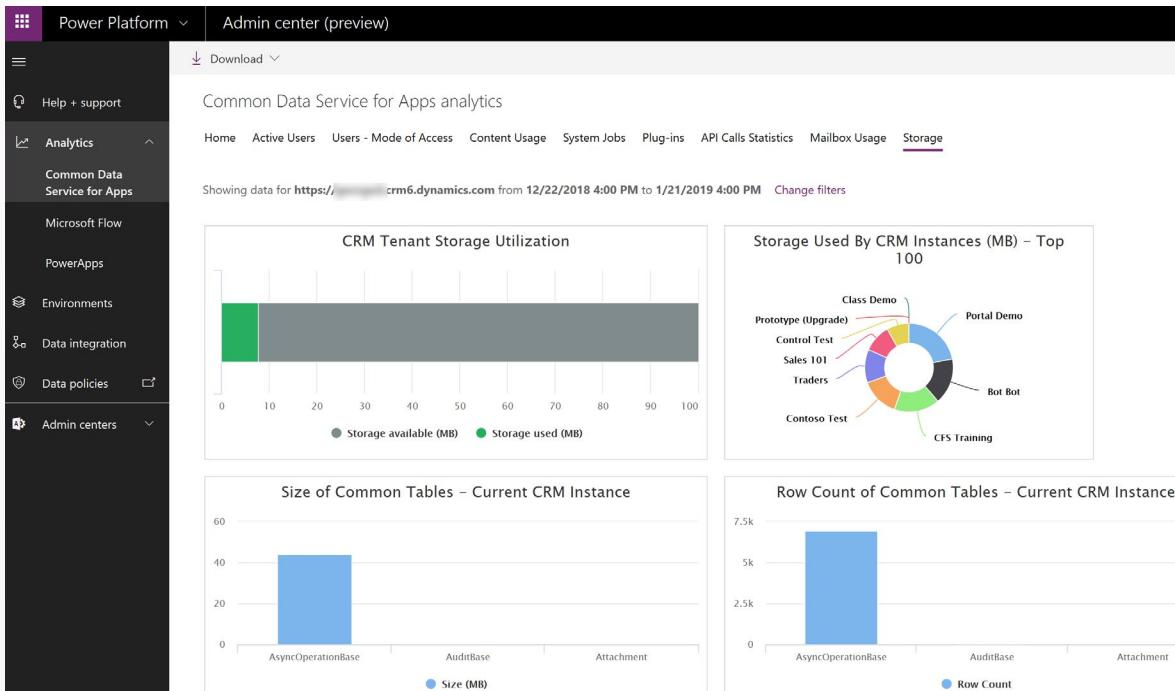
Common Data Service analytics are available via the Power platform admin center <https://aka.ms/ppac>. Various statistics can help identifying potential areas for deeper performance analysis.

## Common Data Service

The Content Usage tab contains information about most used out-of-the-box and custom entities. Identifying the most used entities helps concentrate the optimization efforts on the parts of the app that matter.



The Storage tab provides a number of storage statistics across all of your tenant instances as well as breakdown of the storage within the selected instance.



This can help identify entities with excessive storage use that would be a primary target for further performance investigations.

Other statistics are also available that may be useful. Plug-ins statistics can assist in identifying troublesome plugins, for example those with excessive average execution time.

## Power Apps Analytics

Power Apps Analytics help with performance analysis by providing detailed statistics on Service Performance across various connectors used in your app.

Additional information such as overall app usage, location statistics, or errors seen by end users can be extracted from other tabs. While not directly performance related, this information can be useful in pinpointing overall problematic areas that may be caused by poor app performance such as specific region, specific app, etc.

## Documentation and the customer

Good documentation is key to the successful rollout of a Power Apps application. This video provides some best practices and suggestions on what to document for an app.



<https://www.microsoft.com/videoplayer/embed/RWs1Uz>

## summary

In this module, we covered testing plans, performance optimization and documentation.



<https://www.microsoft.com/videoplayer/embed/RWrMe8>



## Module 3 Master advance techniques and data options in canvas app

### Use imperative development techniques for canvas apps in Power Apps

#### Imperative versus declarative development

After completing the previous recommended learning paths, you should have a solid foundation with Power Apps. In this learning path, you will learn some of the more advanced concepts that Power Apps supports.

This module includes some concepts that might appear to be very developer focused but do not worry. The goal is to help you understand and apply some of these concepts to build better apps.

#### Imperative versus declarative

There are two main ways to do development, imperative and declarative. In imperative development, the focus is on how to achieve the goal. With declarative, the focus is on getting the result. Imperative provides more flexibility because you control every step in the process, but that means more code and more complexity. Declarative is much simpler and straightforward to use but can lack the ability to have the complete control that you might want.

#### Imperative

To better understand imperative programming think about the sandwich that you want for lunch. In imperative programming, you focus on creating the sandwich in your "code". You go to the kitchen, get the ingredients, put the sandwich together, and then send it to the user. You spend a lot of time on the steps, but you have all of the specific functions you want to make it exactly the way you want. No tomatoes? No problem. In a completely made up programming language your code might look something like this.

Function Create Sandwich

```
{  
Go to kitchen;  
Get ingredients;  
Remove tomato;  
Assemble sandwich;}  
  
Function Send Sandwich  
{  
Destination Mouth;}
```

As you can see, there are lots of steps in the process, but you get a sandwich exactly the way you want. This is the approach you will see in languages like C# or other popular coding languages. The focus is on pushing the data.

## Declarative

For declarative programming, think of the same scenario, your sandwich for lunch. The difference is now you are focused on producing the sandwich, not how to make a sandwich. This is much less complex, but you might also run into the issue with tomatoes. If the function you use to get the sandwich doesn't support the option of no tomato you are out of luck. Your code may be as straight forward as follows.

GetSandwich(Kitchen, Mouth)

That made up function takes two inputs, where to get the sandwich from and where to send it. There was no option to remove tomatoes. It would be up to the creator of the GetSandwich function to add an option for no Tomato, which might look like this.

GetSandwich(Kitchen, Mouth, {Tomato: false})

Low-code tools like Excel use this approach to development. The focus is on pulling data.

## Power Apps supports both imperative and declarative methodologies

Power Apps has capabilities for both imperative and declarative logic. Throughout this training there has been a focus on declarative formulas. In the remainder of this module, the focus will be on imperative concepts and methodologies in Power Apps. The key component of imperative logic in Power Apps is variables.

The next unit will cover the different types of variables and how to use them in an imperative mode.

## The three types of variables in Power Apps

In your app, you can use variables. Variables allow you to store information that you need to reference in your app temporarily. Examples where you can use them include to keep a running count or list of information, to dynamically manipulate controls, to optimize performance, or other scenarios where you need to store information temporarily. Variables are a key driver for imperative logic in Power Apps because they allow you to "build the sandwich" piece by piece.

To support you in these needs, Power Apps has three different types of variables.

- **Global variables** – The most traditional type of variable. You use the **Set** function to create and set its value. Then you can reference its values anywhere within your app. A common use is to store a user's DisplayName when the app loads and then reference the variable throughout the app.
- **Context variables** – A context variable is only available on the screen where you create it using the **UpdateContext** function. Context variables are commonly used for functionality that controls a pop-up screen, for example, where you want to use the same variable name on multiple screens but maintain the value separately.
- **Collections** – A collection is a special type of variable for storing a table of data. You can create the collection manually or by loading another data sources table into it. Collections are available throughout your app, like global variables, and they are created using the **Collect** or **ClearCollect** function.

When choosing which type of variable to use, consider where you will use it and the structure of the data you want to store. When in doubt, use a global variable as it has the most flexibility.

## How all of the variable types are the same

With Power Apps, variables are easy to use. You do not have to initialize, declare, or type a variable. You create the variable with the appropriate function, and Power Apps does the rest. When you assign the value to a variable Power Apps will automatically determine the type.

It is also important to note if you are new to variables that variables are temporary and only available to the current user in their current session. When the user closes Power Apps, all of the information stored in variables is no longer available. If you need to store information for use later or by other users, then you will need to write that information to a data source. Variables are temporary by nature.

In the next units, you will explore each variable type in more detail. The next unit covers the global variable.

## Global variables

Global variables are the most commonly used variables because of their flexibility. After you set the variable, you can reference it or update it throughout your app. This allows you to avoid repetitive query for the same information repetitively, to build out the information you need in an imperative way, or sometimes just as a place holder.

## Storing information for your user

A common design pattern in apps is to personalize the app. This can be as simple as saying Welcome and displaying the user's name on each screen of the

app. With Power Apps, you can show a welcome message and get the user's name in a declarative manner by using the following formula in a Label control.

```
"Welcome " & User().FullName
```

This formula displays the string Welcome and then queries Azure Active Directory for the user's DisplayName property and displays it as text. But if you include that function on every screen then each time a screen opens Power Apps has to query that data from Azure AD directly. This creates repetitive calls to the network that slow down your app.

A better approach would be to store that information in a global variable when the app opens, and then reference that variable throughout your app. You could do this by Modifying the **OnStart** property of the app with the following formula.

```
Set(varUserDisplayName, User().FullName)
```

Now for your Label control, you would change the formula to the following.

```
"Welcome " & varUserDisplayName
```

This formula gets you the same output as the previous formula, but instead of having to go back to Azure AD on every screen, Power Apps can reference the value stored in the variable.

## Tracking status in a variable

In a declarative mindset, you might hide or show controls based on a query for data. For example, if you had an app for managing customer's orders, you might have a warning icon that displays only if the customer has more than three outstanding invoices. In addition to the warning, you might have a requirement to get approval from a manager if the customer would like to submit a new order when they have more than three outstanding invoices. This approval workflow will start by the user selecting an approval button.

With a declarative mindset, you would set the Visible property for the warning icon to the following.

```
CountRows(Filter(InvoiceEntity, CustomerNumber = ThisCustomersNumber  
And Status = "Outstanding")) > 3
```

If that is true, then the icon displays, and if it is false the icon does not display. You would then repeat that same formula on the Visible property of the Approval button.

The problem is this becomes a complex formula that you maintain in two different locations, and that query will generate duplicate network traffic, processing in the app, and processing on the data source.

A better approach is only to run the complex call once, store the result in a variable, and then use that variable to control the **Visible** property of each control.

To do this, configure the **OnVisible** property of the screen to set the variable.

```
Set(varOutstandingExceeded, CountRows(Filter(InvoiceEntity, CustomerNumber = ThisCustomersNumber And Status = "Outstanding")) > 3)
```

The variable **varOutstandingExceeded** is either true or false based on the result of the formula. Now set the **Visible** property of the icon and button control to **varOutstandingExceeded**.

No additional formula or functions are necessary. This is because those controls accept either true or false for the **Visible** property and the variable will be either true or false. Based on your **Set** function in the **OnVisible** property of the screen, Power Apps will set the type of variable to Boolean and set the value to true or false based on the result of the formula.

Small changes like this make your app both more performant and easier to maintain. You should incorporate variables anytime you are repetitively retrieving information that is not going to change while you are using it.

## Contextual variables

Contextual variables are similar to global variables except they are only referenced on the screen where you create them. Although it's not possible to set the user's name to a variable to reference throughout your app, there are still advantages to the fact that contextual variables cannot be used on other screens.

Sometimes you have functionality you want to use on multiple screens that is variable driven. For example, many apps use pop-up dialog boxes to confirm things like deleting a record. A common way to implement this is to set a Contextual variable to true when the user selects the delete button. You do that by setting the **OnSelect** property of the button to the following.

```
UpdateContext({varShowPopUp: true})
```

You then set the **Visible** property of the pop-up controls to **varShowPopUp**. This is similar to the example from the global variables. The major difference is reusability. If you copy the controls (using **Ctrl+C**) to an additional screen, then you will have two instances of **varShowPopUp**. These two instances use the same name, but can have different values. The value of **varShowPopUp** on screen1 does not affect the value of **varShowPopUp** on screen2 because each contextual variable, even when they have the same name, are scoped to the screen they are on.

Typically reusing variable names like this is not recommended because it can be confusing, but it's great if you want to reuse functionality independently on different screens.

If you are in doubt about whether you should use global or contextual variables, typically global variables are the default answer. This is because they are available everywhere, making them the most flexible.

One unique behavior of the **UpdateContext** function is that you can declare more than one variable at a time. This is not possible with the **Set** function. To create more than one context variable with a single formula, use a comma between the variables.

```
UpdateContext({varCount: 1, varActive: true, varName: User().FullName})
```

To do the same thing with Global variables, you would use the following.

```
Set(varCount, 1); Set(varActive, true); Set(varName, User().FullName)
```

This is not a reason to sway your choice of variable types but a useful concept to remember. In the next unit, you will learn about storing tables of data in a collection variable.

## Collections

In the previous two units, you learned how global and contextual variables store single values. The third variable type, collections, allow you to store a table of data. This is ideal when you need to store large amounts of structured data for reuse in your app. This data can come directly from a data source, can be created within the app, or a combination of both.

## Using collections to increase performance

The most common reason for using collections is to optimize performance by reducing calls to the same table in a data source. For example, if you have a table that stores all of your active projects and you want to reference that list multiple times in your app, then you should consider querying for that data once and storing it in a collection. To store a copy of the **Projects** table from your data source into a collection called **collectProjects**, use the following formula.

```
Collect(collectProjects, Projects)
```

This will create a collection named **collectProjects** that will have the same rows and columns as the **Projects** table from your data source. Here are a couple of considerations that you need to understand about using collections:

- The Collect function is not delegable. This means by default only the first 500 records from the data source will be retrieved and stored in the collection. For more information about working with delegation, see **Work with data source limits (delegation limits) in a Power Apps canvas app<sup>1</sup>**
- Collections are not linked to the data source after you create them. This means changes to the data in the collection do not automatically save to the data source. If you want to update the data source based on your changes to the collection, you will need to build formulas to do so.
- Collections are temporary. When you close the app, the collection and all of its contents are removed. If you need to store collection data, you need to write it to a data source before closing the app.

## Using dynamic collections

Collections do not have to come from a data source. You can also create a collection from information directly within your app. This is often done to provide values for a drop-down menu or combo box and to store large amounts of data before writing to a data source.

---

<sup>1</sup> <https://docs.microsoft.com/learn/modules/work-with-data-source-limits-powerapps-canvas-app/>

Creating a collection with your own data is very similar to working with the other variable types. The following formula will create a collection named **collectColors** that matches the structure shown in the following table.

```
Collect(collectColors, {Name: "Shane", FavoriteColor: "Orange"},  
        {Name: "Mary", FavoriteColor: "Blue"}, {Name: "Oscar", FavoriteColor:  
        "Yellow"})
```

Name	FavoriteColor
Shane	Orange
Mary	Blue
Oscar	Yellow

After you create the collection, you can then reuse it throughout your app. This also means all of the table functions are available to be used. The one exception where Collections vary from tabular data sources is you cannot use them with the Form control.

For more information about working with collections and the table data they store, see **Author a basic formula that uses tables and records in a Power Apps canvas app**<sup>2</sup>.

Additionally, collections store table data no differently than tabular data sources.

The learning path **Work with data in a Power Apps canvas app**<sup>3</sup> has many concepts that allow you to work with and extend the power of your collections.

In the final unit of this module, you will learn about some additional variable concepts and how to apply them to your apps.

## Additional variable concepts

Now that you have an understanding of Power Apps logic concepts and variable types, there a few additional concepts to expand on that will help you integrate variables into your app.

### Variables can self-reference

This concept applies to both global and context variables. Sometimes you need to make a variable that points to itself. This is often done when you want to either do a counter type variable where it increments a value or you are appending a string. With Power Apps this is easy to implement. Place the following formula on the **OnSelect** property of a button to set up a counter.

```
Set(varCounter, varCounter + 1)
```

The first time that you select the button the value will be 1. If you select the button a second time the value will be 2. Use the following table to see the literal translation.

<sup>2</sup> <https://docs.microsoft.com/learn/modules/author-basic-formula-tables-records-powerapps/>

<sup>3</sup> <https://docs.microsoft.com/learn/pathways/work-with-data-in-a-canvas-app/>

Value of var-Counter before the button press	Button press	Formula	Values	Value of var-Counter after button press
0	First	Set(varCounter, varCounter + 1)	Set(varCounter, 0 + 1)	1
1	Second	Set(varCounter, varCounter + 1)	Set(varCounter, 1 + 1)	2
2	Third	Set(varCounter, varCounter + 1)	Set(varCounter, 2 + 1)	3

When the app first starts, the value of varCounter is 0, and it is incremented by 1 each time the button is selected. It is important to remember that the default value of a variable will vary based on the variable type if you do not set the default property.

- Text variables are ""
- Number variables are 0
- Boolean variables are false

## A variable can store a single record

This concept applies to global and context variables. Collections differ slightly because they are a table made up of one or more records, meaning storing and retrieving a record is different for a collection.

In the previous units, you learned how to store a single value in a global or context variable. You can also store a record in the variable. When you do this, you can then reference the different fields or columns by using the dot (.) notation.

In this example, you will store the entire user record in a global variable named **varUser**. To do so, use the following function.

```
Set(varUser, User())
```

This will store the entire user record in the variable. The user record has 3 columns Email, FullName, and Image. You can retrieve the values of the individual columns using the dot (.) notation. To display the user's email address, add a Label control to the screen and set the text property to:

```
varUser.Email
```

This example stores the record from an action-based data source. You could also use the **LookUp** function as a way to retrieve and store a record from a tabular data source, like Common Data Service, in a variable.

## Variables don't auto update

A common point of confusion for people who are new to variables is that variables don't automatically update. For example, they can use a variable to store the number of customer invoices using OnStart for the app. Then in the app, the user creates a new invoice. The variable doesn't distinguish the number of invoices in the system that have changed. The variable will only update when:

- The user closes the app and then opens it again. This causes OnStart to perform the operation to calculate the number of invoices.

- You implement functionality to update the variable after the user creates an invoice.

Be aware of this common point of confusion if you are new to using variables to track data.

## summary

In this module, you learned about variables in Power Apps and how they enable you to apply both imperative and declarative programming techniques.

Here are some key concepts to remember:

- Power Apps supports both imperative and declarative logic.
- Global variables are for storing information to use throughout your app.
- Context variables are for storing information that is only available on the same screen.
- Collections are a type of variable that stores table data. Most table functions and controls work with collections like all tabular data sources.

See **Performance and optimization tips when building Power Apps<sup>4</sup>** for more information.

See **Tips and tricks to manage your app complexity<sup>5</sup>** for more information.

<sup>4</sup> <https://youtu.be/zrNnQ-PPE2Y?azure-portal=true>

<sup>5</sup> [https://youtu.be/lB06UQ\\_is5k?azure-portal=true](https://youtu.be/lB06UQ_is5k?azure-portal=true)

# Author an advanced formula in a canvas app in Power Apps

## Formulas that process multiple records

Throughout other Power Apps learning paths, you have been focused on using formulas to manipulate single records. That is a great way to get started with Power Apps. As you build more complex apps, it is often necessary to process and work with multiple records in one formula. Throughout this module, the focus will be on learning some of the more common functions that are used to work with tables, records, and collections.

To keep the rest of this module straight forward, the content will refer to tables and records. *Tables* are a data set that is made up of one or more columns with one or more rows of data. A *record* is the name used to refer to the individual rows in the table. Tables can be manually created using the Table function or come from a tabular data source. Collections are a special kind of variable that stores a table. Throughout this module, table and collection are synonymous. This means that any function you can use on a table of data , you can use on a collection.

## Calculations based on multiple records

Sometimes the functions are not about changing the table but performing math operations in relation to the data. The most basic being the **CountRows** function. This simple function is used to count the number of rows (records) in a table. There are also more traditional math functions such as **Sum** or **Average**. These functions process a numerical operation over a table of data.

## Splitting up and combining data

When working with your data, it is often necessary to transform the data to display it or sort it in the proper manner. For example, you cannot display a record in a label. To change those records into a comma-separated list of data you can use the **Concat** function. The function allows you to specify a formula to process over a table of data and the result of each record is separated by a string you specify. The reverse of this is also possible with the **Split** function. You can use the **Split** function to take a string that is separated by a character, like a comma, and turn it into a table of data. Both functions open up the possibilities of working with data the way you want.

## Taking action across a table of data

There might be situations where you want to process your data in ways with no available built-in function. In this instance, you can use the **ForAll** function. ForAll is a dynamic function that allows you to run a formula once for each record in a table while referencing all of the data in the current record. For example, if you wanted to send an individual email to every customer in your table, a ForAll function could run the Office365.SendEmail function once for each customer.

## Tables are flexible

As you work through this module, remember that all of the functions are about working with a table of data. This includes tables that you create with the Table function, tables that come from tabular data sources, collections, and formulas that output tables. Functions like Filter and Search output a table of data and you can use functions with that table of data. For example, you might use Sum(CustomerEntity, InvoiceAmount) to calculate the total amount of invoices in the table. But, you could also filter the data where you only sum the customers where Country equals Germany by modifying the formula to Sum(Filter(CustomerEntity, Country = "Germany"), InvoiceAmount). This concept applies to all of the functions throughout this module.

In the next unit, you will learn how to use math operations across tables.

## Aggregate functions

The aggregate functions are used to provide summary information from a table of data. Information like the average sales price of the standard deviation of scores. Think of it as simple reporting on your data that enables a better app experience.

For example, you could use Max(CustomerOrders, SalePrice) to find the maximum value stored in the SalePrice column of the CustomerOrders table. With this information, you could confirm that the price entered in a Form control did not exceed that price before letting the user submit the form.

Power Apps includes the following aggregate functions:

- **Average** calculates the average, or arithmetic mean, of its arguments.
- **Max** finds the maximum value.
- **Min** finds the minimum value.
- **Sum** calculates the sum of its arguments.
- **StdevP** calculates the standard deviation of its arguments.
- **VarP** calculates the variance of its arguments.

All of these functions support data being passed directly to them, such as:

```
Average(9,10,8)
```

This formula would return the value 9, which is the average of the three input values.

Or by passing a table and expression to them such as:

```
Average(OrdersTable, OrderAmount)
```

This formula would return the average of the OrderAmount column from the OrdersTable. In a later unit, you will learn how to split and combine data with functions.

## Math operation on tables

When working with data in Power Apps, there are many reasons to perform math operations across your data. This math can be counting functions or aggregate functions. Both support tables of data. Not covered in this module are additional math functions and operators for non-table data.

For a complete list of all functions, see **Formula reference for Power Apps<sup>6</sup>**.

### Counting functions

Counting functions are used to compute the number of records in a table of data based on criteria. They are often used to provide visual indicators back to the user.

For example, you could use `CountIf(TasksTable, Complete = "No")` to count the number of records in the TasksTable where the column Complete equals No. Then based on the total, you could conditionally format the Color property of a label or change the Visible property of an icon. These types of visual indicators improve the user experience.

Power Apps includes the following counting functions:

- **Count** counts the number of records that contain a number in a single-column table.
- **CountA** counts the number of records that aren't blank in a single-column table. This function includes empty text ("") in the count.
- **CountIf** counts the number of records in a table that are true for a logical formula. The formula can reference columns of the table.
- **CountRows** counts the number records in a table.

**Count** and **CountA** only support single-column tables, and that is the only argument they accept.

**CountIf** accepts a table and then the logical formula to process. It then returns a count of all of the records that match the logical formula.

**CountRows** accepts only a table as an argument and then returns a count of the number of rows in that table.

### Combine records

When working with data sometimes you need to be able to modify the data. Common scenarios include taking a table of records and combining the records into one string to display, or taking a string and turning it into multiple records in a table. With Power Apps you use the `Concat` and `Split` functions to accomplish this task.

---

<sup>6</sup> <https://docs.microsoft.com/powerapps/maker/canvas-apps/formula-reference>

## Turn table data into a string

The `Concat` function combines the result of a formula applied across all the records of a table, resulting in a single string. Use this function to summarize the strings of a table, just as the `Sum` function does for numbers. This could be used to create a list of comma-separated values to display all of the customers assigned to a sales rep, a semicolon-separated list of email address to pass to an email function, or to dynamically generate an HTML table to be used in the creation of a PDF document with the help of Power Automate.

Fields of the record currently being processed are available within the formula. You simply reference them by name as you would any other value. You can also reference control properties and other values throughout your app.

For example, you could use `Concat(CustomerOrders, Email & ";" )` to create a single string that contains the values of the `Email` column separated by a semicolon. You could use this formula for the `To:` argument in the `Office365.SendEmail` function to send a single email to all of those addresses.

As you begin using the `Concat` function be sure not confuse it with the `Concatenate`<sup>7</sup> function that is used to combine multiple strings into one string.

## Turn a string into a single column table

The `Split` function breaks a text string into a table of substrings. Use the `Split` function to break up comma-delimited lists, dates that use a slash between date parts, to break a word into the individual characters, and in other situations where you need a well-defined delimiter.

A separator string is used to break the text string apart. The separator can be zero, one, or more characters that are matched as a whole in the text string. Using a zero length or blank separator results in each character being broken out individually. The matched separator characters are not returned in the result. If no separator match is found then the entire text string is returned as a single result.

For example, you could use `Split("Canada, Mexico, United States of America", ",")` to create a single column table with three records. The name of the column would be `Result`. This can be useful when retrieving data from a multi-value field and you want to use those values in a drop-down control. In this example, you would set the `Items` property of the drop-down control with the formula.

In the next unit, you will learn how to process a formula once for every record in a table.

<sup>7</sup> <https://docs.microsoft.com/powerapps/maker/canvas-apps/functions/function-concatenate>

## The ForAll function

Previously you have learned to use single-purpose functions to operate against a table. Those are ideal and perform well, but sometimes you need more flexibility or you need to do something not covered by one of those functions. For those situations, there is the **ForAll** function.

The **ForAll** function evaluates a formula for all records of a table.

The formula can calculate a value and/or perform actions, such as modifying data or working with a connection.

Fields of the record currently being processed are available within the formula. You can reference them by name as you would any other value.

You can also reference control properties and other values throughout your app.

For example, you could use ForAll(CustomerOrders, Office365.SendEmail>Email, "Thank you", " You are a great customer " & FirstName)) to send a separate email to all of the email addresses in the CustomerOrders table. The email subject would be "Thank You" and the body would be "You are a great customer" followed by the value stored in the FirstName field.

This example varies from the **Concat** example in two important ways. First, this will send one email for each record in the CustomerOrders table. So, if there are 50 records, 50 different emails will be sent. In the Concat example, only one email was sent with all 50 email addresses in the To: line. Second, this email also references the FirstName field from the CustomerOrders table. When you use the **ForAll** function all of the fields for each record are available to be used.

## Things to know when using ForAll

With **ForAll** there is a tremendous amount of power, but there are also several things to know as you start to build your formula.

- The formula can include functions that take action, such as modifying the records of a data source with the **Patch** and **Collect** functions.
- The formula can call methods on connections.
- You can perform multiple actions per record by using the ; operator.
- You can't modify the table that is the subject of the ForAll function.
- When writing your formula, keep in mind that records can be processed in any order and, when possible, in parallel. You may process the first record of the table after the last record. Be sure to avoid ordering dependencies. For this reason, you can't use the **Set**, **UpdateContext**, **Clear**, and **ClearCollect** functions within a ForAll function because they could easily be used to hold variables that would be susceptible to this effect. You can use **Collect**, but the order in which records are added is undefined.

The **ForAll** function has a lot more rules than most Power Apps functions. Also, because you cannot use functions like Set and UpdateContext sometimes you must find another way to approach your formula. Often the reason you wanted to use **Set** with **ForAll** was to track the number of records that were modified or to capture information about those records. You may find that using Patch to update a collection where you track that same data gets you the same result.

For more information about usage and rules for the ForAll function, see **ForAll function in Power Apps<sup>8</sup>**.

## Summary

In this module, you learned how to use functions that process tables of data. This allows you to do things such as perform mathematical operations, split and combine data, and process almost any function you want on the records. Some key takeaways to consider include:

- Power Apps has functions that process single records, multiple records, and some that can do either.
- Using functions that count or summarize data can provide a better user experience.
- **ForAll** is a powerful function that has a lot more rules and considerations than any other Power Apps function.

In the next module, you will learn more about functions for performing custom updates to your data in Power Apps.

<sup>8</sup> <https://docs.microsoft.com/powerapps/maker/canvas-apps/functions/function-forall>

# Perform custom updates in a Power Apps canvas app

## Sometimes you need something more than forms

When building canvas apps in Power Apps, you go to Galleries to display records from your data source and Forms to view, create, and edit an individual record, but sometimes forms are not enough. In those scenarios, Power Apps has functions for updating your tabular data sources directly.

### Directly create and edit a record

In this module, you will learn about using the **Patch** function to update your data sources without the use of forms directly.

Patch is most often used when you need to take action on the data without user interaction in a repetitive manner, or your app design doesn't allow for the use of forms. For example, if you want to update a logging data source every time a user clicks a button to navigate to another screen you could use the formula for the **OnSelect** property of the button.

```
Patch(LoggingEntity, Defaults(LoggingEntity), {WhoClicked:  
User().FullName, WhenClicked: Now()}); Navigate(NextScreen,  
ScreenTransition.Cover)
```

This formula would create a new record in the data source named **LoggingEntity**. The WhoClicked column sets to the **FullName** property of the user who is signed in, and the WhenClicked column sets to the Date and Time of when they clicked the button. This would open the screen named **NextScreen** using the Cover screen transition.

### Delete a record

There are also functions available for deleting one or more records from your data source. Those functions are:

- **Remove and RemoveIf** - These functions are used to remove or delete records from the data source.
- **Clear** - Use the **Clear** function to remove all of the records from a collection.

For example, if you wanted to give the user the ability to delete a record from a Gallery control, add a Trash icon to the Gallery displaying the data source **CustomerOrders** and then set the **OnSelect** property of the icon to the following.

```
Remove(CustomerOrders, ThisItem)
```

This formula would delete the record for the item that was displaying the Trash icon from the **CustomerOrders** data source. There would be no confirmation, so you might consider implementing a check or pop-up dialog to confirm that the user truly wants to delete the record.

## Bulk changes to records

Patch and Remove are both functions that are used to affect one record. If you need to affect change on more than one record, there are two options:

- Use the **ForAll** function, which was covered in the previous module, to loop through a table of data and run a Patch or Remove function for each record in the table.
- Use the **Collect<sup>9</sup>** function to write from one table to another. Each record of the source table is added as a separate record to the target table.

These topics are covered in other Power Apps learning paths and are not covered in this learning path.

## Collections are data sources

It's important to remember that these functions can use a collection as their target. Patch, Remove, and Removelf can all be used to modify both tabular data sources and collections. As you build more complex apps storing data in collections and working with those items is very common, these functions will be a big part of that manipulation.

The remainder of this module will refer to updating a data source. Remember that a data source can either be a tabular data source or a collection unless stated otherwise.

In the next unit, you will learn how to create and edit records.

## Using the Patch function to create and edit records

The **Patch** function is used to create and edit records in a data source when using a Form control doesn't meet your needs. Patch is used most often when you need to take action on the data without user interaction, in a repetitive manner, or your app design doesn't allow for the use of forms.

### Using Patch to create a record

The **Patch** function can be used to create a new record in your data source. To create a new record, there are three parts to the formula.

1. Include the name of the data source you want to edit. This could be a tabular data source (such as Common Data Service or SharePoint) or a collection. For the example, you will use **CustomerOrders** as the name of the data source.
2. The **Defaults** function returns a record that contains the default values for the data source. If a column within the data source doesn't have a default value, that property won't be present. By

<sup>9</sup> <https://docs.microsoft.com/powerapps/maker/canvas-apps/functions/function-clear-collect-clearcollect>

using Defaults with the data source, this notifies Patch to create a new record.

3. Include the columns that you want to populate in the new record. Here you will specify the name of the column to update followed by the value to write to that column. For this example, you will update the Region and Country column with a string value.

The example formula is as follows:

```
Patch(CustomerOrders, Defaults(CustomerOrders), {Region: "Americas", Country: "Canada"})
```

This formula will create a new record in the **CustomerOrders** data source and will set the Region to Americas and Country to Canada. Notice that you do not define any primary key information (the ID column) that will be updated by the data source according to its settings.

## Using Patch to edit a record

It is also possible to edit a record in the data source. To edit a single record, there are three parts to the formula.

1. Include the name of the data source you want to edit. This could be a tabular data source (such as Common Data Service or SharePoint) or a collection. For the example, you will use **CustomerOrders** as the name of the data source.
2. The record that you want to edit in the data source. The most common way to specify this record is to use the **LookUp** function to retrieve the record from the data source. Another option if you are using a Gallery and you want to update the current record is to use the **ThisItem** function for referencing the record. For this example, you will use a LookUp function.
3. Include the changes that you want to make. Here you will specify the name of the column to update followed by the value to write to that column. For this example, you will update the Region and Country column with a string value.

The example formula is as follows:

```
Patch(CustomerOrders, LookUP(CustomerOrders, ID = 1), {Region: "Asia", Country: "China"})
```

This formula will update the record with an ID of 1 in the **CustomerOrders** table by setting the Region column to Asia and the Country column to China. If there are existing values in those fields, it will be overwritten.

## Updating columns with Patch

The primary logic of most Patch functions is updating the proper columns with the correct information. This will be the source of most of your

troubleshooting of the Patch function. Use the following points to help you work through Patch.

- Make sure you are updating all of the required columns from your data source.
- You can update as many or as few of the optional columns as you would like.
- Make sure your column names are spelled and capitalized correctly. Column names are case-sensitive.
- Make sure you are writing the correct data type. For example, if your column in the data source is a number type, then you cannot write a string value to it, even if that string contains a number.

There are four sources to pass values in your formula to Patch your data source:

- You can hardcode a value. An example is if you want to patch the status of the record with "Pending", your Patch formula would look like:

```
Patch(CustomerOrders, Default(CustomerOrders), {Status: "Pending"})
```

This formula creates a new record and sets the Status column to the string value of "Pending".

- You can reference a variable. For example, you can store the string "Under Review" in a variable named **varStatus** with the following formula.

```
Set(varStatus, "Under Review")
```

Then your Patch formula would be:

```
Patch(CustomerOrders, Default(CustomerOrders), {Status: varStatus})
```

This formula creates a new record and sets the Status column to the string value of "Under Review".

- You can reference the value from the property of a control. An example would be setting the value from a drop-down menu named **Dropdown1** that contained the regions. Your Patch formula would look like:

```
Patch(CustomerOrders, Default(CustomerOrders), {Status: Dropdown1.Selected.Value})
```

This formula creates a new record and sets the Status column to the value of the selected item in the drop-down menu.

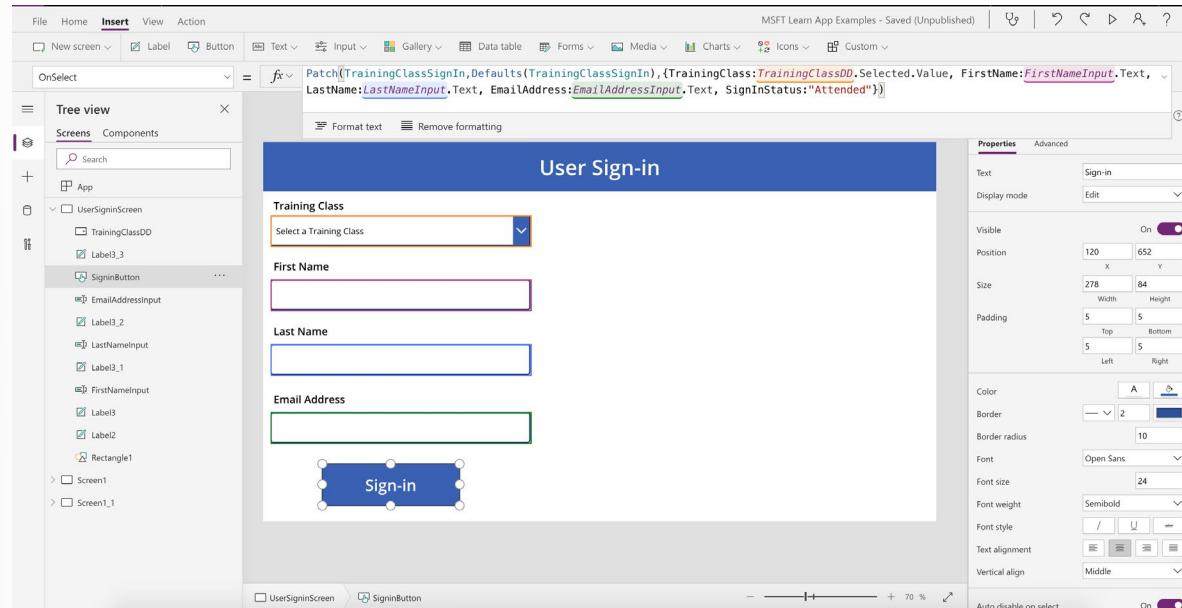
- You can use the output of a formula. An example would be setting the value of the **ModifiedBy** column using the **FullName** from the **User()** function. Your Patch formula would look like:

```
Patch(CustomerOrders, Default(CustomerOrders), {ModifiedBy: User().FullName})
```

This formula creates a new record and sets the ModifiedBy column to the current user's FullName from Azure Active Directory.

## Patch example

Let's take a look at another example, in this example you're trying to build a simple solution for signing users into class as they arrive. This type of Power Apps solution is common and by using the Patch function it can be achieved with only a few steps.



## Solution breakdown

Here we have a simple Canvas app connected to our data source (TrainingClassSignIn). The data source has the following columns, **Training Class**, **FirstName**, **LastName**, **EmailAddress**, **SignInStatus**. This is the information we want to capture when a user selects the **Sign-in** button. In the formula bar, you see the following code:

```
Patch(TrainingClassSignIn, Defaults(TrainingClassSignIn), {TrainingClass:TrainingClassDD.Selected.Value, FirstName:FirstNameInput.Text, LastName:LastNameInput.Text, EmailAddress:EmailAddressInput.Text, SignInStatus:"Attended"})
```

To elaborate, whenever someone selects the **Sign-in** button, a new record will be written to the TrainingClassSignIn data source. As to what data will be written back for the user signing in, you can see it's getting this information from the Controls we added. You will also notice; it is setting the SignInStatus to "Attended" for each new record.

As you can see, Patch is flexible and powerful. Patch has even more capabilities including the ability to update multiple records and merge records. For more information about these scenarios, see

**Patch function in Power Apps<sup>10</sup>**.

<sup>10</sup> <https://docs.microsoft.com/powerapps/maker/canvas-apps/functions/function-patch>

# Deleting record(s) from data sources and collections

One topic that has been not covered in previous content is the concept of deleting a record from a tabular data source or collection. Unlike creating and editing records, which have multiple controls and functions, for deleting records there is only the Remove, Removelf, and Clear functions. This code is often added to the **OnSelect** property of a Button or Icon control.

## Delete a record

To delete a record from your data source use the **Remove** function. Use the **Remove** function to specify the data source and the record that you want to delete. The most common way to specify this record is to use the **LookUp** function to retrieve the record from the data source. Another option is if you are using a Gallery and you want to delete the current record, the **ThisItem** operator is supported for referencing the record.

For example, you could use the following formula to delete a record.

```
Remove(CustomerOrders, LookUp(CustomerOrders, ID = 1))
```

This formula will delete the record where the ID equals 1 from the data source **CustomerOrders**.

## Remove doesn't ask to confirm

Remove does not prompt for any confirmation before deleting the specified record. If you would like to confirm that the user wanted to remove the record, you would need to create confirmation functionality.

## Deleting based on a condition

If you want to delete more than one record from your data source, you can use **Removelf**. The Removelf function allows you to provide a data source to delete from and a condition for selecting the records to delete. This is the same logic that is used by the Filter function.

For example, you could use the following formula to delete all of the records where the Status equals Expired from the **CustomerOrders** data source.

```
RemoveIf(CustomerOrders, Status = "Expired")
```

## Delete all of the records

It is also possible to delete all of the records in a data source. This is most common with collections where you can use the **Clear** function. If you want to delete all of the records from a data source, you can use **Removelf**.

## Delete all of the records in a collection

The **Clear** function deletes all the records of a collection. The columns of the collection will remain. The only input you pass to the function is the collection name.

For example, you could use the following formula to delete all of the records from a collection.

```
Clear(collectSelectedItems)
```

This formula will delete all of the records from the collectSelectedItems collection without changing the columns of the collection.

You will typically see this type of formula when you want to clear out the collection without having to redefine it, like in the case of a reset button or selecting a new order. Note that when working on collections you also have the **ClearCollect** function.

The **ClearCollect** function deletes all the records from a collection and then adds a different set of records to the same collection. With a single function, ClearCollect offers the combination of **Clear** and then **Collect**.

All three functions have their place. One way to think about whether you want to use Clear and Collect versus ClearCollect is when the clearing of the collection happens compared to when you want to add records back. Here are two examples to illustrate this:

- All at once - For example, if you are reloading the items in a collection for a drop-down menu when a screen becomes visible, you would want to use **ClearCollect**. One formula would then remove the old records and add the new records.
- Multi-step - For example, if you are using collections to store user inputs like in a shopping cart you can use **Clear** and **Collect**. This is because the user might want to clear their shopping cart without adding a new record.

## Delete all of the records from a data source

It is possible to delete all of the records from a data source using **RemoveIf**. This is not a very common scenario. Again there will be no confirmation before the formula processes unless you build such functionality. Finally, there is no undo or recycle bin in Power Apps. If you want to recover your data, you would need to go to your data source and use whatever recovery process is available for that data source, outside of Power Apps. Proceed with caution.

For example, you could use the following formula to delete all of the records from a data source.

```
RemoveIf(CustomerOrders, true)
```

This formula will delete all of the records from the **CustomerOrders** data source without changing the columns of the data source.

The reason this works is **Removelf** checks every record in the data source to see if the equation equals true. In this case, the equation has been set to true, so every record will be deleted.

[!NOTE]

Setting the equation portion to **true** also works with the **Filter** function. This can be a valuable setting if you are trying to troubleshoot formulas where you are not sure if Filter is returning data.

## summary

Sometimes it is necessary to work directly with your data source. In this module, you learned how to create, update, and delete data directly from your tabular data source. Here some key things to remember:

- The **Patch** function is the multi-purpose function for creating and editing records directly in your data source when forms don't work for your scenario.
- The **Remove** function allows you to delete a single record from your data source or collection.
- The **Removelf** and **Clear** functions provide the capability to delete multiple or all the records from a given data source.
- When deleting records in Power Apps there is no confirmation, undo, or recycle bin functionality. It is up to you to add that functionality if it is required in your app.

# Complete testing and performance checks in a Power Apps canvas app

## The importance of thinking about performance

The performance of an app is important to keep users happy. Apps can go from mediocre to great just based on performance. And sometimes it can be as simple of a change as caching data in a collection or removing redundant calls to the data source.

In this module, you will learn about common performance problems, ways to mitigate their impact, and how to perform testing to discover the issues.

## The most common performance bottleneck - Data sources

The most common app performance problems come from interactions with data sources. Almost every app has one or more data source. Power Apps natively supports over 200 different connections to these data sources and using these connections is key to a great app.

Calling these data sources from your app is often the largest bottleneck in your app because of the time it takes to call across the network to the data source, process the request on the data source side, return the data to Power Apps across the network, and then for Power Apps to process and display the data. Optimizing these interactions with data sources is key to great performance. The following sections highlight some of the most common mistakes.

### Too many refreshes

Using the Refresh function, you can force Power Apps to update the data it has gathered from a given data source. This seems like a great function to run because you get the freshest data in your app. But, Power Apps will often handle this refresh for you. For example, when you use a Form to submit a new record to a data source displayed in a Gallery control, Power Apps will automatically refresh that connection. If you include a Refresh function when you navigate to the Gallery screen you are now refreshing the data that Power Apps already refreshed. This is redundant and slows your app down for no reason.

[!NOTE]

Do not use the Refresh function until you are certain it is needed.

### Too many lookups

As you start to use relational data (covered in Learning Path Use advanced data options and connectors in Power Apps – Module 1 Work with relational data in a canvas app in Power Apps) a common mistake is not to consider the ramifications of a LookUp function inside of a Gallery. When you place a LookUp function on a Label inside of the Gallery, then that LookUp will be performed once for every record in the Gallery. That means if you have 100 records in the Gallery the app has to perform 100 individual LookUp calls to the data source to render. Depending on the data source this could take minutes to render. A

better approach is only to display the related data using a details screen or to use a Collection to cache the data from the data source, then the LookUp does not have to execute across the network.

[!NOTE]

Be careful when making additional calls to remote data sources if you use controls that display multiple records.

## Storing data in the wrong data source

Different data sources are optimized for different workloads and this should be a consideration when choosing where to store data. One example is storing images or files. A common use of Power Apps is to capture images either using the Camera control or the devices built-in camera app. After the user has taken the image, it needs to be saved. One option is to store the image in the same SQL Server database as the other app data. While possible, it is important to note that SQL Server is very inefficient in storing images. Writing and reading the image file to a SQL database is slow, causing your app to run slow. A better option is to **store Power Apps images in the Azure Blob Storage**<sup>11</sup>.

Azure Blob Storage is much faster than writing the same data to SQL Server. This minor change to the underlying structure of your app can have a useful impact on user satisfaction.

[!NOTE]

Choose the optimal data source for your app to get maximum performance.

## Other performance considerations

While data sources might be the largest bottlenecks, there are other easily overlooked changes you can make to get optimal performance. Some other common issues include:

- **Asset size** - When you are designing your app it is great to include company logos and other visual assets. When you add these assets to your app, make sure the assets are optimized for the size of your app. The higher the resolution of a file, the larger the file size, and the more resources it takes for your app to store and display the image. Use an image editing tool to resize your files to the size you need for your app.
- **Republish your app** - The Power Apps team is constantly updating Power Apps to bring new features and to increase performance. The only way that your app takes advantage of these advancements is for you to open the app and publish again. Your app will stay on the version that it was published on until you do this. So periodically revisiting your app to move to the latest version will provide you with the best possible performance.
- **Build focused apps** - Power Apps supports building apps with as many screens as you can imagine, but too many screens is not a good idea. You should build your apps focused on a specific audience and process. This allows you to optimize the user experience for one audience, makes building and troubleshooting the app easier, and reduces the size of the app. If you have one app for everything, consider breaking it into smaller apps by role.

<sup>11</sup> <https://powerapps.microsoft.com/blog/upload-files-from-powerapps-using-the-azure-blob-storage-connector/>

## Optimize performance in small steps

As you work through this module, you are going to learn about the different techniques and options for optimizing performance. Before you get too deep in optimizing your app, remember that the most important thing is that your app works. A fast-performing app that only throws errors when the user uses it has no value.

It is often easier to build your app so that it accomplishes its goals and is fully functional. After the app works then you can revisit the app for changes you can make to increase performance - making those changes one at a time confirming they don't break functionality. This methodology, of making small changes, will have the highest rate of success. As you become more comfortable with the different performance concepts, you will learn to build the app with them from the start. But in the meantime, build an app that works and then optimize.

## Additional information

To supplement the concepts in this module, there are two additional reading options to help you increase your performance mindset.

- **Performance considerations with Power Apps<sup>12</sup>** - Discusses better ways to load data, patterns, limits, custom APIs, and file optimization.
- **Power Apps Canvas App Coding Standards and Guidelines<sup>13</sup>** - This is a living document that not only covers performance and testing techniques but also explores standards and documentation of your app.

Now that you are aware of the benefits of optimizing performance and some of the common issues to look out for, the remainder of this module will provide you with techniques to increase performance and how to use various methods for testing your app.

## Improve performance with data sources

In the previous unit, you learned that data sources are often the main reason for slow performance in your app. In this unit, you will learn about some of the common techniques you can apply to mitigate those performance issues.

### Use collections to cache data

Often in your app you will find yourself query the same data repeatedly, like in the case of pulling the lists of departments for your drop-down menus. In those instances, you can query for the data once and then reuse that data throughout the app. This reduces the repetitive calls to the data source across the network. The following is an example of this process.

In your app, you have several screens where you provide a drop-down menu for selecting the department. The list of departments is kept in Common Data Service in an entity named **DepartmentList**. On each instance of the menu, you use the following formula.

---

<sup>12</sup> <https://powerapps.microsoft.com/blog/performance-considerations-with-powerapps/>

<sup>13</sup> <https://pahandsonlab.blob.core.windows.net/documents/PowerApps%20canvas%20app%20coding%20standards%20and%20guidelines.pdf>

```
Filter(DepartmentList, Status = "Active")
```

This works and performs okay.

When you finish the app, you realize that you are querying the **DepartmentList** entity multiple times even though it is static information. You could modify your app to create a collection with the **OnStart** property of the app that stores the information using the following formula.

```
Collect(collectDepartmentList, Filter(DepartmentList, Status = "Active"))
```

Now that you have stored that information you could change the **Items** property of the drop-down controls to be **collectDepartmentList** instead of `Filter(DepartmentList, Status = "Active")`. These small changes add up to increase performance in your app. Also, as you become more familiar with the technique, you can build your app this way from the onset which reduces the number of formulas you have to write and maintain.

## Watch out for delegation

One thing to keep in mind is that the Collect function is not delegable. By default, that means only the first 500 items will be returned from your data source and stored in the collection. Be sure to plan for this limitation as you implement the use of collections in your app.

## Delegation also affects performance

When you learned about **delegation**<sup>14</sup>, you focused on returning the right amount of records for your data source. It is also important to remember that delegation, especially for mobile apps, can affect performance.

When a formula delegates to the data source, all of the processing is handled by the data source, and only the matching records are returned across the network to the app to be displayed. If a function is not delegable, then it is very common to change the delegation limit to 2,000 records. This means that the first 2,000 records will be downloaded across the network and then processed locally. In scenarios where you are on a slow cellular connection or a low-end mobile device this processing can take a considerable amount of time, causing a poor experience for the user.

Try to use only delegable functions as much as possible. If your function is not delegable, then plan for the impact on the end user.

## Use the Concurrent function to load multiple data sources

Previously you learned to use collections to cache data in your app. As you implement that functionality in your app it is not uncommon to have

<sup>14</sup> <https://docs.microsoft.com/learn/modules/work-with-data-source-limits-powerapps-canvas-app/>

several collections load when the app launches. Your **OnStart** property might look like the following.

```
Collect(collectDepartmentList, Filter(DepartmentList, Status = "Active")); Collect(collectCompanyList, CompanyList); Collect(collectRegions, RegionList)
```

In that example, you are creating 3 collections, but the collections are processed one at a time. So if each one takes 3 seconds to process then your user has to wait 9 seconds for the app to start.

The Concurrent function allows you to process all of those calls at the same time. You can change your code to the following.

```
Concurrent (  
  
    Collect(collectDepartmentList, Filter(DepartmentList, Status = "Active")),  
  
    Collect(collectCompanyList, CompanyList),  
  
    Collect(collectRegions, RegionList)  
  
)
```

Now all three formulas run at the same time. Reducing your load time to 3 seconds. Concurrent is a great way to avoid long delays from asynchronous calls.

## Preview and experimental features

Within Power Apps there are additional, advanced features you can implement in your app. You can access them by selecting **File** on the menu bar, then choosing **App settings** and **Advanced settings**. The list of options you see are ever changing, but often there are one or more features that relate to performance.

### Preview features

Preview features are features that have been well tested and are close to being released. They will be available for all apps soon. Testing and understanding these features helps you to prepare for when they become standard, and most are enabled by default in new apps.

An example of a current preview feature that helps to increase performance is Delayed load. This feature "Speeds up your app's start time by setting on-demand screen expression calls", which means that screens aren't processed until the user accesses the screen, making the app start and run faster.

## Experimental features

Experimental features are features that might change, break, or disappear at any time. The features are off by default. You can sometimes find performance features here as well, so it is worth taking a look. However, keep in mind you do run a risk by incorporating them into production apps as they can evolve, completely change, or disappear.

An example of a current experimental feature that you might experiment with is "Use longer data cache timeout and background refresh". This feature increases the timeout from 30 seconds to 1 minute for some of the common data sources allowing for more loading time. This might not make for the best user experience but perhaps if you find a way to do this in the background it could let you work with slower queries.

## OnStart versus OnVisible

OnStart and OnVisible are part of your toolkit for building great apps, but from a performance point of view, they can have a major impact.

- **OnStart** - This is an app-level property. Formulas in this property are run once, when the app starts, and never again. All of the formulas must process before the app opens. This is often used to initialize data that you will need throughout the app.
- **OnVisible** - This is a per-screen property. Formulas in this property are run every time a user navigates to the screen. The screen will render before the formula is finished processing.

From a performance perspective, the key consideration is when the code runs. Once, when the app starts, or every time a screen is shown. The collection for departments from earlier in this module is a great example. That code loads in the OnStart property of the app. This means it was loaded once and then always available. If you moved that code from OnStart to OnVisible, you would lose the advantage of the collection. If the formula was in the OnVisible property, each time the user navigated to the screen the data source would be queried. In that case, it would perform just as well to leave the code in the drop down.

This doesn't mean you should not use OnVisible. OnVisible is great for formulas that pertain to the current screen that you need to run. Additionally, OnVisible is non-blocking, so your users don't have to wait on the formula to complete to view the screen, which reduces the amount of time that user needs to view a blank screen.

In most apps, you use a mixture of OnStart and OnVisible to get the optimal experience.

## Summary

As you can see there are a lot of options for how you build your app and interact with data sources. The list in this unit is far from exhaustive. The guidance here is meant to guide you toward better performance, but your results may vary. As you begin applying these techniques to your apps, you will learn what works best for you and your environment.

In the next unit, you will learn about testing and troubleshooting techniques.

## Testing and troubleshooting your app

Now that you have learned about performance bottlenecks and some of the ways to mitigate them this unit will discuss testing techniques. These techniques are a combination of guidance, techniques, and discovery that apply both to performance testing and general debugging.

### Using the Timer control to get metrics

When it comes to working with data connections for retrieving or uploading data, hard numbers are helpful. In Power Apps, you can use a variable and a Timer control to capture how long your formula takes to run. The following scenario will show you how.

Assume you have a Button with the **OnSelect** property set to the following:

```
Collect(collectDepartmentList, Filter(DepartmentList, Status = "Active"))
```

1. Modify that formula to the following to add a variable that you will use to Start and Stop the timer.

```
Reset(Timer1); Set(varTimerStart, true); Refresh(collectDepartmentList);  
ClearCollect(collectDepartmentList, Filter(DepartmentList, Status = "Active")); Set(varTimerStart, false)
```

2. You will see an error about Timer1 because you have not added it to the app yet. You can ignore the error until you complete the next step.
3. To capture exactly how long that formula takes, you will need to first add a **Timer** control to your app and make sure that it is named Timer1. The Timer control is on the **Insert** menu under **Input**.
4. For the **Timer** control, set the **Start** property to the following: varTimerStart.
5. Now insert a **Label** control to display the time.
6. For the **Text** property of the **Label** control set the formula to the following: Timer1.Value
7. Now preview the app and click the **Button**.

In your label, you will see how long your collection takes to create in milliseconds.

This works well for when you want to understand exactly how long a specific query is taking. You could log this data to a different collection and then average the numbers to determine how long it takes. You can also apply this concept to submitting data. Remember to test not only from your local computer but from all of the scenarios for your user's environments.

## Test where your users will use the app

This is more advice than technique. For most app builders the best place to run an app is from the local PC they use to build the app. Testing from that machine generally gives the best case results but may not match your user's experience. Far too often they forget to test the way the user will run the app. For example, if you are going to build a mobile app that runs over a cellular network then make sure your testing includes the same. Understanding the smaller form factor of the mobile device and the latency of a varying internet connection needs to factor into your app's design. The previous timer testing method is great here. Compare the app's query or upload performance between your PC, your mobile phone on Wi-Fi, and your mobile phone on cellular data. Determine if you are satisfied with all three scenarios or if you need to optimize your app for the slower network.

## Use labels to help with testing

As your app incorporates more complex logic and more behind the scenes variables to facilitate that logic consider using label controls as part of your testing toolkit. Simply adding a label to the screen that displays the value of the variable can go a long way to helping you understand why your app is or is not doing something. You can use this during the building and testing phase. When your app is live, you can add additional functionality for hiding and showing these troubleshooting tools.

While you are in the Power Apps Studio you can also select **File** and **Variables** to see all of your variables, their values, where they were created, and where they are used in the app.

Another way to use labels during the build process is to add a label to the welcome screen where you manually display a version number. Power Apps caches your app to optimize your experience. When you are publishing repetitively, like when customizing a SharePoint form, it can be confusing to know which version of the form you are seeing as you may see a cached version. By simply adding a label with v1 or v2 in the corner, you will always be able to check the version.

## Preview versus Published apps

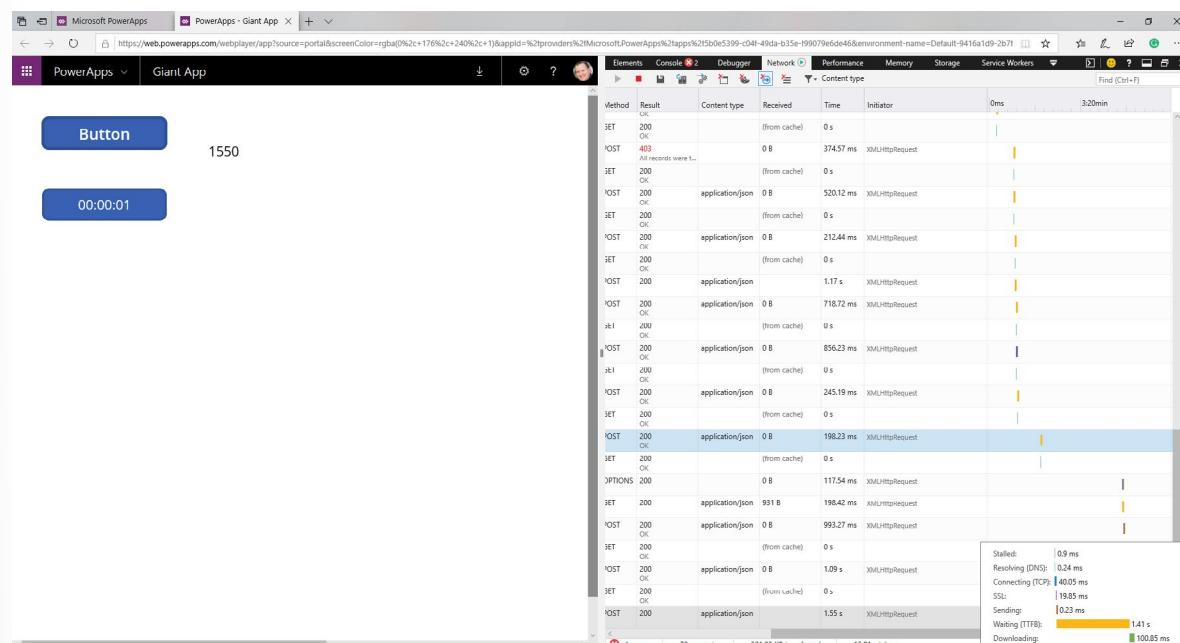
Using the preview capability of the Power Apps Studio gives you great insight into how your app will run when you publish it. But sometimes you may find some inconsistency due to cache or other things your local PC is doing versus what the published app will do in the player. Always remember to test your app after published, in a manner that is consistent with how the user will use the app.

## Looking at the network activity of your app

Now that you have learned about testing from within the app you need to look at actual network calls and performance. To do this you can use your browsers built-in developer tools, Ctrl+Shift+I in Chrome or F12 in Microsoft Edge/IE, or a third-party tool like Fiddler. This will allow you to view the individual

network calls made by your app and view details, such as time that each call takes. From a performance point of view, this can be valuable.

One example of a way you can use this is to determine if performance slowness, as measured by the Timer control from the previous example, is within your app, the network, or the data source. By using the Developer Tools in Microsoft Edge, you can see the breakdown of the query that took 1.55 seconds in Power Apps.



By hovering over the query in the bottom right of the screen, you can see the time breakdown as follows.

Stalled:	0.9 ms
Resolving (DNS):	0.24 ms
Connecting (TCP):	40.05 ms
SSL:	19.85 ms
Sending:	0.23 ms
Waiting (TTFB):	1.41 s
Downloading:	100.85 ms

In this instance, most of the time was spent waiting on the data source to filter the data and respond. This tells you that you cannot make the call faster by changing the app. Instead, you would need to focus on refining the query or speeding up the data source.

This level of information can be overwhelming at first but don't try to learn the whole interface of the developer tools, instead, concentrate on just the pieces you need like the network timelines.

Additionally, you can use the developer tools to get more details about error messages returned in the app or a stronger indication where connectivity issues are occurring.

## summary

App performance is the difference between user adoption or abandonment, and sometimes the difference can be as simple as one change. In this module, you learned about common performance bottlenecks, how to test for them, and some options for mitigating the bottlenecks. Here are some key facts to remember:

- The interaction between the data source and app is the most common cause of poor performance.
- There are many different techniques for mitigating performance issues, such as collections, choosing the correct data source, and the Concurrent function.
- Always perform testing in an environment that is similar to your user's environment. This is especially important when building apps for slower networks.

# Work with relational data in a Power Apps canvas app

## What is relational data?

As you start to build complex apps with Power Apps, one concept that you will need to become familiar with is relating data. Relating data is when you create a connection between two different data sources. An example is a travel expense app where you want to have one record for the trip and then one or more records for the individual expenses like food and lodging.

The following table is an example of storing all of the expense data in one Common Data Service entity.

ID	Destination	Date	Trip reason	Expense type	Expense amount
1	Seattle	4/10/2020	Customer Visit	Hotel	205.75
2	Seattle	4/10/2020	Customer Visit	Dinner	31.33
3	Seattle	4/10/2020	Customer Visit	Flight	450.54
4	Cincinnati	5/2/2020	Training	Cab	23.99
5	Cincinnati	5/2/2020	Training	Lunch	12.44

The example above shows that there is numerous redundant data because each row has all of the information for the entire trip. A better way to store this data would be with parent and child tables, and then create a relationship between the two entities. The next example shows what the two entities would look like.

First, the parent table with only one entry per trip.

ID	Destination	Date	Trip reason
1	Seattle	4/10/2020	Customer Visit
2	Cincinnati	5/2/2020	Training

Then, the child table that has one entry per expense item with a reference to the parent record.

ID	Trip reason	Expense type	Expense amount	TripID
1	Customer Visit	Hotel	205.75	1
2	Customer Visit	Dinner	31.33	1
3	Customer Visit	Flight	450.54	1
4	Training	Cab	23.99	2
5	Training	Lunch	12.44	2

Here you see one entry for each expense. There is also a new column for TripID. This column creates the relationship by specifying the record ID of the parent record. This allows you to query the details of the associated trip.

When building relationships, it is also possible to relate data from two different data sources. An example of this is storing customer information in a CRM system, such as Dynamics 365, and then using that information as part of a SharePoint list for sales regions. With Power Apps, the setup and design of referencing the ID of the customer in

Dynamics 365 from your SharePoint list works the same. This is one of the many benefits of Power Apps, connecting to multiple data sources from within one app is seamless.

[!NOTE]

Common Data Service can define relationships using Lookup columns, creating the structure and connecting the tables for you. This is outside the scope of this primer on relationships but worth noting as you consider data sources.

In the next section, you will learn the functions and formulas for utilizing relationships in Power Apps.

## Working with relationships in Power Apps

As you would expect, Power Apps has all of the functions and controls that you need to build and utilize relationships in your apps. In most apps, there are two primary ways that relationships are used. The first is the scenario discussed in the previous module, storing data in multiple tables and then using relationships to connect the data back together. In this unit, you will learn how to connect a customer and an invoice table.

The second common use of relationships is to query the parent from the child record. For this example, you will learn how to query the customer name when you are directly looking at the invoice record.

### Connecting a parent and child table in Power Apps

In this example, you will walk through how to reference a parent and child relationship using the Power Apps filter and a LookUp function.

The Customer table is the parent in this relationship, meaning one customer can have many invoices. Notice there is no reference in the Customer table to the Invoice table. For this example, the data source name for this table is **CustomerTable**. The table looks like the following.

ID	CustomerName	CustomerPhoneNumber
1	Contoso	513-555-1212
2	Fabrikam	206-555-1313
3	Tailspin Toys	404-555-1414

The Invoice table is the child in this relationship. For this example, the data source name for this table is **InvoiceTable**. The table looks like the following.

ID	InvoiceDate	InvoiceDescription	InvoiceAmount	CustomerID
1	5/16/2020	Parts	412.33	1
2	5/1/2020	Service	205.44	3
3	5/6/2020	Travel	132.98	1
4	5/31/2020	Parts	75.55	2

In Power Apps, use the Gallery control to display the contents of the Customer table. Use the Gallery control with the following steps:

1. Insert a **Gallery** control onto your canvas.
2. Set the **Items** property to **CustomerTable**.

The Gallery will display a list of all of the customer records in the table. This is **Gallery1**.

To display the full contents of the InvoiceTable, use the following steps:

1. Insert a **Gallery** control onto your canvas.
2. Set the **Items** property to **InvoiceTable**.

This Gallery will display a list of all of the invoice records in the table. This is **Gallery2**.

Now to display only the invoices for the customer selected in Gallery1, you need to modify the items property of Gallery2.

1. Set the **Items** property of **Gallery2** to:

```
Filter(InvoiceTable, CustomerID.ID = Gallery1.Selected.ID)
```

The formula will do the following.

Formula Argument	Formula Input	Notes
source	InvoiceTable	
logical_test	CustomerID.ID = Gallery1.Selected.ID	CustomerID is the column name from the InvoiceTable. Gallery1.Selected.The ID is the value of the ID column for the selected record in the gallery.

Now, Gallery2 will only display the invoice records for the selected customer in Gallery1.

## Looking up information stored in the parent from the child

The previous example looked showed how to go from the top down. Sometimes you need to go from the bottom up. For example, in the InvoiceTable, if you are looking at the record for the invoice with an ID of 2, then you know that it is associated with the customer with the ID of 3. This is different if you want to know the customer's name or phone number. To do this in Power Apps, you can use the **LookUp** function.

The **LookUp** function allows you to query a data source for a single record that meets the evaluation criteria. In this example, you will use the same tables as the previous example, but you will start with a blank screen to avoid confusion.

To display the full contents of the InvoiceTable, use the following steps:

1. Insert a **Gallery** control onto your canvas.
2. Set the **Items** property to **InvoiceTable**.
3. Set the **Layout** to Title, Subtitle, and body.

4. In the **Data** pane set **Title** to **InvoiceDate**, **Subtitle** to **InvoiceAmount**, and **Body** to **CustomerID**.

This Gallery will display a list of all of the invoice records in the table. This is **Gallery3**.

5/16/2020	412.33	>
5/1/2020	205.44	>
5/6/2020	132.98	>
5/31/2020	75.55	>

Showing the ID value for each customer does not provide help for the app user. To display the customer name, instead of the ID, do the following.

1. In **Gallery3**, click the label for **Body** and set the **Text** property to:

```
LookUp(CustomerTable, ID = ThisItem.CustomerID, CustomerName)
```

The formula will do the following:

Formula Argument	Formula Input	Notes
source	CustomerTable	
logical_test	ID = ThisItem.CustomerID	The ID is the column name from the CustomerTable. ThisItem. CustomerID is the value of the CustomerID column for the current record in the gallery.
result	CustomerName	This is the column that will be returned for the records that matched the logical_test.

After making that change, Gallery3 is much more user-friendly.

5/16/2020

412.33

Contoso >

---

5/1/2020

205.44

Tailspin Toys >

---

5/6/2020

132.98

Contoso >

---

5/31/2020

75.55

Fabrikam >

## Performance Notes

The previous example is used to demonstrate the concept of looking up from the child to the parent. There can be performance ramifications of doing lookups from within a Gallery. In this example, the data source would be queried four times by the LookUp function, one for each record in the InvoiceTable. If the Invoice table has hundreds of records, then the LookUp would execute hundreds of times. Be sure to consider the performance ramifications before implementing data source calls in a Gallery.

A better option to consider for this scenario is using a collection or other means to query and store all of the customer records and then perform your lookup against the collection. For more information about data performance, see the blog post on **Performance considerations with Power Apps**<sup>15</sup>.

In the next unit, you will learn how Common Data Services removes all of the issues associated with related data by automatically drilling down for you.

## Common Data Service for apps makes relationships even easier

In the previous unit, you learned how to use functions to connect related tables. Functions allow you to build apps upon relational data. With Common Data Service, you define relationships at the entity level. With this in place, Power Apps provides a dot (.) notation for drilling into the related data, which reduces the number of LookUp functions that you need in your app.

The following video shows how to configure a relationship in Common Data Service.

---

<sup>15</sup> <https://powerapps.microsoft.com/blog/performance-considerations-with-powerapps/>



<https://www.microsoft.com/videoplayer/embed/RE2ORdo>

Now that we have a relationship configured in Common Data Service, let's take a look at how to use a relationship in a Power Apps app.



<https://www.microsoft.com/videoplayer/embed/RE4wnzn>

## summary

Using relational data is key to building apps that work well and scale well. In this module, you learned about relational concepts, how Power Apps supports relationships, and how Common Data Service makes working with relational data even easier. These items are important to remember:

- Using relational data with your Power Apps apps allows you to build scalable solutions.
- You can use the Filter and LookUp functions to connect parent and child tables.
- Common Data Service has built in capabilities for creating relationships and allows you to easily use the dot (.) notation to expand on related data.

In the next module, you will learn about data source limits and how Power Apps uses delegation to enhance performance.

# Work with data source limits in a canvas app

## Delegation overview

Before you choose your data source in Power Apps, it's important to understand delegation. By using delegation, Power Apps optimizes its interaction with data sources to minimize the amount of transferred data. Put more simply, Power Apps uses delegation to offload the processing (which includes filtering, searching, and sorting) of data to the data source when available. Delegable processing is dependent on the data source and function being utilized. If you have a lot of data and need to rely on the back-end data source for operations such as filtering, then you might want to consider moving (or replicating) your data into an environment (such as Common Data Service) that works well with delegation. To replicate your data from a different source, you can use the Data Integrator to move data into Common Data Service.

## Delegation in action

There's an example to help you understand delegation. You have a list of 5,000 projects stored in SharePoint. The **ProjectStatus**

column stores the values **Active** or **Inactive**. Half (2,500) of the records are set to each of these values. You could show the list in a gallery and filter it by using this formula.

```
Filter(SharePointList, ProjectStatus = "Active")
```

Because the **Filter** function is delegable to a SharePoint data source, Power Apps would send your formula to SharePoint. SharePoint would process all 5,000 records and return to Power Apps the 2,500 records for which **ProjectStatus** is set **Active**. Those records would all be available in your gallery. In this scenario, Power Apps didn't process any data, and only the matching records were sent from SharePoint to Power Apps, which is efficient.

## When delegation isn't available

Some functions cannot be delegated to some data sources. An example of a non-delegable action is the **Search** function against the SharePoint data source. The **Search** function is similar to **Filter**, but you can use **Search** to check across multiple fields and to find partial matches. For example, `Search(SharePointList, "Rob", "FirstName", "LastName")` would return all of the records where the string "rob" is in either the **FirstName** or the **LastName** column. In this example, Power Apps would return records for Robert Smith, Rob Jones, and Suzy Robinson.

Because the **Search** function isn't delegable to the SharePoint data source, Power Apps has to process the records locally, which means first the records are sent from SharePoint to Power Apps. By default, the data source will only return the first 500 records. It is not the first 500 records that match your formula, but the first 500 records in the data source. In this example, when you add this formula to your gallery, SharePoint returns the first 500 records from the list to Power Apps, and then Power Apps performs the search operation locally. If your list

contained 5,000 items, the other 4,500 records in your data source are not processed or displayed. You can increase the default of 500 records to a maximum of 2,000 records in the advanced settings of Power Apps Studio. Under those circumstances, 3,000 records still would not be processed or displayed.

## Consider delegation when choosing a data source

The reason there is an entire module dedicated to delegation is that it is a key consideration when choosing your data source. You need to consider the types of functions you need, like Search, and the amount of data you will have as you chose the best data source for your application.

In the following unit, you will learn more about how delegation works with various data sources.

## Functions, predicates, and data sources combine to determine delegation

Determining when delegation will and will not happen is a combination of several variables. The first thing to consider is the data source. The following table shows the different functions and whether they support delegation for Common Data Service. In this table, **Yes** means the data source does the processing across all of the records. **No** means the data source returns only the first 500 (default) records to Power Apps, and Power Apps then processes the function locally.

Item	Number [1]	Text [2]	Option Set	DateTime [3]	Guid
Filter	Yes	Yes	Yes	Yes	Yes
Sort	Yes	Yes	No	Yes	-
SortByColumns	Yes	Yes	No	Yes	-
Lookup	Yes	Yes	Yes	Yes	Yes
=, <>	Yes	Yes	Yes	Yes	Yes
<, <=, >, >=	Yes	Yes	No	Yes	-
And/Or/Not	Yes	Yes	Yes	Yes	Yes
StartsWith	-	Yes	-	-	-
IsBlank	Yes [4]	Yes [4]	No [4]	Yes [4]	Yes
Sum, Min, Max, Avg	Yes [5]	-	-	No	-

1. Numbers with arithmetic expressions like `Filter(entity, field + 10 > 100)` aren't delegable. Language and TimeZone aren't delegable.
2. Doesn't support Trim[Ends] or Len. Supports other functions such as Left, Mid, Right, Upper, Lower, Replace, and Substitute.
3. DateTime can be delegated except for DateTime functions Now() and Today().

4. Supports comparisons. For example, Filter(EntityName, MyCol = Blank()).
5. The aggregate functions are limited to a collection of 50,000 records. If needed, use the Filter function to select 50,000 records from a larger set before using the aggregate function.

**Common Data Service**<sup>16</sup> has more info about using the Common Data Service as a data source, and about its delegable functions.

This table is only for supported delegable functions if you use the Common Data Service as a data source. But what if you use a different data source, like SharePoint or SQL?

## Other data sources: SharePoint and SQL

If you use SharePoint or SQL as a data source, review the supported delegable functions and associated documentation for these data sources. As stated at the beginning of this module, the supported delegable functions vary depending on the data source you use. Every data source is slightly different as to what is delegable and what is non-delegable. It's highly recommended to review the different data sources before you begin the app building process. That way, you'll have a better understanding of what's supported and also of any limitations. Every Power Apps solution has specific business requirements. It's important to ensure that the data source supports those requirements with the functions you need for the amount of data you have.

Additionally, if you use the Filter or LookUp function, then you also use a predicate. The predicate is what allows you to evaluate the formula. The function FirstName = "Rob" uses the = predicate. Some data sources don't support certain predicates. For example, Salesforce doesn't support the IsBlank predicate. So although the formula Filter(SalesforceCustomers, Name = "Contoso") is delegable, the formula Filter(SalesforceCustomers, IsBlank(Name)) isn't delegable.

## The Column type can also factor in

The column type can also affect whether delegation is possible. Complex columns, like a SharePoint lookup column, aren't delegable. These columns have deeper logic and are only processable locally by Power Apps. The good news is that if you use a complex column, Power Apps provides visual warning indicators so you can correct the issue. The visual warning indicator is the key to identifying any delegation issues in your Power App. A blue underline anywhere in your formula bar indicates that there's a delegation issue with your formula. It may not return all of your records unless it's corrected. In the next section you'll learn more about delegation warnings, limits, and how to work around them if you ever run into a delegation issue.

## Delegation warnings, limits, and non-delegable functions

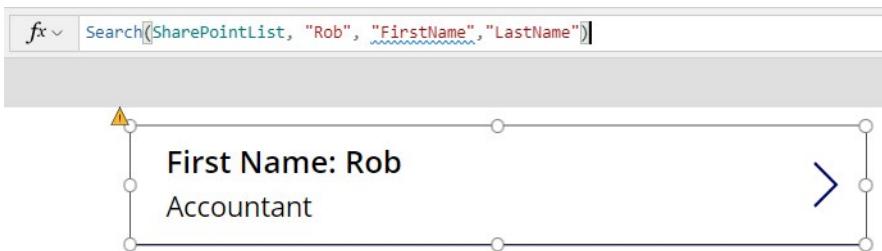
Power Apps uses visuals to help you, the app maker, understand when delegation is occurring. The maker portal also has one setting you can adjust to increase the amount of data returned when delegation is not possible.

---

<sup>16</sup> <https://docs.microsoft.com/connectors/commondataservice/>

## Delegation warnings

Anytime you use a non-delegable function, Power Apps underlines it with a blue line and displays a yellow warning triangle as shown below.



This gives you a clear visual indicator that delegation is not happening, which means you may not be seeing all of your data. It is important to understand a couple of things about this visual indicator.

- Power Apps will provide this warning regardless of the size of your data source. Even if your data source only has a few items and delegation isn't technically causing you a problem (remember the first 500 items are returned by default and processed locally) the warning will still show. The warning appears anytime that your formula is not delegated.
- The warning indicator only processes through the first thing that causes delegation. Notice in the above screenshot that only the underlined field "FirstName" is in blue. That is because it was the first item that caused delegation. "LastName" would also cause delegation in this scenario. This can be confusing because people try to troubleshoot what is the difference between FirstName and LastName instead of the real issue, which is the Search function. If you encounter this confusion, rearrange your formula. This will validate and whichever field is first will show the issue.



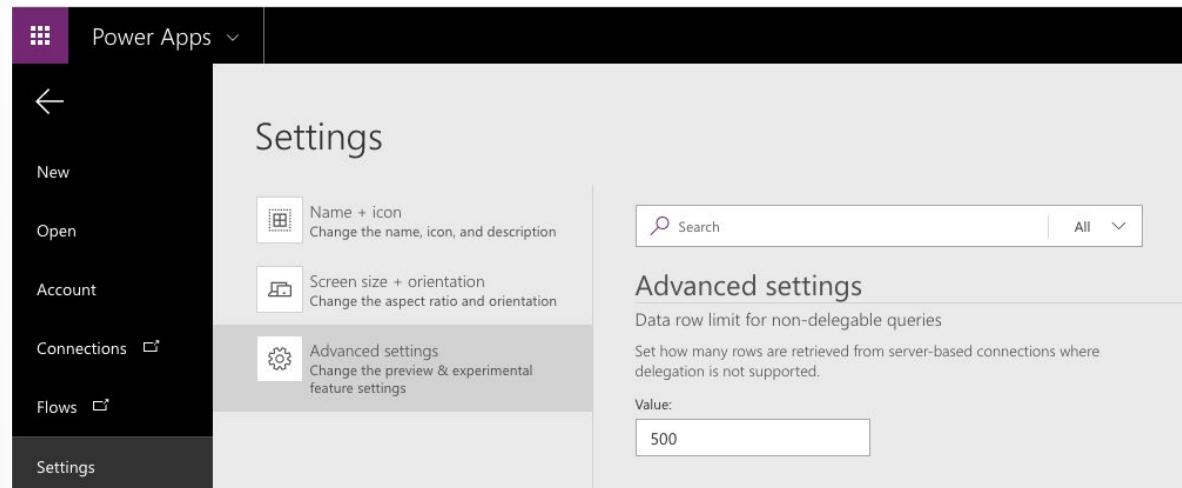
- This visual indicator is only present when you are in the maker portal, building the app. When a user is running the app, they will not receive any notification that delegation is not occurring and that they may only be seeing partial results. Keep this in mind when designing your app and build accordingly.

## Changing the number of records returned when delegation isn't available

When a formula cannot delegate to the data source, for any reason, then by default Power Apps retrieves the first 500 records from that data source and the processes the formula locally. Power Apps does

support adjusting this limit from 1 to 2000. You can adjust this in the Advanced settings.

1. From the Maker portal, select **File** in the upper-left corner.
2. In the left-most menu, select **Settings**.
3. Under **App settings**, select **Advanced settings**
4. Set the Data row limit for non-delegable queries for any value between 1 to 2000.
5. After you have set the limit, select the **arrow** in the upper left to save your change and return to the Maker portal.



There are two primary reasons that you might want to adjust this limit.

- To increase the limit if you are working with data where 500 records aren't enough, but less than 2000 is. For example, if you have a customer list and you know you will never have more than 1000 customers, then you could design your app to ignore delegation and always return all 1000 records.
- To decrease the limit to 1 or 10 to help with testing. If you are running into scenarios where you are not sure if a non-delegable function is causing problems with your app, you can lower the limit and then test. If you set the limit to 1 and your gallery is only presenting one record, you know that you had a non-delegable function. This speeds up your troubleshooting process.

## Non-delegable functions

In the previous unit, you learned about the functions that are delegable and how they relate to the various data sources. These other functions, not covered in that unit, are not delegable. The following are notable functions that do not support delegation.

- First, FirstN, Last, LastN
- Choices
- Concat

- Collect, ClearCollect
- CountIf, Removelf, Updatelf
- GroupBy, Ungroup

All of these functions are not delegable, so by adding them to a formula you might take a previously delegable function and make it not delegable, as seen in the previous example.

## Partially supported delegable functions

The Table shaping functions below are considered to be partially delegable. This means, formulas in their arguments can be delegated. However, the output of these functions is subject to the non-delegation record limit.

- AddColumns
- DropColumns
- ShowColumns
- RenameColumns

A common pattern is to use AddColumns and LookUp to merge information from one table into another, commonly referred to as a Join in database parlance. For example:

```
AddColumns( Products, "Supplier Name", LookUp( Suppliers,  
Suppliers.ID = Product.SupplierID ).Name )
```

Even though Products and Suppliers may be delegable data sources and LookUp is a delegable function, the AddColumns function is partially delegable. The result of the entire formula will be limited to the first portion of the Products data source.

Because the LookUp function and its data source are delegable, a match for Suppliers can be found anywhere in the data source, even if it's large. A potential downside is that LookUp will make separate calls to the data source for each of those first records in Products, causing much chatter on the network. If Suppliers is small enough and doesn't change often, you could cache the data source in your app with a Collect call when the app starts (using OnVisible on the opening screen) and do the LookUp to it instead.

## summary

Building apps require a strong understanding of what your data source can and cannot support. This module covered what is delegation in Power Apps, the variables that determine when delegation is and is not available, and how to adjust the limits of delegation. These items are important to remember:

- Delegation is used to increase performance and reduce the amount of data sent between Power Apps and the data source.
- Delegation is function, predicate, and data source dependent.

- When delegation is not occurring, Power Apps provides warnings in the Power Apps Studio to the maker but does not warn the app user.
- The default amount of records returned by a non-delegable query is 500, it can be increased to a maximum of 2,000.

In the next module, you'll focus on gaining access to additional data sources by building custom connectors.

# Connecting to other data in a Power Apps canvas app

## Overview of the different data sources

In the previous units, you learned about directly referencing data sources from Power Apps. The two types of data sources were Tabular and Action based. As a refresher, they are as follows:

**Tabular data** - A tabular data source is one that returns data in a structured table format. Power Apps can directly read and display these tables through galleries, forms, and other controls. Additionally, if the data source supports it, Power Apps can create, edit, and delete data from these data sources. Examples include Common Data Service, SharePoint, and SQL Server.

**Action-based data** - An action-based data source is one that uses functions to interact with the data source. These functions could be used to return a table of data or to make updates to the data, such as send an email, update permissions, or create a calendar event. Examples include Office 365 Users, Project Online, and Azure Blob Storage.

Both of these data source types are commonly used to bring data and additional functionality to your app. Also, there is one type of special connection with Power Apps, the connection to Power Automate.

**Power Automate** - Is a cloud-based service that makes it practical and simple for line-of-business users to build workflows that automate time-consuming business tasks and processes across applications and services. Flow is a part of Office 365, and if you have a Power Apps license, then you also have a Flow license, meaning you can integrate its functionality to extend your Office 365 apps with no additional cost.

Power Automate can be used to enhance the capabilities of your app using its built-in actions. For example, you can build complex approvals, add business logic, better integrate with external data sources, and create PDFs. Later in this module you will learn more about Power Automate and how to integrate it with Power Apps.

In the next unit, you will learn about action-based connectors by adding the Office 365 Users connector to your app to both retrieve and update information.

## Work with action-based data sources

Action-based data sources differ from the more popular tabular data sources in that you use functions to interact with the data source instead of just reading and writing data. One important difference to note between tabular and action-based data sources is action-based data sources cannot be used with the Forms control. Forms only work with tabular data sources. This doesn't mean action-based data sources can only read data; most action-based data sources provide functions for updating the data as appropriate. An action-based data source can also be for things like sending emails or other notifications, not necessarily only for reading and writing data.

In the following examples you will learn how to add the Office 365 Users connector to your app, query for users, find the logged in user's manager, and finally update the logged in user's profile.

## Add the Office 365 Users data source

In this example, you will add the Office 365 Users data source, an action-based data source, to an app. The process is the same as adding a tabular data source.

1. In the Power Apps Studio, select **View** from the top menu bar.
2. On the menu bar, select **Data sources**.
3. From the pop-out list, select **Add data source**.
4. Select **New connection** in the pop-out list.
5. In the search box, type “Office 365”.
6. The list of options will filter, select **Office 365 Users**.
7. In the pop-out list, you will see an overview of the data source, select **Connect**.

You have now added the Office 365 Users data source to your app. Next time that you want to use the data source in this app or any app, the data source will be available from the **Add data source** screen in step 3 above. You will not have to add a new connection to this data source again.

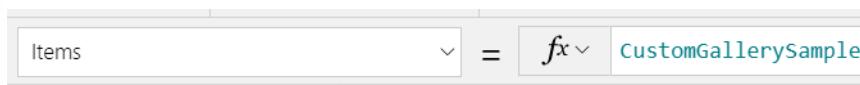
## Display a list of users in a gallery

Now that you have the data source added to your app, you can display a list of the Office 365 users in a gallery.

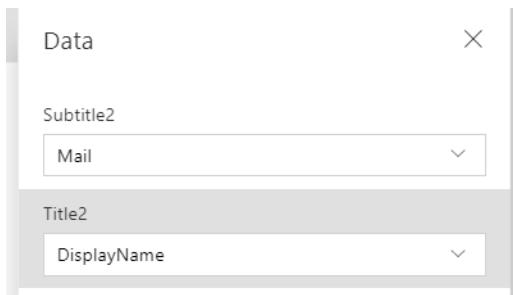
1. On the menu bar, select **Insert**.
2. Select **Gallery**.
3. From the flyout menu, select **Vertical**.
4. In Gallery panel, to the right of Data source, select **None**.

Notice that your Office 365 users data source does not show up. That is because the gallery only shows tabular data sources. You can still use your action-based connecting with a gallery, it just takes a different process.

1. In the formula bar, there's an **Items** property. Delete **CustomGallerySample** from the formula bar.



2. Type in the following for the **Items** property:  
`Office365Users.SearchUser()`
3. In the gallery panel, change the **Layout** to **Title** and **Subtitle**.
4. Set the **Title** label to **DisplayName**
5. Set the **Subtitle** label to **Mail**.



Now your gallery shows all of your Microsoft 365 users' DisplayNames and Mail properties just like you were using a tabular data source. This is because that function of the Office365Users data connection returns tabular data. You could use the output of this data with any function, like SortByColumns or Sum, that accepts a table of data as an input.

## Find the email address of the logged in user's manager

Another common use of the Office 365 data source is to query the user's manager. With tabular data sources, you would use the LookUp function to find this type of information. With this action-based data source, you use a function to directly query the information as shown in the example below.

1. Click an area outside of your gallery from the previous example.
2. Select the **Insert** link on the menu bar.
3. Select **Label**.
4. In the formula bar, delete "Text".
5. In the formula bar, type the following formula:

`Office365Users.ManagerV2(User().Email).mail`

[!NOTE]

If you receive an error after entering this formula then the user does not have a manager assigned in Office 365.

In the label, you will now see the current user's manager email address.

The following is a breakdown of the formula.

Formula argument	Formula input	Notes
id	User().Email	User() is a built-in function that returns information about the current logged in user including FullName, Email, and Image.
.property	.mail	The function returns the whole user record for the manager. To display only the email address in a Label, use the dot (.) notation.

## Update the logged in user's profile info

Another way you might use an action-based connector is to update data. With a tabular data source, you could update directly using a Form or a Patch function. Those capabilities do not work with action-based data sources. Instead, for each action-based data source, you are dependent on the functions provided by that connector for your options. The following example shows how to update your Microsoft User Profile by using the provided function.

1. On the canvas, select the **Insert** menu option.
2. Add a **Button**.
3. For the **OnSelect** property of the button, set the formula to

```
Office365Users.UpdateMyProfile({aboutMe: "Project manager with 5 years of technical project management experience."})
```

That will update your Microsoft 365 Profile. The following is a breakdown of the formula.

Formula argument	Formula input	Notes
Optional	{aboutMe: " Project manager with 5 years of technical project management experience."}	Additional optional parameters: birthday, interests, mySite, pastProjects, schools, skills

Another example would be to update the profile by referencing a **text input** control on the screen. If you had a text input control on the screen named `textInput1`, the formula would update to:

```
Office365Users.UpdateMyProfile ({aboutMe: TextInput1.Text})
```

The Office 365 Users action-based data source is a rich data source and commonly used in many apps. You should take some time to get more familiar with this data source. To learn more about this and all of the available data sources, see [Connectors<sup>17</sup>](#).

These examples demonstrate how to integrate an action-based data source into your app. The concepts can be similar to tabular data sources, like when displaying users in a gallery, but also different, like when writing back to a user's profile.

In the next unit, you will learn more about Power Automate and how to integrate it with your Power Apps apps.

## Power Automate is a companion to Power Apps

Power Automate is a standalone component in the Microsoft 365 ecosystem but also a great Power Apps companion. This is because Power Automate has actions and triggers for interfacing directly with Power Apps. This connection allows you to easily leverage Power Automates functionality in your app.

---

<sup>17</sup> <https://docs.microsoft.com/connectors/>

## Business logic

Power Apps is great in scenarios where you have a direct action you want to be taken after a user performs an activity. An example might be sending an email after a user submits a new expense report. If you need to notify someone the expense report was submitted, then you can do that easily directly from Power Apps.

But what if instead of just notifying someone, you also want to start an approval process. This is a great example of where connecting to Power Automate makes your app stand out. You can have Power Apps trigger a Power Automate flow when the user submits the data. Power Automate can then look up who the user's manager is, and send the manager an approval request. Power Automate will then facilitate getting a response from the manager, updating the data source with the status based on their response, and ultimately sending the original submitter an update.

Approvals are just one example of how you can use Power Automate's native abilities to augment your app's capabilities.

## Data connections

Power Apps offers many of options for connecting to data through the built-in connectors, premium connectors, and custom connectors, however sometimes you need more than that. Some data sources, like custom APIs, can provide data back in a difficult to use format. For example, complex JSON structures are not user-friendly in your app. This is where Power Automate can help. Power Automate has additional actions and expressions natively built in that are better at handling these complex data structures. And more importantly, these actions and expressions can parse through the data and restructure it into a much easier to use object. After the data is parsed, Power Automate can respond to Power Apps with just the friendly data.

Use Power Automate to interact with complex data sources, restructure the objects, and then return to Power Apps only the data that you need. Watch this video to get started with creating a Power Automate flow to connect to Power Apps.



<https://www.microsoft.com/videoplayer/embed/RE2ORdw>

The following video shows how to use Power Automate as a data source to your Power Apps app.



<https://www.microsoft.com/videoplayer/embed/RE4wsDJ>

Connecting Power Apps and Power Automate can bring greater functionality into your apps.

## summary

In this module, you discovered non-tabular ways to bring data and functionality into your app. Action-based data sources can be used to read and write data but also to perform additional functionality like sending notifications or triggering other actions. You also learned how Power Automate, which you already have a license for if you have a Power Apps license, can be incorporated into your app to increase the functionality. With its support for approvals, complex business logic, and working with complex data, Power Automate is the perfect companion to Power Apps. These items are important to remember:

- Power Apps supports both tabular and action-based data sources.
- With action-based data sources, you can use the data sources provided functions to interact with the data source.
- You can easily connect Power Apps to Power Automate, providing your app with additional capabilities. With this connection, you can pass data back and forth between the two platforms.

# Use custom connectors in a Power Apps canvas app

## Overview of custom connectors

If you built your first canvas app in Power Apps, you may be thinking how you can extend your app by calling a publicly available API, or a custom API you're hosting in a cloud provider, such as Azure. To support such tailored scenarios, you can build custom connectors with triggers and actions. Because these connectors are function-based, they will call specific functions in the underlying service of the API to return the corresponding data.

An advantage of building custom connectors is that they can be used in different platforms, such as

**Power Apps**<sup>18</sup>, **Power Automate**<sup>19</sup>, and **Azure Logic Apps**<sup>20</sup>.

## Creating custom connectors

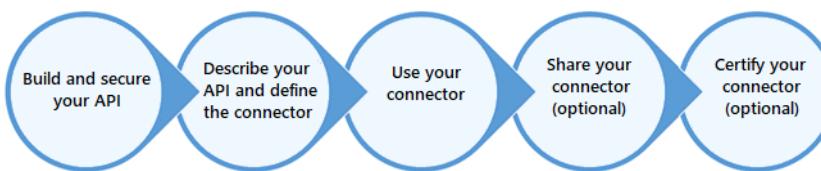
You can create custom connectors using three different approaches:

- **Using a blank custom connector**<sup>21</sup>
- **From an OpenAPI definition**<sup>22</sup>
- **From a Postman collection**<sup>23</sup>

While the requirements for each approach will vary, they all require either a per user/app/month or a per user/month for all users using the app. Each link above points to the instructions for each approach. You'll learn how to create a custom connector from a Postman collection in a later unit.

## Overview of the custom connector lifecycle

Before you start creating custom connectors, you need to know all the details for each of the steps involved in the custom connector lifecycle. The following diagram shows all the tasks involved in creating and using custom connectors.



<sup>18</sup> <https://powerapps.microsoft.com/>

<sup>19</sup> <https://flow.microsoft.com/>

<sup>20</sup> <https://azure.microsoft.com/services/logic-apps>

<sup>21</sup> <https://docs.microsoft.com/connectors/custom-connectors/define-blank>

<sup>22</sup> <https://docs.microsoft.com/connectors/custom-connectors/define-openapi-definition>

<sup>23</sup> <https://docs.microsoft.com/connectors/custom-connectors/define-postman-collection>

## Build and secure your API

A custom connector is a wrapper around a REST API that lets an underlying service communicate with Power Apps, Power Automate, or Azure Logic Apps. Make sure that you have a fully functioning API before you start creating custom connectors.

You can use any language and platform for your API, as long as it's made available as a REST API or SOAP API. Here are a few examples:

- Publicly available APIs like  
[NOAA<sup>24</sup>](#), [US Census API<sup>25</sup>](#), or [EU Open Data Portal<sup>26</sup>](#).
- An API that you create and deploy to any cloud hosting provider, such as Azure, Heroku, or Google Cloud.

A custom line-of-business API that's deployed to your network. You can connect to the API if it's available over the public internet, or you can connect to it through a gateway (currently available in Power Automate and Power Apps)

For Microsoft technologies, we recommend one of these platforms:

- [Azure Functions<sup>27</sup>](#)
- [Azure Web Apps<sup>28</sup>](#)
- [Azure API Apps<sup>29</sup>](#)

To secure your API and connectors, you can use one of these standard authentication methods ([Azure Active Directory<sup>30</sup>](#) is recommended):

- [Generic OAuth 2.0<sup>31</sup>](#)
- OAuth 2.0 for specific services, such as Azure Active Directory, Dropbox, GitHub, and Salesforce
- [Basic Authentication<sup>32</sup>](#)
- [API Key<sup>33</sup>](#)

You can set up Azure AD authentication for your API in the Azure portal so you don't have to implement authentication through code. If needed, you can require and enforce authentication through your API's code. For more information about Azure AD for custom connectors, see

---

<sup>24</sup> <https://www.ncdc.noaa.gov/cdo-web/webservices/v2>

<sup>25</sup> <https://www.census.gov/developers/>

<sup>26</sup> <https://data.europa.eu/euodp/en/developerscorner>

<sup>27</sup> <https://azure.microsoft.com/services/functions/>

<sup>28</sup> <https://azure.microsoft.com/services/app-service/web/>

<sup>29</sup> <https://azure.microsoft.com/services/app-service/api/>

<sup>30</sup> <https://azure.microsoft.com/develop/identity/>

<sup>31</sup> <https://oauth.net/2/>

<sup>32</sup> <https://swagger.io/docs/specification/authentication/basic-authentication/>

<sup>33</sup> <https://swagger.io/docs/specification/authentication/api-keys/>

Secure your API and connector with Azure AD<sup>34</sup>.

## Describe the API and define the custom connector

After you have an API, the next step to consider is how to describe the API's interface and its operations so that Power Apps, Power Automate, and Azure Logic Apps can communicate with the API. After you determine how to describe the API, you can create the connector, which then registers it with the appropriate services.

Use one of the following approaches to describe your API:

- An OpenAPI definition (formerly known as a Swagger file). For more information, see **Create a custom connector from an OpenAPI<sup>35</sup>** and **What is Swagger<sup>36</sup>**.
- A Postman collection. For more information, see **Create a Postman collection for a custom connector<sup>37</sup>**, **Create a custom connector from a Postman collection<sup>38</sup>**, and **Postman documentation<sup>39</sup>**.
- You can also start with a blank custom connection using the custom connector wizard in Power Apps and Power Automate. For more information, see **Create a custom connector from scratch<sup>40</sup>** for additional information.

OpenAPI definitions and Postman collections use a different format, but both are language-independent, machine-readable documents that describe your API's operations and parameters. You can generate these documents from various tools, based on the language and platform used by your API. Behind the scenes, Power Apps, Power Automate, and Azure Logic Apps use OpenAPI to define connectors.

## Use the custom connector

You can use your custom connector the same way that you use any built-in connectors. You create the connection to the API and call any operations provided by the API the same way that you call operations for built-in connectors.

Connectors created in Power Apps or Power Automate can be used in both services, but connectors created in Azure Logic Apps cannot be used directly in other services. However, a connector can be easily recreated using the same OpenAPI definition or Postman collection used to create the Azure Logic Apps connector.

<sup>34</sup> <https://docs.microsoft.com/connectors/custom-connectors/azure-active-directory-authentication>

<sup>35</sup> <https://docs.microsoft.com/connectors/custom-connectors/define-openapi-definition>

<sup>36</sup> <http://swagger.io/getting-started/>

<sup>37</sup> <https://docs.microsoft.com/connectors/custom-connectors/create-postman-collection>

<sup>38</sup> <https://docs.microsoft.com/connectors/custom-connectors/define-postman-collection>

<sup>39</sup> <https://www.getpostman.com/docs/>

<sup>40</sup> <https://docs.microsoft.com/connectors/custom-connectors/define-blank>

For more information about using custom connectors from Power Apps, Power Automate, and Azure Logic Apps:

- **Use a custom connector from a Power Apps app<sup>41</sup>**
- **Use a custom connector from a flow<sup>42</sup>**
- **Use a custom connector from a logic app<sup>43</sup>**

## Share the custom connector

Connectors can be shared with other users in your organization the same way you share resources in Power Apps, Power Automate, and Azure Logic Apps. For more information, see **Share a custom connector in your organization<sup>44</sup>**.

## Certify the custom connector

If you want to share the connector with all users in Power Apps, Power Automate, and Azure Logic Apps, you can submit the connector for Microsoft certification. During this process, Microsoft reviews the connector, checks for technical and content compliance, and validates functionality for Power Apps, Power Automate, and Azure Logic Apps. For more information, see **Submit your connector for Microsoft certification<sup>45</sup>**.

## Use postman for your custom connector

Now that you understand what custom connectors are and how their lifecycle works, you can learn how to build custom connectors using a Postman collection.

Postman is an app for making HTTP requests. Postman collections help you organize and group related API requests. Collections can make custom connector development faster and easier if you don't have an OpenAPI definition for your API. For more information about collections, see **Creating collections<sup>46</sup>**.

## Create a Postman collection for a custom connector

There are some prerequisites needed to create a Postman collection for a custom connector:

- **Postman<sup>47</sup> app**

---

<sup>41</sup> <https://docs.microsoft.com/connectors/custom-connectors/use-custom-connector-powerapps>

<sup>42</sup> <https://docs.microsoft.com/connectors/custom-connectors/use-custom-connector-flow>

<sup>43</sup> <https://docs.microsoft.com/connectors/custom-connectors/use-custom-connector-logic-apps>

<sup>44</sup> <https://docs.microsoft.com/connectors/custom-connectors/share>

<sup>45</sup> <https://docs.microsoft.com/connectors/custom-connectors/submit-certification>

<sup>46</sup> [https://www.getpostman.com/docs/postman/collections/creating\\_collections](https://www.getpostman.com/docs/postman/collections/creating_collections)

<sup>47</sup> <https://www.getpostman.com/>

- **API key<sup>48</sup>**  
for the Cognitive Services Text Analytics API

## Create an HTTP request for the API

1. In Postman, on the **Builder** tab, select the HTTP method, enter the request URL for the API endpoint, and select an authorization protocol.

The screenshot shows the Postman application interface. The 'Builder' tab is active. In the top bar, 'POST' is selected under 'Method' and the URL 'https://westus.api.cognitive.microsoft.com/text/analytics/v2.0/sentiment' is entered. Under the 'Authorization' section, a dropdown menu is open with 'No Auth' selected. The status bar at the bottom right indicates 'This request does not use any authorization.'

Parameter	Value
HTTP method	"POST"
Request URL	https://westus.api.cognitive.microsoft.com/text/analytics/v2.0/sentiment
Authorization	"No Auth" (you specify an API key in the next step)

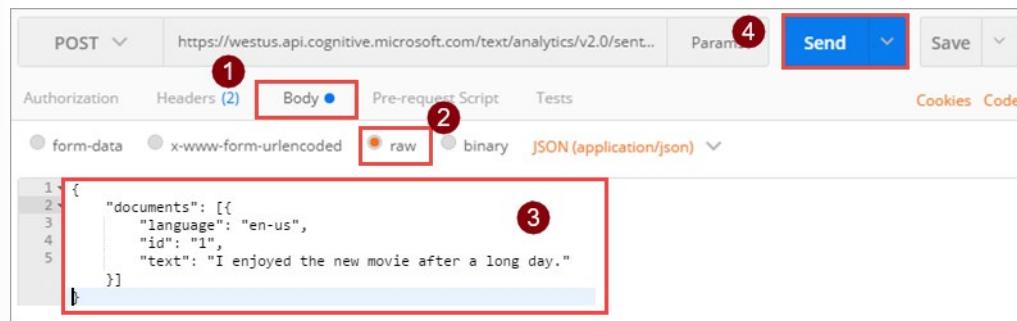
2. Enter key-value pairs for the request header. For common HTTP headers, you can select from the drop-down list.

The screenshot shows the 'Headers' section of the Postman interface. A red circle labeled '1' is over the 'POST' method. A red box labeled '2' highlights the 'Headers' tab. Below it, two header entries are listed: 'Ocp-Apim-Subscription-Key' with value '<your-API-subscription-key>' and 'Content-Type' with value 'application/json'. Both entries have checkboxes checked.

Parameter	Value
Headers	Key: "Ocp-Apim-Subscription-Key" Value: your-API-subscription-key, which you can find in your Cognitive Services account Key: "Content-Type" Value: "application/json"

3. Enter content that you want to send in the request body. To check that the request works by getting a response back, select **Send**.

**48** <https://docs.microsoft.com/connectors/custom-connectors/index#get-an-api-key>



```
{  
  "documents": [  
    {  
      "language": "en-us",  
      "id": "1",  
      "text": "I enjoyed the new movie after a long day."  
    }  
  ]  
}
```

4. The response field contains the full response from the API, including the result or an error, if any occurred.

The screenshot shows the Postman interface with the response tab selected. The status bar indicates `Status: 200 OK Time: 99 ms Size: 444 B`. The response body (highlighted with a red box) is:

```
{  
  "documents": [  
    {  
      "language": "en-us",  
      "id": "1",  
      "text": "I enjoyed the new movie after a long day."  
    }  
  ]  
}
```

Below the response, the status bar shows `Status: 200 OK Time: 99 ms Size: 444 B`.

For more information about HTTP requests, see [Requests](#)<sup>49</sup>.

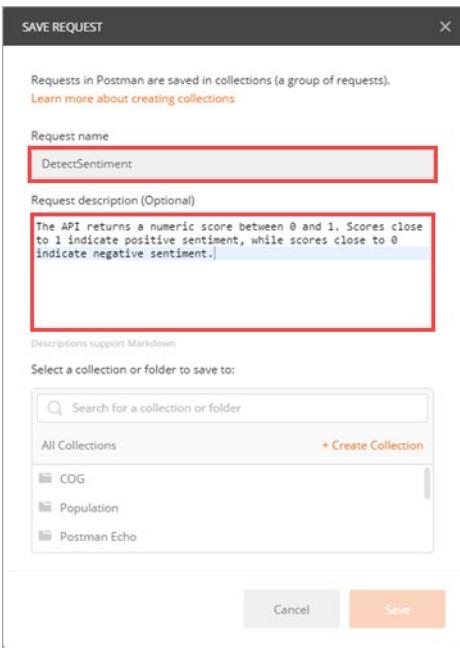
## Save the collection

1. Select **Save**.

<sup>49</sup> [https://www.getpostman.com/docs/postman/sending\\_api\\_requests/requests](https://www.getpostman.com/docs/postman/sending_api_requests/requests)

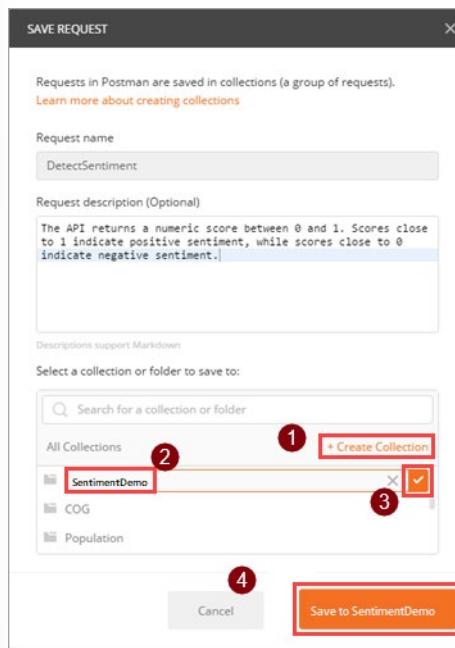


- Under **Save Request**, enter a request name and a description. The custom connector uses these values for the API operation summary and description.



Parameter	Value
Request name	"DetectSentiment"
Request description	"The API returns a numeric score between 0 and 1. Scores close to 1 indicate positive sentiment, while scores close to 0 indicate negative sentiment."

- Choose **+ Create Collection** and provide a collection name. The custom connector uses this value when you call the API. Select the check mark, which creates a collection folder, then choose **Save to SentimentDemo**.



Parameter	Value
Collection name	"SentimentDemo"

## Save the request response

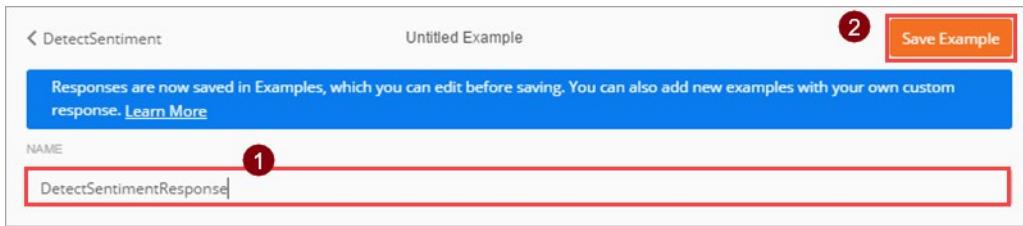
Now that you've saved your request, you can save the response. That way, the response appears as an example when you load the request later.

1. Above the response window, select **Save Response**.

```
1
2
3
4
5
6
7
8
9
{
  "documents": [
    {
      "score": 0.99358940124511719,
      "id": "1"
    }
  ],
  "errors": []
}
```

Custom connectors support only one response per request. If you save multiple responses per request, only the first one is used.

2. At the top of the app, provide a name for your example response, and select **Save Example**.

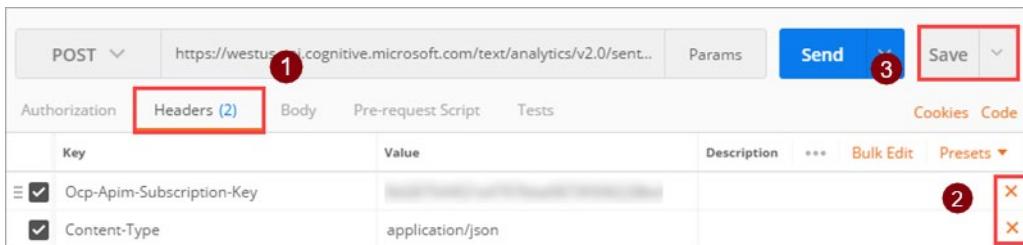


If your API had additional features, you would continue to build your Postman collection with any additional requests and responses.

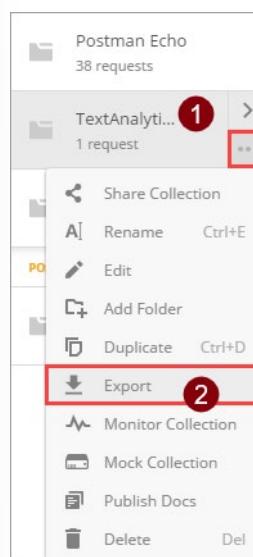
## Export the Postman collection

You can now export the collection as a JSON file, which you will then import using the custom connector wizard. However, before you export the collection, remove the content type and security headers - these were required to make API requests, but they are handled differently in the custom connector.

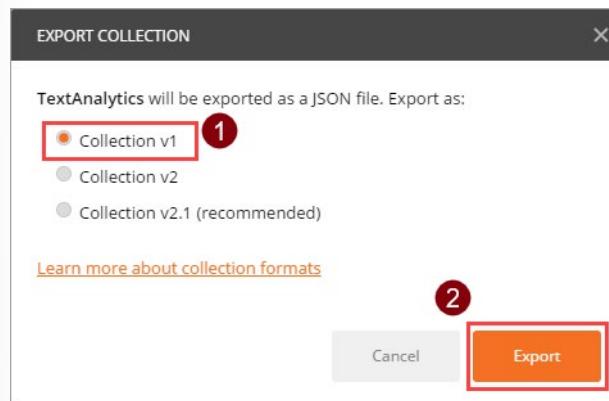
- Under **Headers**, hover over each header, and choose the **X** next to the header to remove it. Select **Save** to save the collection again.



- Select the ellipsis (...) next to the collection, and then choose **Export**.



3. Select the **Collection v1** export format, select **Export**, then browse to the location where you want to save the JSON file.



## Create a custom connector from the Postman collection

You created the Postman collection and now want to use it to create a custom connector. Before proceeding, there are some prerequisites needed to create the custom connector from the Postman collection created earlier:

- Download the Postman collection you created
- An **API key**<sup>50</sup> for the Cognitive Services Text Analytics API
- A **Power Apps**<sup>51</sup> subscription

<sup>50</sup> <https://docs.microsoft.com/connectors/custom-connectors/index#get-an-api-key>

<sup>51</sup> <https://docs.microsoft.com/powerapps/signup-for-powerapps>

## Import the Postman collection

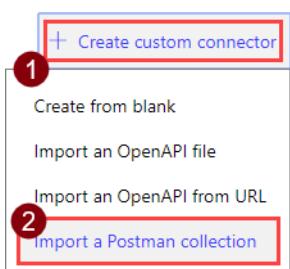
Now you're ready to work with the Postman collection you created earlier. The collection contains most of the required information. You can review it and update it as you go through the custom connector wizard. You need to import the Postman collection for Power Apps and Power Automate.

## Import the Postman collection for Power Apps and Power Automate

1. Go to [powerapps.com](https://powerapps.microsoft.com/)<sup>52</sup> or [flow.microsoft.com](https://flow.microsoft.com/)<sup>53</sup> and sign in.
2. In the upper-right corner, select the gear icon and then select **Custom connectors**.



3. Choose **Create custom connector**, then select **Import a Postman collection**.



4. Enter a name for the custom connector, then navigate to the Postman collection that you created, and select **Continue**.



Parameter	Value
Custom connector title	"SentimentDemo"

The wizard imports the collection then converts it to an OpenAPI definition named generatedApiDefinition.swagger.json.

<sup>52</sup> <https://powerapps.microsoft.com/>

<sup>53</sup> <https://flow.microsoft.com/>

## Update general details

This example shows the Power Automate user interface, but the steps are similar same across all three technologies. However, any differences will be pointed out.

1. On the **General** page, review the imported information from the Postman collection, including the API host and the base URL for the API. The connector uses the API host and the base URL to determine how to call the API.

The screenshot shows the 'General information' configuration page. It includes fields for an icon (a green globe with a magnifying glass), an icon background color (#007ee5), a description ('SentimentDemo'), and connection settings (Scheme: HTTPS, Host: westus.api.cognitive.microsoft.com, Base URL: /). There is also a checkbox for connecting via an on-premises data gateway.

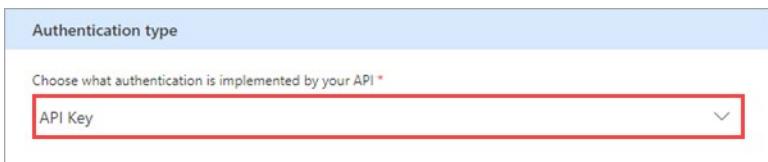
2. Update the description to something meaningful. This description displays in the custom connector's details, and it can help others decide if the connector would be useful to them.

Parameter	Value
Description	"Uses the Cognitive Services Text Analytics Sentiment API to determine whether the text is positive or negative."

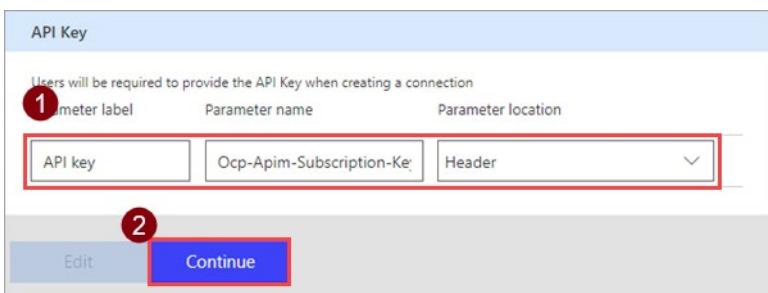
## Specify the authentication type

There are several options available for authentication in custom connectors. For this tutorial, the Cognitive Services APIs use API key authentication.

1. On the **Security** page, under **Authentication type**, choose **API Key**.



2. Under **API Key**, specify a parameter label, name, and location. Specify a meaningful label, because this is displayed when someone first makes a connection with the custom connector. The parameter name and location must match what the API expects (in this case, the header you specified in Postman). Select **Connect**.



Parameter	Value
Parameter label	"API key"
Parameter name	"Ocp-Apim-Subscription-Key"
Parameter location	"Header"

3. At the top of the wizard, make sure the name is set to "SentimentDemo", then select **Create connector**.

## Review and update the connector definition

The custom connector wizard gives you many options for defining how your connector functions, and how it is exposed in Power Apps, Power Automate, and Azure Logic Apps. The next section explains the user interface (UI) and covers a few options, but you are also encouraged to explore this on your own.

## Review the UI and definition

Before you learn about the specific steps on the **Definition** page, here's a review of the user interface.

- This area displays any actions, triggers (for Logic Apps and Power Automate), and references that are defined for the connector. In this case, the DetectSentiment action from the Postman collection is displayed. There are no triggers in this connector, but you can learn about triggers for custom connectors in **Use webhooks with Azure Logic Apps and Power Automate**<sup>54</sup>.

<sup>54</sup> <https://docs.microsoft.com/connectors/custom-connectors/create-webhook-trigger>

The screenshot shows the Logic App designer interface. It includes three main sections: 'Actions (1)', 'Triggers (0)', and 'References (0)'. The 'Actions' section contains one action named 'DetectSentiment...' with a 'New action' button. The 'Triggers' section has a 'New trigger' button. The 'References' section notes that references are reusable parameters used by both actions and triggers.

2. The **General** area displays information about the action or trigger currently selected. This information comes from the Postman collection. You can edit the information here, including the **Visibility** property for operations and parameters in a logic app or flow:
  - **none** - Typically displayed in the logic app or flow.
  - **advanced** - Hidden under an additional menu.
  - **internal** - Hidden from the user.
  - **important** - Always shown to the user first.

The screenshot shows the 'General' tab in the Logic App designer. It displays the following fields:

- \* Summary: DetectSentiment
- \* Description: The API returns a numeric score between 0 and 1. Scores close to 1 indicate positive sentiment.
- \* Operation ID: DetectSentiment
- Visibility: A radio button group where 'none' is selected, while 'advanced', 'internal', and 'important' are unselected.

3. The **Request** area displays information based on the HTTP request included in the Postman collection. In this case, the HTTP verb is **POST**, and the URL is "/text/analytics/v2.0/sentiment" (the full URL to the API is "<https://westus.api.cognitive.microsoft.com/text/analytics/v2.0/sentiment>").

The Request area in Postman. It includes fields for Verb (POST), URL (/text/analytics/v2.0/sentiment), Path, Headers, and Body.

4. The **Response** area displays information based on the HTTP response included in the Postman collection. In this case, the only response defined is for "200" (a successful response), but you can define additional responses.

The Response area in Postman, showing a single response entry for status code 200.

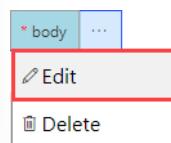
5. The **Validation** area displays any issues detected in the API definition. Make sure to check this area before you save a connector.

The Validation area in Postman, showing a green checkmark and the message "Validation succeeded."

## Update the definition

Next you will change a few things so the connector is more user-friendly when someone uses it in Power Apps, Power Automate, or Azure Logic Apps.

1. In the **General** area, update the summary to "Returns a numeric score representing the sentiment detected".
2. In the **Request** area, select **body** then **Edit**.



3. In the **Parameter** area, you now see the three parameters that the API expects: ID, language, and text. Select **id** then **Edit**.



4. In the **Schema Property** area, update values for the parameter, then select **Back**.

Parameter	Value
Title	"ID"
Description	"An identifier for each document that you submit."
Default value	"1"
Is required	"Yes"

5. In the **Parameter** area, select **language** and then select **Edit**. Repeat the process you used for ID with the following values.

Parameter	Value
Title	"Language"
Description	"The 2 or 4 character language code for the text."
Default value	"en"
Is required	"Yes"

- In the **Parameter** area, choose **text** then **Edit**, and repeat the process you used for ID and language, with the following values.

Parameter	Value
Title	"Text"
Description	"The text to analyze for sentiment."
Default value	None
Is required	"Yes"

- In the **Parameter** area, choose **Back** to take you back to the main definition page.
- At the top right of the wizard, select **Update connector**.

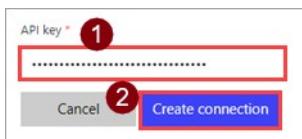
## Test the connector

Now that you've created the connector test it to make sure it works properly.

- On the **Test** page, choose **New connection**.

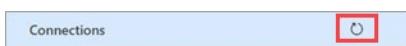


- Enter the API key from the Text Analytics API, then select **Create connection**.



- Return to the **Test** page.

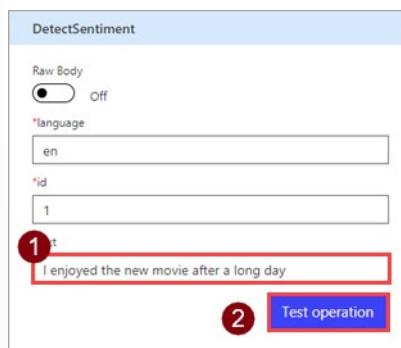
In Power Automate, go back to the **Test** page. Choose the refresh icon to make sure that the connection information is updated.



In Power Apps, go back to the list of connections available in the current environment. In the upper-right corner, select the gear icon, then select **Custom connectors**. Choose the connector that you created, then go back to the **Test** page.



- On the **Test** page, enter a value for the **text** field (the other fields use the defaults that you set earlier), and then select **Test operation**.



5. The connector calls the API. You can review the response, which includes the sentiment score.

The screenshot shows the Postman interface with the 'Response' tab selected. It displays the API response. The 'Status' is 'OK (200)'. The 'Headers' section shows a JSON object with 'content-type': 'application/json; charset=utf-8'. The 'Body' section shows a JSON response with a 'documents' array containing one item. The item has a 'score' of 0.9935894012451172, an 'id' of '1', and an empty 'errors' array. A large red box highlights the entire 'Body' section.

```
[{"documents": [{"score": 0.9935894012451172, "id": "1"}], "errors": []}]
```

## summary

Custom connectors are an excellent way of extending apps because they can access resources outside of the readily available connectors for Power Apps. This module covered what custom connectors are and how to use them across different platforms, such as Power Apps, Power Automate, and Azure Logic Apps. This module also covered the custom connectors lifecycle, and how to create one using Postman. Here are some points to remember:

- Custom connectors provide a way to extend apps by calling APIs, services, and systems for scenarios tailored toward your business needs.
- After you create a custom connector, it can be shared with other users inside the organization. To share it outside of the organization, submit it to Microsoft for certification.
- Using Postman is one of the most common approaches to create custom connectors.

## Module 4 Automate a business process using Power Automate

### Get started with Power Automate

#### Introducing Power Automate

Welcome to Power Automate! In this module, you'll learn how to build flows.

If you're a beginner with Power Automate, this module will get you going. If you already have some experience, this module will tie concepts together and fill in the gaps.

#### Learning objectives

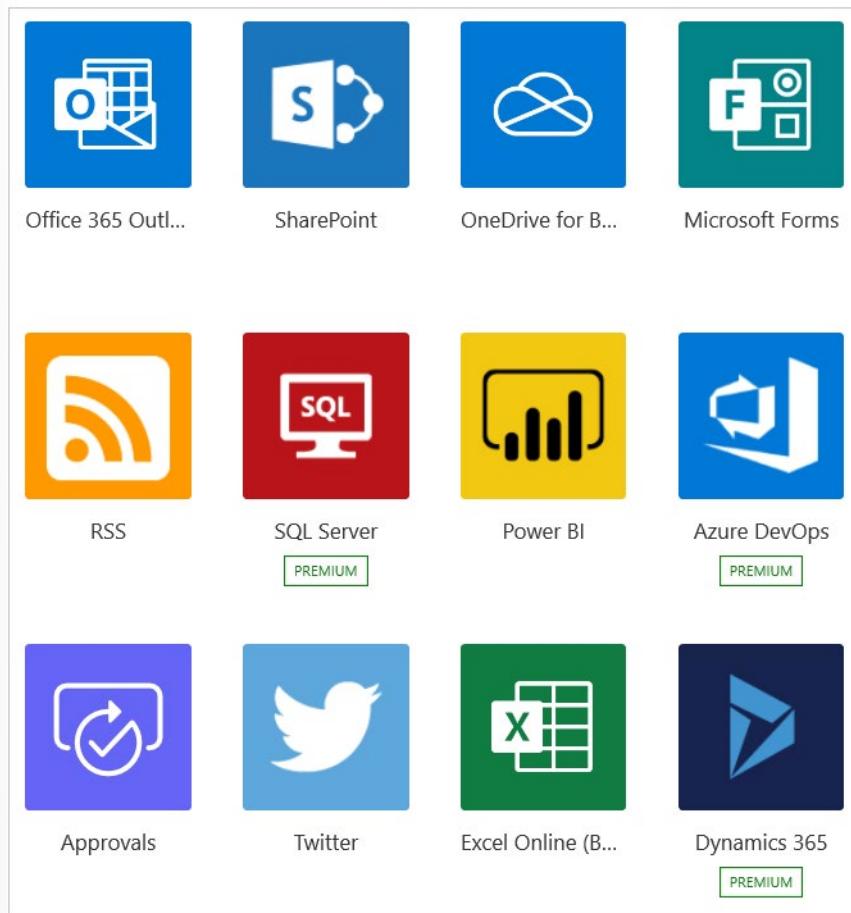
In this module, you will:

- Learn what Power Automate is and how it can be used
- Create a flow that automatically saves email attachments
- Learn how to create a button flow to send yourself a reminder
- Create a flow that sends you notifications
- Create a flow that copies files
- Create a flow that runs on a schedule
- Create a flow that posts tweets
- Create a flow that your team can use

#### What is Power Automate?

Power Automate is an online workflow service that automates actions across the most common apps and services. For example, you can create a flow that adds a lead to Microsoft Dynamics 365 and a record in MailChimp whenever someone with more than 100 followers tweets about your company.

When you sign up, you can connect to more than 220 services, and can manage data either in the cloud or in on-premises sources like SharePoint and Microsoft SQL Server. The list of applications you can use with Power Automate grows constantly.



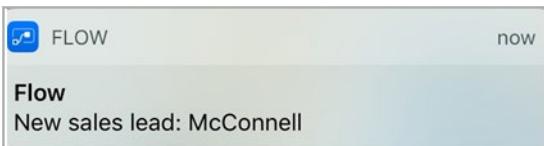
## What can you do with Power Automate?

You can use Power Automate to automate workflows between your favorite applications and services, sync files, get notifications, collect data, and much more.

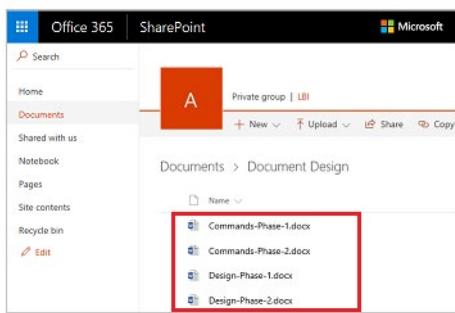
For example, you can automate these tasks:

- Instantly respond to high-priority notifications or emails.
- Capture, track, and follow up with new sales leads.
- Copy all email attachments to your OneDrive for Business account.
- Collect data about your business, and share that information with your team.
- Automate approval workflows.

A common use of Power Automate is to receive notifications. For example, you can instantly receive an email or a push notification on your phone whenever a sales lead is added to Dynamics 365 or Salesforce.



You can also use Power Automate to copy files. For example, you can ensure that any file that's added to Dropbox is automatically copied to SharePoint, where your team can find it.



You can monitor what people are saying about your business by creating a flow that runs whenever someone sends a tweet with a certain hashtag. The flow can add details about each tweet to a Facebook post, a SQL Server database, a SharePoint list, or even a Microsoft Excel file that's hosted on OneDrive for Business--whichever service works for you.

You can create actions to connect the data you collect to Microsoft Power BI, spot trends in that data, and ask questions about it.

The following example shows a flow that saves tweets with the hashtag #MicrosoftFlow to an Excel file.



	Sensitivity:	General	Tweet Text	Tweeted by	Retweet count	Created at	PowerAppsid
A2			#microsoftflow blogpost Don't miss these community events at the Data Insights Summit! https://t.co/				
31			#Microsoft is once again out of stock on all Lumia handsets. Read more: https://t.co/y9amIyBTJ \$MSFT Translated using #MicrosoftFlow	VillamizarITPro	0	2017-06-06T06:01:08. xEJlaFaugEM	
32			#MicrosoftFlow arrives on #Windows10 #Mobile #windows #Office365 #o365 https://t.co/FIevlQz209 RT @MicrosoftFlow: We're pleased to announce that the #MicrosoftFlow app is now publicly available for #WindowsPhone devices! Read: https://...	enterinit	0	2017-06-06T06:20:12. 2GhfaBo2rJw	
33			adi_regev		9	2017-06-06T06:35:59. G-A-AOKr9_w	

Also, you can automate approval loops for things like vacation requests on a SharePoint list.

Vacation Requestors	Start Date	End Date	Status
Amy Phillips	5/15/2017	5/19/2017 12:00 AM	Approved
Charles Fergus	8/28/2017	9/4/2017 12:00 AM	Denied
George Adams	6/12/2017	6/19/2017 12:00 AM	Approved
Grace Thompson	6/26/2017	6/28/2017 12:00 AM	Approved
Shawn Delany	6/14/2017	6/21/2017 12:00 AM	Denied
Soledad Verdi	7/03/2017	7/10/2017 12:00 AM	Approved

For more ideas, browse our list of templates. Templates help you build flows by making a few configuration changes. For example, you can use templates to easily build flows to send yourself weather forecasts, reminders at regular intervals, or phone notifications whenever your manager sends you mail.

	<b>Send myself a reminder in 10 minutes</b>	By Microsoft	Instant	303832
	<b>Get today's weather forecast for my current location</b>	By Microsoft	Instant	216639
	<b>Get a push notification when you receive an email from your boss</b>	By Microsoft	Automated	181614
	<b>Send a customized email when a new SharePoint list item is added</b>	By Microsoft	Automated	144443

Have an idea for a flow that you don't see in the list? Create your own from scratch and, if you want, share it with the community!

## Where can I create and administer a flow?

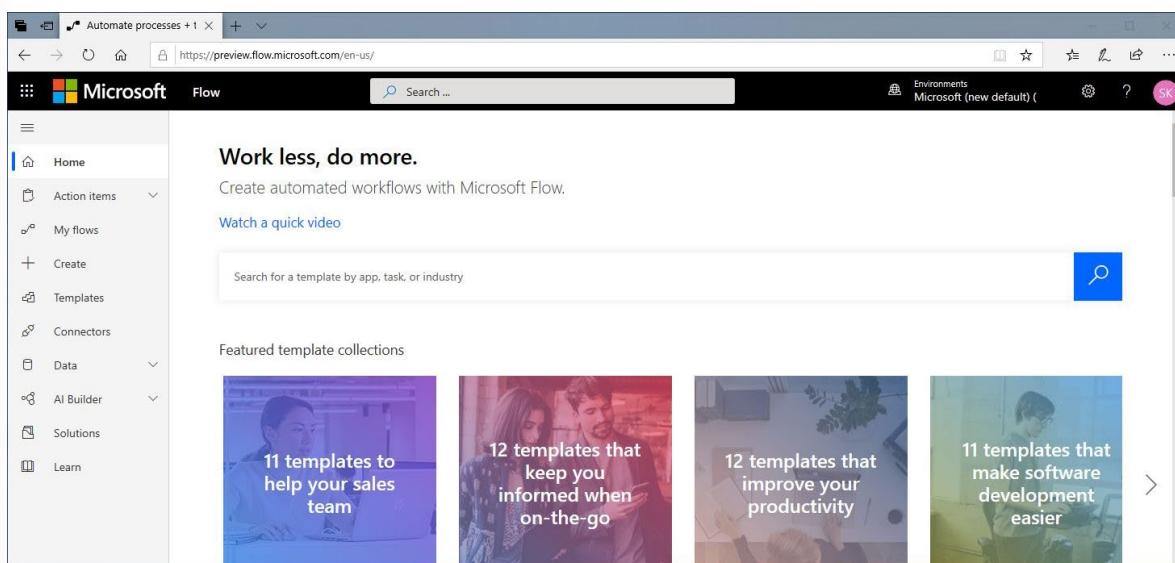
You can create a flow and perform administrative tasks in a browser or, if you download the Power Automate mobile app, on your phone.

Here are some of the tasks you can perform with the mobile app:

- Turn flows on or off from wherever you are.
- See when a flow has failed.
- Review detailed run history reports.
- View and filter runs by notification type.

## A brief tour of Power Automate

Let's jump into Power Automate, and we'll show you around. We have tons of information for you to learn about how to use Power Automate.



When you sign in to Power Automate, you'll find these menus:

- **Action items**, where you can manage approvals and business process flows.
- **My flows**, where your flows reside.
- **Create**, where you start a new flow.
- **Templates**, where you can take a look at some of the most popular templates. These should give you some great ideas for flows you want to try.
- **Connectors**, where you can connect from one service to another.
- **Data**, where you can access entities, connections, custom connectors and gateways.
- **Solutions**, where you can manage your solutions.
- **Learn**, where you can find information that will help you quickly ramp up on Power Automate.

For now, let's focus on the ? menu next to your login, which has these options:

- **Documentation** is where our advanced topics reside. If you want to really understand a feature or function, you can do a deep dive here to figure things out.
- **Learn** has learning paths to guide you through using Power Automate, all the way from beginning techniques to advanced scenarios.
- **Support** is a great landing place to find help.
- **Roadmap** is where you can get a glimpse into what will be made in the next product update.
- **Community** is a place to plug into and find out how other people use Power Automate.
- **Give Feedback** taps into a community of power users, and is where you can send comments and questions to developers and other experienced users.
- **Blog** keeps you up to date about the most recent developments and releases in the Power Automate ecosystem.
- **Pricing** can help you choose the right plan for you or your business.

## What's next?

Now that you have a taste of what Power Automate is and what it can do, let's take a look at what makes a flow.

## Exercise - Create your first flow

In this unit, you'll see more of Power Automate as you build your first flow.

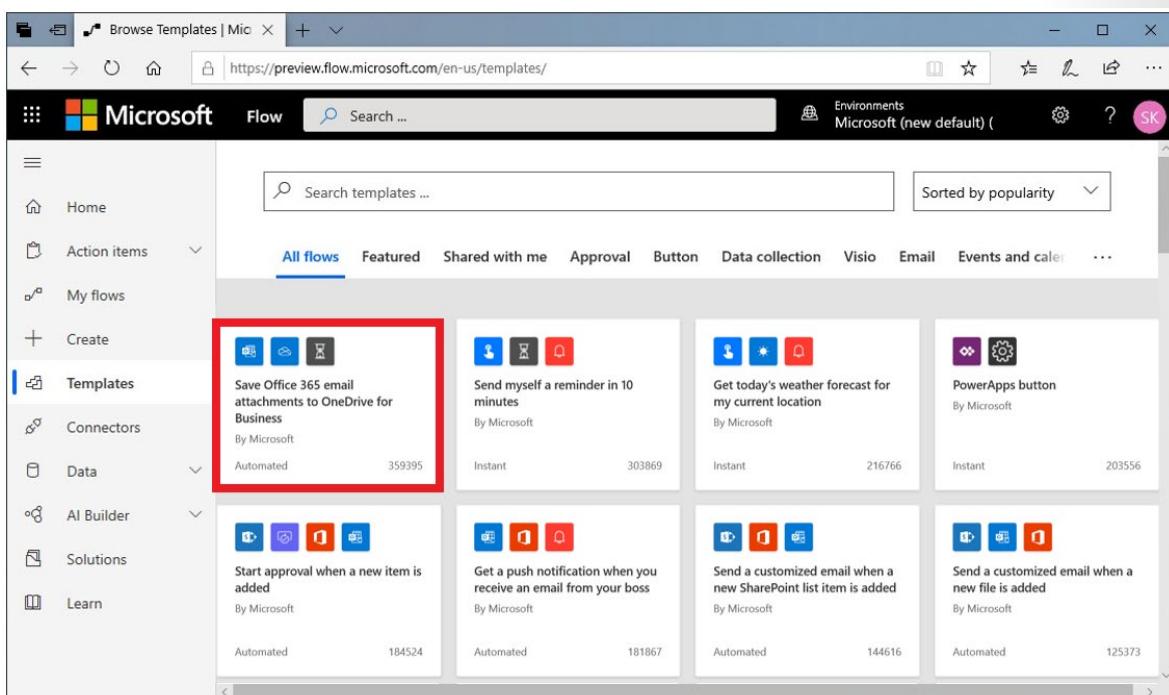
It can be time consuming to search for attachments through email. The flow that you'll build saves time by storing all your email attachments in a folder on your Microsoft OneDrive for Business account.

## Choose a template

Our many templates will get your flows flowing quickly. They'll help you connect the services you're already using in more meaningful ways.

Sign in to **Power Automate**<sup>1</sup>, and select the **Templates** menu. You can sign up for Power Automate with a Microsoft account.

Select the **Save Office 365 email attachments to OneDrive for Business** template.



## Create the flow

**Save Office 365 email attachments to OneDrive for Business** is one of our one-click templates, in which you can answer questions that are necessary to build the flow, so that you don't have to write a line of code.

On the template graphic, there's a description of what the template does and what it needs to succeed.

<sup>1</sup> <https://ms.flow.microsoft.com>

You'll be asked to provide credentials for the Microsoft Office 365 Outlook and Microsoft OneDrive for Business services. If you regularly use both services, you'll already be signed in.

1. Click on the template and select **Create Flow**.
2. On the next page, Power Automate creates the flow for you.
  - It will connect to your work email to get any attachments.
  - It will then create a folder on your OneDrive for Business account to automatically put every attachment that's sent to your work email address in that folder.

The screenshot shows the 'Save Office 365 email attachments to OneDrive for Business' flow details page. At the top, there are standard navigation and action buttons: Edit, Share, Save As, Delete, Send a copy, Submit as template, Export, Analytics, Turn off, and Repair tips off. Below the header, the title 'Flows > Save Office 365 email attachments to OneDrive for Business' is displayed. The main content area is divided into two sections: 'Details' and 'Connections'. The 'Details' section contains information such as the flow name ('Save Office 365 email attachments to OneDrive for Business'), status ('On'), creation date ('Oct 15, 09:39 AM'), modification date ('Oct 15, 09:39 AM'), type ('Automated'), and plan ('Per-user plan'). It also lists the owner ('Sari'). The 'Connections' section shows two connected services: 'Office 365 Outlook' and 'OneDrive for Business'. Both connections have a checkmark indicating they are active. Below these sections is an 'Owners' section showing 'Sari' as the owner. At the bottom left, there is a link to the 'Original template'. The 'Runs' section indicates that no runs have occurred yet.

3. Select the **My flows** menu.
4. Select the flow you just created and click **Edit** to see how it works.

The screenshot shows the 'Flows' page. At the top, there is a toolbar with buttons for New, Edit (highlighted with a red arrow), Share, Save As, and a search bar. Below the toolbar, the word 'Flows' is centered. Underneath, there are three tabs: 'My flows' (underlined), 'Team flows', and 'Business process flows'. The 'My flows' tab is selected. A table below lists the flows, showing columns for Name, Modified, and Type. The first flow listed is 'Save Office 365 email attachments to OneDrive for ...', which was created 20 minutes ago and is of the 'Autom...' type. The flow icon includes a checkmark and a blue square with a white icon.

5. Send an email with an attachment, or have another user send an email with an attachment. You then should see a green check mark, which indicates that the flow succeeded.
6. Select **Edit** to see how the flow is defined.
7. Select **Succeeded** to see the run history and the results.

The screenshot shows the 'Save Office 365 email attachments to OneDrive for Business' flow details page. It includes sections for 'Details', 'Connections', 'Owners', and 'Runs'. The 'Runs' section shows four successful executions.

Start	Duration	Status
Oct 16, 01:29 PM (1 d ago)	00:00:03	Succeeded
Oct 15, 12:41 PM (2 d ago)	00:00:02	Succeeded
Oct 15, 12:10 PM (2 d ago)	00:00:02	Succeeded
Oct 15, 10:30 AM (2 d ago)	00:00:01	Succeeded

In this case, all parts of the flow were successful.

The screenshot shows the Microsoft Power Automate flow editor with a successful run history. The flow consists of three main steps: 'On new email' trigger, 'Apply to each Attachment on the email' action, and 'Create file' action. All steps are marked as completed successfully.

## Important concepts in Power Automate

Keep these concepts in mind when building flows:

- Every flow has two main parts: a *trigger*, and one or more *actions*.
- You can think of the trigger as the starting action for the flow. The trigger can be something like a new email arriving in your inbox or a new item being added to a SharePoint list.
- Actions are what you want to happen when a trigger is invoked. For example, the new email trigger will start the action of creating a new file on OneDrive for Business. Other examples of actions include sending an email, posting a tweet, and starting an approval.

These concepts will come into play later, when you build your own flows from scratch. In the next unit, we'll look at the Power Automate mobile app and its capabilities.

## Exercise - Learn to use the Power Automate mobile app

Of course we have an app – the Power Automate mobile app! From this app, you can access these features:

- Activity Feed
- Browsing
- Buttons
- Managing Flows

First, you'll need to download and install the Power Automate mobile app from your app store.

After it's installed, start it and sign in.

When you first start the app, you'll see the Activity Feed. The Activity Feed is the place to see what's happening with your flows. It won't be the full experience you'd expect from your PC, but it will show you useful details.

For example, you'll see a flow's last activity. You can see whether the flow succeeded or failed. If it failed, you'll see which step it failed on.

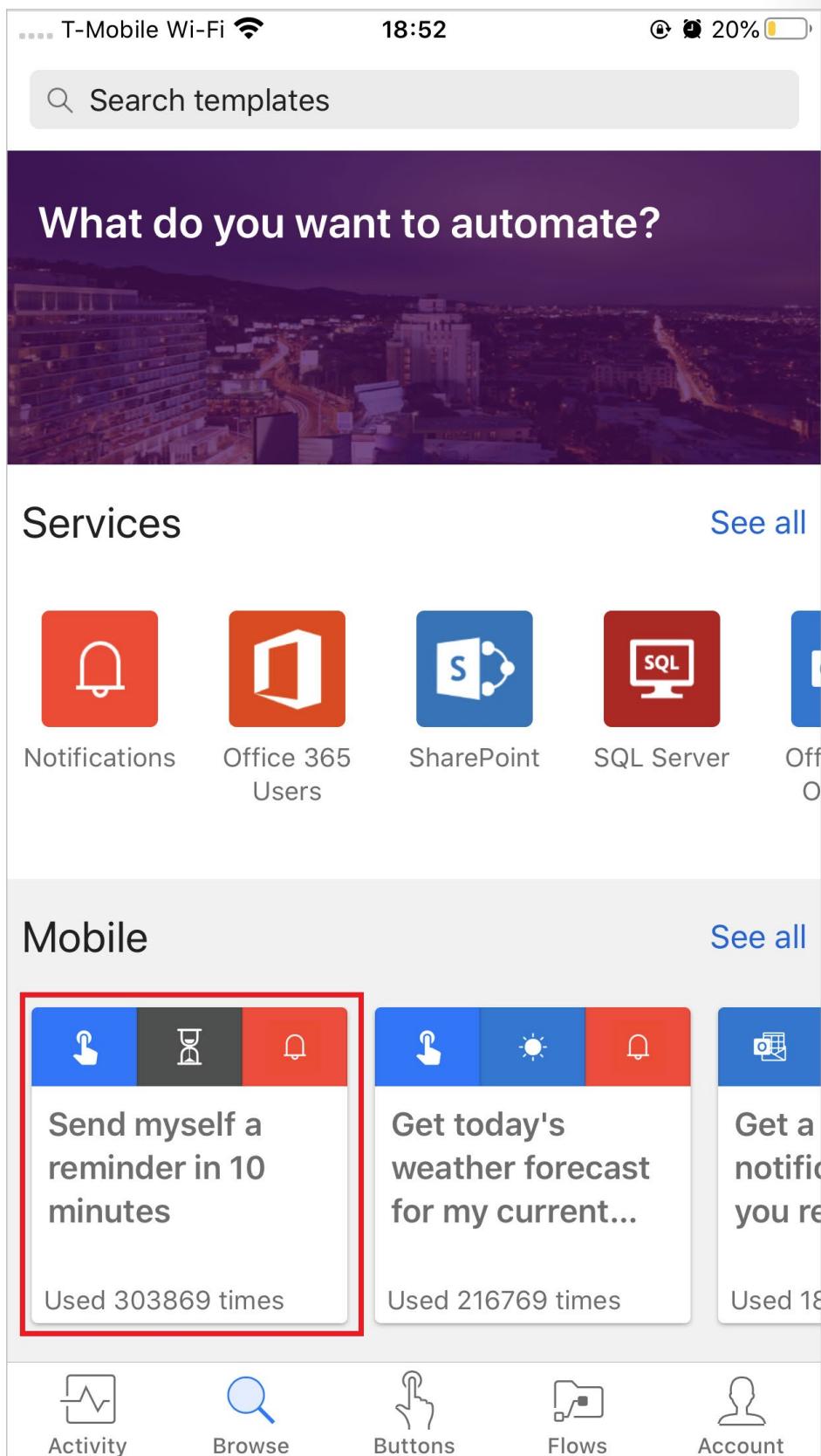
## How button flows are started

Buttons are flows that are started through a manual action. For example, you can create a button to send a *Working from home today* email to your manager. If you live far from your workplace, you can then use this button on days when the traffic is a mess!

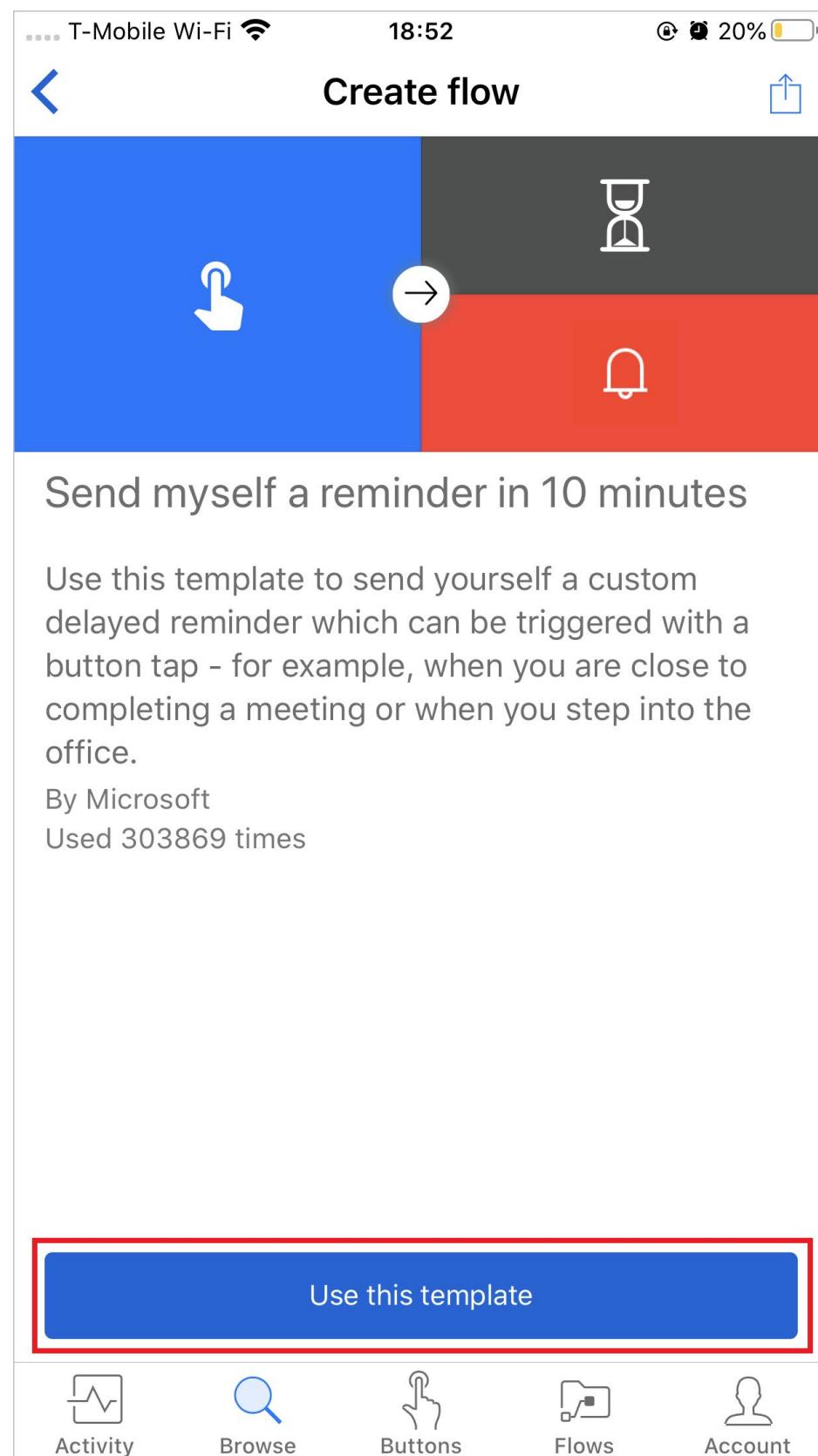
- Select **Buttons** to use some of these flows.
- Select **Browse** to check out templates for more button flows that you can add to your collection.

To show you how you can use buttons, we'll use the **Send myself a reminder in 10 minutes button** template.

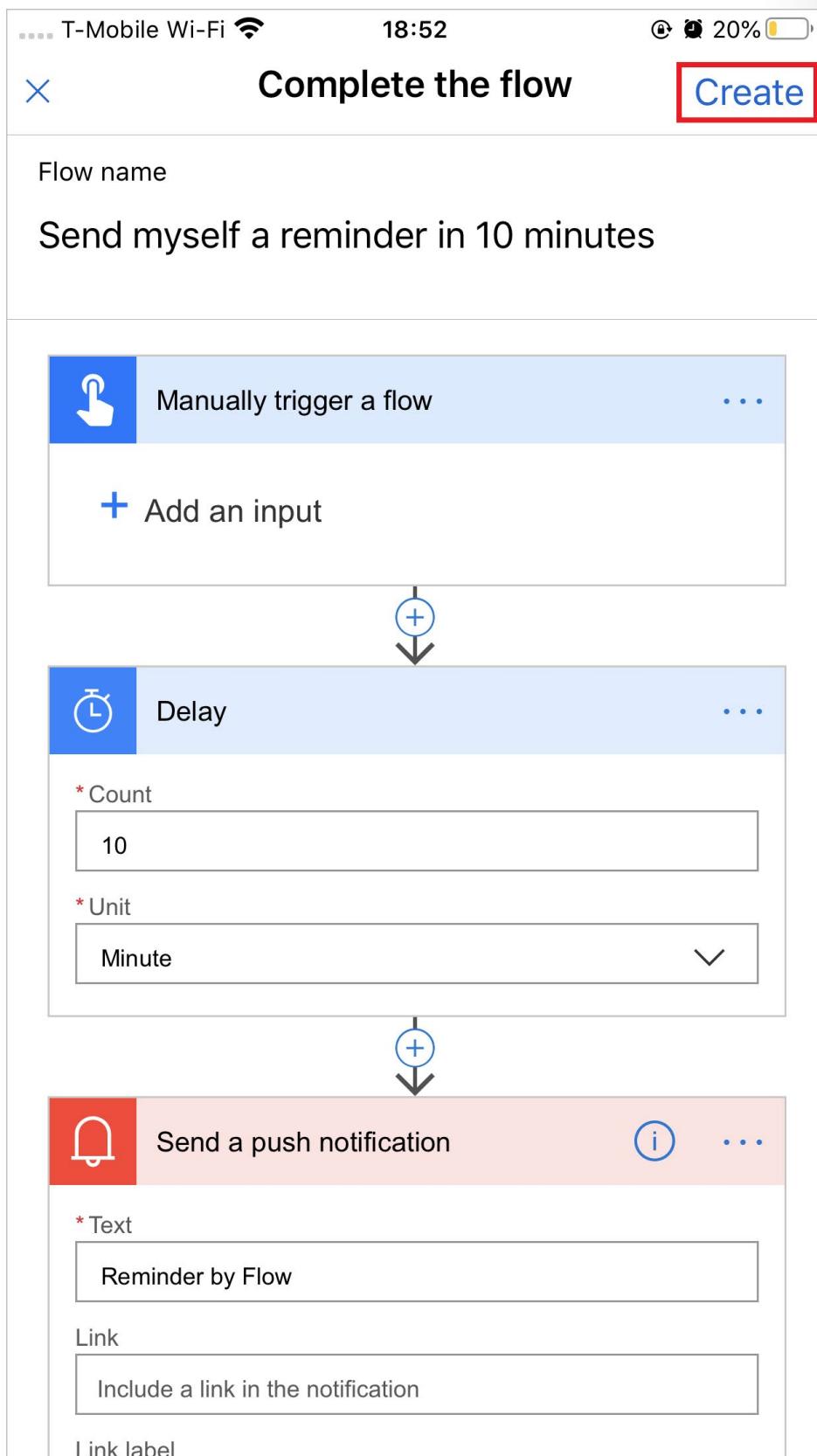
1. Select **Browse**.
2. Select the **Send myself a reminder in 10 minutes button** flow.



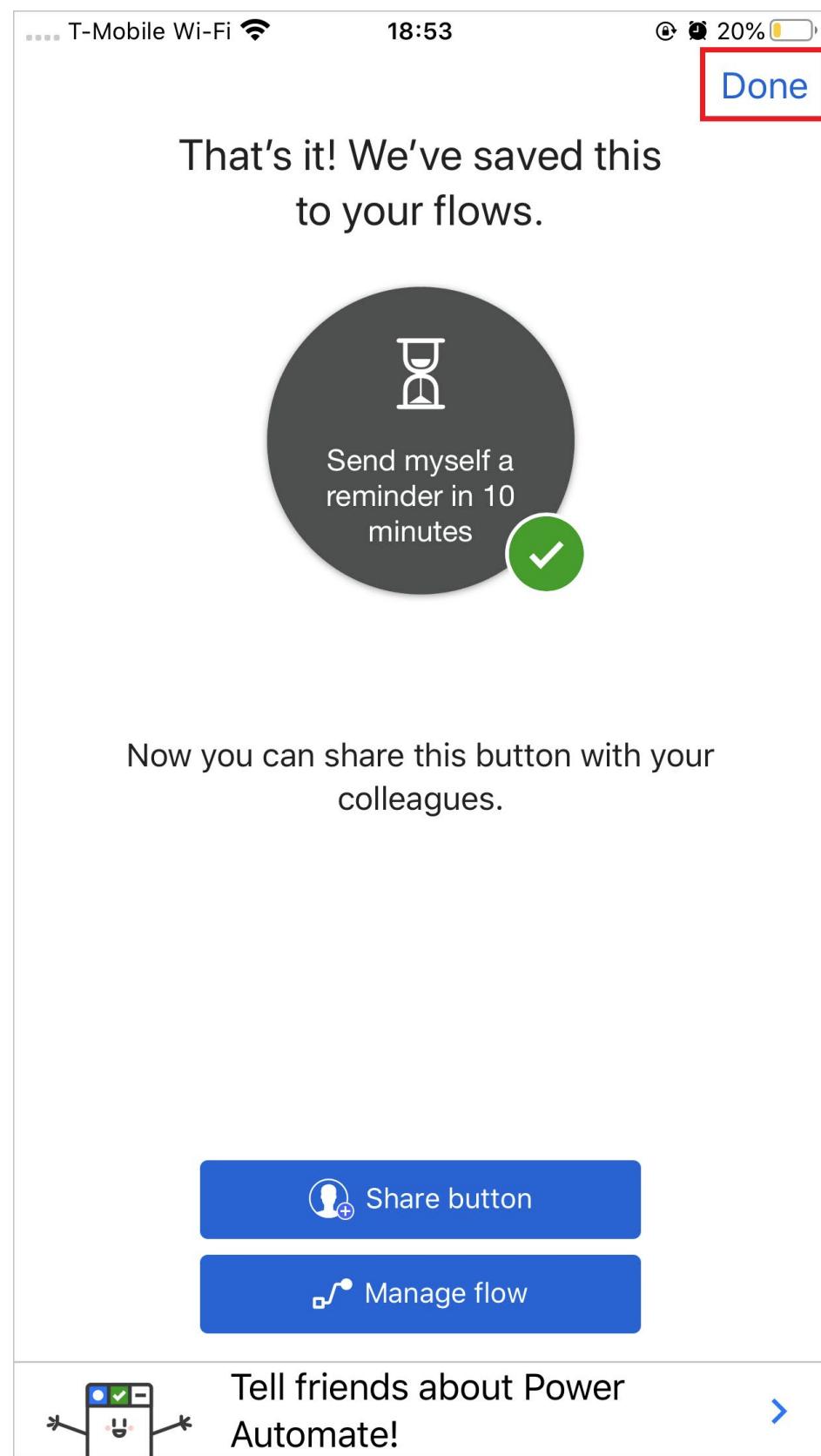
3. Select **Use this template**.



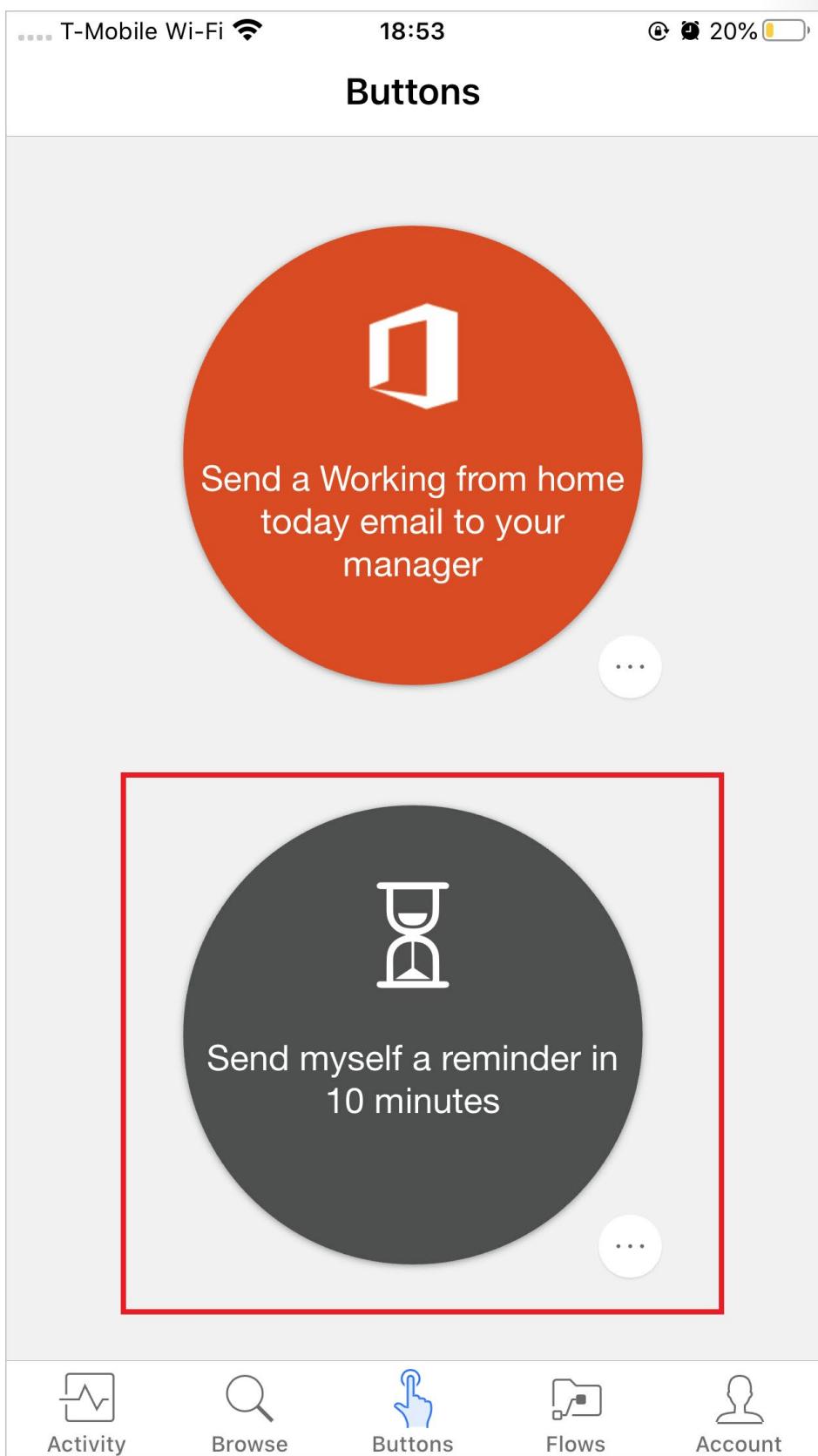
4. Select **Create**, and then select **Done**.



The flow is saved.



5. Select **Buttons** to see the new flow.



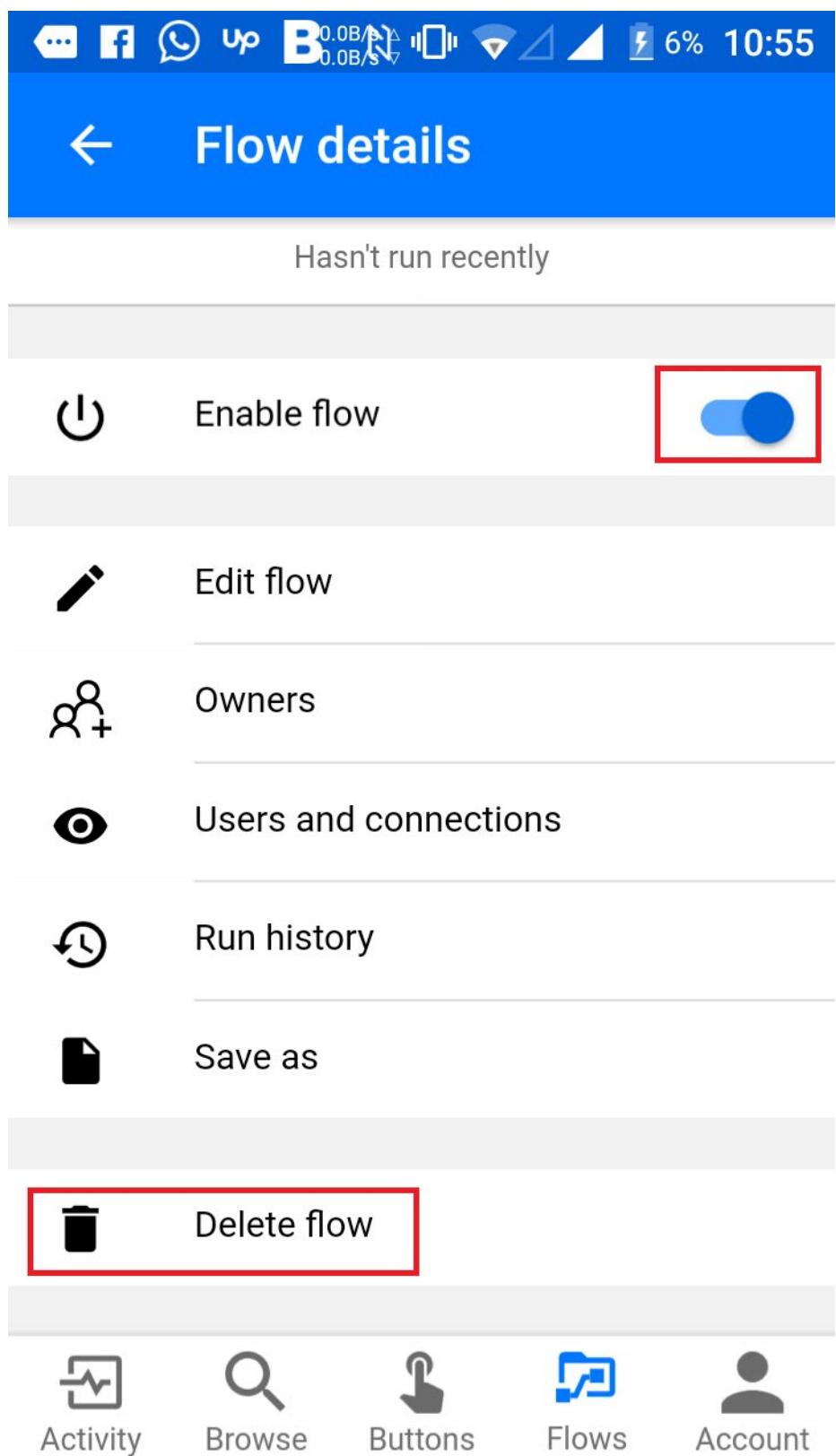
6. Select the flow. In 10 minutes, you'll get a reminder.

It's simple to add more buttons to your collection.

## Modify or delete a flow

If you want to change or delete one of your flows, it's easy.

1. Select **Flows**.
2. Select one of your flows.
3. Select one of the options:
  - To enable or disable the flow, toggle the **Enable flow** option on or off.
  - To change the flow, select **Edit flow**.
  - To get an idea of the successful and unsuccessful runs of the flow, select **Run history** to view the history of the flow.
  - To delete a flow, select **Delete flow**.



The next unit shows how to receive text and email notifications from flows.

# Exercise - Receive text and email notifications from flows

A common use of Power Automate is to get a notification when something happens. Notifications can be emails, text messages, or push notifications on your phone.

In this unit, you'll create a flow that generates a push notification whenever you receive an email from your manager.

## Get the mobile app

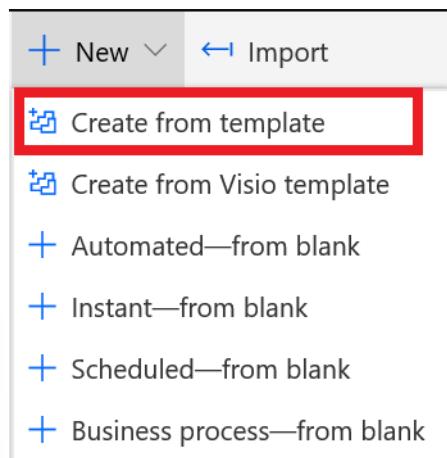
It's easy to create a flow that sends push notifications to your mobile device. Keep in mind that you'll need the Power Automate mobile app to receive push notifications. The mobile app is available for **Google Android<sup>2</sup>**, **Apple iOS<sup>3</sup>**, and **Windows Phone<sup>4</sup>**.

If you're using an unsupported mobile device, consider using Short Message Service (SMS) messages (that is, text messages) instead of push notifications to receive notifications.

## Create a flow that sends push notifications

Power Automate comes with many templates to get you started with creating flows. Let's create a flow by using a template.

1. Sign in to **Power Automate<sup>5</sup>** by using your organizational account.
2. Select **My flows**.
3. Select **New**, and then select **Create from template**.



4. Scroll down, and select **Get a push notification when you receive an email from your boss**.  
You can also quickly find this template by entering *notifications* in the search field.
5. Select **Create Flow**. The flow will automatically render and open to the details and run history page.
6. To edit the flow and see the steps which will be used to get your email profile and your boss's, select **Edit**.

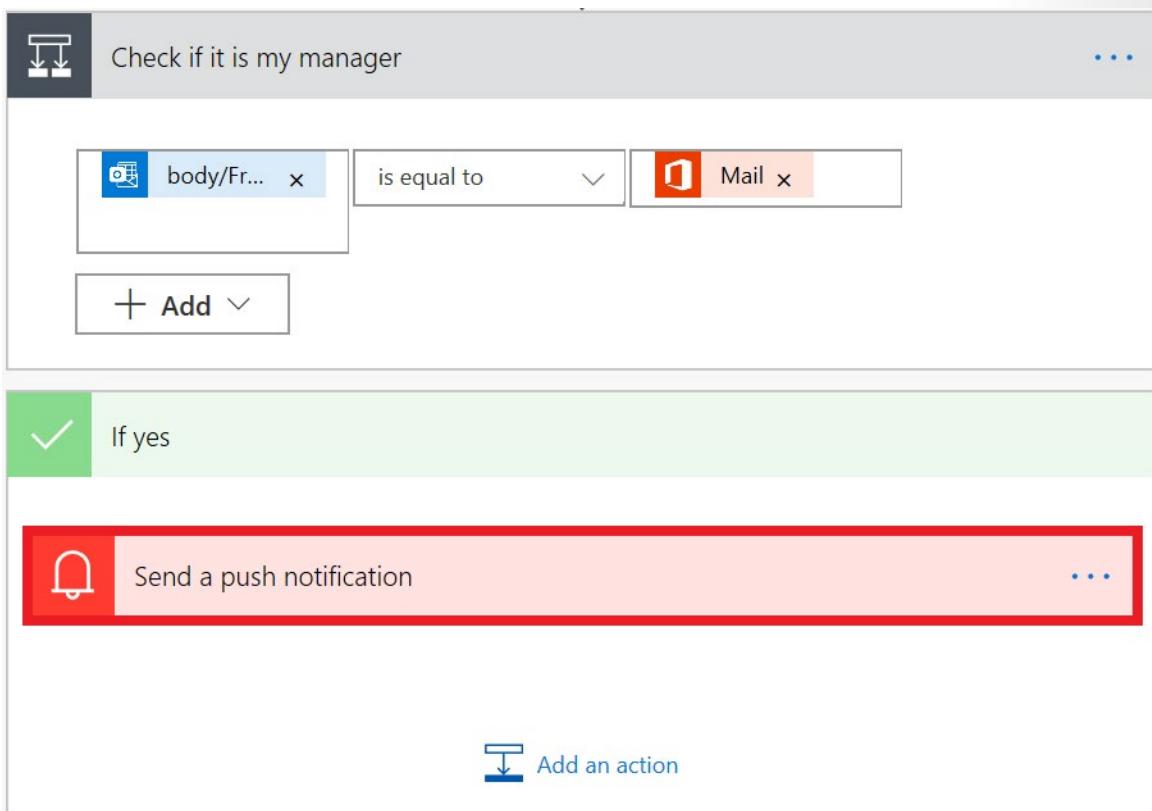
<sup>2</sup> <https://play.google.com/store/apps/details?id=com.microsoft.flow>

<sup>3</sup> <https://itunes.apple.com/app/apple-store/id1094928825>

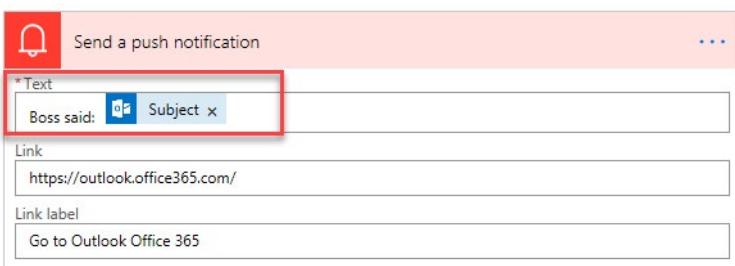
<sup>4</sup> <https://www.microsoft.com/p/microsoft-flow/9nkn0p5l9n84>

<sup>5</sup> <https://ms.flow.microsoft.com>

7. Scroll to the bottom of the flow steps to find the **Check if it is my manager** section. Expand it by clicking on the title of that section. Your email address and your manager's are automatically filled in from the profile information that you entered. If your organization does not store your manager information in the active directory, you can manually input your manager's email here. Only administrators have the right to change this information so you cannot edit this portion of your profile data unless you have administrator rights.
8. In the **Send a push notification** section, select the title bar to change the text of the notification that you'll get when an email is received from your boss.



9. To change the text of the email, in the **Text** field, enter a new message. You can also select dynamic content fields in the list.



10. Select **Save** to save and test the flow.

Now, when emails arrive from your boss, you'll get a push notification on your phone.

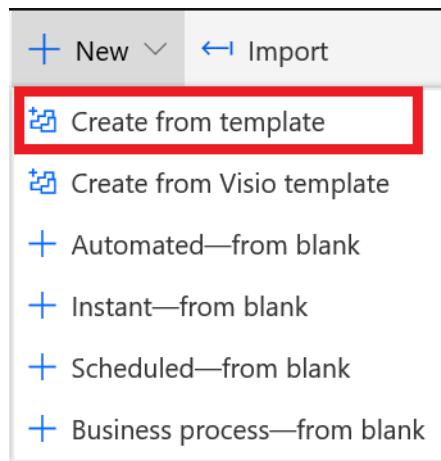
## Exercise - Copy files with flows

At some point, most of us have needed to copy files from one storage service to another. Power Automate makes it easy to automatically move and copy files between two services, like Microsoft OneDrive and Google Drive.

In this unit, you'll use a flow template to copy files from your personal OneDrive to Microsoft OneDrive for Business. Power Automate must have permissions to your OneDrive folders to do this.

### Create a flow that copies files

1. Sign in to **Power Automate**<sup>6</sup> by using your organizational account.
2. Select **My flows**.
3. Select **New**, and then select **Create from template**.



4. Scroll down, and select **Copy files to OneDrive for Business when they're added to OneDrive**.

You can also quickly find this template by entering *OneDrive for Business* in the search field.

5. Select **Continue**.
6. In the **OneDrive Folder** field, select the folder button.

A screenshot of a configuration screen for a 'OneDrive Folder'. It has two fields: '\*OneDrive Folder' and '\*OneDrive for Business Folder Path'. The first field contains the placeholder 'The unique identifier of the folder.' with a small folder icon button to its right, which is highlighted with a red box. The second field contains the placeholder 'The unique path of the folder.' with a similar folder icon button.

7. Select the OneDrive folder that files should be copied from.
8. In the **OneDrive for Business Folder Path** field, select the folder button, and then select the folder that files should be copied to.
9. Select **Create Flow**.
10. To change the flow, select **Edit flow**.

---

<sup>6</sup> <https://ms.flow.microsoft.com>

Now, whenever a file is put in the selected folder on OneDrive, it will be copied to the selected folder on OneDrive for Business.

## Exercise - Create recurring flows

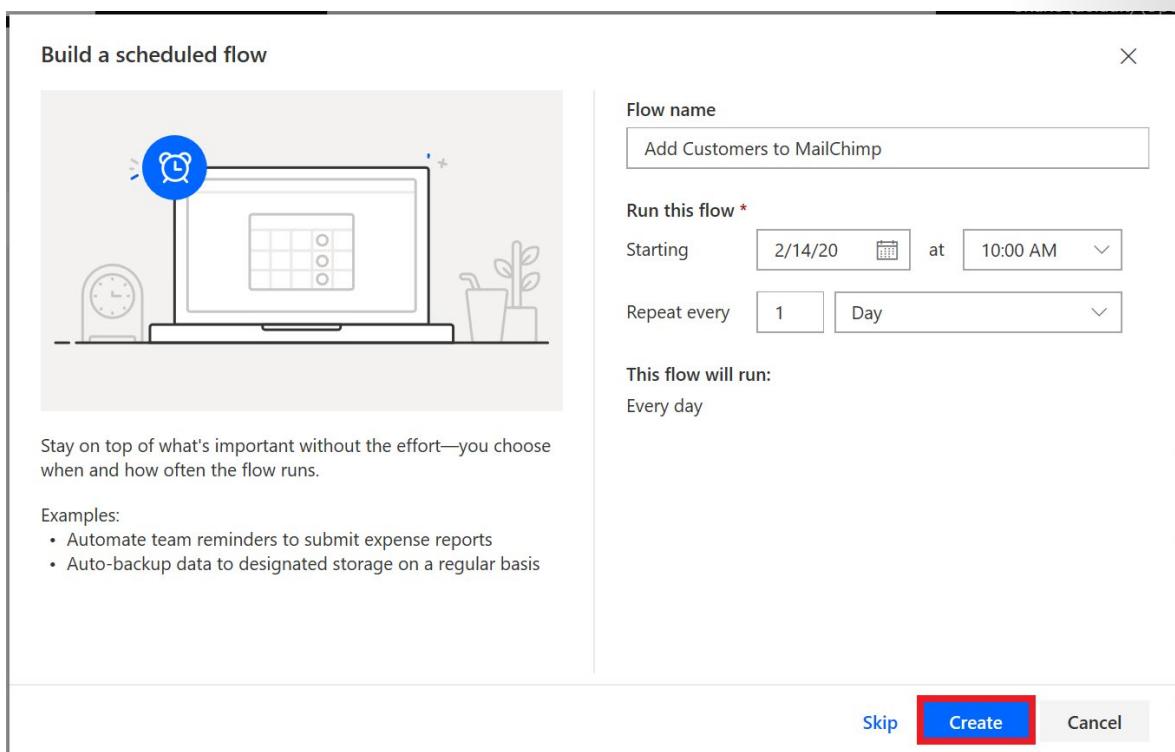
In this unit, you'll learn how to build prescheduled flows by using a trigger called *recurrence*. You'll build a flow for the Contoso marketing team that automatically pulls customer email addresses from a Microsoft Excel workbook on Microsoft OneDrive. You'll then set up the flow so that, once a day, any new email addresses that were added to the workbook are added to a MailChimp customer list.

### Prerequisites

For this scenario, you will need to make an excel file with a table that contains the following columns: ContactEmail, FirstName, and LastName.

### Create a scheduled flow

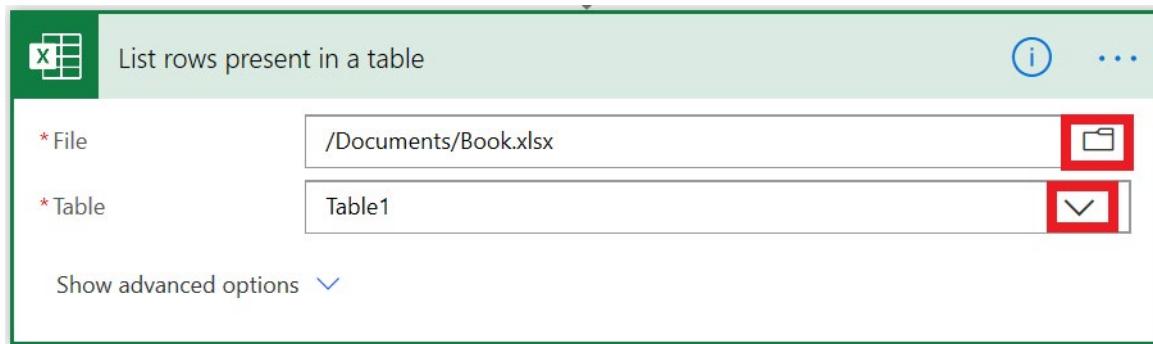
1. Sign in to **Power Automate**<sup>7</sup> by using your organizational account.
2. Select **My flows**.
3. Select **New**, and then select **Scheduled-from blank**.
4. Name your flow and under **Run this flow** set the flow to repeat every one Day.
5. Select **Create**.



6. Select **New step**, to add an action.

<sup>7</sup> <https://ms.flow.microsoft.com>

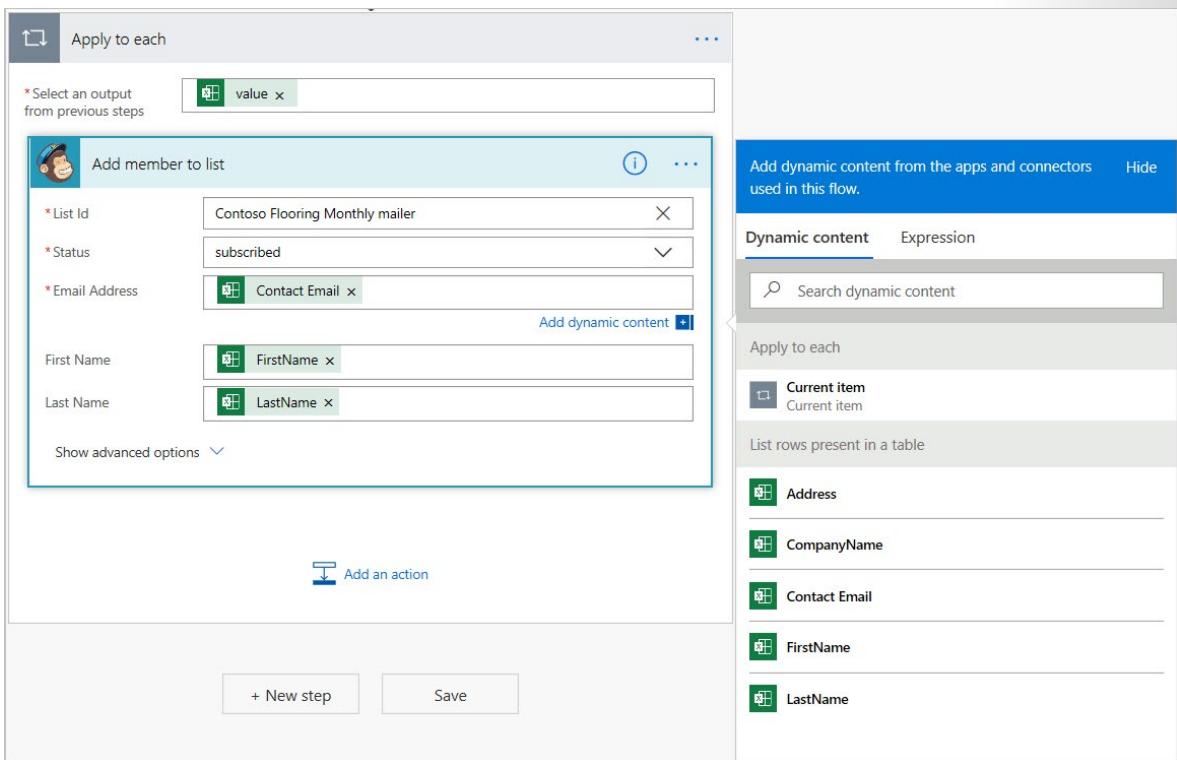
7. In the search field, enter *excel*, select the **Excel Online (Business)** service, and then select the **List rows present in a table** action.
8. In the **File name** field, select the folder button, and then select the Excel file to use.
9. In the **Table name** box, select the drop-down arrow, and then browse to and select the worksheet to use.



10. Select **New step**, and then select **Add an action**.
11. In the search field, enter *chimp*, select the **MailChimp** service, and then select the **MailChimp - Add member to list** action.

Note: MailChimp is a premium connector. Depending on your Power Automate license, you might need to sign up for a trial to use this connector.
12. In the **List Id** field, select the desired MailChimp mailing list. In the **Status** field, select *subscribed*.
13. In the **Email Address** field, use the dynamic content feature to add the **ContactEmail** field.

Notice that the flow automatically creates an additional step. Flow detects that you're setting up an action that requires an additional action. Whenever the flow reads a new email address, it will also create a new action for each row.
14. Use the dynamic content feature to fill in the **First name** and **Last name** fields.



And there you have it!

This flow will now run once a day, get the new rows from the Excel worksheet, grab the email address and name from each row, and enter the email address and name in the Contoso MailChimp mail list, saving you both time and money.

## Exercise - Send an email when a tweet is posted

You can create a flow that automatically performs one or more actions after it's triggered by an event. For example, the flow can notify you by email when someone posts a tweet that includes a keyword that you specify. In this example, posting a tweet is the event (also known as a *trigger*), and sending an email notification is the action. In this unit, you'll learn how to create this example flow.

### Prerequisites

- An account on [flow.microsoft.com](https://flow.microsoft.com)<sup>8</sup>
- A Twitter account
- Microsoft Office 365

### Specify an event to start the flow

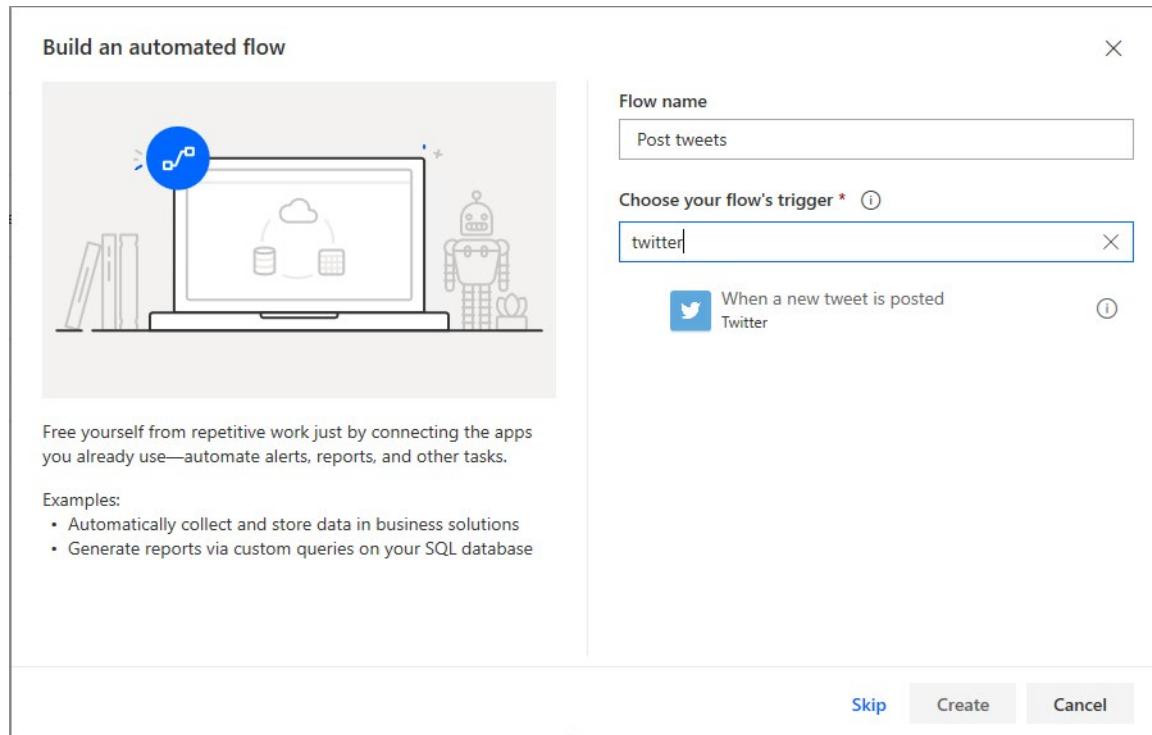
First, you must select the trigger (event) that starts the flow.

1. Sign in to **Power Automate**<sup>9</sup> by using your organizational account.

<sup>8</sup> <https://flow.microsoft.com>

<sup>9</sup> <https://flow.microsoft.com>

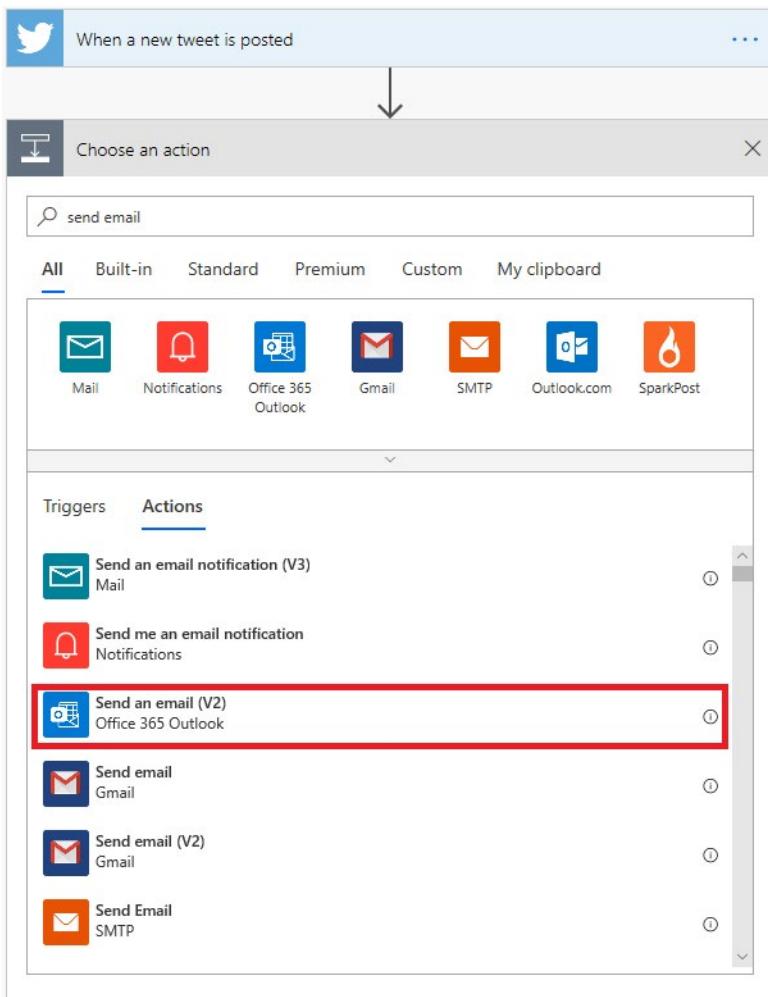
2. Select **My flows**.
3. Select **New**, and then select **Automated—from blank**.
4. Under **Choose your flow's trigger**, enter *twitter*, select the **Twitter - When a new tweet is posted** trigger and select **Create**.



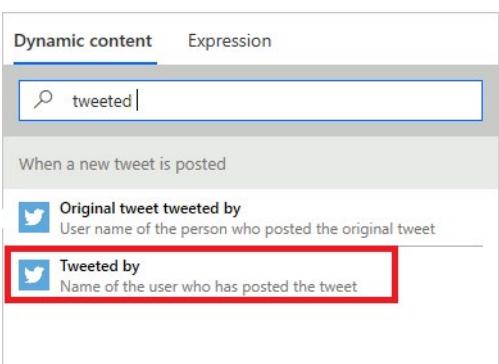
5. If you haven't already connected your Twitter account to Power Automate, select **Sign in to Twitter**, and then enter your credentials.
6. In the **Search text** box, enter the keyword to find.

## Specify an action

1. Select **New step**, and in the search field, enter *send email*, and then select the **Office 365 Outlook - Send an email** action.



2. If you're prompted to sign in, select the sign-in button, and then enter your credentials.
3. In the **To** field, enter or paste your email address, and then select your name in the list of contacts that appears.
4. In the **Subject** field, enter **New tweet from:** followed by a space.
5. In the list of dynamic content, select the **Tweeted by** token to add a placeholder for it.



6. Select the **Body** field, and then, in the list of dynamic content, select the **Tweet text** token to add a placeholder for it.

7. Optional: Add more tokens, other content, or both to the body of the email.
8. Select **Save** to save the flow.
9. Post a tweet that includes the keyword that you specified, or wait for someone else to post such a tweet.

Within a minute after the tweet is posted, an email message will notify you of the new tweet.

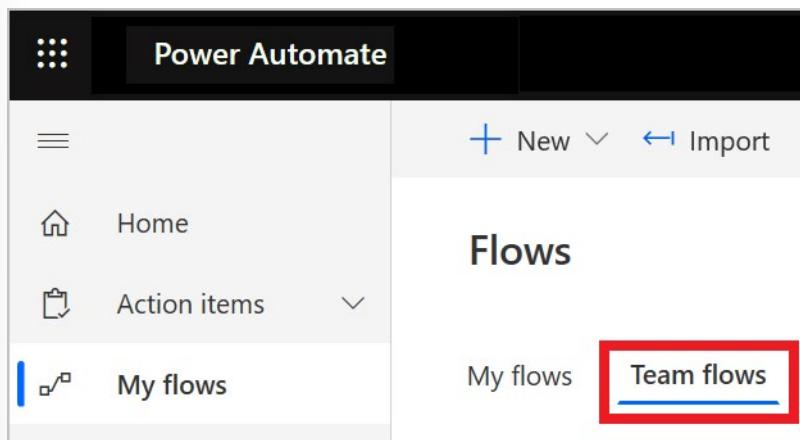
## Exercise - Create team flows

Team flows extend the potential of Power Automate to groups of people. After all, why should just one person enjoy the benefits of increased automation in his or her work environment?

Here are some advantages of team flows:

- Multiple people can own and manage a flow together.
- If the creator of a team flow leaves the organization, the other owners of the flow can continue to run it.
- All owners of a team flow can view its history, manage its properties, edit it, add and remove owners, and delete it.

If you're the creator or an owner of a team flow, you'll find it listed on the **Team flows** tab in **Power Automate**<sup>10</sup>.



*Note:*

- Shared connections can be used only in the flow in which they were created.
- Owners can use services in a flow, but they can't change the credentials for a connection that another owner created.

## Prerequisites

To create a team flow, you must have a **paid Power Automate plan**<sup>11</sup>. Additionally, to add other owners to a team flow or remove owners from it, you must be the creator or an owner.

---

<sup>10</sup> <https://flow.microsoft.com>

<sup>11</sup> <https://flow.microsoft.com/pricing/>

## Create a team flow

You create a team flow by adding other owners to an existing flow. After new owners are added to a flow, the flow will appear on the **Team flows** tab.

1. Sign in to **Power Automate**<sup>12</sup> by using your organizational account.
2. Select **My flows**.
3. Select the **Share** button for the flow that you want to change.

The screenshot shows the 'Flows' interface with the 'Team flows' tab selected. A single flow named 'Post tweets' is listed. The 'Share' button for this flow is highlighted with a red box. Other tabs visible include 'My flows', 'Business process flows', and 'UI flows (preview)'. The flow details show it was modified 49 minutes ago.

4. Enter the name, email address, or group name of the person or group that you want to add as an owner.
5. In the list that appears, select the user or group.

The user or group becomes an owner of the flow.

Keep in mind that when you create a team flow, it appears on the **Team flows** tab. It no longer appears on the **My flows** tab.

## Add a SharePoint list as a co-owner of a flow

You can add Microsoft SharePoint lists as co-owners of a flow. In that way, everyone who has edit access to the list automatically gets edit access to the flow. After the flow is shared, you can just distribute a link to it.

## Restrictions on changes to flows

Any owner of a team flow can contribute connections to a flow. After another person has access to the flow, that person can use any connections in it, but only within the scope of that flow.

For example, John creates a flow that updates items in SharePoint with his account, and he shares the flow with Mary. In this case, Mary will be able to change the use of SharePoint inside that flow, but not in any of her personal flows. Likewise, Mary can change the flow so that it uses her SharePoint connection, but John still won't be able to use that new SharePoint connection in any of his flows.

To view all the connections that are used by a flow, select the **Share** button, and inspect the list of embedded connections.

<sup>12</sup> <https://flow.microsoft.com>

## Remove an owner

Important: If you remove an owner whose credentials are used to access Power Automate services, be sure to update the credentials for those connections, so that the flow continues to work correctly.

1. On the **Team flows** tab, select the **Share** button for the flow that you want to change.

The screenshot shows the 'Flows' interface with the 'Team flows' tab selected. A single flow is listed: 'Post tweets'. The flow icon is a blue Twitter bird, and its name is 'Post tweets'. To the right of the flow, there are edit, delete, and more options buttons. The delete button is highlighted with a red box. The table has columns for Name and Modified.

Name	Modified
Post tweets	49 min ago

2. Select the **Delete** button for the owner that you want to remove.

## Embedded and other connections

The connections that are used in a flow fall into two categories:

- **Embedded:** These connections are used in the flow.
- **Other:** These connections have been defined for the flow, but they aren't used in it.

If you stop using a connection in a flow, that connection appears in the **Other** connections list. It will remain there until an owner includes it in the flow again.

To view the list of connections, change the team flow.

- On the **Team flows** tab, select the **Invite another owner** button for the flow that you want to change.  
The list of connections appears under the list of owners in the flow's properties.

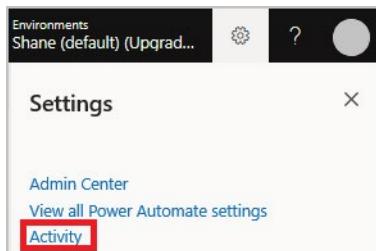
## Troubleshoot flows

In this unit, you'll learn how to troubleshoot common issues that might occur while you run your flows.

### Identify the error

Before you can fix a flow, you must identify why it failed. You will get an email with a list of failures each week.

1. Select the **Settings** button (the gear symbol) at the top of the web portal and select **Activity** (or select the **Activity** tab in the mobile app), and then select your flow in the list that appears.



2. Details about the flow appear, and at least one step has a red exclamation point (!) symbol. Open that step, and review the error message.

A screenshot of the Power Automate flow details page for a flow named '← Skype room support ticket'. The flow consists of two steps: 'Manually trigger a flow' and 'Apply to each'. The 'Apply to each' step has an error message: 'An action failed. No dependent actions succeeded.' Below it, the 'HTTP' action step also has an error message: 'Unauthorized'. The 'HTTP' step details include: Code: Unauthorized, Method: POST, URI: https://prod-24-westus.logic.azure.com:443/workflows/7db14b8c9cf4382a291c1b041a, and Status code: 401.

## Authentication failures

In many cases, flows fail because of an authentication error. If this type of error occurs, the error message includes the word "Unauthorized," or an error code of 401 or 403 appears. You can usually fix authentication errors by updating the connection.

1. You can view the connections by opening up the flow details by selecting the flow from **My Flows**.
2. Scroll to the connection that you saw the "Unauthorized" error message for.
3. Next to the connection, select the **Verify password** link in the message that states that the connection hasn't been authenticated.
4. Verify your credentials by following the instructions that appear. Then return to your flow-run failure, and select **Resubmit**.

The flow should now run as expected.

## Action configuration issues

Flows sometimes fail if a setting in one of the flow's actions doesn't work as expected. In this case, the error message includes the phrase "Bad request" or "Not found," or an error code 400 or 404 appears.

The error message should indicate how to fix the failure.

1. Select the **Edit** button, and then fix the issues inside the flow definition.
2. Save the updated flow, and then select **Resubmit** to try to run the flow again with the updated configuration.

## Temporary issues

If error code 500 or 502 appears, the failure is temporary or transient.

- Select **Resubmit** to try to run the flow again.

## Issues with your pricing plan

Sometimes your flows might behave unexpectedly because you aren't using the correct plan.

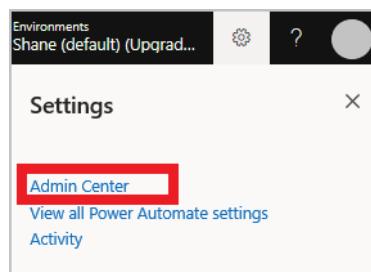
- To view your plan, in Power Automate, select **Learn**. It will redirect you to another page. Here select **Learn**, and then select **Pricing**.

Learn more about **pricing and how to switch plans<sup>13</sup>**.

## Issues with data usage

You might have run out of data that you can use.

- If you're on a free plan or a trial plan, select the **Settings** button (the gear symbol) to show your current usage against your plan.



- If you're on a paid plan, runs are pooled across all users in your organization. We're working on features that will show information about available quotas and usage across an organization.

Important: If you exceed your data limit, Power Automate throttles your flow runs.

Learn more about **usage limits<sup>14</sup>**.

## You might be running flows too often

Your plan determines how often your flows run. For example, your flows might run every 15 minutes if you're on the free plan. If a flow is triggered less than 15 minutes after its last run, it's queued until 15 minutes have passed.

Whenever a flow is triggered, whether by an automatic trigger or because you manually start it, the action counts as a run. Checks for new data don't count as runs.

---

<sup>13</sup> <https://flow.microsoft.com/pricing/>

<sup>14</sup> <https://flow.microsoft.com/pricing/>

Learn more about **usage limits**<sup>15</sup>.

## You might be using an incorrect account

If you sign in by using a Microsoft account (for example, an account that ends with @outlook.com or @gmail.com), you can use only the free plan. To take advantage of the features of the paid plan, sign in by using your organizational account or school email address.

To upgrade, use an organizational account or a school account, or create a **Microsoft Office 365 trial account**<sup>16</sup>.

## Some flows run more often than expected

Some flows might run more often than you expect. For example, you create a flow that sends you a push notification whenever your manager sends you an email. That flow must run every time you get an email from anyone, because the flow must check whether the email came from your manager. This action counts as a run.

## Other issues that are based on limits, and caveats

You might have issues that are based on other limits:

- Each account can have up to:
  - 250 flows.
  - 15 custom connectors.
  - 20 connections per application programming interface (API) and 100 connections total.
- You can install a gateway only in the default environment.
- Some external connectors, like Twitter, implement connection throttling to control the quality of service. Your flows might fail when throttling is in effect. If your flows are failing, review the details of the run that failed in the flow's run history.

## Summary

Let's quickly review what we covered in this module.

In this module, you learned the basics of Power Automate, including the difference between triggers and actions and how to create flows for yourself or your team. You created a flow that automatically saves email attachments, one that alerts you of relevant tweets and a button flow to send yourself a reminder.

## Next steps

**Congratulations!** You've finished the first module of this learning path for Power Automate. Continue learning by checking out the next module in this learning path.

See you in the next module!

<sup>15</sup> <https://flow.microsoft.com/pricing/>

<sup>16</sup> <https://powerbi.microsoft.com/documentation/powerbi-admin-signing-up-for-power-bi-with-a-new-office-365-trial/>

## Build more complex flows

### Learn to build more complex flows

In the previous module for Power Automate, you learned how to build some basic flows that save email attachments and send you reminders from your phone.



In this module, you'll now build some more complex flows and increase your knowledge.

If you're a beginner with Power Automate, this module will expand your business flow skills. If you already have some experience, this module will tie concepts together and hopefully help fill in the gaps.

Be more productive, automatically: that's what Power Automate is all about.

So, let's get started!

### Exercise - Build an approval request

In this unit, you'll learn how to build a business-friendly scenario that uses approvals.

In this scenario, anyone who has access to the Microsoft SharePoint list can contribute tweets without knowing anything about Twitter. The social media team can then approve or reject those tweets allowing that team to remain in control of the account and the content that goes out to customers.

#### Step one: Create a SharePoint list for tweets

You'll use a template that starts an approval process whenever a new item is created in a specific list. If the item is approved, a tweet is posted to Twitter. For this unit, you'll change the process by adding steps that update a SharePoint list with the approval response, indicate whether the item was approved, and add any comments that the approver added to the proposed tweet.

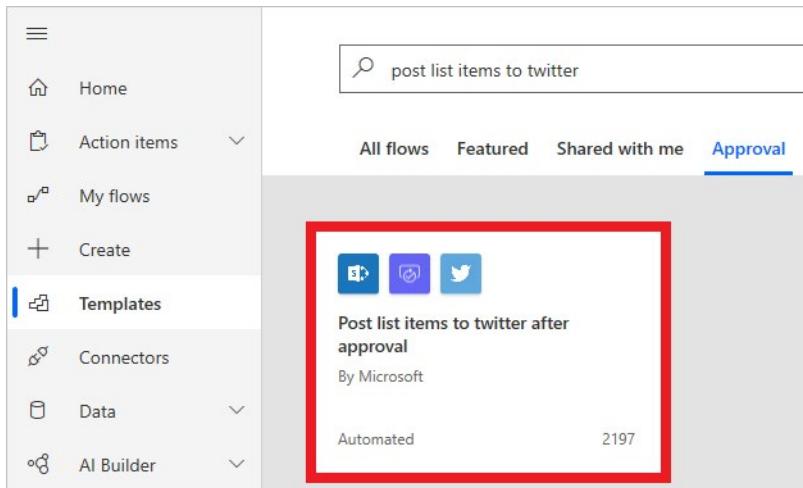
First, let's create the SharePoint list.

1. On your SharePoint site, create a SharePoint list named *ContosoTweets*.
2. Open the list, and select **Add column**.
3. Select + **Add column** to add the following columns. Select **Save** after you create each column.
  - Add a column of the *Multiple lines of text* type that's named *TweetContent*. This column will hold the content of the tweets that will be approved later.
  - Add a column of the *Date* type that's named *TweetDate*. Toggle the **Include Time** option to yes.
  - Add a column of the *Yes/No* type that's named *ApprovalStatus*. The approver can then select **Yes** or **No** to approve or reject the tweet.
  - Add a column of the *Single line of text* type that's named *ApproverComments*. The approver can then add a comment about the approval status.

4. Copy the URL of the SharePoint list. You'll use it when you create the flow.

## Step two: Create an approval request flow

1. Sign in to **Power Automate**<sup>17</sup>, and then select **Templates**.
2. Search **Post list items to Twitter** under **Approvals** and select the appropriate template.



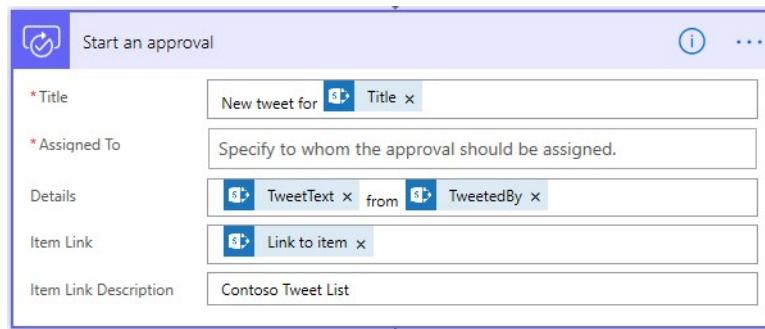
3. Make sure that your account credentials for **SharePoint**, **Approvals**, and **Twitter** are correct, and then select **Continue**.
4. In the **When a new item is created** action, enter the following values:
  - **Site Address:** Enter the URL of your team's SharePoint site.
  - **List Name:** Select *ContosoTweets*.
5. In the **Start an approval** action, in the **Title** field, enter *New tweet for*, and then select **Title** in the dynamic content list.

The screenshot shows the configuration of a 'Start an approval' action. The 'Title' field is populated with 'New tweet for'. To the right, a 'Dynamic content' pane is open, showing a list of available fields. The 'Title' field is listed and highlighted with a red box. Other visible fields include 'Created By JobTitle', 'Modified By JobTitle', and 'Title' again, which is also highlighted with a red box.

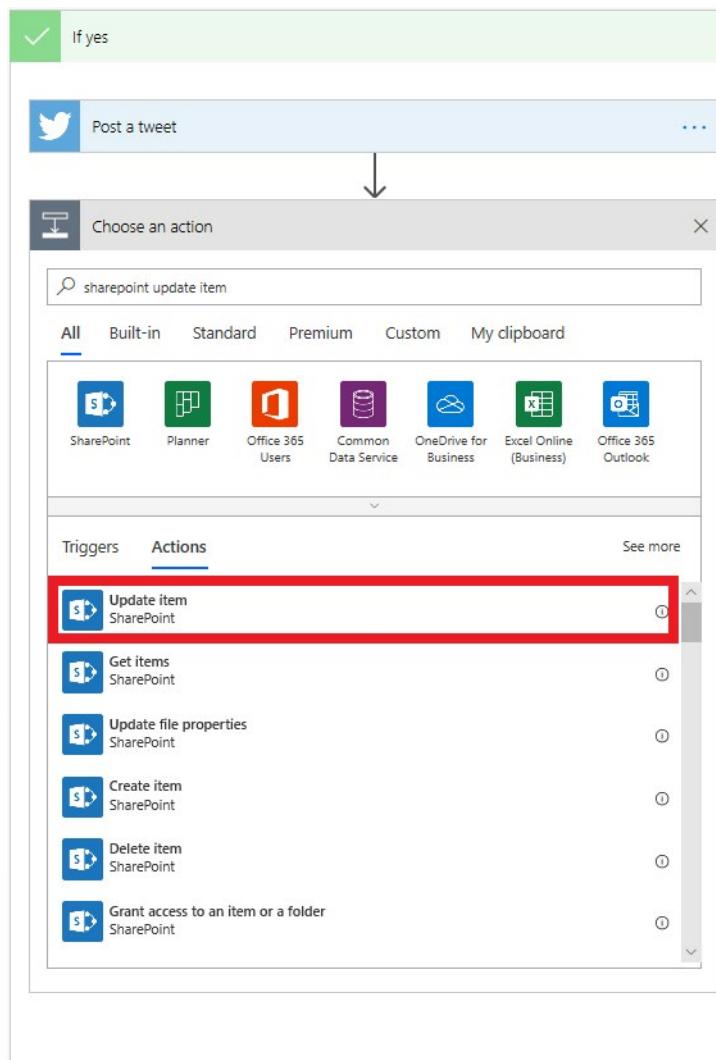
6. In the **Assigned to** field, enter and select either your name or the name of a test user.
7. In the **Details** field, remove the default items, and add dynamic fields and text to get **TweetContent** on **TweetDate** by **Created by DisplayName**.

<sup>17</sup> <https://ms.flow.microsoft.com>

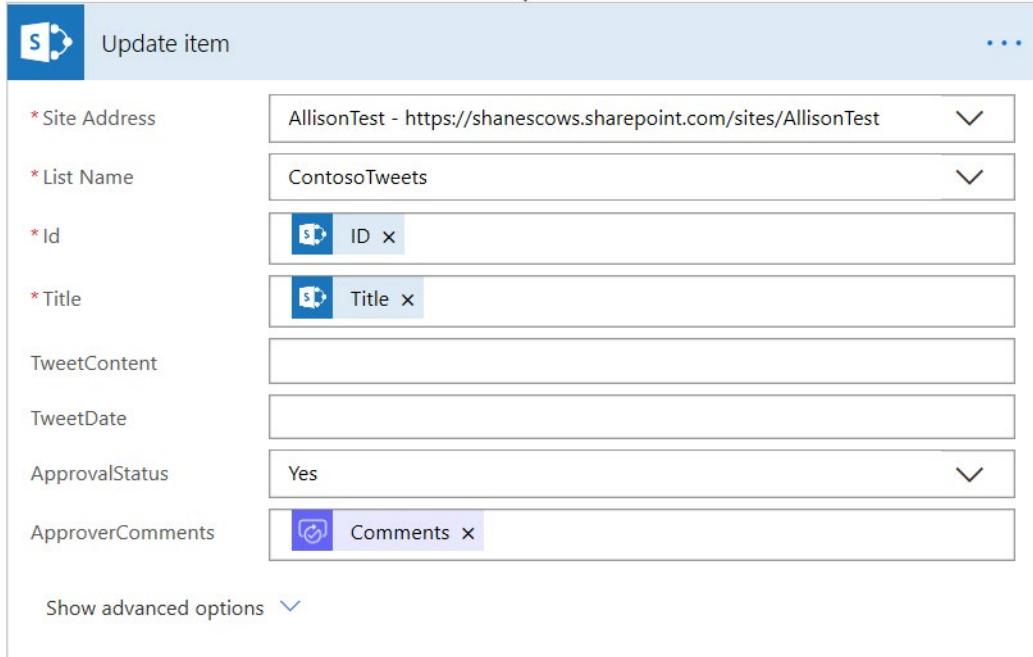
8. In the **Item Link** field, add the dynamic content **Link to Item**. In the **Item Link Description** field, enter *Contoso Tweet List*.



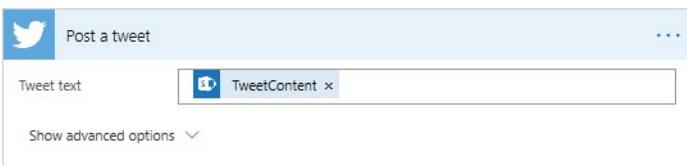
9. In the **If yes** section of the Condition, select **Add an action**.  
10. Search for *update item*, select the **SharePoint** connector, and then select the **SharePoint – Update item** action.



11. In the **Site Address**, enter the URL of the team's SharePoint site again. In the **List Name** field, select **ContosoTweets** again. In the **Id** field, add **ID** from the dynamic content list. The **Id** field is used to match the actual tweet request in the SharePoint list.
12. Select the **Title** field, and then, in the dynamic content list, search for *title*. Add **Title** from the **When a new item is created** action.
13. In the **ApprovalStatus** field, select **Yes**. Then select the **ApproverComments** field, and add **Comments** from the dynamic content list.



14. Click and drag **Post a Tweet** to below the **Update Item** action.
15. In the **If no** section of the Condition, select **Add an action**.
16. Repeat steps 10 through 13 to create a **SharePoint – Update item** action. Set the same values that you set for the **IF YES** condition. The only difference is that you set the **ApprovalStatus** field to **No** this time.
17. Expand the **Post a tweet** action by selecting the title bar. Then select the **Tweet text** field, and add **TweetContent** from the dynamic content list. This step will create the actual tweet and then post it to Twitter when it's approved.



18. Select **Save**.

Congratulations! You just created your first approval flow.

This unit showed just one way that Power Automate can empower your team to be more productive. Your team can contribute ideas, relevant news, or product guidance, but you maintain control over what's tweeted out to customers.

In the next unit, you'll see what it looks like when an approver receives a new request for a proposed tweet.

## Exercise - Build a flow that processes an approval request

In the previous unit, you learned how to build an approval process for tweets that are stored in a Microsoft SharePoint list. In this unit, you'll see what the experience looks like when an approver receives a new approval request.

### Step one: Change the SharePoint list

First, we need to add an item to our SharePoint list. We can then process an approval request for that item.

1. In SharePoint, open the **ContosoTweets** list that you set up in the previous unit, and then select **New** to create a list item (tweet).
2. Enter the following values, and then select **Save**:
  - **Title:** Promotions
  - **TweetContent:** Check out the new product line of Contoso Flooring #OhSoContoso
  - **Tweet Date:** Today's date.

The screenshot shows the 'New item' form for the 'ContosoTweets' list. At the top right, there are buttons for 'Save' (highlighted with a red box), 'Cancel', 'Copy link', and 'Edit form'. The main area has a title 'New item'. There are three fields: 'Title \*' with the value 'Promotions', 'TweetContent' with the value 'Check out the new product line of Contoso Flooring. #OhSoContoso', and 'TweetDate' with the value '2/25/2020' and a calendar icon. Below the date is a dropdown menu showing '12:00 AM'.

### Step two: Change the flow

1. In Power Automate, select **My flows**.
2. Select the **Post list items to Twitter after approval** flow that you set up in the previous unit, and then, under **Run history**, select the flow that's running.

3. Select the **When a new item is created** trigger. Make sure that the information for the list item that you just created is shown.

OUTPUTS	
ID	1
Title	Promotions
TweetContent	Check out the new product line of Contoso Flooring. #OhSoContoso
TweetDate	2020-02-25T08:00:00Z

4. In Microsoft Outlook, open the automated approval mail in the inbox, and then select **Approve**. Add a comment and press **Submit**.



5. In Power Automate, click on **Approvals** under **Actions**. You can see the approval you just submitted in your history.
6. In SharePoint, refresh the **ContosoTweets** list. Make sure that the **ApprovalStatus** field is set to **Yes**, and that the comment that you just entered is shown.

ContosoTweets				
Title	TweetContent	TweetDate	ApprovalStatus	ApproverComments
Promotions	Check out the new produc #OhSoContoso	2/25/2020 12:00 AM	Yes	Good Tweet

In this unit, you saw the experience from the approver's point of view, from receiving an approval request email to processing the request in the Approval Center.

## Exercise - Create a flow that stores documents

In this unit, you'll see how Contoso Flooring uses Power Automate to automatically convert documents to a standard format and then store them in Microsoft SharePoint Online. You'll create a flow that detects when a new file has been added to a Microsoft OneDrive for Business folder. The flow then converts that file to a PDF and stores it in a SharePoint Online folder.

### Prerequisites

For this scenario, you need an account with Muhimbi, a PDF conversion service. If you don't already have a Muhimbi account, you can sign up for a [free 30-day trial<sup>18</sup>](#).

### Create the source and target folders

First, you must create the source and target folders in OneDrive for Business and SharePoint Online.

1. In OneDrive for Business, under **Files**, create a folder named **Finished Documents**.
2. In SharePoint Online, in **Shared Documents**, create a folder named **PDF – Finished files**.

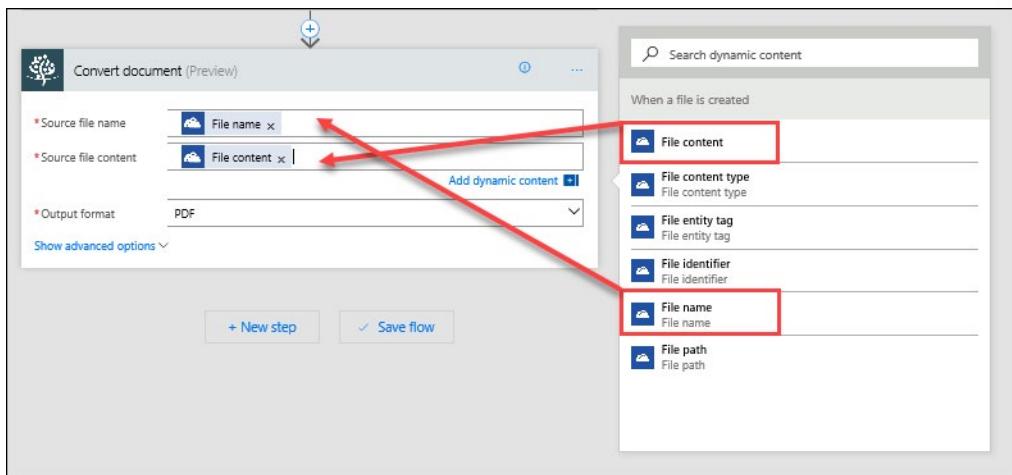
### Create the flow

1. In Power Automate, select **My Flows**, and then select **Automated–from blank**.
2. In the **Choose your flow's trigger** search field, enter *OneDrive*, and select the **OneDrive for Business - When a file is created** trigger. Name your flow and press create.
3. In the **Folder** field, select the folder button, and then select the **Finished Documents** folder that you created in the previous step.
4. Select **New step**.
5. In the search box, enter *muhimbi*, select the **Muhimbi PDF** connector, and then select the **Muhimbi PDF – Convert document** action.
6. If Power Automate prompts you to sign in to Muhimbi, sign in. If you don't have a subscription to Muhimbi, you can use a [free 30-day trial<sup>19</sup>](#).
7. In the **Convert document** action, set the following values:
  - **Source file name:** In the dynamic content list, select **File name**.
  - **Source file content:** In the dynamic content list, select **File content**.
  - **Output format:** Select **PDF**.

---

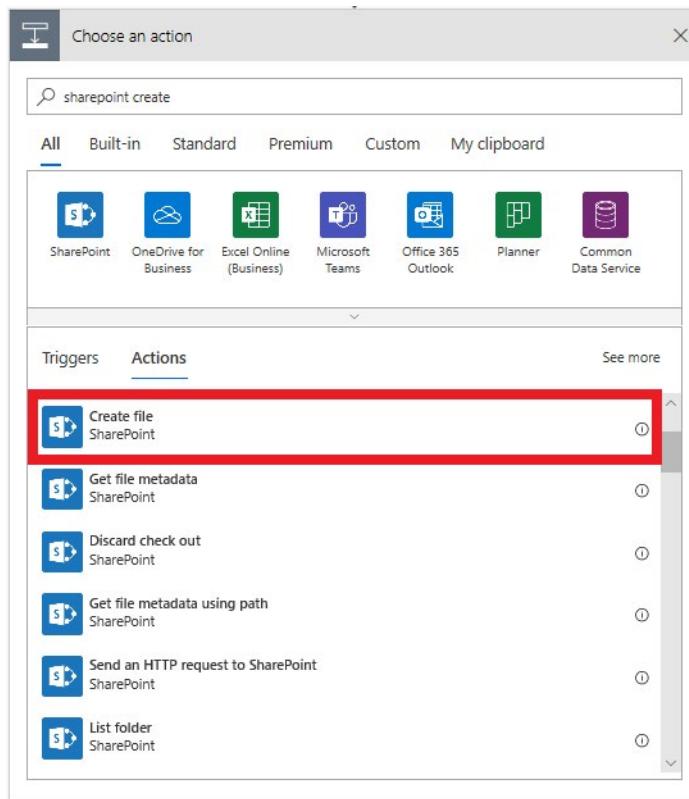
<sup>18</sup> <https://api.muhimbi.com/Auth/Pages/Signup.aspx>

<sup>19</sup> <https://api.muhimbi.com/Auth/Pages/Signup.aspx>



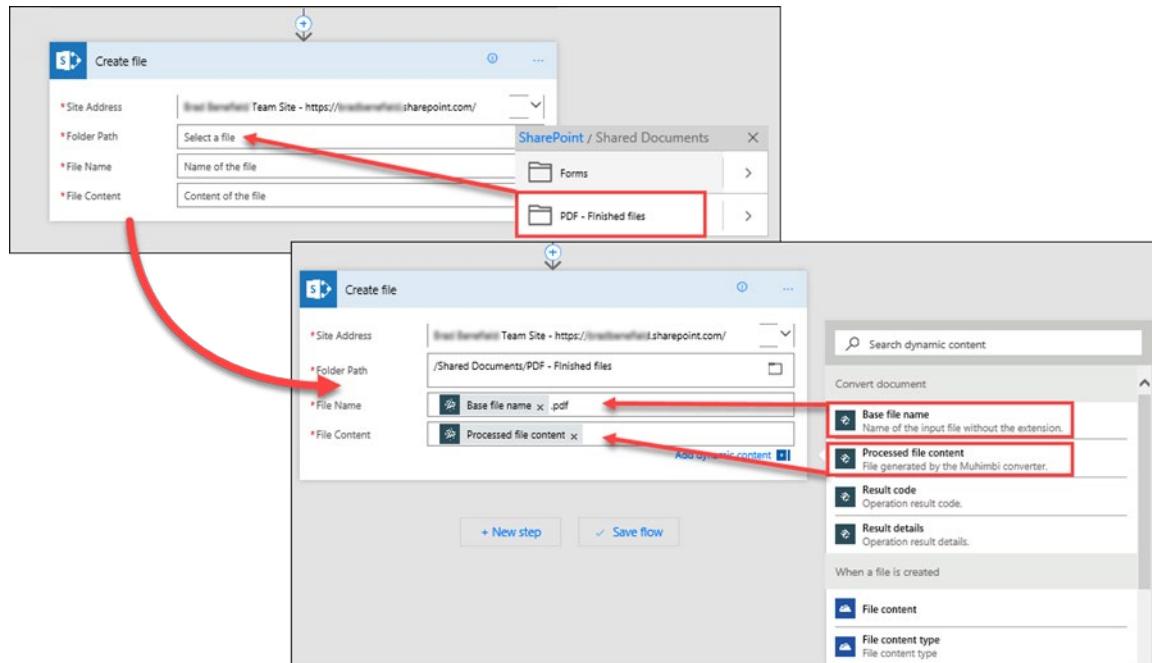
So far, you've set up these steps for your flow:

1. The flow is triggered whenever a new file is added to a specific OneDrive for Business folder.
2. The Muhimbi service converts that file to PDF.
- For the final step, you'll add an action that moves the PDF document to a SharePoint Online folder where the team can access it.
8. Select **New step**.
9. In the search field, enter *sharepoint*, and then select the **SharePoint – Create file** action.



10. In the **Create file** action, set the following values:

- **Site address:** Enter the URL of your SharePoint site.
- **Folder path:** Select the folder button, and browse to the **PDF - Finished files** folder.
- **File name:** In the dynamic content list, under **Convert document**, select **Base file name**. Then enter **.pdf** so that the file will be saved with the .pdf file name extension in SharePoint.
- **File content:** In the dynamic content list, under **Convert document**, select **Processed file content**.



11. Select **Save** at the top of the page to save your work.

## Test the flow

1. To test the flow, add a new file to your **Finished Documents** folder in OneDrive for Business.
2. In Power Automate, select **My flows**, and then select the new flow to view the run history.
3. After the flow runs, make sure that the file was converted to a PDF and saved to the **PDF – Finished files** folder in SharePoint.

## Exercise - Build a flow that uses information like locations or date

You can build button flows that use information like Global Positioning System (GPS) data, date information, or email. This information is available as *trigger tokens*. Trigger tokens are data points that are known and available to the device that a button flow is running on. These tokens change, based on factors like the current time or the current geographic location of the device.

For example, if you run a button flow on a phone, the phone probably knows the time at your current location, the date, and your current address. In other words, the time and date, and the address where

the phone is located, are all determined when the button flow runs. They're automatically available for use in any button flows that are run on the device.

You can use these trigger tokens to build useful flows that minimize repetitive tasks like providing your location to someone or tracking how much time you spent on a particular job/service call.

## List of button trigger tokens

Here's the list of button trigger tokens that are available to you when you create button flows.

Parameter	Description
City	The city where the device that's running the flow is located.
Country/Region	The country/region where the device that's running the flow is located.
Full address	The full address where the device that's running the flow is located.
Latitude	The latitude where the device that's running the flow is located.
Longitude	The longitude where the device that's running the flow is located.
PostalCode	The postal code where the device that's running the flow is located.
State	The state where the device that's running the flow is located.
Street	The street where the device that's running the flow is located.
Timestamp	The time in the area where the device that's running the flow is located.
Date	The date in the area where the device that's running the flow is located.
User name	The user name of the person who's signed in to the device that's running the flow.
User email	The email address of the person who's signed in to the device that's running the flow.

## Create a button flow that uses trigger tokens

When you create a button, you can use trigger tokens to add rich functionality to it.

Let's create a button flow on a mobile device. The button flow will use trigger tokens to send the date and your full address in a "Working from home" email to your boss.

Although the procedures in this unit show screenshots from an Apple iOS device, the experience is similar on Android and Windows Phone devices.

## Prerequisites

- A work or school email address, or a **Microsoft account**<sup>20</sup> that has access to Power Automate

<sup>20</sup> <https://account.microsoft.com/about?refd=www.microsoft.com>

- The Power Automate mobile app for **Android<sup>21</sup>**, **iOS<sup>22</sup>**, or **Windows Phone<sup>23</sup>**

## Create the button flow

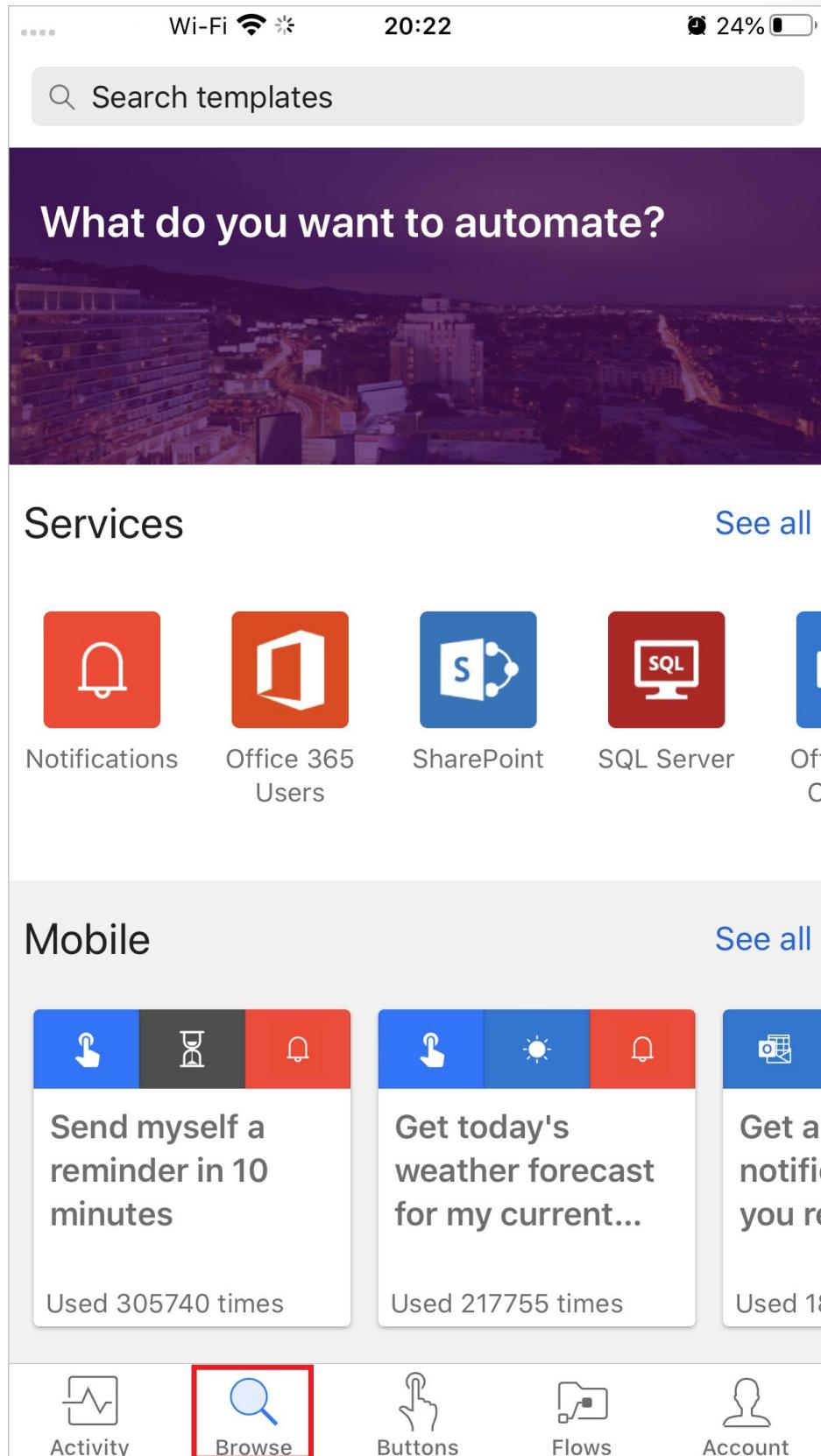
1. Launch the Power Automate mobile app and sign in using your organizational account.
2. Select **Browse**.

---

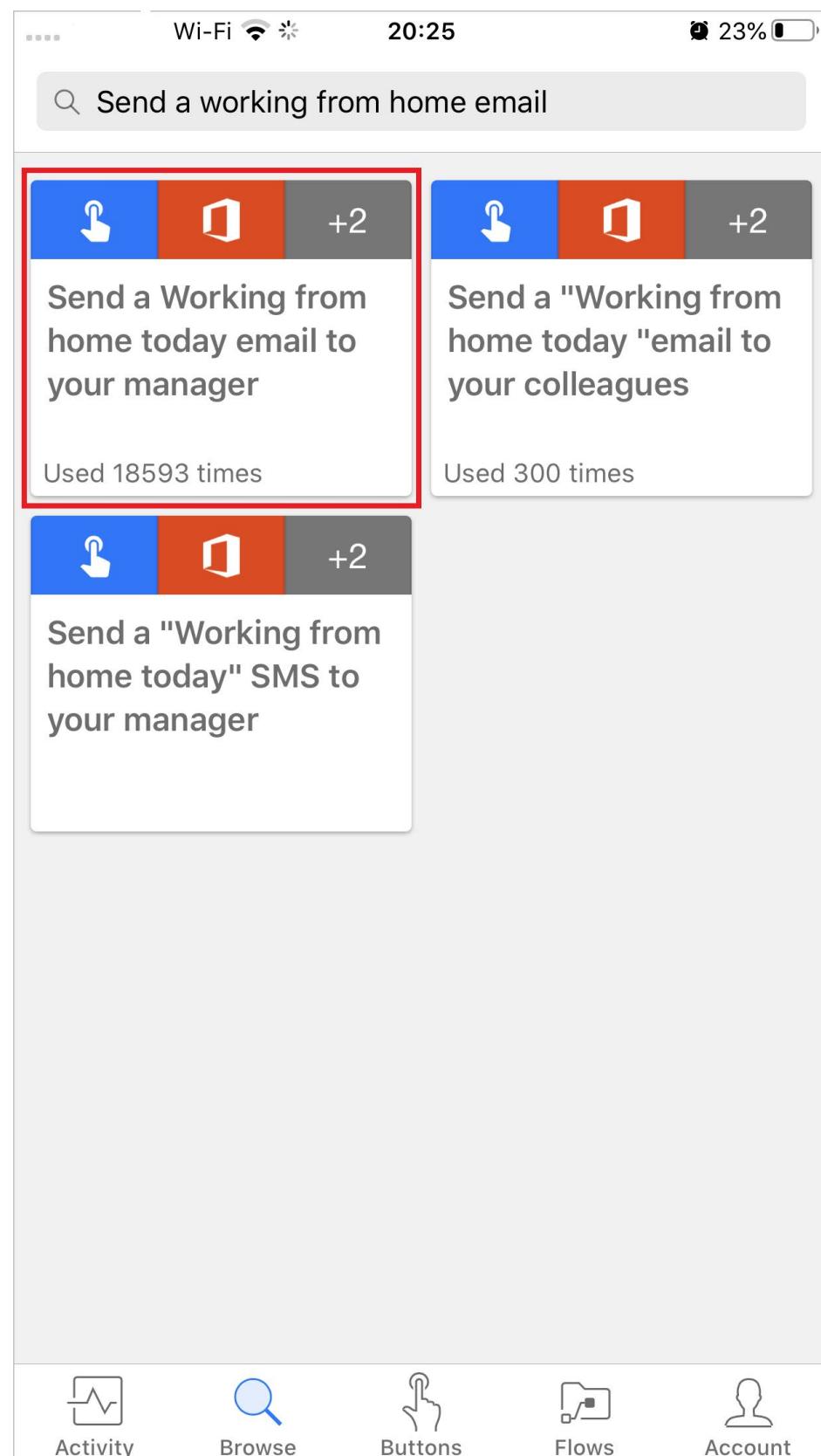
<sup>21</sup> <https://aka.ms/flowmobiledocsandroid>

<sup>22</sup> <https://aka.ms/flowmobiledocsios>

<sup>23</sup> <https://aka.ms/flowmobilewindows>



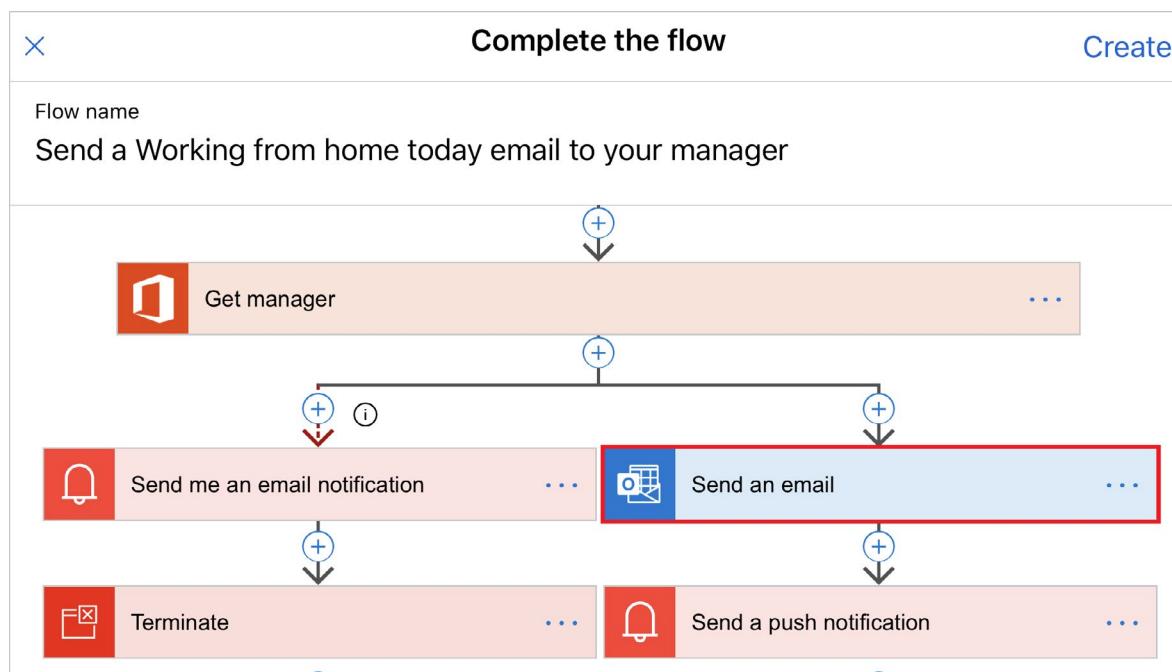
3. Search for and select the **Send a 'Working from home today' email to your manager** service.



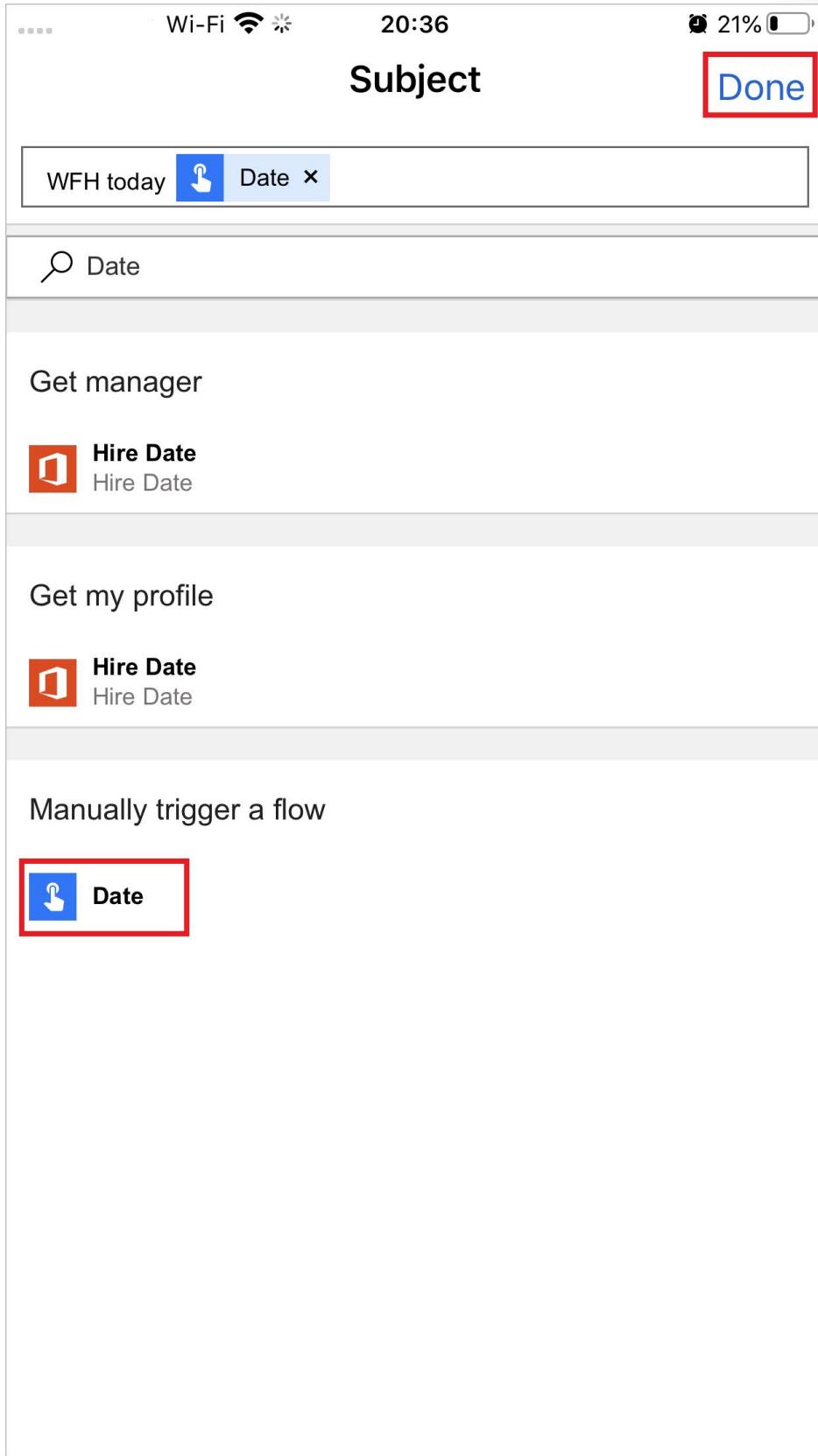
4. Select **Use this template**.

The screenshot shows the Microsoft Flows mobile application interface. At the top, there's a navigation bar with a back arrow, the text "Create flow", and a share icon. Below this is a large grid of four colored boxes: blue, orange, red, and blue. The blue box contains a hand icon pointing to a central white circle with an arrow. The orange box contains a Microsoft logo. The red box contains a bell icon. The bottom-right blue box contains a camera icon with a grid. Below the grid, the title "Send a Working from home today email to your manager" is displayed in bold. A descriptive text follows: "Email a custom message, such as WFH to your manager with a button click, and get a notification when the message has been sent." Below this, it says "By Microsoft" and "Used 18593 times". At the bottom of the screen, there's a blue button with the text "Use this template" enclosed in a red rectangular border. At the very bottom, there are five navigation icons: "Activity" (chart), "Browse" (magnifying glass), "Buttons" (hand), "Flows" (file folder), and "Account" (person icon).

5. Press the title of the **Send an email** card to expand your options. Note that this is a part of simultaneous actions. We've turned the screen so you can see more of the flow here.

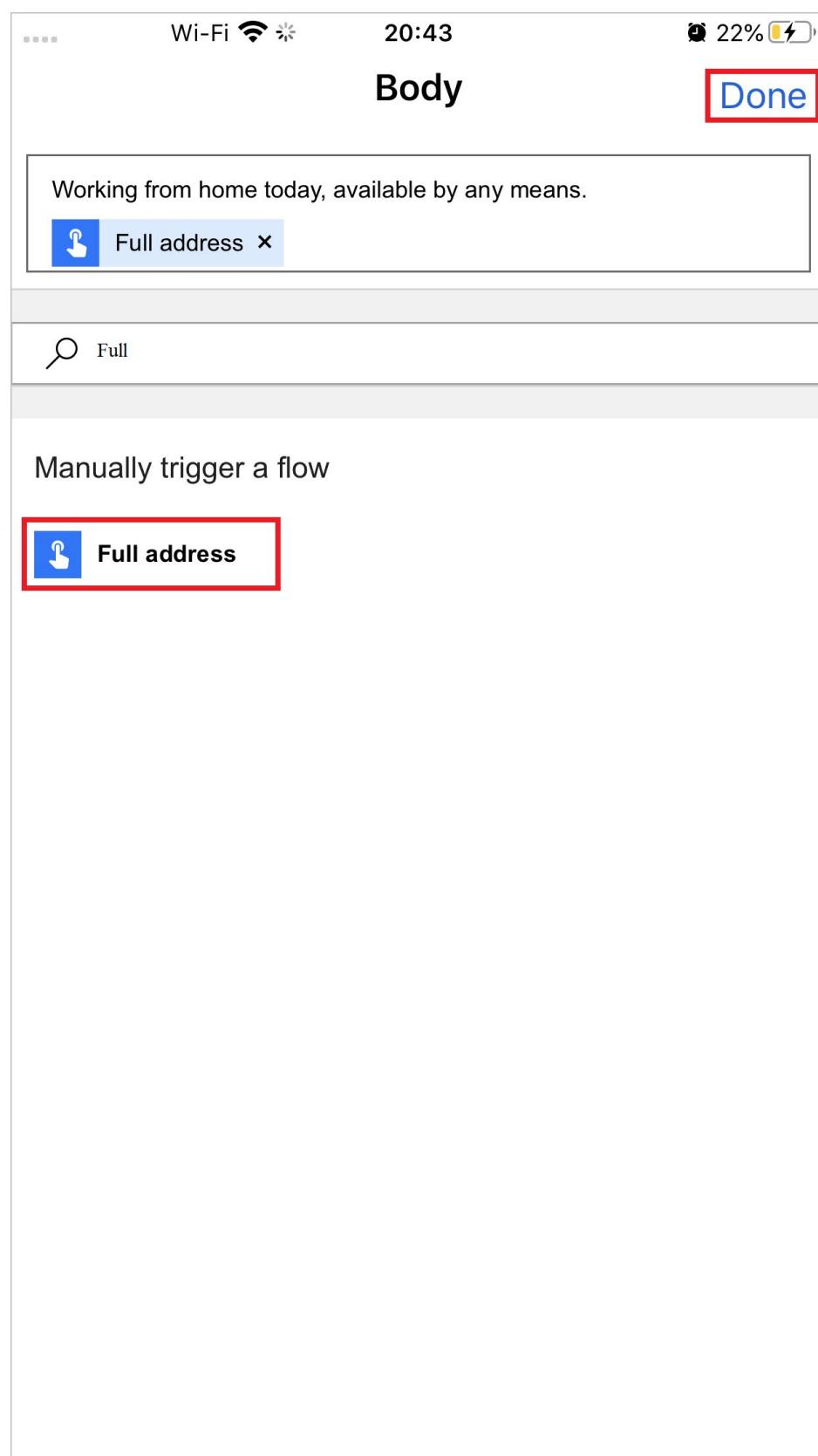


6. Select the **Subject** field, and enter *WFH today*. Notice that when you selected the **Subject** field, a list of tokens appeared. While the cursor is still in the **Subject** field, scroll through the list of tokens, and select **Date**. Notice that the date token now appears in the **Subject** field.

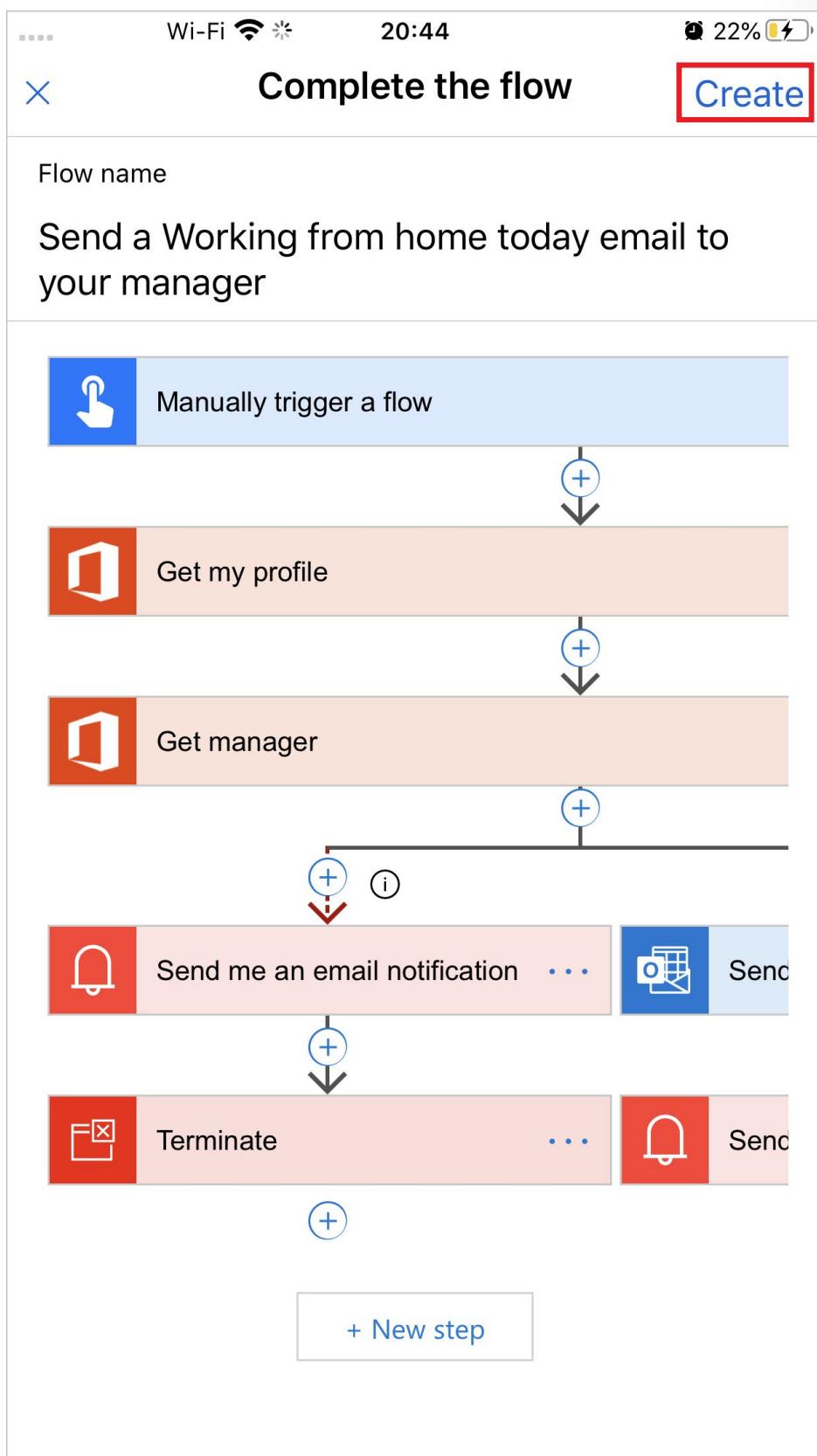


7. Scroll to the **Body** field, and select the default message so that you can add tokens there.

8. Select the **Full address** token.



9. Select **Create**.

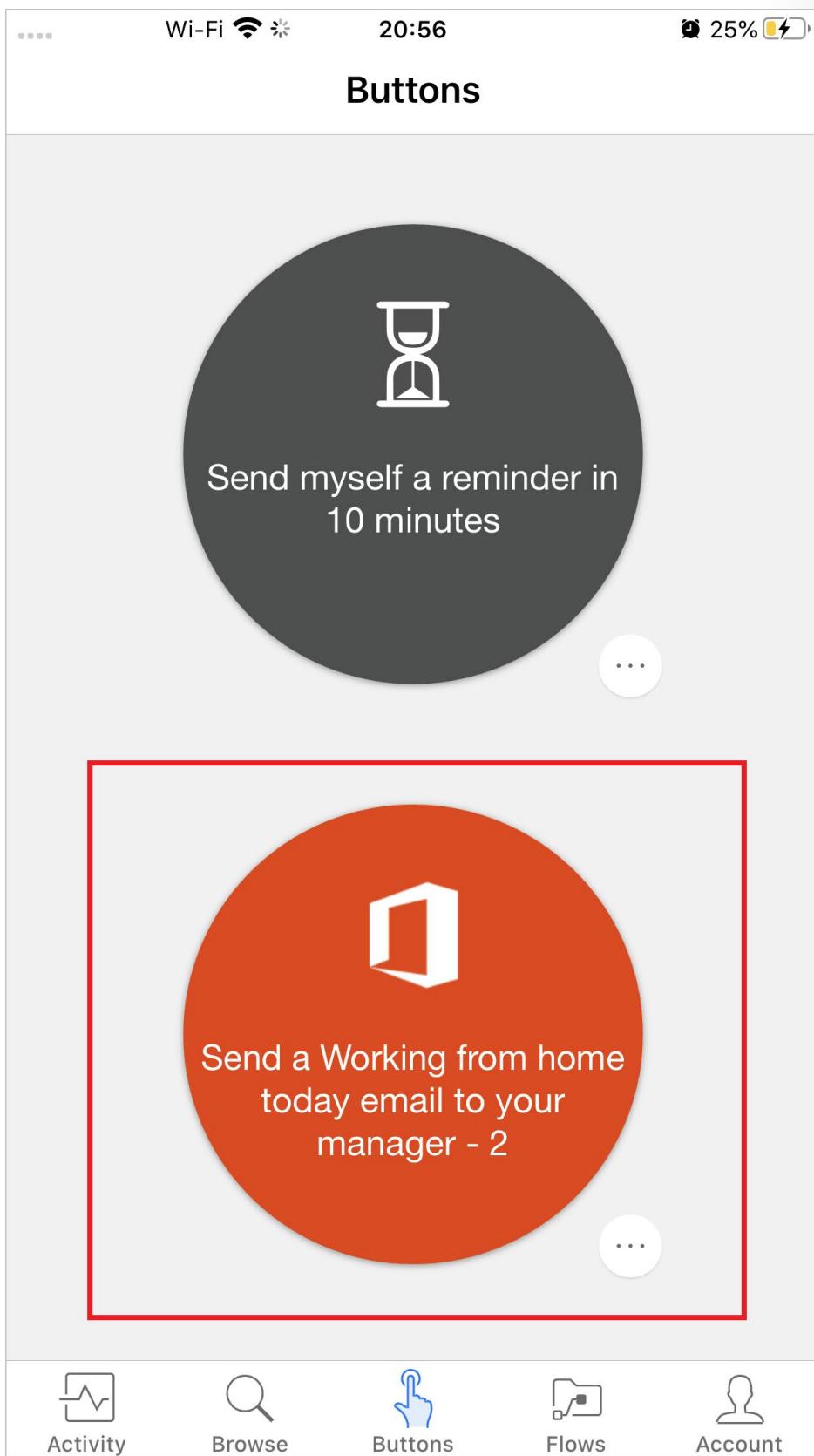


10. Select **Done**. Your button flow is now created.

## Run the button flow

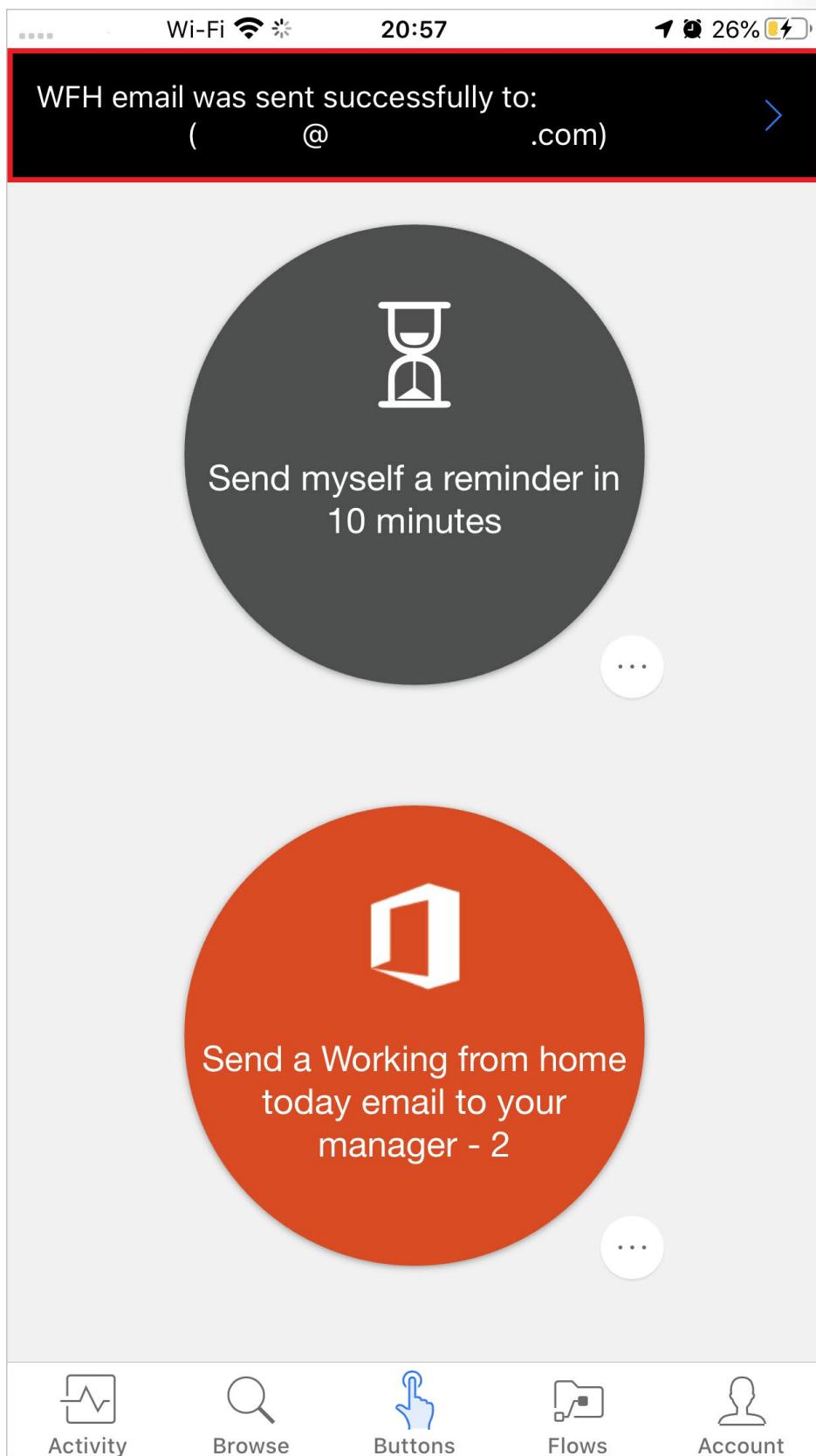
Note: This button flow will send your current location via email.

1. Select the **Buttons** tab at the bottom of the window. You'll see a list of the buttons that you have permissions to use. Select the button that represents the button flow that you just created:



2. Your device may prompt you to let the button flow access your device's location information. If necessary, press **Allow**

In a few moments, you'll notice that the email was sent to your boss.



Congratulations! You just created a button flow that uses both the **Date** and **Full address** trigger tokens.

# Exercise - Build a flow that accepts user input when run

You can customize button flows by letting the user provide specific details that will be used when the flow runs.

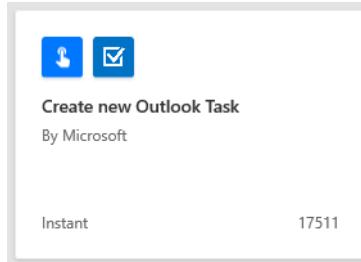
You can create a button flow either on the Power Automate website or in the mobile app for Power Automate. For this unit, you'll use the website.

## Prerequisites

You must have an account on the Power Automate website.

## Open the template

1. Launch Power Automate and sign in using your organizational account.
2. Select **Templates** and search *Create a New Outlook Task*.

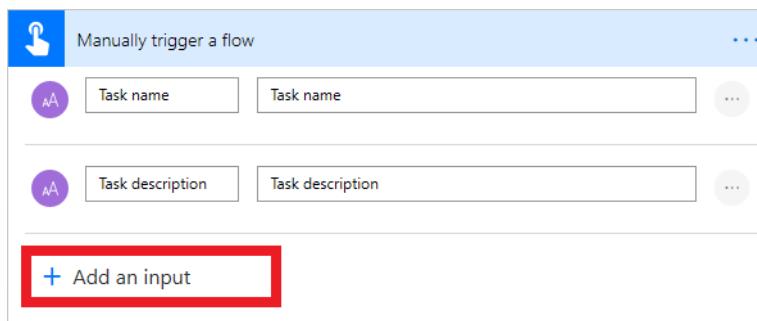


3. Sign in if you are prompted to do so and select **Create Flow**.

## Customize the user input

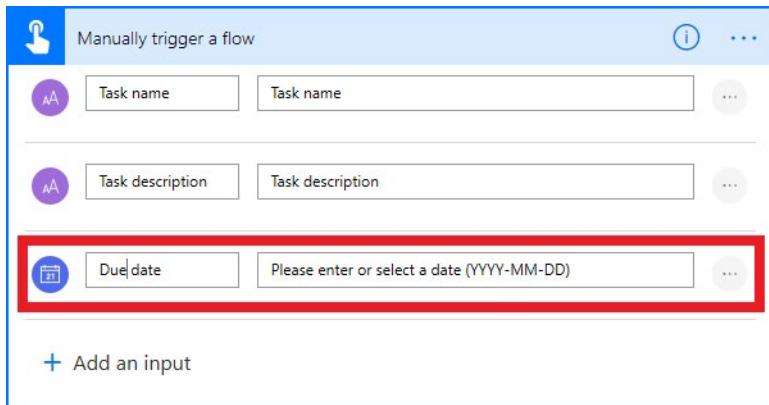
You'll notice that this flow already requests additional input, but let's add another field.

1. On the trigger card, select **Add an input**.



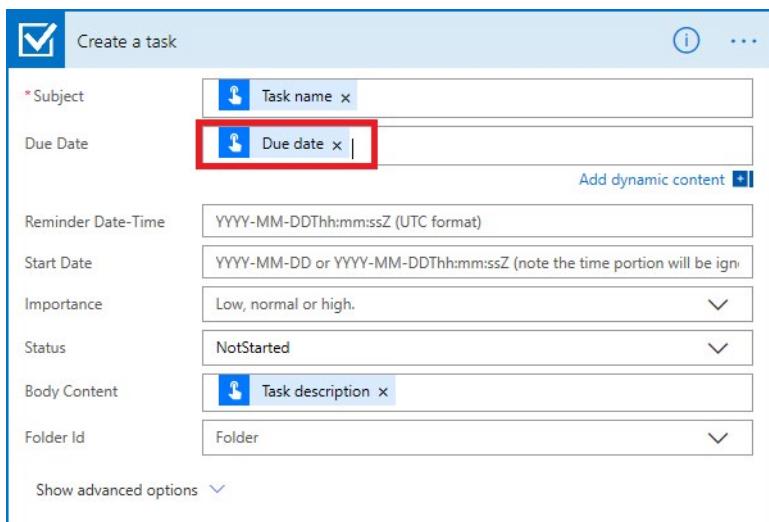
2. For each custom field that should be available when someone runs your flow, enter values in the **Input title** and **Input description** field.

In this example, you'll create one custom input field, **Due Date**.



## Customize the task

1. On the **Create a task** card, select the title bar to expand the card.
2. In the Due Date field, select **Due Date** from the dynamic content.

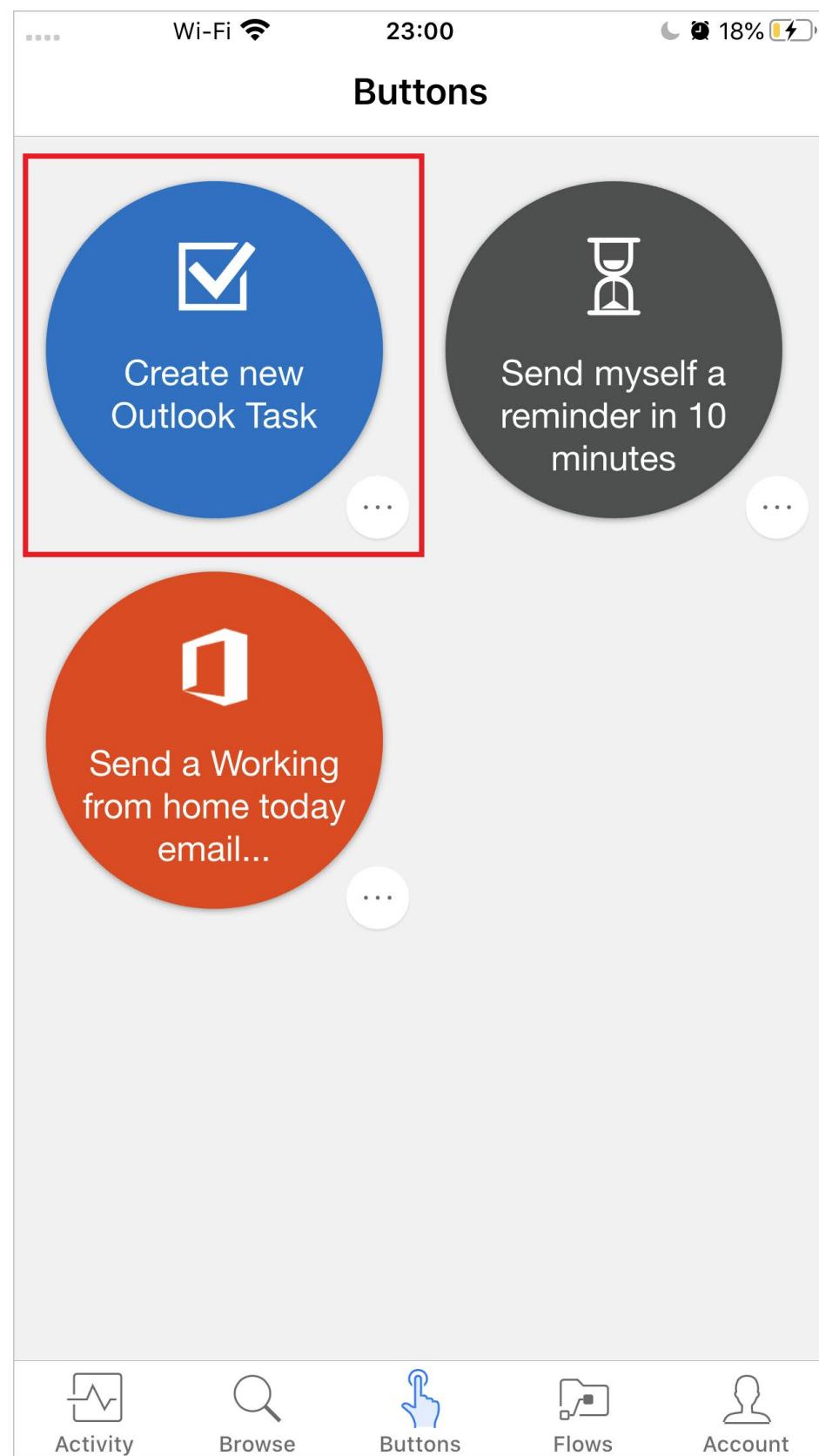


3. Select **Save**.

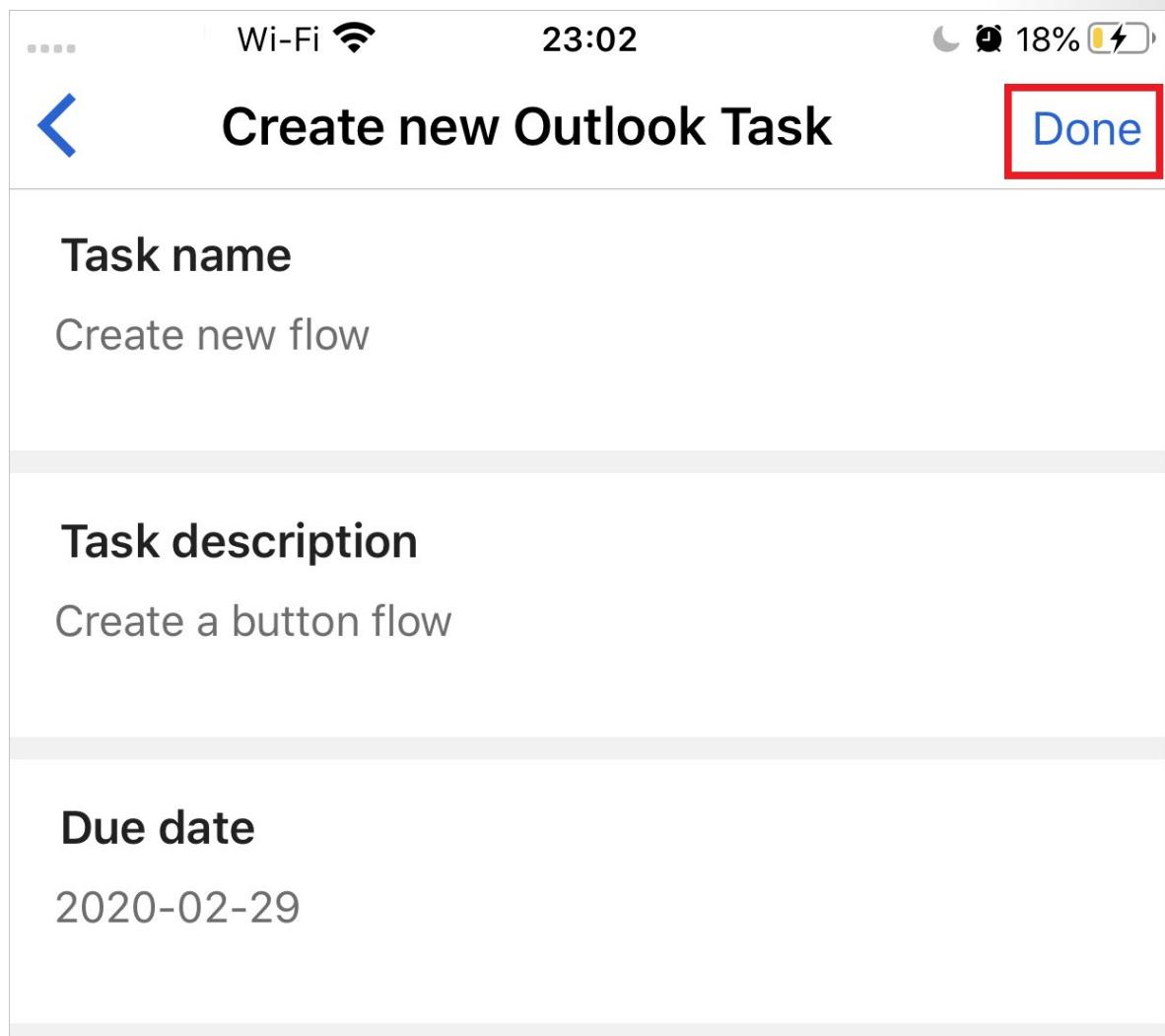
## Run the flow

You'll now use the mobile app for Power Automate to run the button flow that you just created. You'll provide all the user input that's needed to create a task with a name, description, and due date.

1. In the mobile app for Power Automate, select the **Buttons** tab at the bottom of the window, and then select the **Create new Outlook Task** button.



2. Enter the requested inputs and select **Done**.



The flow runs.

3. Select the **Activity** tab at the bottom of the window to view the results.

The screenshot shows a mobile application interface for managing Power Automate flows. At the top, there are status icons for Wi-Fi, signal strength, time (23:02), battery level (19%), and a lightning bolt icon. Below the header, there are two tabs: "Feed" and "Approvals". The "Feed" tab is selected.

The main content area is divided into sections by date: "TODAY" and "YESTERDAY".

**TODAY Section:**

- Create new Outlook Task**: This flow was triggered once and completed in seconds. It is highlighted with a red border. The thumbnail icon shows a blue square with a white checkmark and a smaller blue square with a white lock icon.
- Get a push notification when you receive an email from your boss**: This flow has run 20 times and completed 19 minutes ago. The thumbnail icon shows a blue square with a white envelope icon, a red square with a white Microsoft logo, and a red square with a white bell icon.
- Boss said: Optometric Billing Solutions - PowerBI Help and...**: A message from the "Boss" flow. It includes a "Go to Outlook Office 365" link and was posted 6 hours ago. The thumbnail icon is a light gray square.

A "Boss Refer" section is visible on the right side of this card.

**YESTERDAY Section:**

- Get a push notification when you receive an email from your boss**: This flow has run 8 times and completed at 11:38 PM. The thumbnail icon is identical to the one in the TODAY section.
- Send a Working from home today email to your manager - 2**: This flow ran 1 time and completed at 8:57 PM. The thumbnail icon shows a 2x2 grid of icons: blue (lock), red (Microsoft), red (bell), and blue (envelope).
- WFH email was sent successfully to: Jennifer Hammond...**: A message from the "Send a Working from home today email to your manager" flow. It was posted 8:57 PM.

At the bottom of the screen, there is a navigation bar with five items: "Activity" (highlighted with a red box), "Browse", "Buttons", "Flows", and "Account".

4. To view the detailed results of the flow run, select the **Create a task** step.

Activity details

Create new Outlook Task  
Flow successfully ran 1 time

LAST RUN SECONDS AGO

Run details

Manually trigger a flow → Create a task

Inputs

Subject: Create new flow

Due Date: 2020-02-29

Status: NotStarted

Body Content: Create a button flow

Activity Browse Buttons Flows Account

This screenshot shows the 'Activity details' screen for a flow named 'Create new Outlook Task'. The flow has run successfully once. The last run was just now (seconds ago). The run details show two steps: 'Manually trigger a flow' followed by 'Create a task'. Both steps were completed successfully (indicated by green checkmarks). The 'Create a task' step triggered the creation of a new Outlook task with the subject 'Create new flow', due date '2020-02-29', and status 'NotStarted'. The body content of the task is set to 'Create a button flow'. At the bottom, there are navigation links for Activity, Browse, Buttons, Flows, and Account.

Now you can not only run button flows with the already available information, but also request inputs from the user.

## Exercise - Learn to build a flow that runs at recurring time intervals

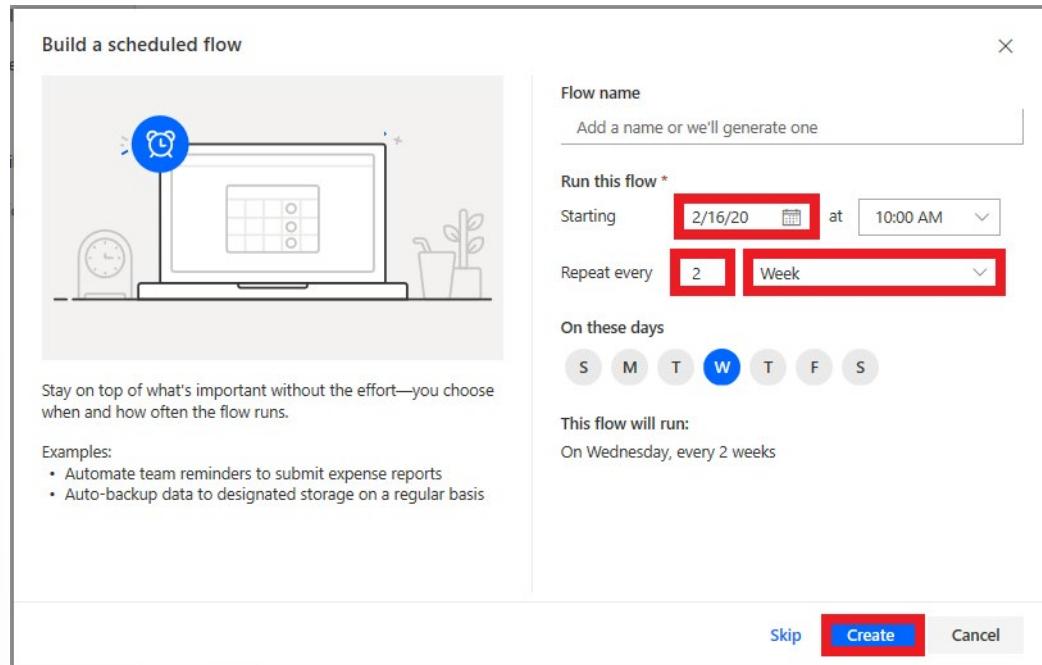
You can create a flow that performs one or more tasks (for example, sending a report by email) on a specific schedule:

- Once a day, an hour, or a minute
- On a date that you specify
- After a number of days, hours, or minutes that you specify

### Create the flow

1. Launch Power Automate and sign in using your organizational account.
2. In the left pane, select **My flows**.
3. Select **New**, and then select **Scheduled—from blank**.
4. In the dialog box, specify the flow's name and how often the flow should run.

For example, if you want the flow to run every two weeks, enter 2 in the **Interval** field, and select **Week** in the **Frequency** field. You can also specify the day of the week your flow should run. The text at the bottom of the dialog box explains your inputs in plain language.



5. Once you are satisfied with your inputs, select **Create**.

## Specify advanced options

- Follow the steps in the previous section. Once your flow is created, select the title of the **Recurrence** card to expand it. Select **Edit** and then **Show advanced options**.

*Note:*

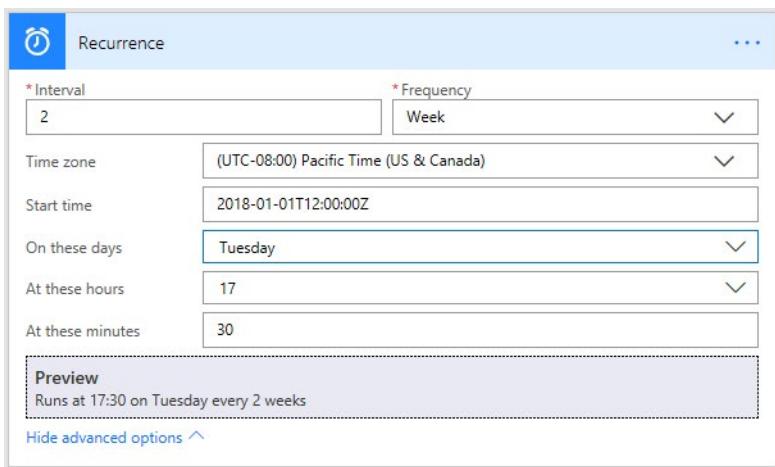
The advanced options vary, depending on the value of the **Interval** and **Frequency** fields. If the dialog box that you see doesn't match the graphic that follows, make sure that the **Interval** and **Frequency** fields are set to the same values that are shown in the graphic.

- Here you can specify a time zone to reflect the local time zone, Universal Coordinated Time (UTC), or another time zone.

- If you selected **Day** in the **Frequency** field, you can specify the time of day when the flow should run.

If you selected **Week**, specify the day or days of the week when the flow should run, and the time or times of day when the flow should run.

For example, set up the flow as shown in the following graphic to start it no earlier than noon (Pacific time) on Monday, January 1, 2018, and to run it every two weeks, at 5:30 PM (Pacific time) on Tuesday.

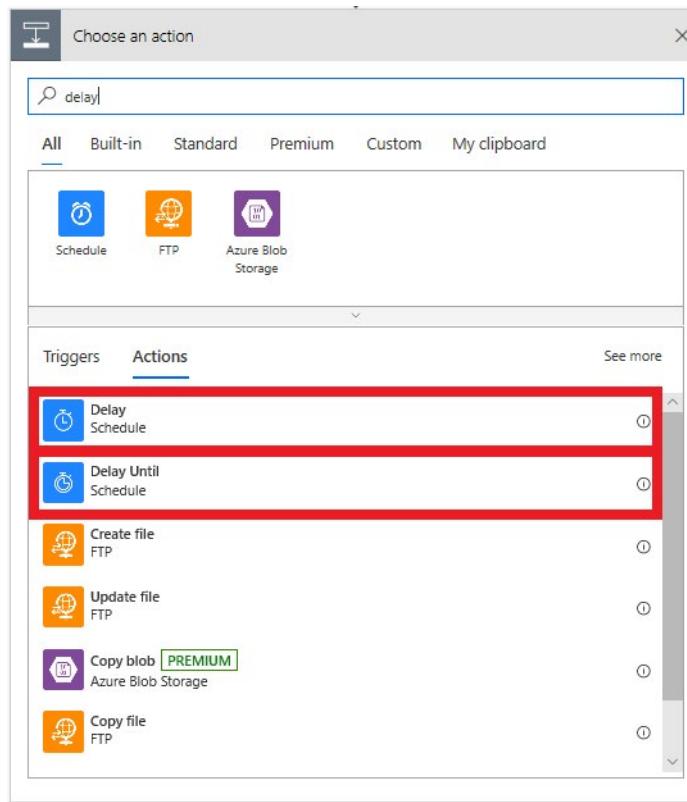


- Add the action or actions that the flow should take.

## Delay the flow

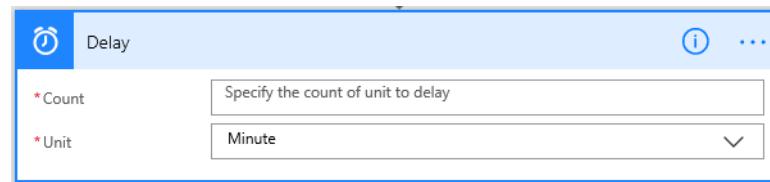
Next, you'll learn how to delay a flow.

- On the top navigation bar, select **My flows**, and then select **Automated—from blank**.
- In the **Search all triggers** field, enter *Twitter*, and then select **Twitter - When a new tweet is posted**. select **Create** and finish the steps in the flow.
- Select **New step**.
- In the list of actions, search **Delay** and select either **Delay** or **Delay until**.

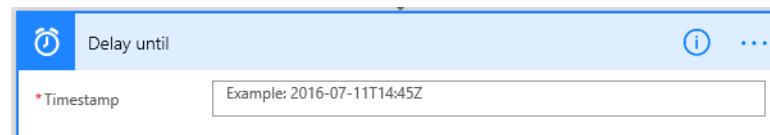


5. Follow one of these steps, depending on the action that you just selected:

- If you selected **Delay**, specify a count and a unit of time, like second, minute, or hour.



- If you selected **Delay until**, specify a date in this format: YYYY-MM-DDTHH:MM:SSZ



## Exercise - Build a flow that runs when an event in Dynamics 365 occurs

You can create flows that start when an event occurs in Microsoft Dynamics 365 or some other service. These flows then perform an action in that service.

In Power Automate, you can set up automated workflows between your favorite apps and services to sync files, get notifications, collect data, and more.

In this unit, we'll look to build two flows:

- The first flow creates a task in Dynamics 365 when a lead is created in another instance of Dynamics 365.
- The second flow copies a list item to Microsoft Planner when a task is created in Dynamics 365.

Here are some other examples of flows that you can create by using Dynamics 365:

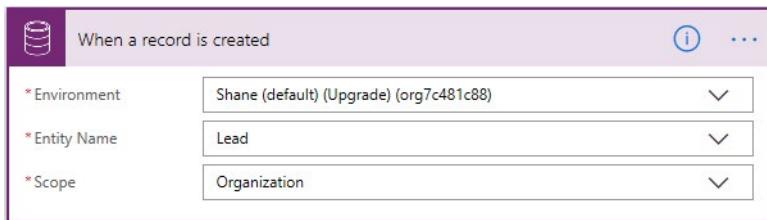
- Create a list item in Microsoft SharePoint when an object is created in Dynamics 365.
- Create Dynamics 365 lead records from a Microsoft Excel table.
- Copy Dynamics 365 accounts to customers in Microsoft Dynamics 365 for Finance and Operations.

Please note: To invoke a flow trigger, the Dynamics 365 customer engagement entity that's used with the flow must have change tracking turned on. For more about how to turn on change tracking, see **Enable change tracking to control data synchronization<sup>24</sup>**.

## Example one: Create a task from a lead

This example shows how to create a task in Dynamics 365 whenever a lead is created in another instance of Dynamics 365.

1. Sign in to **Power Automate<sup>25</sup>** by using your organizational account.
2. In the left pane, select **My flows**.
3. Select **New**, and then select **Automated—from blank**.
4. In the list of flow triggers, select **Common Data Service - When a record is created**. Dynamics 365 keep information in the Common Data Service, so we will use this connector throughout.
5. If you're prompted to sign in to Common Data Service, do so.
6. In the **Environment** field, select the instance where the flow should listen.
7. In the **Entity Name** field, select the entity to listen to. This entity will act as a trigger that starts the flow.  
For this unit, select the **Lead** entity.
8. For Common Data Service, you also need to specify the **Scope**. This will determine if your flow runs if you create a new record, if a new record is created by a user within your business unit, or if a new record is created by any user in your organization. For this example, choose **Organization**.



9. Select **New step**.
10. Select **Common Data Service – Create a new record**.

<sup>24</sup> <https://docs.microsoft.com/dynamics365/customer-engagement/admin/enable-change-tracking-control-data-synchronization>

<sup>25</sup> <https://ms.flow.microsoft.com>

11. In the **Environment** field, select the environment where the flow should create the record. Note that this doesn't have to be the same environment that the event is triggered from.
12. In the **Entity Name** field, select the entity that will create a record when the event occurs.  
For this unit, select the **Tasks** entity.
13. More fields appear on entity selection. Select the **Subject** field. A dynamic content pane appears, where you can select fields from the previous steps.  
For this unit, select **Full name**.

The screenshot shows the 'Create a new record' dialog in Microsoft Power Automate. The dialog has a header 'Create a new record' with a database icon. Below the header are several input fields:

- \* Environment: (default) (Upgrade)
- \* Entity Name: Tasks
- \* Subject: Full name (with a small 'x' icon to close the dynamic content pane)
- Description: Type additional information to describe the task.
- Due Date: Enter the expected due date and time.
- Duration: Type the number of minutes spent on the task. The duration is used in reporti
- Priority Value: Select the priority so that preferred customers or critical issues are hand ✓
- Owner: Enter the user or team who is assigned to manage the record. This field is upd
- Owner Type: Enter the user or team who is assigned to manage the record. This field ✓
- Regarding: Choose the record that the task relates to.
- Regarding Type: Choose the record that the task relates to. ✓

At the bottom left, there is a link 'Show advanced options' with a dropdown arrow. The dynamic content pane is partially visible behind the 'Subject' field, showing options like 'See more' and 'Company Name', 'Customer', 'Description', and 'Email'.

*Tip:*

In the dynamic content pane, select **See more** to see more fields that are associated with the entity. For example, you can also insert the **Company Name**, **Customer**, **Description**, or **Email** field for the lead into the **Subject** field for the task.

14. Select **Save**.

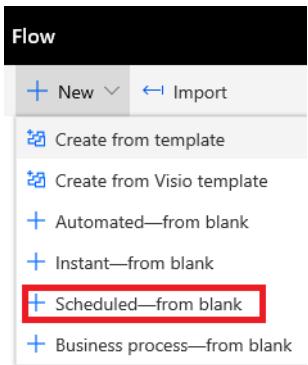
## Example two: Create a Planner task from a Dynamics 365 task

This example shows how to create a task in Microsoft Planner whenever a task is created in Dynamics 365. Planner is a service that you can use to create to-do lists, add reminders, and track errands.

1. Sign in to **Power Automate**<sup>26</sup> by using your organizational account.
2. In the left pane, select **My flows**.
3. Select **New**, and then select **Automated-from blank**.

---

<sup>26</sup> <https://ms.flow.microsoft.com>



4. In the list of flow triggers, select **Common Data Service - When a record is created**. Dynamics 365 keep information in the Common Data Service, so we will use this connector throughout.
5. If you're prompted to sign in to Common Data Service, do so.
6. In the **Environment** field, select the instance where the flow should listen.
7. In the **Entity Name** field, select the entity to listen to. This entity will act as a trigger that starts the flow.  
For this unit, select the **Tasks** entity.
8. In the **Scope** field, choose **Organization**.
9. Select **New step**.
10. In the search field, enter *Planner*, and then select **Planner – Create a task**.
11. See **create a task parameters<sup>27</sup>** for information about the fields.
12. In the **Title** field, add **Subject** from the dynamic content pane.
13. Select **Save**.

## Limitations of trigger-based logic

Triggers like **When a record is created**, **When a record is updated**, and **When a record is deleted** will start your flow within a few minutes after the event occurs. But in rare cases, it might take up to two hours for your flow to be triggered.

When the trigger occurs, the flow receives a notification, but the flow runs on the data that exists when the action runs. For example, if your flow is triggered when a new record is created, and you update the record twice before the flow runs, your flow runs only once with the latest data.

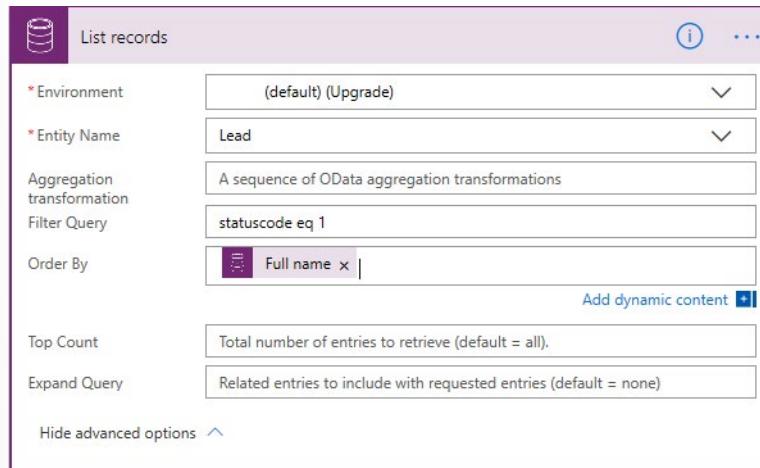
## Specify advanced options

When you add a step to a flow, you can select **Show advanced options** to add a filter or order-by query that controls how the data is filtered in the flow.

For example, you can use a filter query to retrieve only active contacts, and you can order them by last name. Enter the `statuscode eq 1` Open Data Protocol (OData) filter query, and select **Full name** in

<sup>27</sup> <https://docs.microsoft.com/connectors/planner/#create-a-task>

the dynamic content pane. For more about filter and order by queries, see [MSDN: \\$filter<sup>28</sup>](#) and [MSDN: \\$orderby<sup>29</sup>](#).



## Best practices for advanced options

When you add a value to a field, you must match the field type, regardless of whether you enter a value or select a value in the dynamic content pane.

Field type	How to use	Where to find	Name	Data type
Text fields	Text fields require a single line of text or dynamic content that's a text-type field. Examples include the <b>Category</b> and <b>Sub-Category</b> fields.	<b>Settings \ Customizations \ Customize the System \ Entities \ TaskFields</b>	category	Single Line of Text
Integer fields	Some fields require an integer or dynamic content that's an integer-type field. Examples include the <b>Percent Complete</b> and <b>Duration</b> fields.	<b>Settings \ Customizations \ Customize the System \ Entities \ Task \ Fields</b>	percent complete	Whole Number

---

<sup>28</sup> [https://msdn.microsoft.com/library/gg309461.aspx#Anchor\\_1](https://msdn.microsoft.com/library/gg309461.aspx#Anchor_1)

<sup>29</sup> [https://msdn.microsoft.com/library/gg309461.aspx#Anchor\\_2](https://msdn.microsoft.com/library/gg309461.aspx#Anchor_2)

Field type	How to use	Where to find	Name	Data type
Date fields	Some fields require a date that's entered in <i>mm/dd/yyyy</i> format or dynamic content that's a date-type field. Examples include the <b>Created On</b> , <b>Start Date</b> , <b>Actual Start</b> , <b>Last on Hold Time</b> , <b>Actual End</b> , and <b>Due Date</b> fields.	<b>Settings \ Customizations \ Customize the System \ Entities \ Task \ Fields</b>	created on	Date and Time
Fields that require both a record ID and a lookup type	Some fields that reference another entity record require both the record ID and the lookup type.	<b>Settings \ Customizations \ Customize the System \ Entities \ Account \ Fields</b>	accountid	Primary Key

## Exercise - Build a flow that uses SQL

This unit shows how to create a flow that monitors a source for new or changed items, and then copies those changes to a destination. You might create a flow of this type if your users enter data in one location, but your team needs that data in a different location or format.

In this unit, you'll copy data from a **Microsoft SharePoint list**<sup>30</sup> (the source) to a **Microsoft Azure SQL Database**<sup>31</sup> table (the destination).

Keep in mind that you can copy data over more than **275 services**<sup>32</sup> that Power Automate supports.

**Important:** Changes that you make in the destination aren't copied back to the source, because two-way synchronization isn't supported. If you try to set up two-way synchronization, you'll create an infinite loop where changes are sent endlessly between the source and destination.

## Prerequisites

- Access to a data source and a destination. This unit doesn't include the steps to create the source and destination.
- Access to **Power Automate**<sup>33</sup>.
- A basic understanding of how your data is stored.
- Familiarity with the basics of creating flows. For this unit, it's assumed that you know how to perform these actions.

<sup>30</sup> <https://support.office.com/article/SharePoint-lists-1-An-introduction-f11cd5fe-bc87-4f9e-9bfe-bbd87a22a194>

<sup>31</sup> <https://docs.microsoft.com/azure/sql-database/sql-database-technical-overview>

<sup>32</sup> <https://flow.microsoft.com/connectors/>

<sup>33</sup> <https://flow.microsoft.com>

*Tip:*

Column names in the source and destination don't need to match, but you must provide data for all required columns when you insert or update an item. Power Automate identifies the required fields for you.

## Quick overview of the steps

If you're comfortable with Power Automate, use these quick steps to copy data from one data source to another.

1. Identify the source that you'll monitor and the destination that you'll copy changed data to. Confirm that you have access to both the source and the destination.
2. Identify at least one column that uniquely identifies items in the source and destination. In the example that follows, we use the **Title** column, but you can use any columns.
3. Set up a trigger that monitors the source for changes.
4. Search the destination to check whether the changed item exists.
5. Use a condition like this:
  - If the new or changed item doesn't exist in the destination, create it.
  - If the new or changed item exists in the destination, update it.
6. Trigger your flow, and then confirm that new or changed items are being copied from the source to the destination.

*Note:*

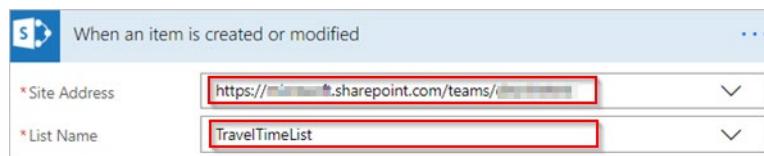
If you haven't previously created a connection to SharePoint or a SQL Database, follow the instructions when you're prompted to sign in.

Here are the detailed steps to create the flow.

## Monitor the source for changes

First, we'll set up the SharePoint site to monitor changes.

1. Launch Power Automate and sign in using your organizational account.
2. In the left pane, select **My flows**.
3. Select **New**, and then select **Automated—from blank**.
4. Name your flow. Search all triggers triggers for the **SharePoint - When an item is created or modified** trigger and select it. Press **Create**.
5. On the **When an item is created or modified** card, enter the site address, and then select the name of the SharePoint list that your flow monitors for new or updated items.



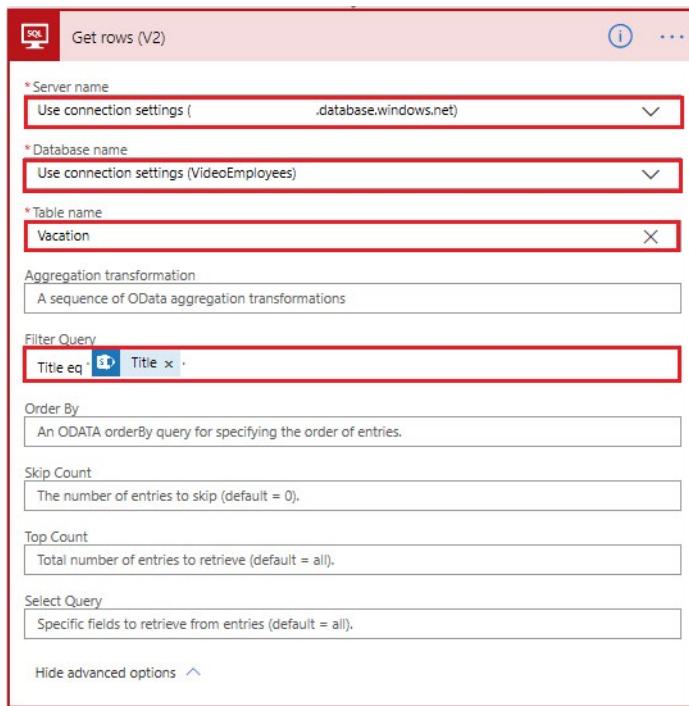
## Search the destination for the new or changed item

Next, we'll use the **SQL Server - Get rows** action to search the destination for the new or changed item.

1. Select **New step**.
2. Under **Choose an action**, search for *Get rows*, and then select **SQL Server - Get rows (V2)**.
3. Set the **Server name**, **Database name**, and **Table name** for the table you wish to monitor.
4. Select **Show advanced options**.
5. In the **Filter Query** box, enter *Title eq* followed by a space and a single quotation mark ('). Then select the **Title** token in the dynamic content list, and enter another single quotation mark.

This step assumes that you're matching the titles of the rows in the source and destination.

The **Get rows** card should now look like this image.



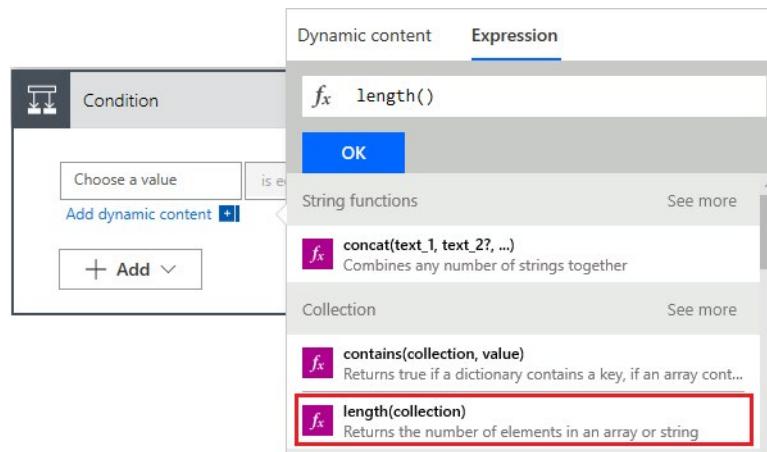
## Check whether the new or changed item was found

Next, we'll check whether the new or changed item was found.

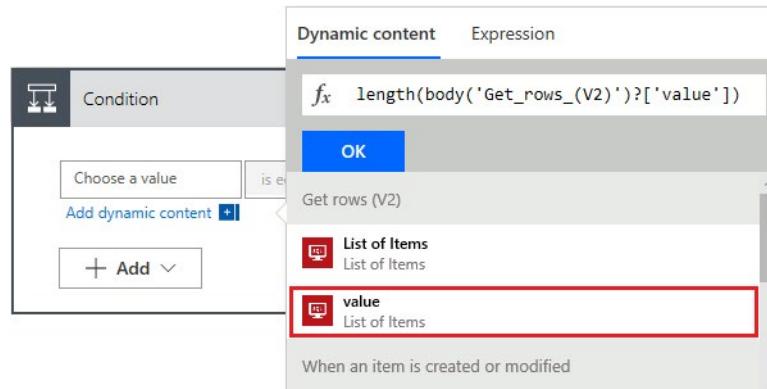
1. Select **New step**, and then select **Condition**.
2. On the **Condition** card, select the field on the left.

The **Add dynamic content from the apps and connectors used in this flow** list opens.

3. Select **Expression** and choose **length**. Your cursor should be between the parentheses in the equations.



- Without leaving the open pane, select **Dynamic content**. In the **Get rows** category, select **value**.

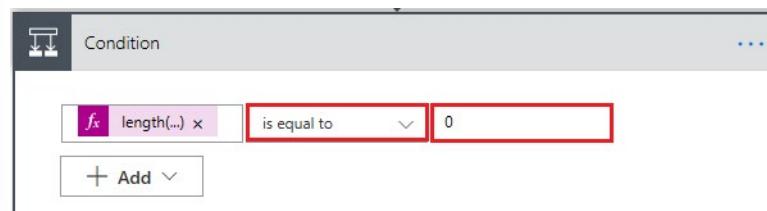


*Tip:*

Confirm that you've selected **value** in the **Get rows** category. Don't select **value** in the **When an item is created or modified** category.

- In the field in the center, select *is equal to*.
- In the field on the right, enter 0 (zero).

The **Condition** card should now look like this image.



*Tip:*

The addition of the `length()` function lets the flow check the **value** list and check whether it has any items.

When your flow gets items from the destination, there are two possible outcomes.

Outcome	Next step
The item exists.	Update the item.
The item doesn't exist.	Create a new item.

## Create the item in the destination

If the item doesn't exist in the destination, create it by using the **SQL Server - Insert row** action.

1. On the **If yes** branch of the condition, select **Add an action**, search for *insert row*, and then select **SQL Server - Insert row (V2)**.
2. On the **Insert row** card, set the **Server name**, **Database name**, and **Table name** for the table to insert the new item into (the information you entered above).

The **Insert row** card expands and shows all fields in the selected table. Fields that are marked with an asterisk (\*) are required and must be filled in for the row to be valid.

3. Select each field that you want to fill in, and enter the data.

You can manually enter the data, select one or more tokens in the dynamic content pane, or enter any combination of text and tokens into the fields.

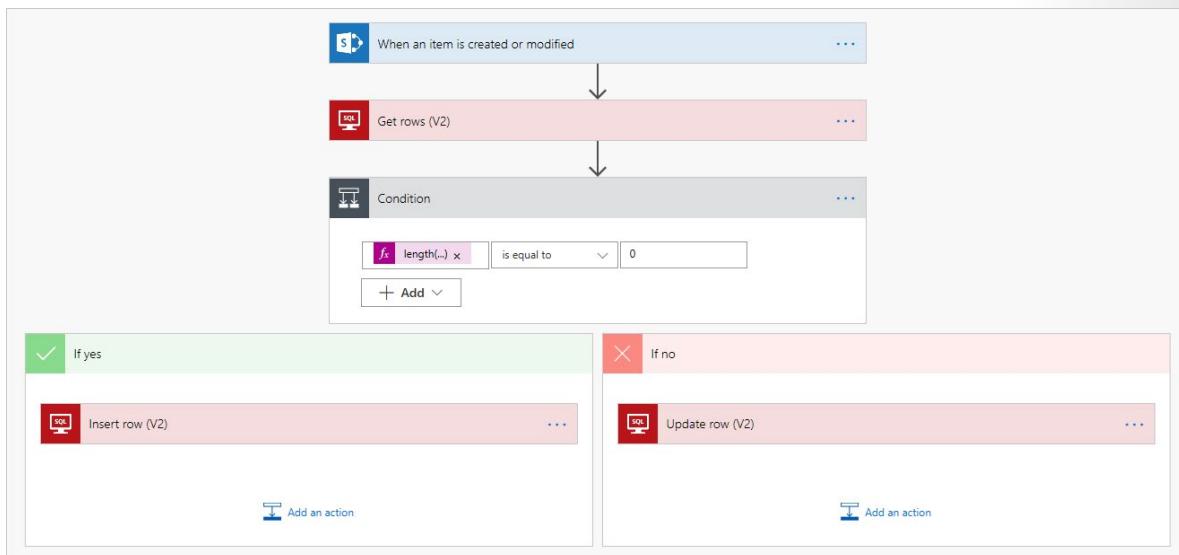
*Note:*

The **Insert row** and **Update row** cards show the names of the columns in the SQL Database table that's being used in the flow. Therefore, the cards that are shown in the images in this procedure might differ from the cards that you see.

## Update the item in the destination

Next, if the item exists in the destination, update it with the changes.

1. Add the **SQL Server - Update row** action to the **If no** branch of the condition.
2. Select **Save** to save the flow.



Now, whenever an item in your SharePoint list (the source) changes, your flow is triggered. It either inserts a new item or updates an existing item in SQL Database (the destination).

**Note:**

Your flow isn't triggered when an item is deleted from the source. If this scenario is important to you, consider adding a separate column that indicates when an item is no longer needed.

## Exercise - Integrate Power Apps, Power Automate, and Sharepoint

The data that fuels business processes is often buried in separate systems that are difficult to connect to and navigate. This is one reason why business processes don't stop becoming complex, and why people rarely stop worrying about them.



But the Power Platform with Power Automate, Power Apps, and Power BI along with tools like Microsoft SharePoint make it all easier. Together, these apps and services provide these advantages:

- The data can easily be tapped.
- Critical business decisions can be made more quickly and more intelligently.
- People can worry less about what their data is doing and concentrate more on moving their business forward.

This unit gives an overview of:

- Integration of Power Automate with Power Apps.
- Integration of Power Automate and Power Apps with SharePoint for easy sharing of data in lists.

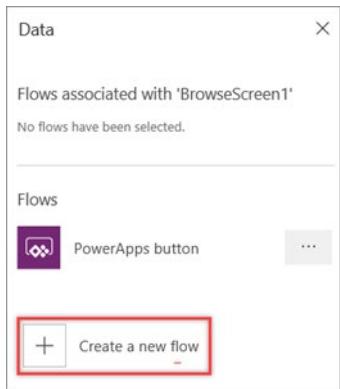
## Add a flow in Power Apps

Adding a flow to a Power Apps application is very straightforward.

1. Go to <https://make.powerapps.com>, and sign in by using your organizational account.
2. Open your app for editing.
3. On the **Action** tab, select **Power Automate** on the toolbar.



4. In the **Data** dialog box, select **Create a new flow**.



Power Automate is started and shows templates filtered by the trigger **PowerApps Button**.

Template	Provider	Last Modified	Popularity
PowerApps button	By Microsoft	Instant	204600
Send approval and follow up via email	By Microsoft	Instant	69735
Send approval email and follow up via email	By Microsoft	Instant	15615
Add an item to SharePoint and send an email	By Microsoft	Instant	5587

For more about how to create flows, see [Create a flow from a template in Power Automate<sup>34</sup>](#).

## Add a Power Apps application from Power Automate

You can also go in the other direction. You can start in Power Automate and then select a template to add an app from Power Apps.

1. Launch Power Automate and sign in using your organizational account.
2. In the left pane, select **Templates**.
3. Select one of the many Power Apps templates.

To see all the Power Apps templates that are available, you can search for *Power Apps*.

Once the template is selected and opened, you can start building your flow.

For more about how to create apps by using Power Apps, see [Create a canvas app from a template in Power Apps<sup>35</sup>](#).

<sup>34</sup> <https://docs.microsoft.com/flow/get-started-logic-template>

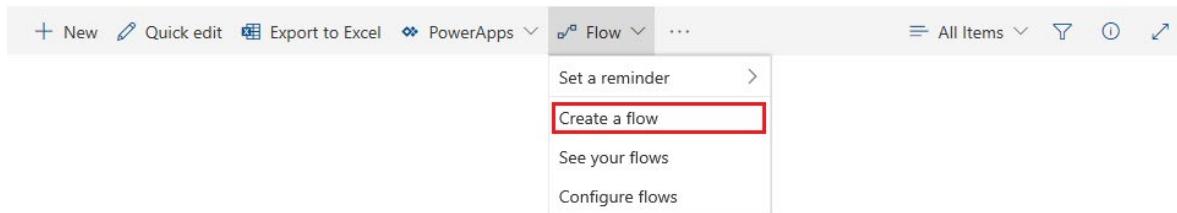
<sup>35</sup> <https://docs.microsoft.com/powerapps/maker/canvas-apps/get-started-test-drive>

## Integration of SharePoint with Power Automate

Customers regularly exchange data between SharePoint lists and other systems to support business processes. These scenarios become more powerful through the deep integration of Power Automate with SharePoint lists.

Power Automate allows for automating the exchange of workflows and data between SharePoint and a variety of Microsoft and third-party services. You can create and start flows directly from a SharePoint list, and store and change that data in SharePoint.

1. From a SharePoint list, select **Flow** on the top toolbar, and then select **Create a flow**.



2. In the **Create a flow** pane, select the template to use.

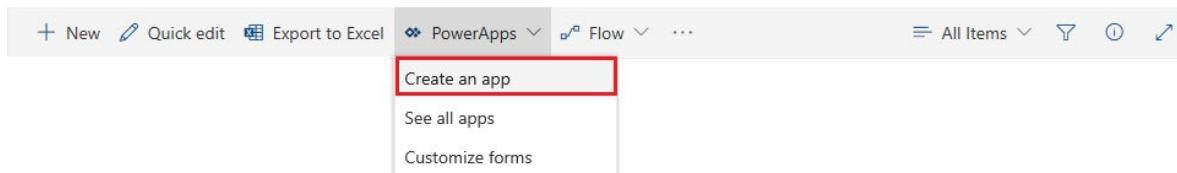
Power Automate is started, and you can finish creating the flow.

## Integration of SharePoint with Power Apps

Power Apps lets you connect to, create, and share business apps on any device in minutes. You can build efficient mobile forms and apps directly from a SharePoint list, without writing a line of code.

Power Apps and Power Automate share a common connector framework that lets you weave in dozens of data sources that are located on premises or in the cloud. These data sources include Microsoft Exchange, Microsoft SQL Server, Microsoft Dynamics, Salesforce, Google, MailChimp, Twitter, and Wunderlist.

1. From a SharePoint list, select **Power Apps** on the top toolbar, and then select **Create an app**.



2. In the **Create an app** pane, enter a name for your app, and then select **Create**.

Power Apps is started, and you can finish creating the app.

## Exercise - Create a business process flow

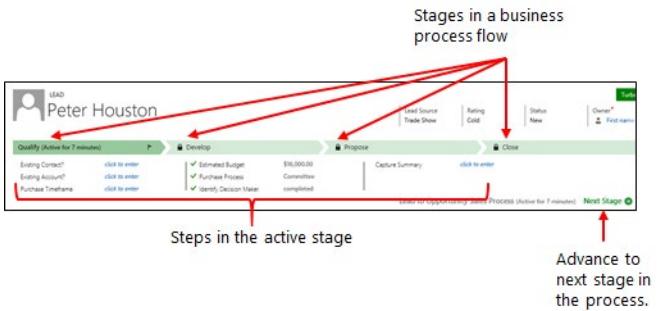
This unit shows how to create a business process flow by using Microsoft Power Apps.

For more about how to create a mobile task flow, see [Create a mobile task flow<sup>36</sup>](#).

When a user starts a business process flow, the process bar at the top of the page shows the stages and steps of the process.

---

<sup>36</sup> <https://docs.microsoft.com/dynamics365/customer-engagement/customize/create-mobile-task-flow>



*Tip:*

When you create a business process flow definition, you can define who has privileges to create, read, update, or delete instances of the business process flow. For example, for service-related processes, you might give customer service reps full access to change the business process flow instance. But you might give sales reps just read-only access to the instance, so that they can monitor post-sales activities for their customers. To set security for a business process flow definition that you create, select **Enable Security Roles** on the action bar.

## Create a business process flow

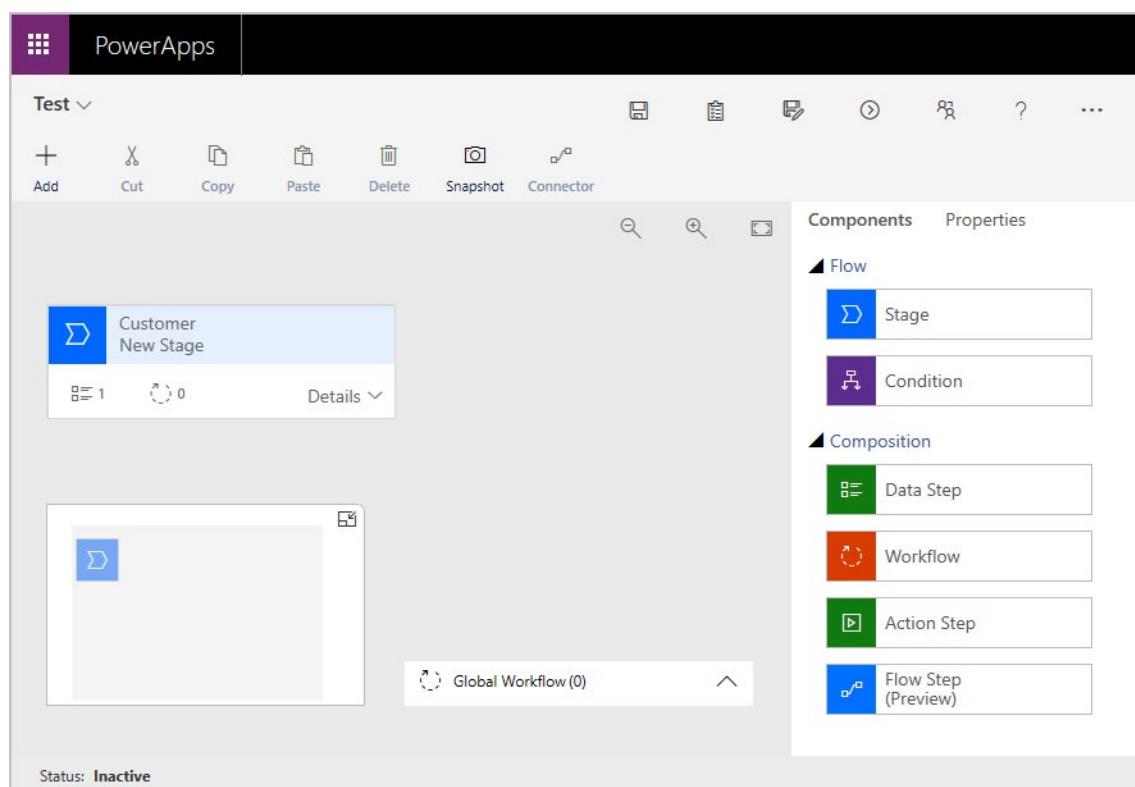
1. Launch Power Automate and sign in using your organizational account.
2. In the left pane, select **Flows**.
3. On the top bar, select **New** and **Business process—from blank**.
4. In the **Build a business process flow** pane, fill in the required fields:
  - **Flow name:** The display name of the process doesn't have to be unique, but it should be meaningful for people who must choose a process. You can change this name later.
  - **Name:** A unique name that's based on the display name. You can change the name when you create the process, but you can't change it after the process has been created. Power Automate can generate this for you.
  - **Common Data Service entity:** Select the entity on which to base the process.

The entity that you select affects the fields that are available for steps that can be added to the first stage of the process flow. If you can't find the entity that you want, make sure that the **Business process flows (fields will be created)** option is set for the entity in the entity definition. You can't change the entity after you save the process.

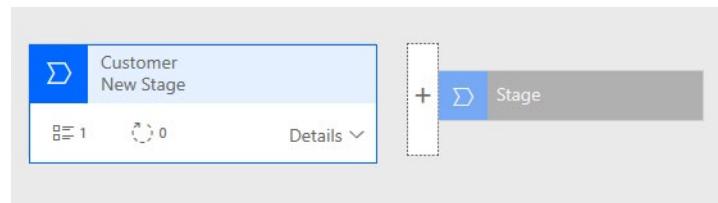
5. Select **Create**.

The new process is created, and the business process flow designer is started. The designer page has three sections:

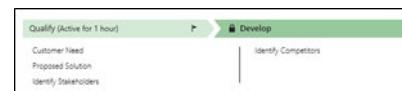
- On the left, a single stage named *Customer New Stage* has already been created for you.
- Beneath this stage is the mini map, which lets you see the whole process or quickly go to a part of the process.
- On the right are components that you can drag to the designer. You can also set properties to create a business process flow.



6. Add stages so that users can proceed from one business stage to another in the process:
  1. Drag the **Stage** component from the **Components** tab to the plus sign (+) in the designer.



2. Select the stage, and then, on the **Properties** tab on the right, set the properties:
  1. Enter a display name.
  2. Optional: Select a category for the stage (for example, **Qualify** or **Develop**). This category appears as a chevron on the process bar.

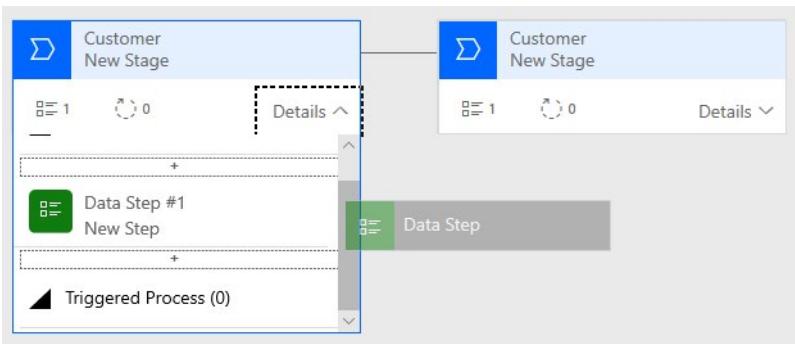


3. When you've finished setting the properties, select **Apply**.
7. Add steps to each stage:

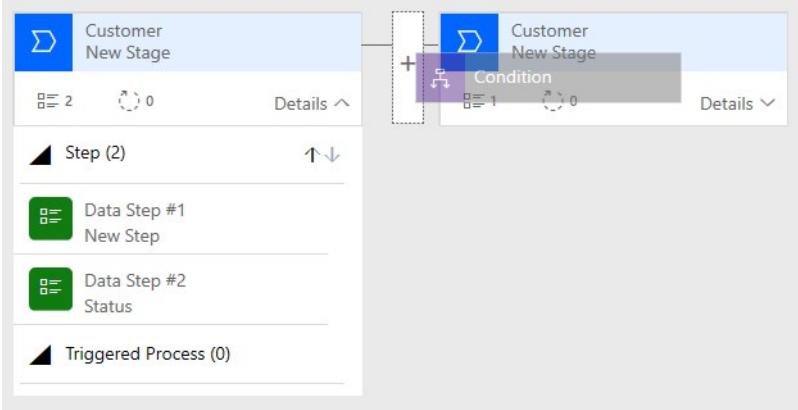
*Tip:*

To see the steps in a stage, select **Details** in the lower-right corner of the stage.

1. Drag the **Step** component from the **Components** tab to the stage.



2. Select the step, and then, on the **Properties** tab, set the properties:
  1. Enter a display name for the step.
  2. If users should be able to enter data to finish a step, select the appropriate field in the drop-down list.
  3. If users must fill in the selected field to finish the step before they can proceed to the next stage of the process, select **Required**.
  4. When you've finished, select **Apply**.
8. Add a branch (condition) to the process:
  1. Drag the **Condition** component from the **Components** tab to the plus sign (+) between two stages.



2. Select the condition, and then, on the **Properties** tab, set the properties. When you've finished, select **Apply**.
9. Add a workflow to the process:
  1. Drag the **Workflow** component from the **Components** tab to either a specific stage or the **Global Workflow** item:
    - Drag the **Workflow** component to a specific stage if the workflow should be triggered when the process enters or exits that stage. The **Workflow** component must be based on the same primary entity as the stage.
    - Drag the **Workflow** component to the **Global Workflow** item if the workflow should be triggered when the process is activated or archived (that is, when the status changes to

**Completed or Abandoned).** The **Workflow** component must be based on the same primary entity as the process.

2. Select the Workflow, and then, on the **Properties** tab, set the properties:
  1. Enter a display name.
  2. Select when the workflow should be triggered.
  3. Search for an existing on-demand active workflow that matches the stage entity, or create a workflow by selecting **New**.
  4. When you've finished, select **Apply**.



10. To validate the business process flow, select **Validate** on the action bar.

11. To save the process as a draft while you continue to work on it, select **Save** on the action bar.

**Important:** No one can use a process while it's a draft.

1. To activate the process and make it available to your team, select **Activate** on the action bar.
2. To define who has privileges to create, read, update, or delete the business process flow instance,



select **Edit Security Roles** on the action bar. For example, for service-related processes, you might give customer service reps full access to change the business process flow instance. But you might give sales reps just read-only access to the instance, so that they can monitor post-sales activities for their customers.

1. In the **Security Roles** pane, select the name of a role to open the details page for that role.
2. On the **Business Process Flows** tab, select options to assign the role appropriate privileges for the business process flow.

*Note:*

By default, the System Administrator and System Customizer security roles have access to new business process flows.



#### Security Role: Common Data Service User

Entity	Create	Read	Write	Delete	Append	Append To	Assign	Share
Expired Process	●	●	●	●	●	●	●	●
Idea to Project Business Process	○	○	○	○	○	○	○	○
My BPF	○	○	○	○	○	○	○	○
New Process	●	●	●	●	●	●	●	●
Challenge Management Process	○	○	○	○	○	○	○	○
Translation Process	●	●	●	●	●	●	●	●

3. Select **Save**.

*Tip:*

Keep these tips in mind as you work on your business process flow in the designer:

- To take a snapshot of everything in the business process flow designer, select **Snapshot** on the action bar. This option is useful if you want to share and get comments about the process from a team member.

- Use the mini map to quickly go to different parts of the process. This option is useful when you have a complicated process that scrolls off the screen.
- To add a description of the business process, select the arrow beside the process name in the upper-left corner of the page. You can enter up to 2,000 characters in the description field.

## Edit a business process flow

You can edit the business process flow after it has been created.

1. On the Power Apps main page, select **Flows** in the left pane.
2. In the list of business process flow, select the flow that you created, and then select **Edit** at the top.

Keep the following points in mind when you edit the stages of a business process flow:

- Business process flows can have up to 30 stages.
- You can add or change the following properties of a stage:
  - **Stage Name:** You can change the stage name after you create the stage.
  - **Entity:** You can change the entity for any stage except the first one.
  - **Stage Category:** A category lets you group stages by the type of action. It's useful for reports that will group records by the stage that they're in. The options for the stage category come from the Stage Category global option set. You can add more options to this global option set and change the labels of existing options. You can also delete options, but we recommend that you keep the existing options. If you delete an option, you won't be able to add it back later. If you don't want an option to be used, change the label to *Do not use*.
- **Relationship:** Enter a relationship when the preceding stage in the process is based on a different entity than the current stage. For the current stage, select **Select relationships**, and then specify the relationship that should be used when the flow moves between the two stages. We recommend that you specify relationships, because they provide the following benefits:
  - Attribute maps are often defined for relationships. These attribute maps automatically carry over data between records. Therefore, they help minimize the amount of data entry that's required.
  - When you select **Next Stage** on the process bar for a record, any records that use the relationship are listed in the process flow. Therefore, the reuse of records in the process is promoted. In addition, you can use workflows to automate the creation of records. Users then just have to select the workflow instead of creating a record. Therefore, the process is streamlined.
- **Set Process Flow Order:** If you have more than one business process flow for an entity (record type), you must specify which process is automatically assigned to new records. On the action bar, select **Order Process Flow**. For new records or records that don't already have a process flow associated with them, the first business process flow to which a user has access will be used.
- **Enable Security Roles:** A user's access to a business process flow depends on the privileges that are defined for the business process flow in the security role that's assigned to the user. By default, only the System Administrator and System Customizer security roles can view a new business process flow.

## Exercise - Create a business process flow that has conditions

Business process flows guide you through the different stages of sales, marketing, or service processes, toward completion. For a simple process, a linear business process flow is a good option. But in more complex scenarios, you can use an enhanced business process flow that branches into different directions, depending on conditions within the flow.

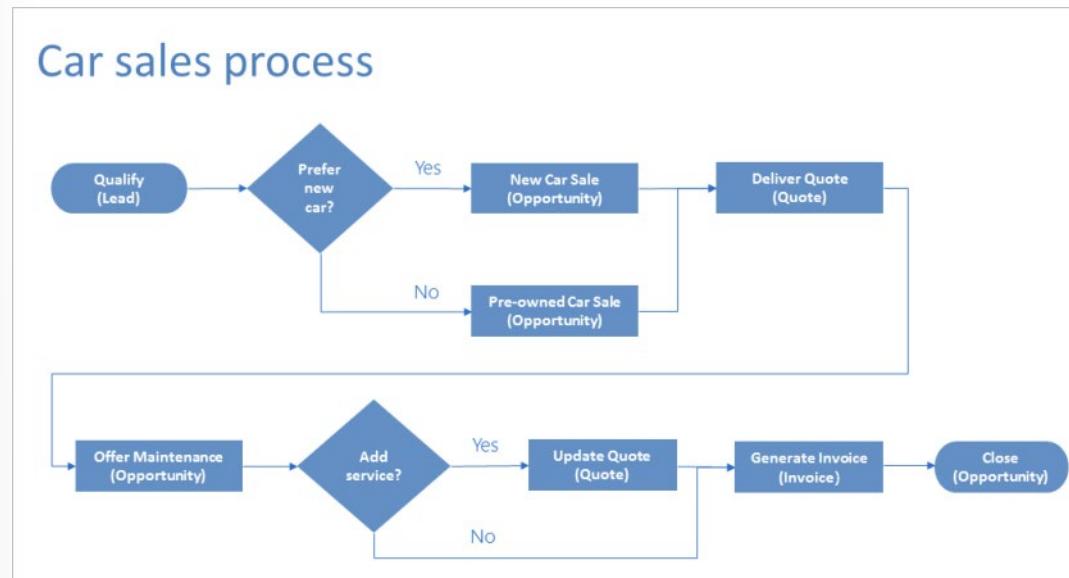
Branches are automatically selected in real time, based on rules that are defined in the process definition.

- If you have **Create** permissions on business process flows, you can use **If-Else** logic to create business process flows that have multiple branches.
- The branching condition can be formed from multiple logical expressions that use a combination of **AND** or **OR** operators.

For example, for the process of selling cars, you can set up a single business process flow that starts with a common qualification stage but then splits into separate branches, based on a rule:

- One branch manages the case of a customer who prefers a new car or a pre-owned car.
- Another branch manages the case of a customer whose budget is above or below \$20,000.
- A third branch might be for purchasing or declining a maintenance or service plan.

The following diagram shows a business process flow that has branches.



## Guidelines for business process flows that have branches

Keep the following points in mind when you design a business process flow that has branches:

- A process can span a maximum of five unique entities.
- You can use a maximum of 30 stages per process and a maximum of 30 steps per stage.
- Each branch can be no more than five levels deep.
- Branching rules must be based on the steps in the stage that immediately precedes them.

- You can combine multiple conditions in a rule by using the AND operator or the OR operator, but not both.
- When you define a process flow, you can optionally select an entity relationship. This entity relationship must a one-to-many (1:N) relationship.
- More than one active process can run concurrently on the same data record.
- When branches are merged, either all the peer branches must be merged to a single stage, or each peer branch must end the process. A peer branch can't merge with other branches and end the process at the same time.

*Note:*

- An entity that's used in the process can be revisited multiple times (that is, there can be multiple closed entity loops).
- A process can go back to the previous stage, regardless of the entity type. For example, if the active stage is **Deliver Quote** on a quote record, process users can move the active stage back to the **Propose** stage on an opportunity record.

## Dynamics 365 customer engagement example: Car selling process flow that has two branches

Let's look at an example of a business process flow that has two branches. In this example, the business process flow is used for sales of new and pre-owned cars.

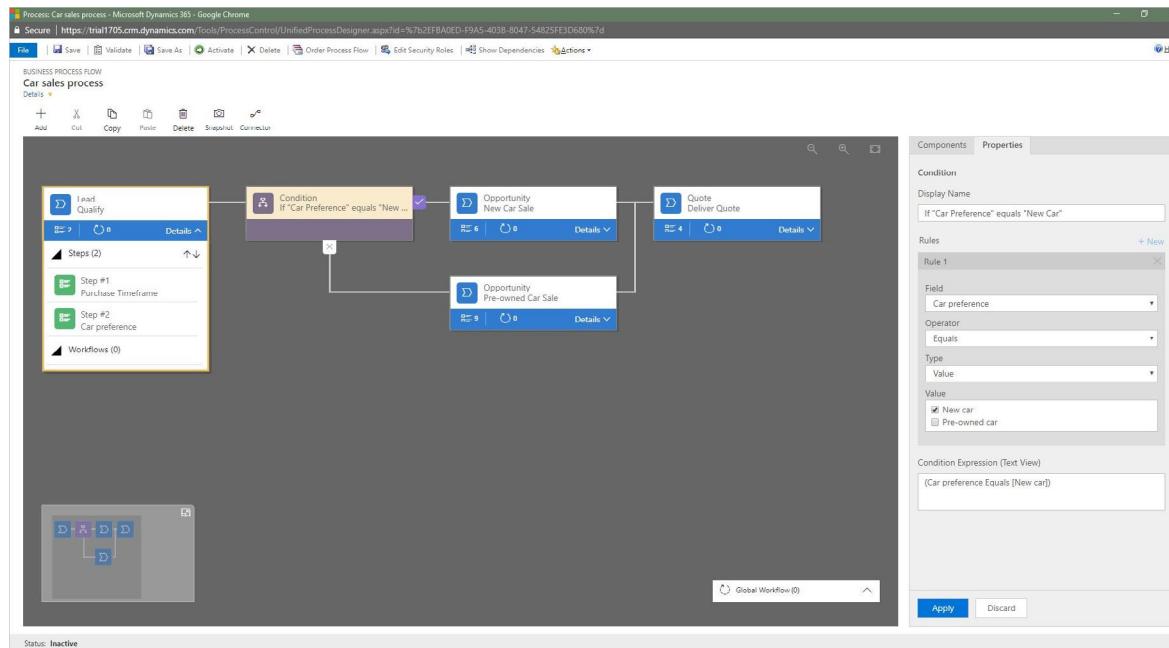
First, we'll create a process named **Car Sales Process**.

1. Launch Power Automate and sign in using your organizational account.
2. In the left pane, select **Flows**.
3. On the top bar, select **NewBusiness process—from blank**.
4. In the **Create business process flow** pane, fill in the required fields:
  - **Display name:** Enter *Car sales process*.
  - **Name:** A unique name that's based on the display name that you entered. You can change the name when you create the process, but you can't change it after the process has been created.
  - **Common Data Service entity:** Select the *Lead* entity. The entity that you select affects the fields that are available for steps that can be added to the first stage of the process flow. If you can't find the Lead entity, make sure that the **Business process flows (fields will be created)** option is set for that entity in the entity definition. You can't change the entity after you save the process.
5. Select **Create**.
6. Once the new page spins up in PowerApps, add the first stage to the process, name the stage *Qualify*, and add two data steps to it: *Purchase Time frame* and *Car Preference*.
7. After the common Qualify stage, split the process into two separate branches by adding a **Condition** component:
  1. Set up the condition with rules that meet your business requirements.
  2. To add the first branch, which will be run when the condition is satisfied, add a **Stage** component to the **Yes** path of the **Condition** component.

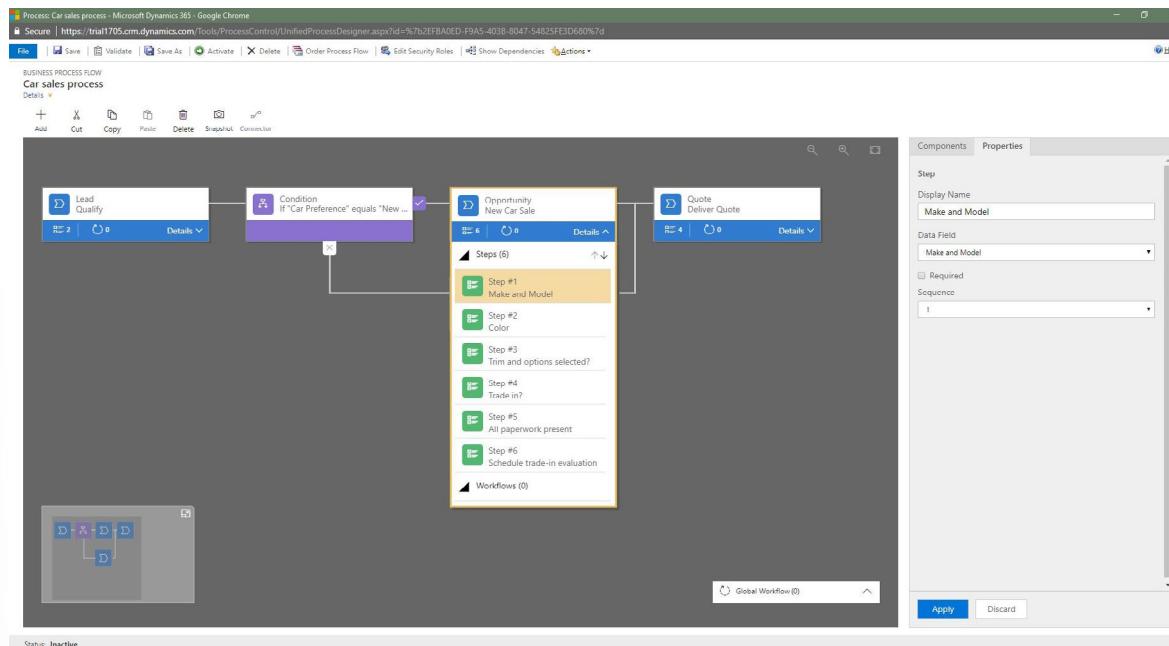
3. To add the second branch, which will be run when condition isn't satisfied, add a **Stage** component to the **No** path of the **Condition** component.

*Tip:*

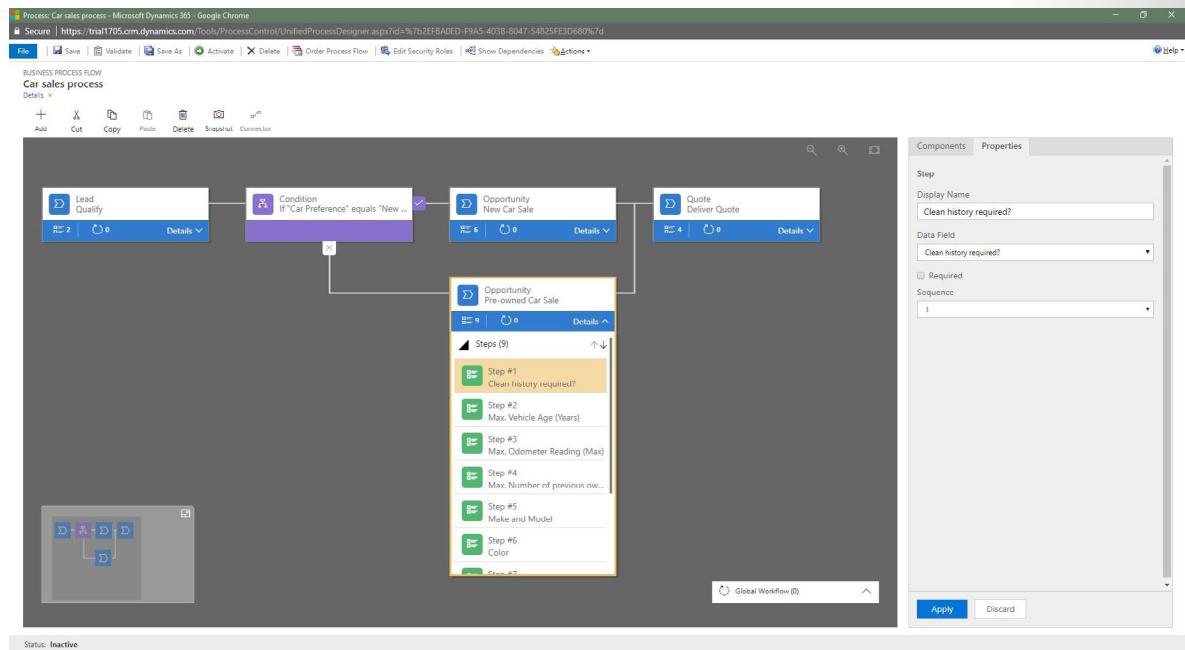
To create more complex branching, you can add another a **Condition** component to the **No** path of an existing **Condition** component.



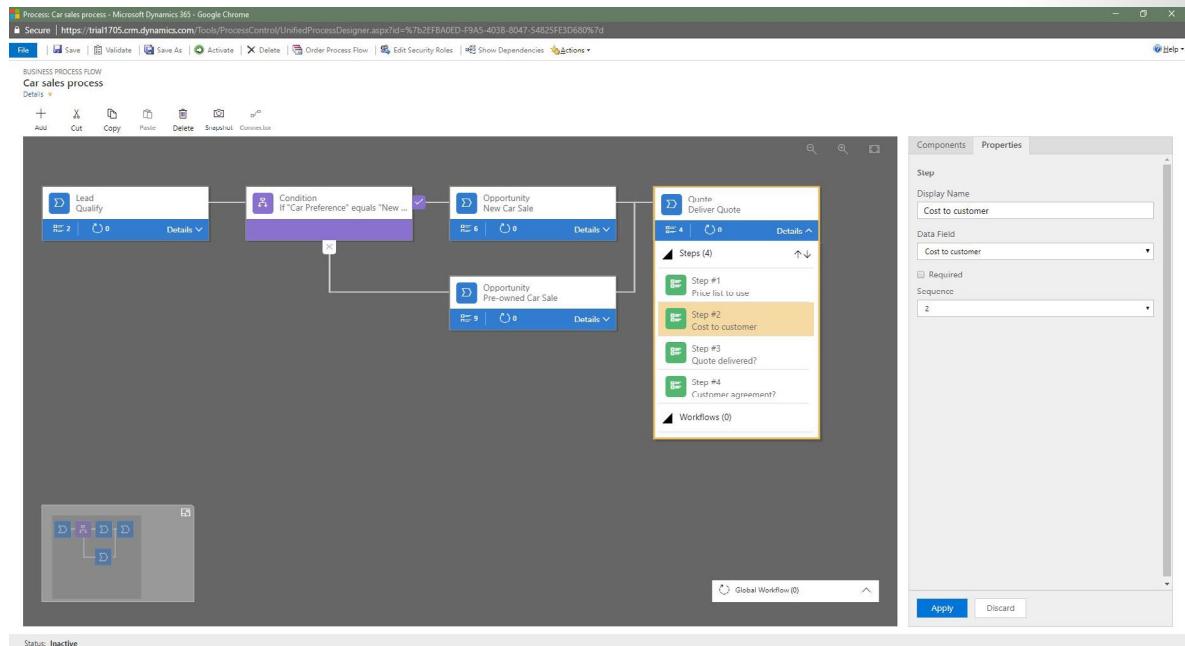
If **Car preference = New**, the process branches out to the **New Car Sales** stage, as shown here.



Otherwise, the process goes to the **Pre-Owned Car Sales** stage in the second branch, as shown here.



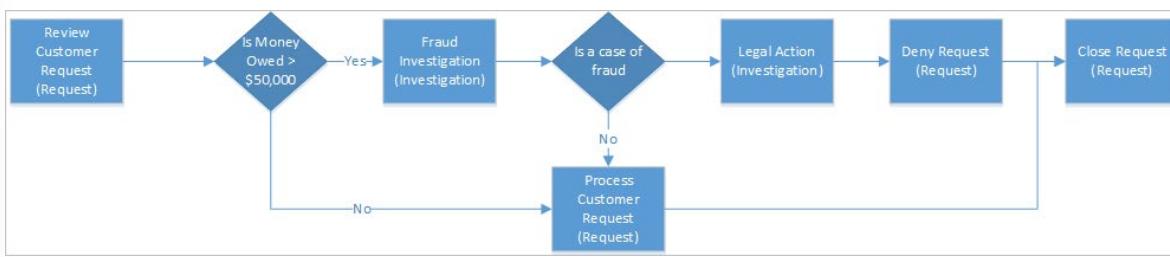
After all the steps in either the **New Car Sales** stage or the **Pre-Owned Car Sales** stage are finished, the process returns to the main flow, at the **Deliver Quote** stage.



## Prevent information disclosure example

There are a few things that you need to consider to prevent people from seeing specific information about a process flow.

This section uses the example of a business process flow that has branches for processing a loan request at a bank. In the following diagram, the custom entities that are used in the stages are shown in parentheses.



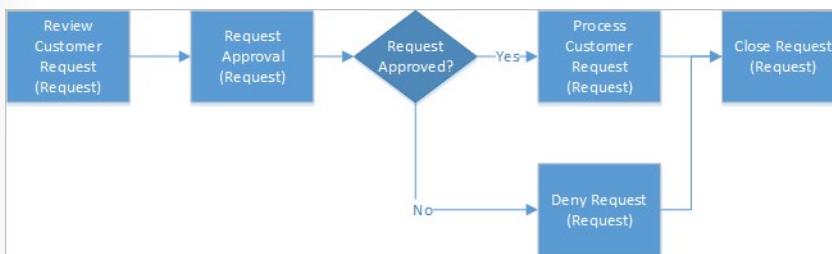
In this scenario, the bank loan officer needs access to the Request record, but she shouldn't have any visibility into the investigation of the request. At first glance, it looks as though we can easily meet this requirement by assigning the loan officer a security role that doesn't grant access to the Investigation entity. But let's look at the example in more detail to see whether things will really be that easy.

Let's say that a customer submits a loan request for more than \$60,000 to the bank. Here is a high-level view of the stages and branches:

- In the first stage, the loan officer reviews the request.
- A branching rule checks whether the amount that's owed to the bank will exceed \$50,000. If this branching rule is satisfied, the next stage in the process is to investigate whether the request is fraudulent.
- If it's determined that the request is fraudulent, the process moves on to taking legal action against the requestor.
- The loan officer shouldn't have visibility into the two investigative stages, because she doesn't have access to the Investigation entity.
- But if the loan officer opens the Request record, she can see the entire end-to-end process. Not only will she be able to see the Fraud Investigation stage, but she'll also be able to identify the outcome of the investigation, because she can see the Legal Action stage in the process.
- The loan officer can preview the steps in the investigative stages by choosing the stage. Although she won't be able to see the data or the step completion status, she'll be able to identify the potential actions that were taken against the requestor during the Fraud Investigation and Legal Action stages.

In this process flow, the loan officer will be able to see the Fraud Investigation and Legal Action stages, and this ability constitutes improper information disclosure.

Pay special attention to the information that might become disclosed because of branching. In our example, to prevent information disclosure, split the process into two separate processes: one for the request processing and one for the fraud investigation. The process for the loan officer will then look like this.



The process for the investigation will be self-contained and will include the following stages.



You'll have to provide a workflow to synchronize the Approve/Deny decision from the Investigation record to the Request record.

## Exercise - Monitor flows

You can view a summary of the number of times that each flow succeeded or failed today, yesterday, and on previous days. You can also explore details about each run, like when it ran, how long each step took, and, if a step failed, why it failed.

### Prerequisites

- Install the Power Automate mobile app for **Google Android<sup>37</sup>**, **Apple iOS<sup>38</sup>**, or **Windows Phone<sup>39</sup>** on a supported device. The screenshots in this unit were taken on the Apple iPhone version of the app, but the mobile app for Android and Windows Phone are similar.
- If you don't already have a flow, create one on the **Power Automate website<sup>40</sup>**. For easier testing, use a flow that you can trigger yourself instead of waiting for an external event.

The flow in this tutorial runs when you receive email from a specific address.

*Tip:*

For testing, you can set up the flow with your personal email address. Then, when the flow is ready for real use, you can set it up with a different address (for example, your manager's).

When the flow runs, it sends a custom push notification to your phone.

### Show a summary of activity

- If your flow hasn't run before, trigger a run to generate data.

It might take some time for the data to appear in the app.

- Start the mobile app.

Select the **Activity** tab. This tab organizes data by day, and today's data appears at the top.

<sup>37</sup> <https://aka.ms/flowmobiledocsandroid>

<sup>38</sup> <https://aka.ms/flowmobiledocsios>

<sup>39</sup> <https://aka.ms/flowmobilewindows>

<sup>40</sup> <https://flow.microsoft.com/>

Wi-Fi 17:59 30%

Feed Approvals

**TODAY**

**Get a push notification when you receive an email from your boss** 1h >  
Flow successfully ran 17 times

Boss said: Optometric Billing Solutions - PowerBI Help and...   
[Go to Outlook Office 365](#) 1h

**YESTERDAY**

**Get a push notification when you receive an email from your boss** 11:38 PM >  
Flow successfully ran 8 times

**Send a Working from home today email to your manager - 2** 8:57 PM >  
Flow successfully ran 1 time

WFH email was sent successfully to: Jennifer Hammond...  
<https://outlook.office365.com/> 8:57 PM

**Post list items to twitter after**

Activity Browse Buttons Flows Account

Each entry shows the name of the flow and icons that correspond to the flow's trigger events and actions.

The screenshot shows the Microsoft Flow history for today. It displays two entries. The first entry is for a flow named "Get a push notification when you receive an email from your boss". This flow has a blue icon (document with a gear) and an orange icon (Microsoft logo). A red box highlights the flow name and its description. Below the entry, it says "Flow successfully ran 17 times" and "1h >". The second entry is for a flow named "Boss said: Optometric Billing Solutions - PowerBI Help and...". It has a blue icon (document with a gear) and an orange icon (Microsoft logo). A blue link "Go to Outlook Office 365" is present. To the right, there are buttons for "Boss Refer" and "Go to".

If at least one run of a flow has succeeded in a day, an entry shows the number of successful runs and the time of the most recent success. A different entry shows similar information if a flow has failed.

This screenshot shows the Microsoft Flow history for today, identical to the one above. The first entry for the push notification flow is highlighted with a red box around the "17 times" run count. The second entry for the boss message flow is also present. The "1h >" timestamp is highlighted with a red box. The "Boss Refer" and "Go to" buttons are visible on the right.

If a flow sends push notifications, the text of the most recent notification appears at the bottom of the entry for successful runs.

TODAY

 Get a push notification when you receive an email from your boss 1h >

Flow successfully ran 17 times

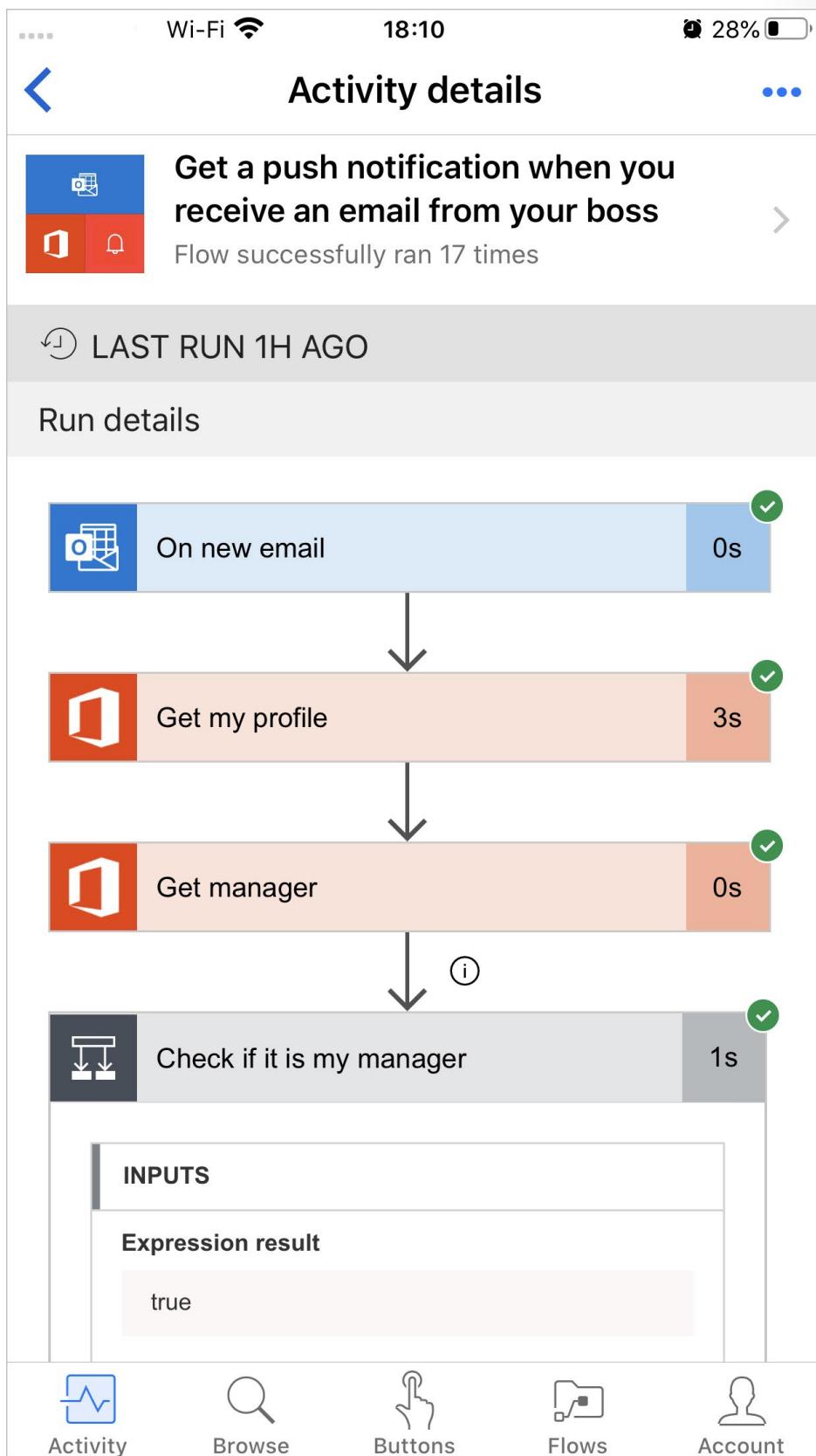
Boss said: Optometric Billing Solutions - PowerBI Help and...  
[Go to Outlook Office 365](#) 1h

Boss Refer Go to

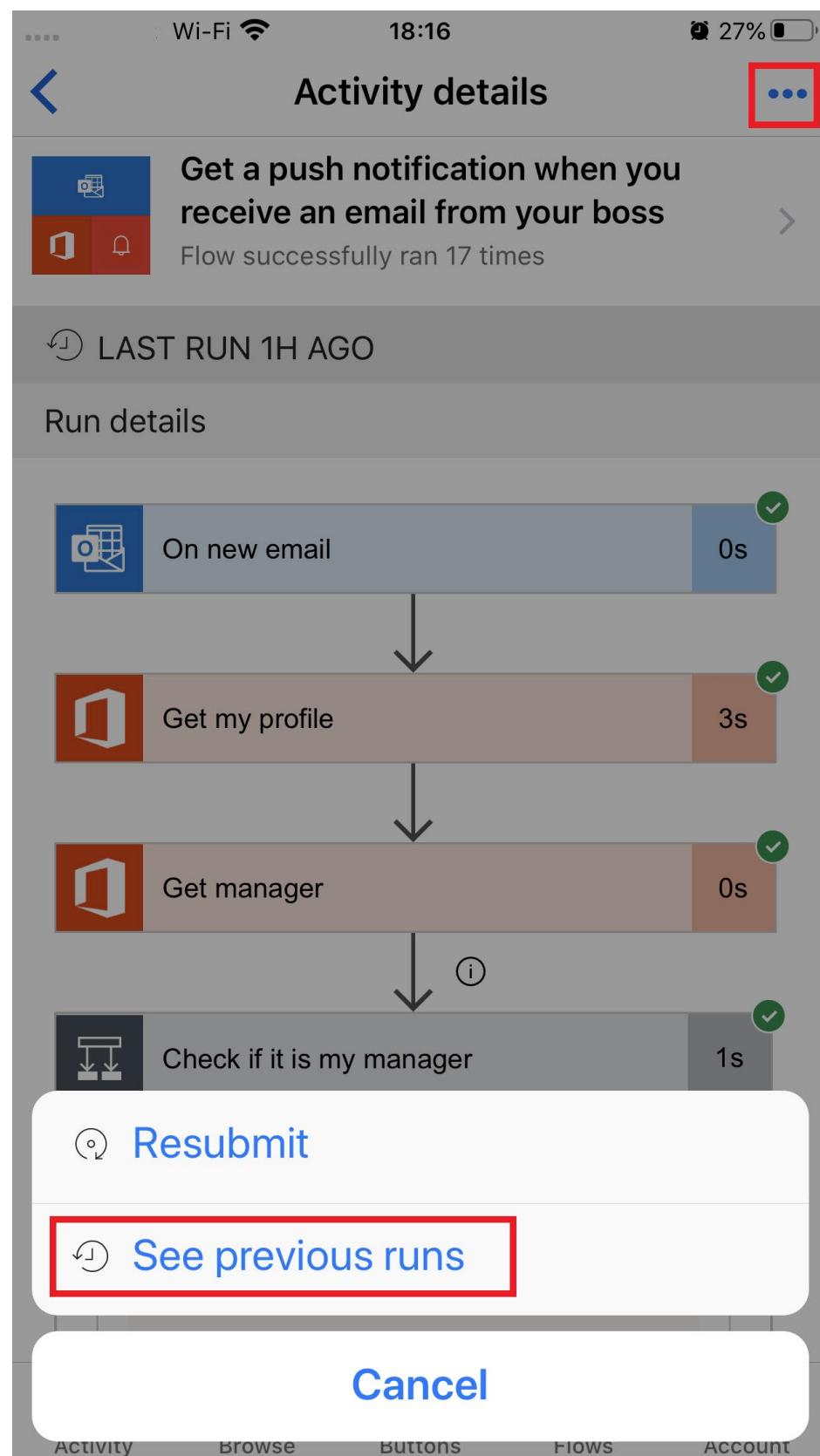
## Show details of a run

1. In the activity summary, select an entry to show details for the most recent run.

For each event and action, a symbol indicates whether the event or action succeeded or failed. If it succeeded, the amount of time that it took (in seconds) also appears.



2. Tap **Run history** to list all runs of the flow. Then select a specific run to view its details.



Run history

TODAY (17)

Flow successful	Feb 17, 2020 5:09 PM
Flow successful	Feb 17, 2020 2:47 PM
Flow successful	Feb 17, 2020 1:59 PM
Flow successful	Feb 17, 2020 1:37 PM
Flow successful	Feb 17, 2020 12:43 PM
Flow successful	Feb 17, 2020 11:34 AM
Flow successful	Feb 17, 2020 11:30 AM
Flow successful	Feb 17, 2020 11:15 AM
Flow successful	Feb 17, 2020 11:03 AM

Activity    Browse    Buttons    Flows    Account

## Summary

Congratulations! You've expanded your skills for creating flows that let you do more while working less.

In the previous modules for Power Automate, you learned how to build simple flows. You've now increased your knowledge by learning about complex data sources, flow scheduling, integration, and complex business processes.



## Continue your journey

Microsoft Learn provides several learning paths, based on your role and interests.

Here are some ways that you can use Power Automate to get more done with less work when you use dynamics 365 and the Power Platform. Because these technologies work together, it's easy to measure your business, act on the results, and automate your workflows.

- In this **Power Apps**<sup>41</sup> learning path, you'll learn how to build apps that use flows to track or update business processes on any device.
- In this **Power BI**<sup>42</sup> learning path, you'll learn how to turn your unrelated sources of data into coherent, visually immersive, and interactive insights.
- In this **Customer Service**<sup>43</sup> learning path, you'll learn how to capture, track, and follow up on sales leads, and how to connect to your customer relationship management (CRM) platform.

What's even more exciting is that you can do all this without writing a line of code!

---

<sup>41</sup> <https://docs.microsoft.com/learn/patterns/create-powerapps/>

<sup>42</sup> <https://docs.microsoft.com/learn/modules/get-started-with-power-bi/>

<sup>43</sup> <https://docs.microsoft.com/learn/modules/get-started-with-dynamics-365-for-customer-service/index>

# Introduction to business process flows in Power Automate

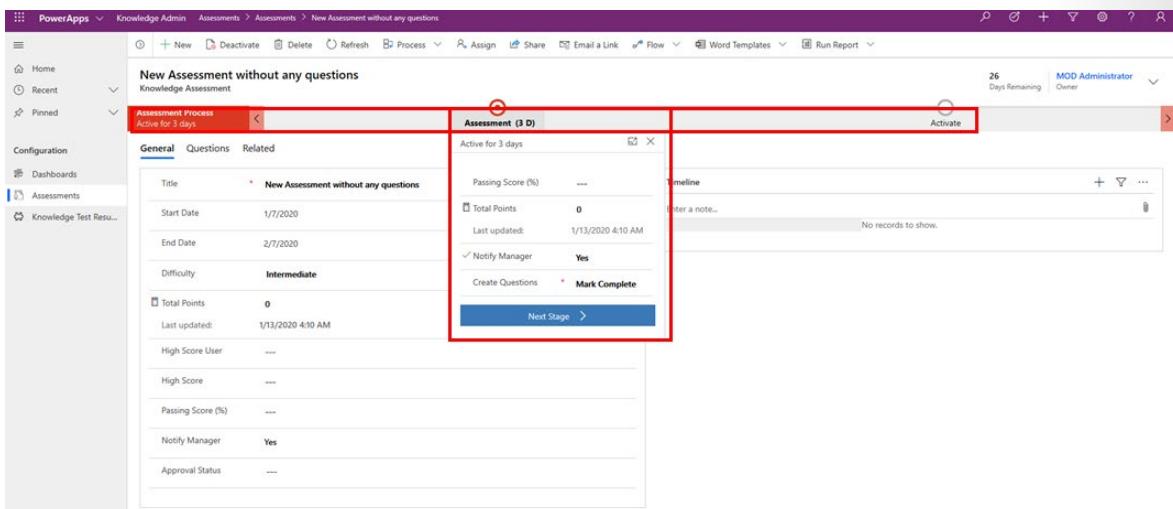
## Introduction to business process flows

A business process flow is a series of ordered work steps that a user completes within a business process. In Microsoft Power Automate, a business process flow is composed of a series of discrete stages that leads a user along a path toward process completion. Each stage contains one or more fields (called data steps) that you should complete before proceeding to the next stage in the business process flow.

A business process flow visually guides a user through stages within a process and shows progress toward process completion. A user can also see which stages that they have completed and which stages that they still need to complete within an instance of a process.

Business process flows can be configured to require users to enter information into certain fields (data steps) before completing the stage and, if needed, you can also allow users to jump stages. All data that is collected while you are completing a business process flow is stored in one or more entities in Common Data Service.

The following illustration shows a simple business process flow with two stages for adding questions to a survey. The business process flow is shown as a component of a model-driven solution in Power Apps.



Business process flows are created and managed by using Power Automate, and they are available for out-of-the-box entities and for your own custom entities.

Microsoft has many prebuilt business process flows, or you can create your own. Additionally, an entity can have none, one, or many business process flows associated with it. Business process flows are customizable to fit many organizational needs.

*Tip:*

Business process flows are meant to guide users through steps that are required to complete a business process. You must have a valid Power Apps Dynamics 365 license or a valid Power Apps license to create or use a business process flow.

## Business process flow vs regular

A business process flow is a visual guide that is meant to help users complete a business process by using a set of predefined stages. Users are not limited on how long they run a business process or how long they have a stage open. All data that is associated with the business process flow can only be stored in one or more entities in Common Data Service, and you cannot associate a business process flow with another data source behind it.

Common Data Service has many different out-of-the-box business process flows that you can use in its current state or modified to fit your needs.

A Power Automate flow does not have any visual components like a business process flow. Power Automate flows can be configured to work with many different data sources and a flow can connect to many different data sources within the same flow. A flow can be configured to time out if it is not completed in a certain time and can be triggered to move between steps based on data or user interaction. Flows support complex logic and looping and a Power Automate flow can call another Power Automate flow as needed.

## Business process flows value to organizations

Business process flows allow organizations to quickly standardize how processes are completed and what data is collected at each stage. Business process flows support logical branching so they can be used to standardize many common business processes within an organization.

Business process flows offer the following benefits:

- Improved outcomes
- Consistent stages and work steps across all instances of the process
- Improved data collection and reporting
- Decreased time to complete the process
- Predictable outcomes

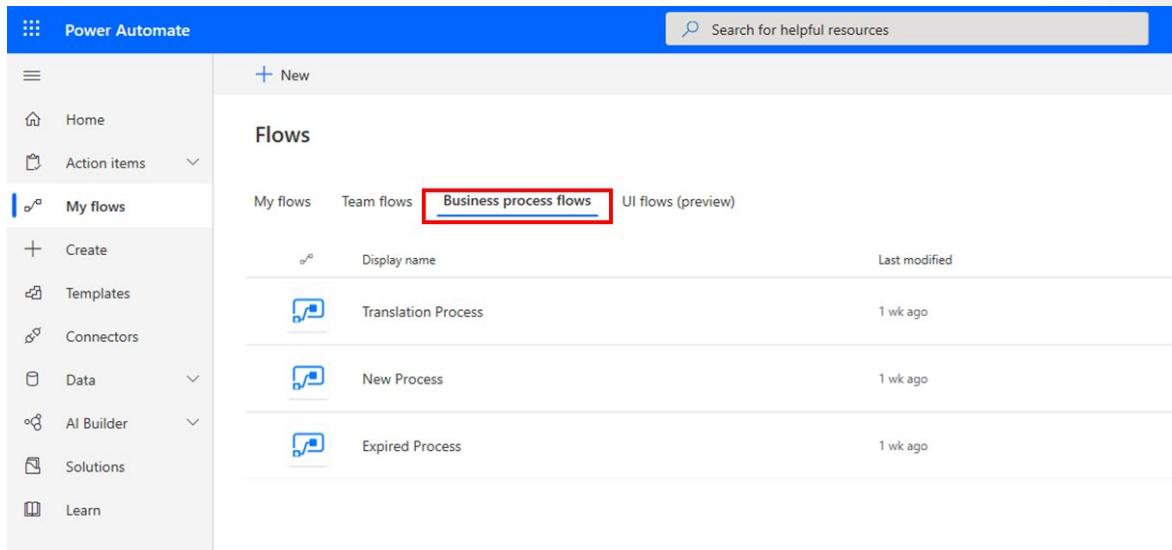
Business process flows are simple to set up and administer. Business process users who are close to business operations and processes can create new business process flows or modify out-of-the-box business process flows by using Power Automate.

Business process flows can be customized based on security roles, allowing access to the appropriate stages and steps based on a security role. Finally, the process of each instance of a business process flow can be monitored, and the data from the process flow can be used in Power BI dashboards and reporting for simplified administration.

## Business process flows and the larger Power Platform

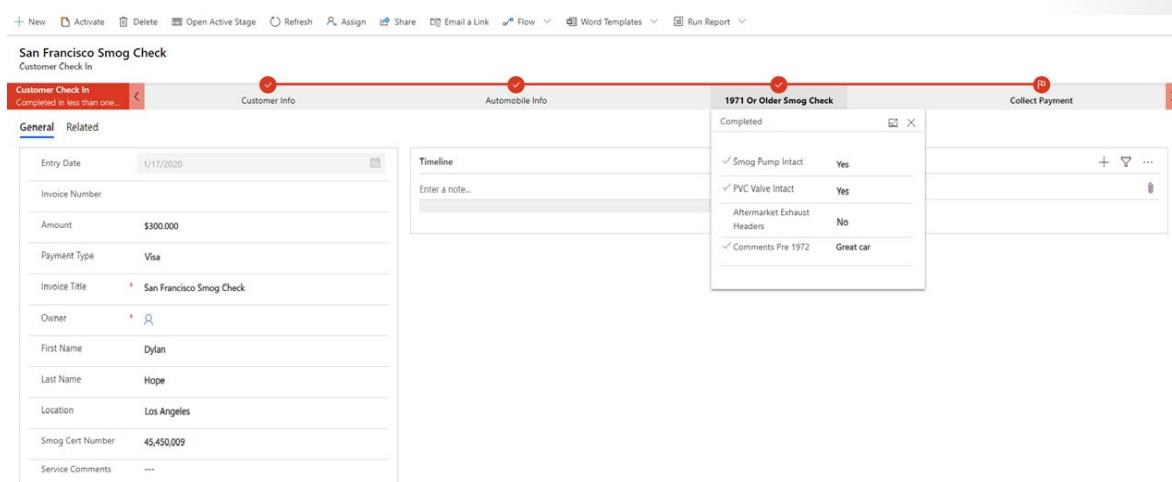
Business process flows are deeply integrated with the Microsoft Power Platform. They are created, customized, and managed by using Power

Automate. You can manage or create new business process flows by launching Power Automate and selecting **Business process flows** under **My Flows** in Power Automate, as shown in the following image.



The screenshot shows the Power Automate interface with the 'Flows' section selected. The 'Business process flows' tab is highlighted with a red box. The table lists three flows: 'Translation Process', 'New Process', and 'Expired Process', each with a small icon, display name, and last modified date.

You can launch the new business process flows as a component of a model-driven Power App or a stand-alone application within Power Automate (called an immersive business process flow), as shown in the following image.



The screenshot shows a Power App for 'San Francisco Smog Check'. A modal window titled '1971 Or Older Smog Check' is open, listing several items with checkboxes and dropdowns. The items include 'Smog Pump Intact' (Yes), 'PVC Valve Intact' (Yes), 'Aftermarket Exhaust Headers' (No), and 'Comments Pre 1972' (Great car).

As mentioned previously, all data that is associated with a business process flow is stored in one or more Common Data Service entities (custom or standard). You could launch an instant Power Automate flow in conjunction with a business process flow to store data outside of Common Data Service if needed. Additionally, you can create Power BI dashboards from the data that was collected within the business process flow.

Business process flows are deeply integrated within Power Platform and offer powerful ways to improve how you manage common business processes.

## Uses of business process flows

Knowing when to use a business process flow or a regular Power Automate flow to automate a processes or task can be difficult. The following guidelines can help you decide if you should use a business process flow or a standard Power Automate flow.

Use a business process flow if you:

- Want to create automated business processes with Dynamics 365 Solutions.
- Want a simple visual guide to help users complete a process.
- Want to use out-of-the-box business process flows.
- Have a Dynamics 365 license and want to create automated business processes with Common Data Service.

Use a Power Automate flow if you:

- Want to schedule a workflow to start based on a predefined time interval or after X minutes, hours, or days of some action or event.
- Want to trigger a flow based on data outside of Common Data Service (SharePoint, for example).
- Do not want to store data that is captured in the flow in Common Data Service.
- Want to push notifications outside of Outlook (SMS or Gmail, for example).
- Want to use and create workflows with only an Office 365 license.

## Summary

Business process flows are a powerful way to visually standardize and guide users through a process. They ensure consistent implementation and data collection, can improve the quality of outcomes, and decrease the cycle times to complete a process.

You can create business process flows as a custom solution, or you can choose to use or customize out-of-the box business process flows that are shipped with Common Data Service and Dynamics solutions.

Additionally, you can create business process flows as a stand-alone solution (called an immersive business process flow), or you can include them as a panel within a model-driven app.

An important parameter to remember is that business process flows only store data within Common Data Service.

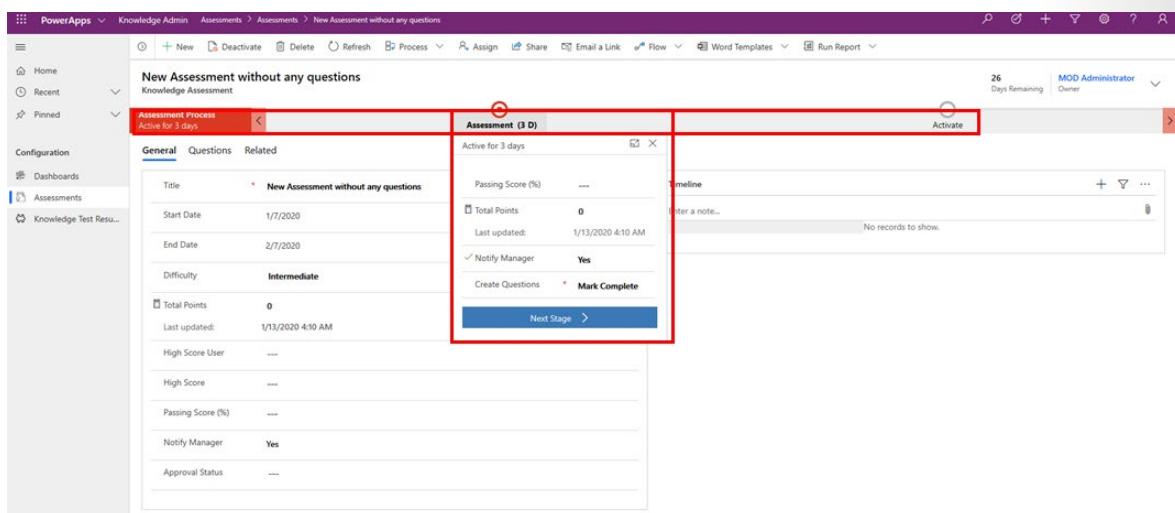
You should use a regular Power Automate flow if you want to store data in a data source outside of Common Data Service, or if you want to trigger the flow based on a time or recurring condition or if you only have an Office 365 license available.

# Create an immersive business process flow

## Introduction to immersive business process flows

Business process flows are available in two different varieties: embedded within a model-driven app and as a stand-alone solution called an immersive business process flow. You should examine the differences between each before learning how to create an immersive business process flow.

Embedded business process flows are visual representations of a series of work steps within an end-to-end process. They always appear along the top of a model-driven app in their own control, as shown in the following image.



An immersive business process flow can exist on its own, and it is built entirely within the Power Automate editor. They will always exist within the default solution in an instance of Common Data Service, and are only accessible within Power Automate or the data tab that is associated with an entity in Common Data Service. Immersive business process flows offer many of the same advantages of an embedded business process flow, but with the added advantage of simplified creation and streamlined management.

You will learn how to build an immersive business process flow in the next units of this module. The following screenshot shows a simple two-step immersive business process flow.

By taking a closer look, you'll see that immersive and embedded business process flows look similar. The map of the business process flow is shown along the top of the screen and a main form is shown beneath it.

The name of the business process flow is shown on the left side of the control, while each step (called a Stage) is shown as a red circle. A user selects the red circle of the current Stage and fills out information in a drop-down screen that includes various fields (called data steps).

The key difference between immersive and embedded business process flows is that an embedded business process flow is a component within the larger model-driven app, while the immersive business process flow is a stand-alone solution. Embedded and immersive business process flows help users complete a process within the context of a larger solution.

A user always views and completes an embedded business process flow within a model-driven app. An immersive business process flow is a stand-alone app, and it is built solely for the purpose of completing the business process. The immersive business process flow exists as the entire solution rather than a component of a larger model-driven app.

Many of the concepts and techniques that are discussed and demonstrated within this learning path apply to both embedded and immersive business process flows. Immersive and embedded flows are similar. The main difference between them is how they are launched and the context that they are viewed in.

The remainder of this module is focused on exploring and building an immersive business process flow.

## Exercise - Create an immersive business process flow

The following scenario and exercise will help you practice building an immersive business process flow with Power Automate and Common Data Service. Remember, all data that is associated with any business process flow is always stored in one or more entities within Common Data Service and business process flows.

**Note:**

To complete this exercise, you will need access to an account that has permission to create entities and fields in Common Data Service and you should have a Power Apps or Dynamics 365 license. Ask your Power Platform administrator for proper permissions or sign in and create a personal Power Apps and Power Automate development environment by using the Power Apps Community Plan (which is free). Sign up by accessing the **Power Apps Community Plan**<sup>44</sup> page.

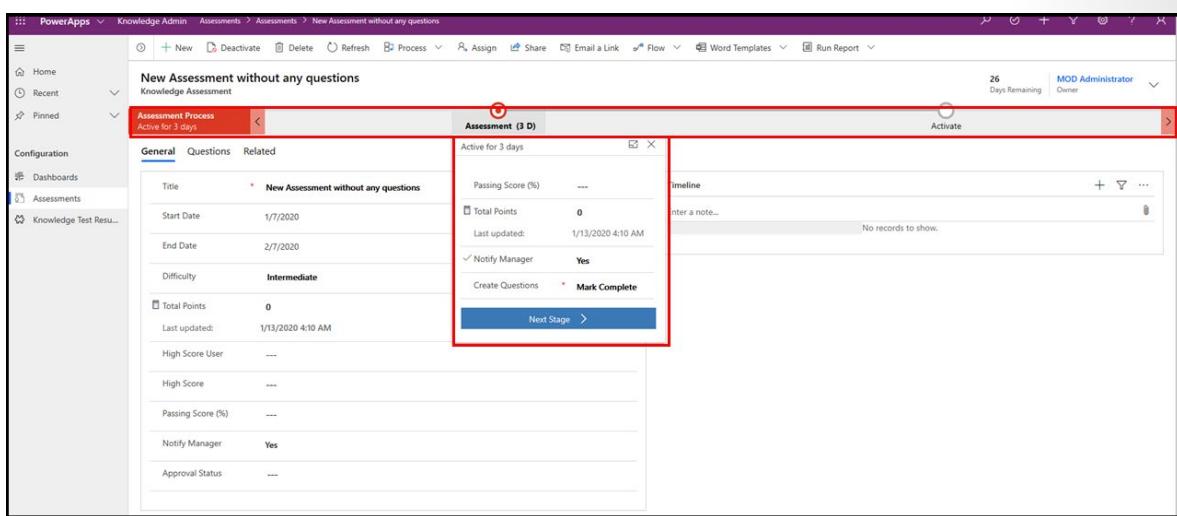
## Scenario

You work for SmogChecksRUs, a rapidly growing auto repair company that specializes in performing automotive smog checks and other auto services. SmogChecksRUs has been using a Microsoft Excel spreadsheet to collect customer and vehicle information, but now the marketing department has asked for a better way to collect information so they can follow up with customers and schedule checkups every two years, improve customer retention, and increase sales.

Management believes that improved data collection and standardized processes will improve customer satisfaction, improve customer loyalty, and increase recurring business and overall sales revenue. You have decided to create an immersive business flow by using Power Automate to meet management goals.

## Create an immersive business process flow

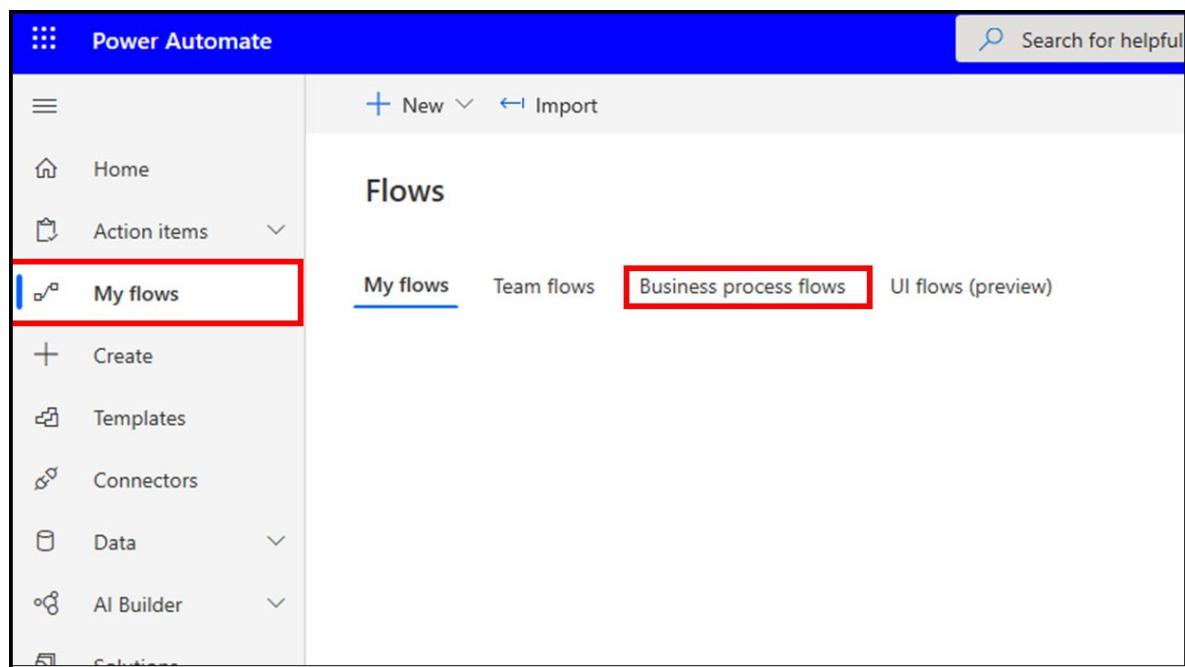
1. Go to **Power Automate**<sup>45</sup> and sign in to your local instance.
2. Select the proper environment in the upper-right corner of the screen.



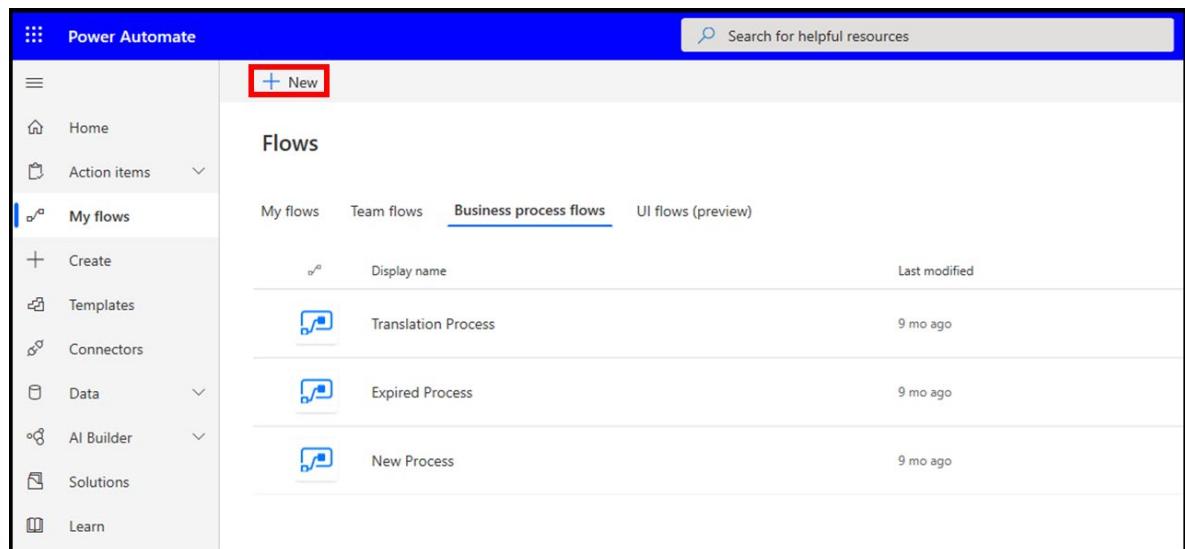
3. Select **My Flows** on the menu on the left side of the screen and then select the **Business process flows** tab.

<sup>44</sup> <https://powerapps.microsoft.com/communityplan/?azure-portal=true>

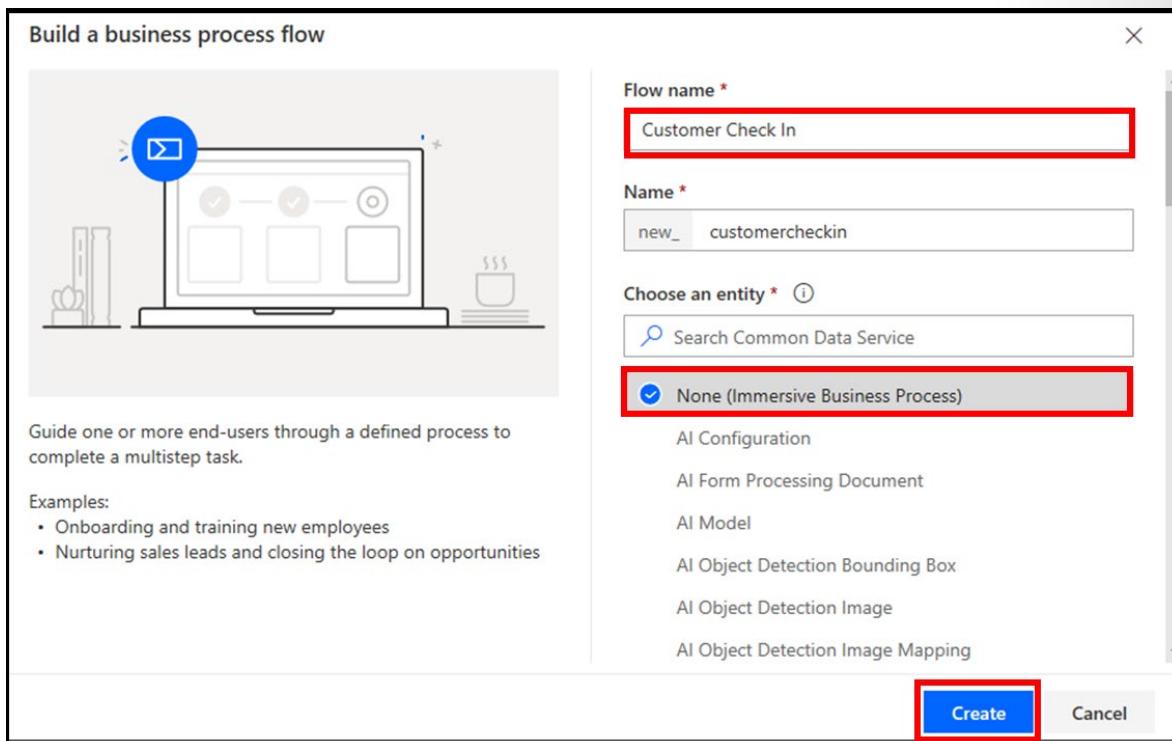
<sup>45</sup> <https://preview.flow.microsoft.com/?azure-portal=true>



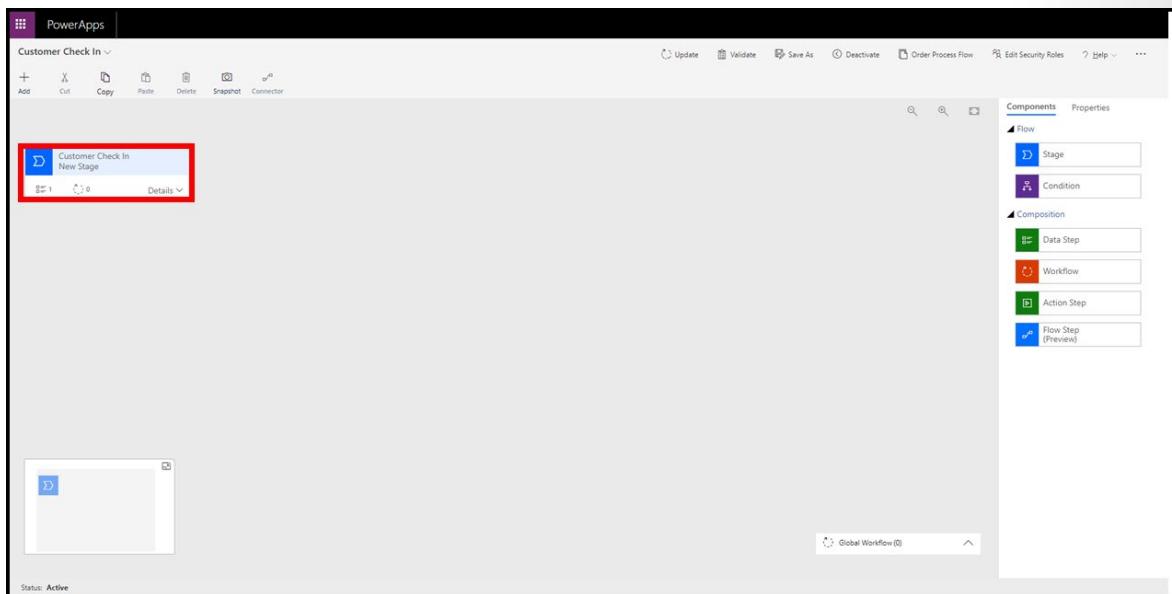
4. Select the **+ New** button on the upper-left corner of the screen.



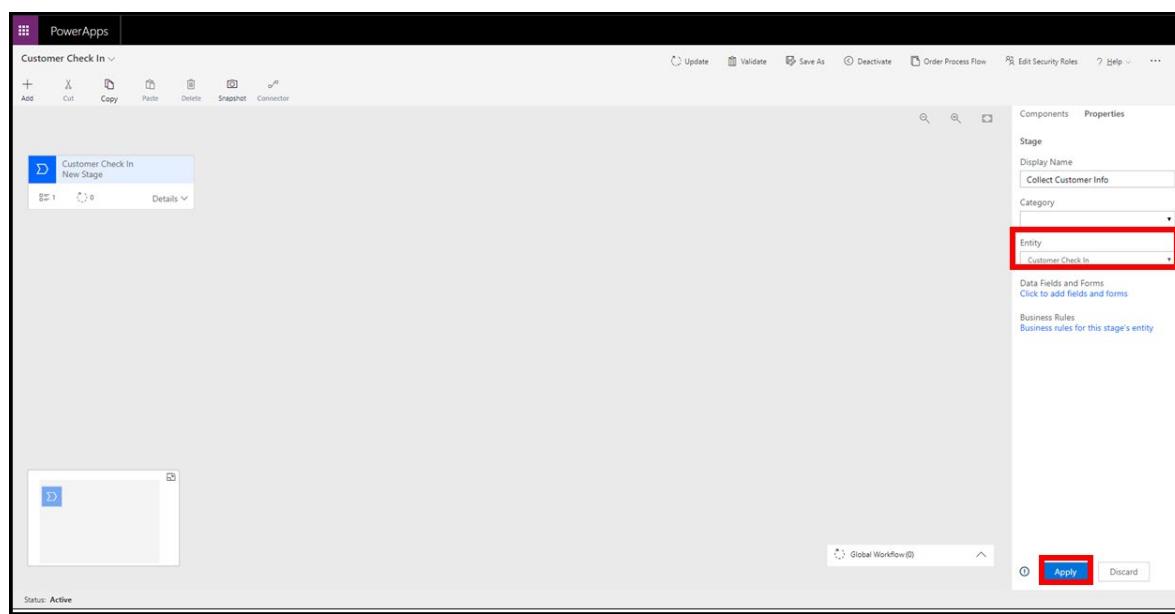
5. Enter **Customer Check In** as the **Flow name**, select **None (Immersive Business Process)** under the entity drop-down list, and then select the **Create** button, as shown in the following screenshot. Wait a minute for the entity to be created, and then the **Business process flow** editor will launch.



- In the **Business process flow** editor, add fields to the new entity (`customercheckin`) that was automatically created in the previous step. Select **Customer Check In New Stage**, as shown in the following figure.

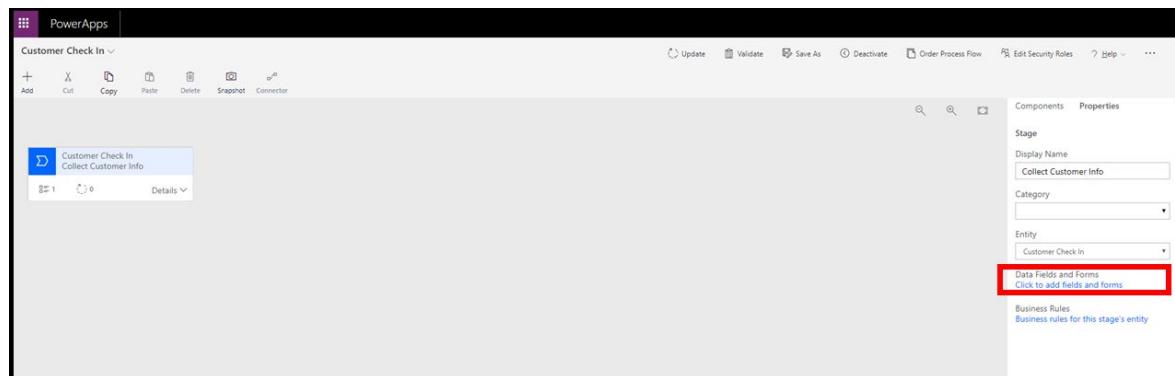


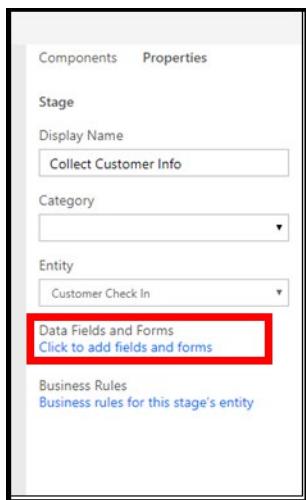
- Select **New Stage** within the designer, rename the Stage to **Collect Customer Info**, and then select the **Apply** button, as shown in the following screenshot.



Next, you will add fields to the new **CollectCustomerInfo** entity in Common Data Service so you can capture customer information.

8. Select the **Collect Customer Info** stage and then select the **Click to add fields and forms** hyperlink, as shown in the following figure.





**Note:**

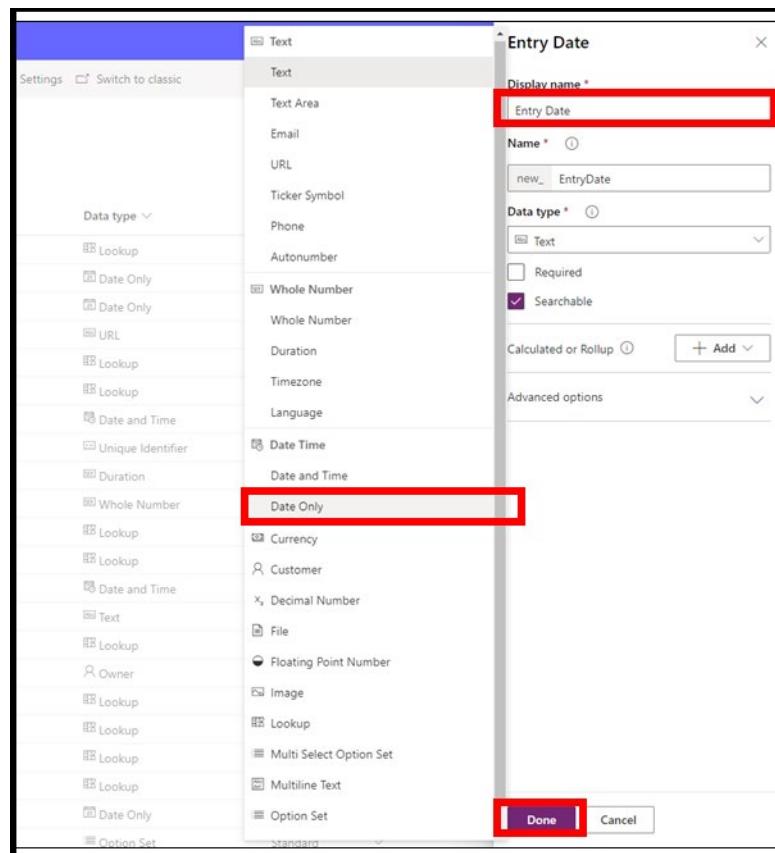
If you select **Click to add field and forms** link from Power Automate, you may see the classic UI. To see the modern UI, Open Power Apps in a new window and sign-in. On the left, select **Data** then **Entities**.

Find and select the **Customer Check In entity**. Add the fields as listed in the next step.

9. Add fields to the collectcustomerinfo entity by selecting the **Add Field** button, as shown in the following screenshot.

The screenshot shows the Power Apps modern UI for the 'Customer Check In' entity. The left sidebar shows navigation options like Home, Learn, Apps, Create, Data, and Flows. The main area displays entity details with tabs for Fields, Relationships, Business rules, Views, Forms, Dashboards, Charts, and Keys. Under the Fields tab, there is a table listing three fields: 'Active Stage' (Name: activestageid), 'Active Stage Started On' (Name: activestagestartedon), and 'Completed On' (Name: completedon). A red box highlights the '+ Add field' button in the top navigation bar.

10. Add each of the fields from the following list. Enter the name and data type and then select the **Done** button each time you add a new field, as shown in the following screen.



- Entry Date - Date Only
- First Name - Text
- Last Name - Text
- Phone Number - Phone
- Address - Text
- City - Text
- State - Text
- Postal Code - Text
- Phone Number - Phone Number
- Comments - Text Area

11. When you are finished, select the **Save Entity** button to save the new fields. Make sure that you select the **Save Entity** button or none of the fields will be added.

Display name	Name	Data type	Type	Customizable	Required	Searchable
Active Stage	... activestagedid	Lookup	Custom	✓	✓	✓
Active Stage Started On	... activestagestartedon	Date Only	Custom	✓	✓	✓
Address	... new_address	Text	Custom	✓	✓	✓
City	... new_city	Text	Custom	✓	✓	✓
Comments	... new_comments	Text Area	Custom	✓	✓	✓
Completed On	... completedon	Date Only	Custom	✓	✓	✓
Context URL	... bpf_contexturl	URL	Custom	✓	✓	✓
Created By	... createdby	Lookup	Standard	✓	✓	✓
Created By (Delegate)	... createdonbehalfby	Lookup	Standard	✓	✓	✓
Created On	... createdon	Date and Time	Standard	✓	✓	✓
Customer Check In	... businessprocessflowinstanceid	Unique Identifier	Standard	✓	✓	✓
Duration	... bpf_duration	Duration	Custom	✓	✓	✓
Entry Date	... new_entrydate	Date Only	Custom	✓	✓	✓
First Name	... new_firstname	Text	Custom	✓	✓	✓
Import Sequence Number	... importsequencenumber	Whole Number	Standard	✓	✓	✓
Last Name	... new_lastname	Text	Custom	✓	✓	✓
Modified By	... modifiedby	Lookup	Standard	✓	✓	✓
Modified By (Delegate)	... modifiedonbehalfby	Lookup	Standard	✓	✓	✓
Modified On	... modifiedon	Date and Time	Standard	✓	✓	✓
Name (Primary Field)	... bpf_name	Text	Custom	✓	✓	✓

Discard Save Entity

12. Close the current tab of the browser that is showing the entity fields, and then return to the business process flow designer screen.

## Add fields as steps and finish the flow

Now, you will add the fields as Steps in the first Stage in the **Customer Check In** business process flow.

1. Refresh your browser and then select the **Customer Check In** Stage.

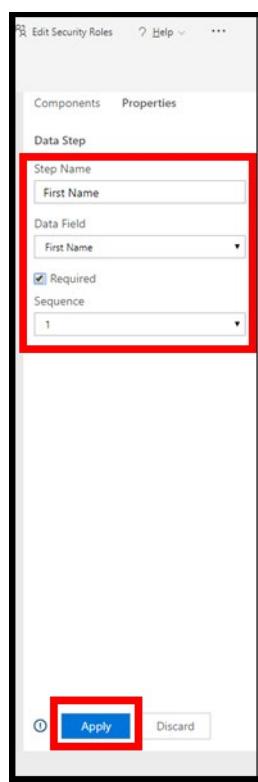
Data Step  
 Step Name: First Name  
 Data Field: First Name  
 Required  
 Sequence: 1

Apply Discard

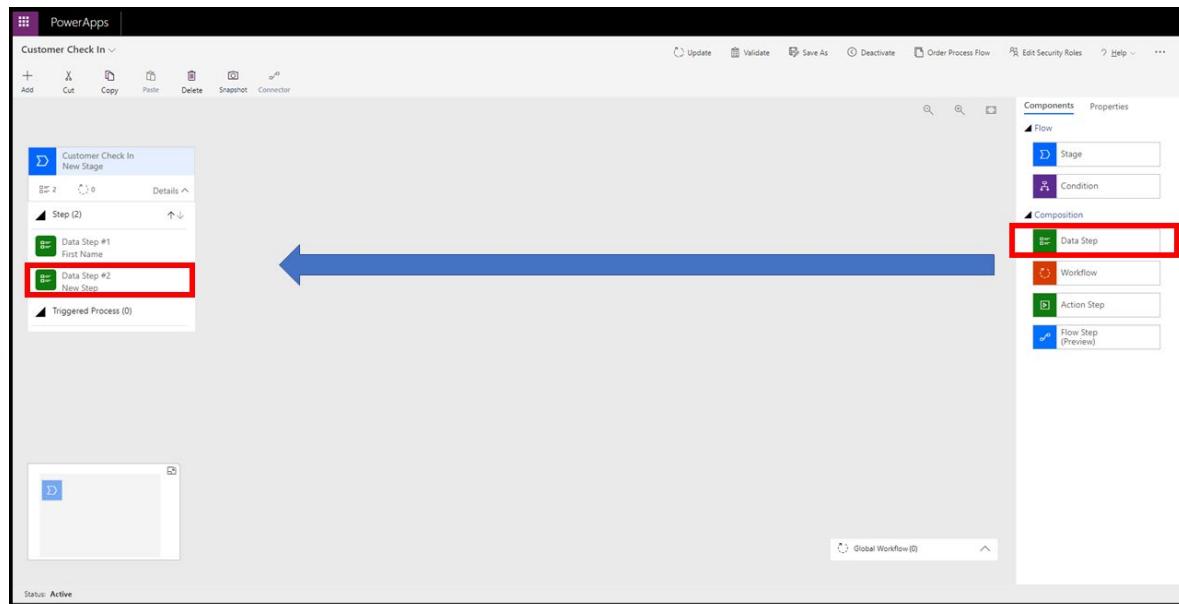
2. Select **Data Step # 1**, and then enter the following information:

- Step Name - Enter **First Name**.
- Data Field - Select **First Name** from the drop-down menu.
- Select the **Required** check box.

- Select the **Apply** button.



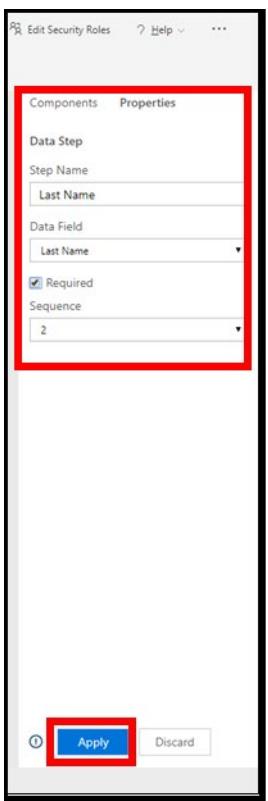
3. Select the **Components** tab and then drag a Data Step under **Data Step #1**, as shown in the following figure.



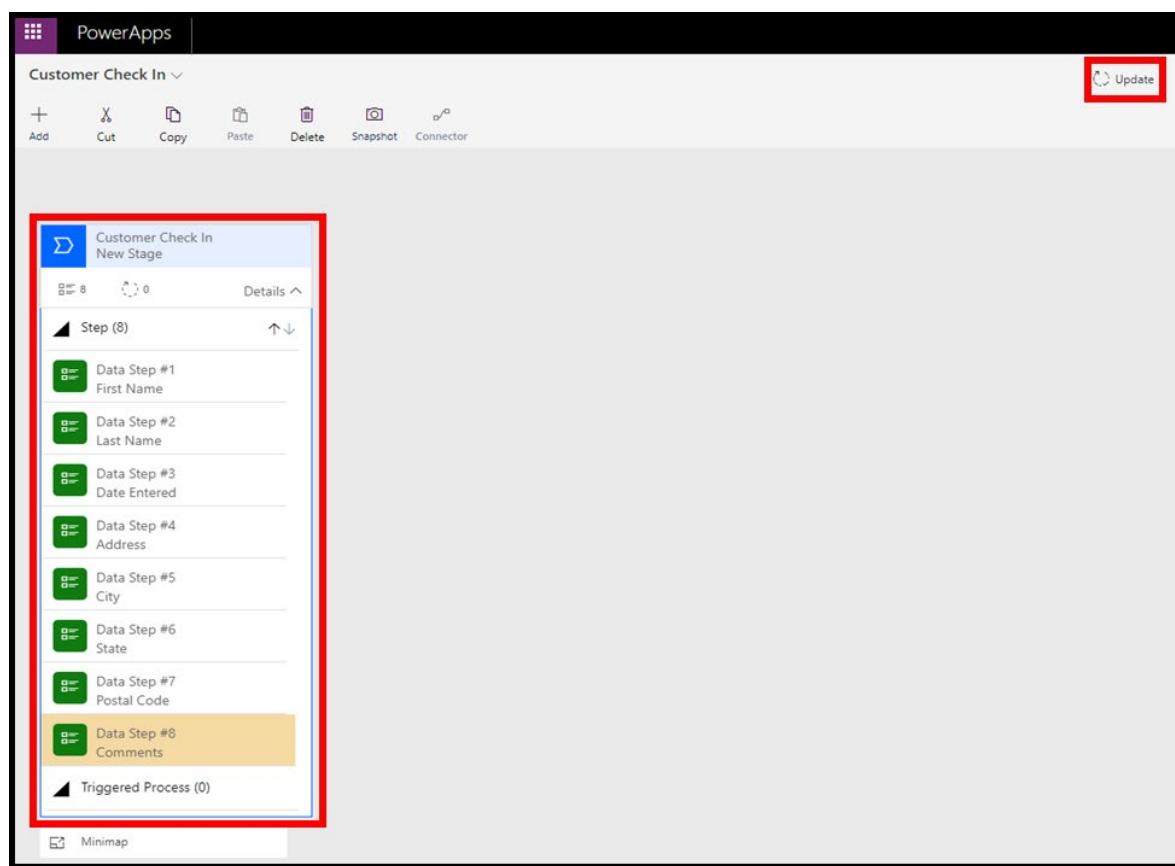
4. Select **Data Step #2** and enter the following information:

- Step Name - Enter **Last Name**.
- Data Field - Select **Last Name** from the drop-down menu.

- Select the **Required** check box.
- Select the **Apply** button.



5. Select the **Components** tab, and then drag additional data steps and add fields that were added earlier under Stage One (Entry Date, Address, City, State, and so on).
6. When you are done, Stage 1 should appear as shown in the following figure. If all appears correct, select the **Update** button in the ribbon.

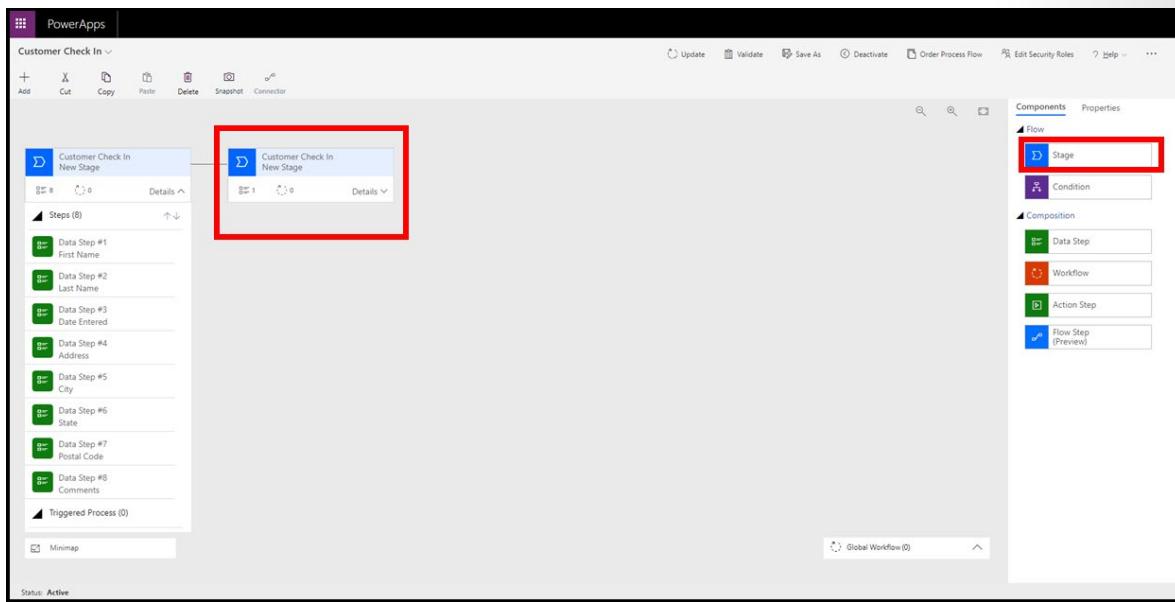


**Note:**

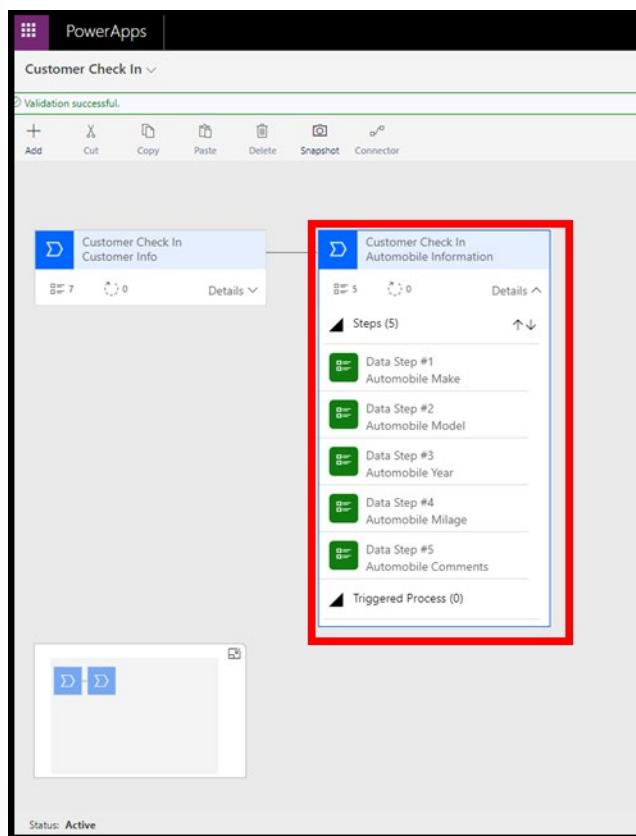
When the data process flow is used, data will be saved into the fields that you created in the customercheckin entity in Common Data Service.

Now, you will collect information about the automobile being serviced, so you'll add a new stage and add fields about the automobile to the customercheckin entity.

7. Select the **Components** tab and drag a new stage to the right of Stage 1. Make sure to drop the new stage into the plus (+) sign. The new stage should resemble the following screenshot.



8. Repeat the steps that were covered for Stage 1, and then add the following fields and steps:
  - Automobile Make - Text
  - Automobile Model - Text
  - Automobile Year - Number
  - Automobile Mileage - Number
  - Automobile Comments - Text Area
9. Make sure to save the entity after you add the new fields.
10. When you are done adding the data steps, select the **Update** button in the ribbon. Your completed Stage 2 should look like the following screenshot.



## Exercise - Run the business process flow and view data

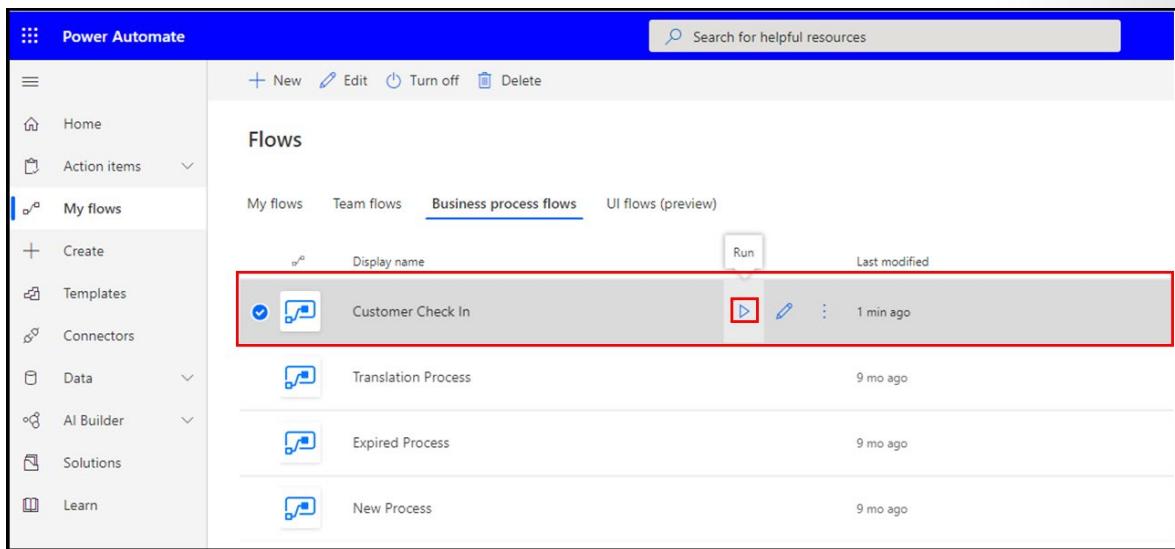
Now that you have a new business process flow, you can try it out and discover how the data is stored after it has been run.

1. Select the **Update** button in the top ribbon.
2. Close the business process flow designer after you see the message that the process flow has updated successfully.
3. Go back to **My Flows**. Select **Business process flows**, where you should see the new flow listed.

*Note:*

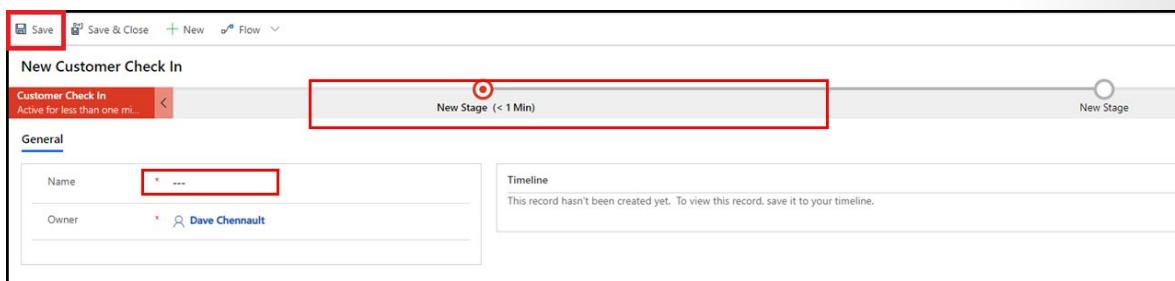
You can see all immersive business process flows that are in process or that have been run by going to the main Power Automate page, selecting the **My flows** icon on the left menu, and then selecting the **Business process flows** tab.

4. Select the **Run** arrow, as shown in the following screenshot.

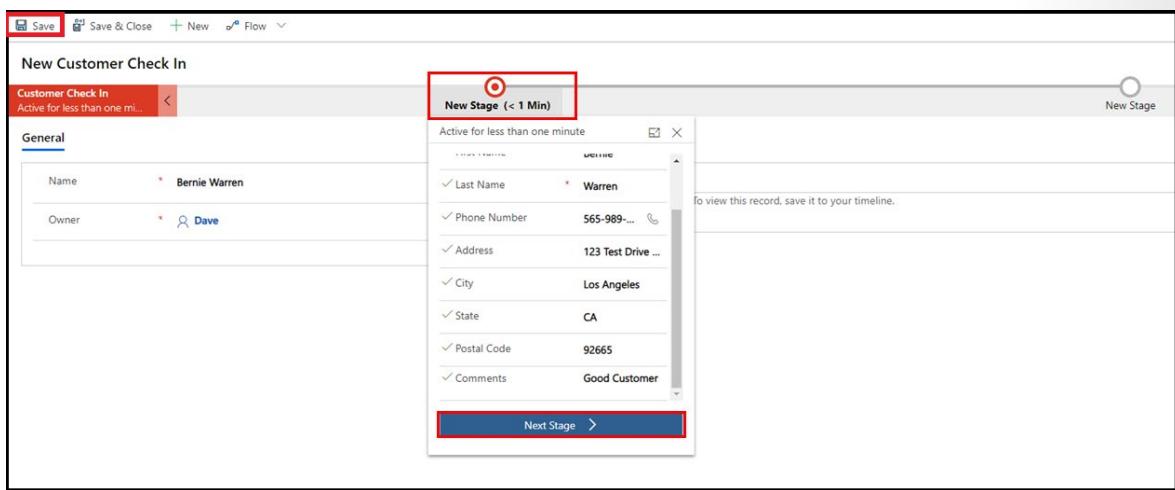


This selection will launch a new instance of the flow that you created.

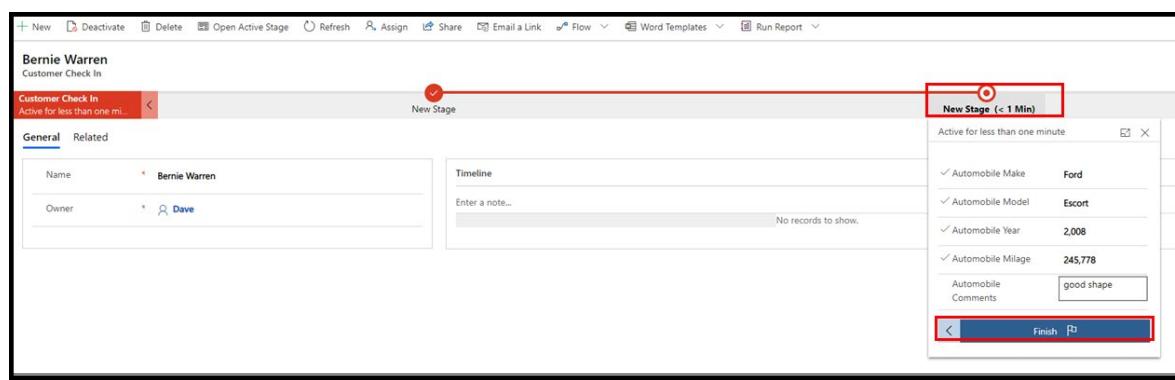
- Enter a name in the **Name** field on the **General** form (main form) and then select **Save**, as shown in the following figure.



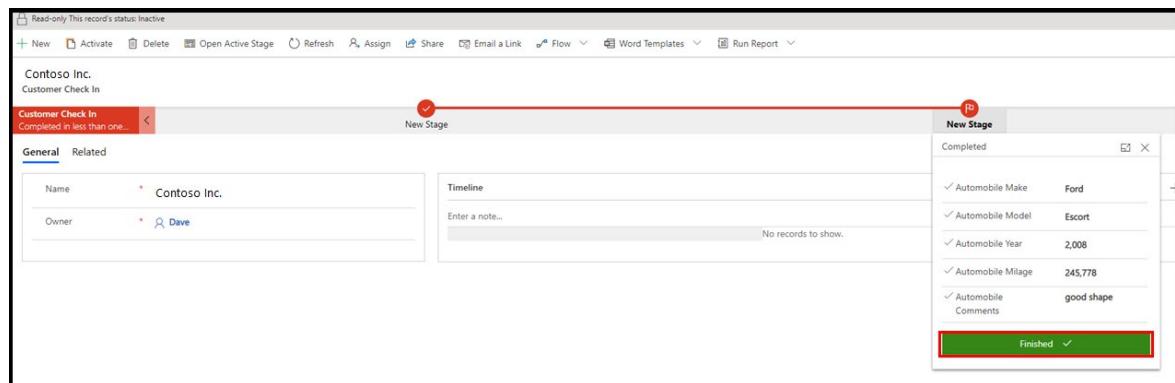
- After you select the **Save** button, select the red circle for the first stage, fill out the information, and then select the **Next Stage** button.



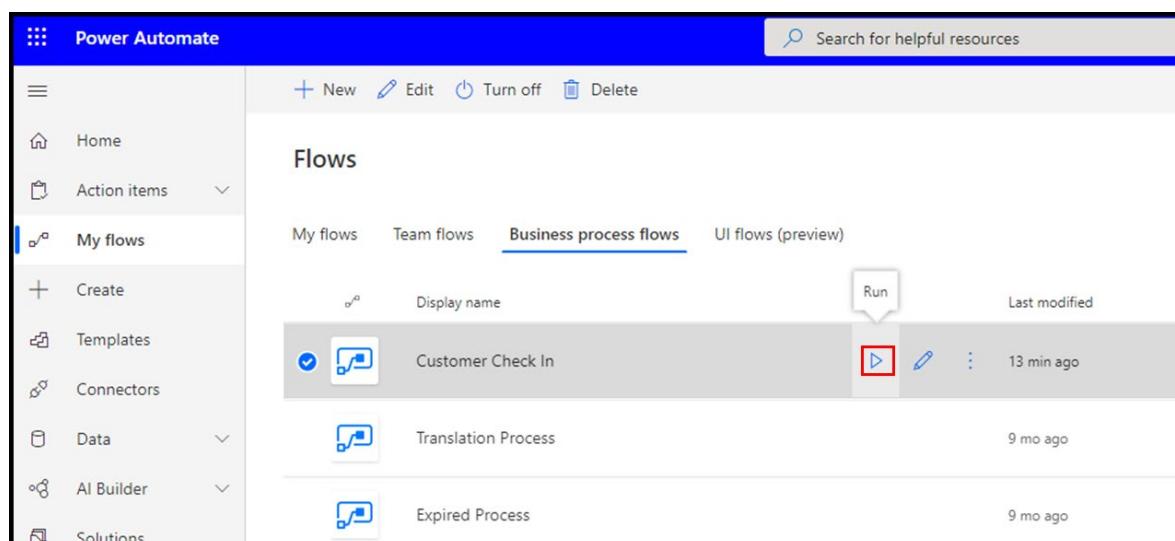
- Select the second stage, fill out the vehicle information, and then select the **Finish** button.



After you select the **Finish** button, the button in Stage 2 should turn green and the data that you entered is automatically saved.



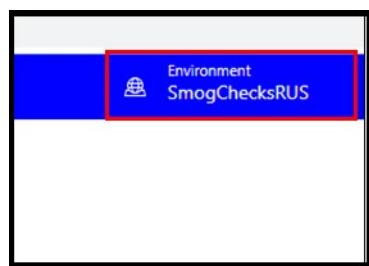
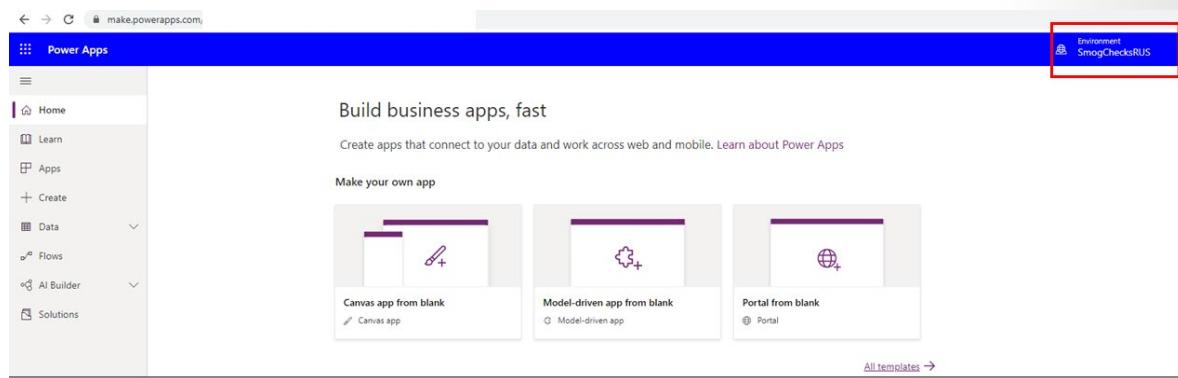
8. You can create additional records by launching Power Automate, selecting **My Flows** and **Business process flows**, and then selecting the arrow next to the **Customer Check In** business process flow, as shown in the following screenshot.



## View the created data

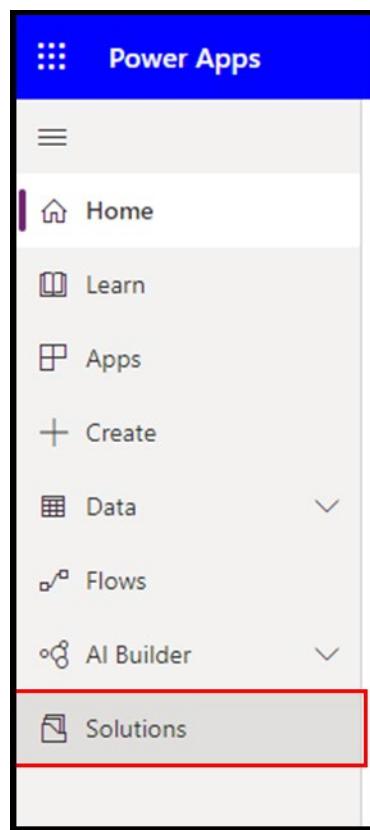
You can view the data that you created for running the flow by following these steps:

1. Go to **Power Apps**<sup>46</sup> and sign in.
2. Select the environment that you used to create the **Customer Check In** business process flow, as shown in the following screenshot.



3. Select the **Solutions** menu on the left side of the screen.

<sup>46</sup> <https://make.powerapps.com/?azure-portal=true>



4. Double-click the **Common Data Services Default Solution** to open it, as shown in the following figure.

Solutions				
	Display name	Name	Created	Version
	AI Sample Data	... msdyn_AIBuilderSam...	1/11/2020	1.0.0.5
	Asset Checkout	... AssetCheckout	1/11/2020	0.0.0.1
	Innovation Challenge	... InnovationHub	1/11/2020	0.0.0.1
	Fundraiser	... msdyn_SampleApp	1/11/2020	1.0.0.2
	Contextual Help Base	... msdyn_ContextualH...	1/11/2020	1.0.0.10
	Contextual Help	... msdyn_ContextualH...	1/11/2020	1.0.0.10
	Common Data Services Default Solution	... Crd83b3	1/11/2020	1.0.0.0
	Default Solution	... Default	1/11/2020	1.0

5. Select the **Customer Check In** entity.

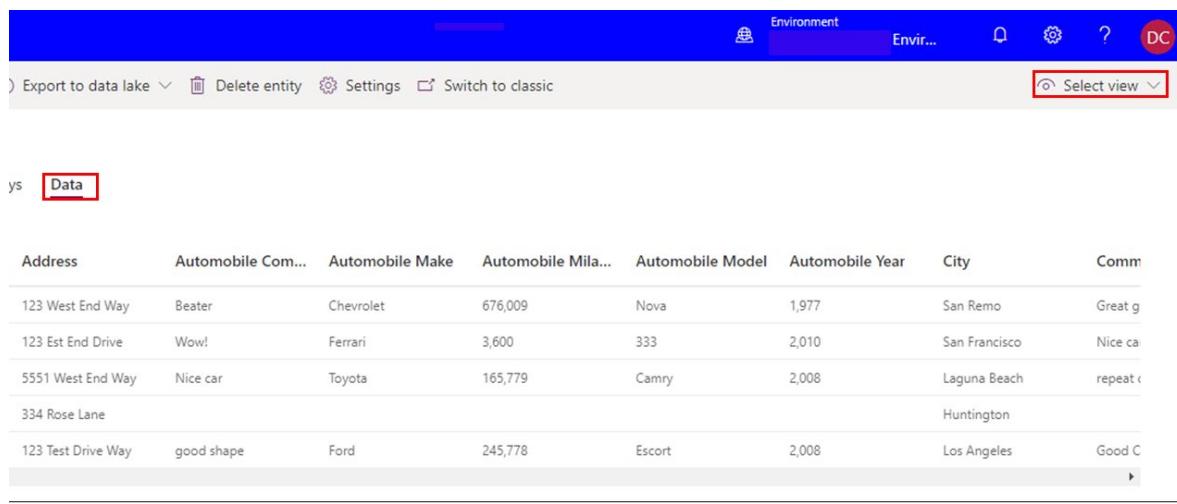
The screenshot shows the 'Power Apps' interface with the 'Solutions' tab selected. The main area displays a list of entities under the 'Common Data Services Default Solution'. One entity, 'Customer Check In', is highlighted with a red box. The columns in the list are: Display name, Name, Type, and Managed by.

Display name	Name	Type	Managed by
Author	cr5be_author	Entity	
Boat Name	cr5be_boatname	Option Set	
Books	cr5be_books	Entity	
Captain	cr5be_captain	Option Set	
Cruises	cr5be_cruises	Entity	
Destination	cr5be_destination	Option Set	
Dogs	cr5be_dogs	Entity	
Gender	cr5be_gender	Option Set	
Identification	cr5be_identification	Option Set	
Library Book	cr5be_librarybook	Entity	
Passengers	cr5be_passengers	Entity	
PC	cr5be_pc	Entity	
PC Type	cr5be_pctype	Option Set	
Screen Size Field	cr5be_screensizefield	Option Set	
Sex	cr5be_sex	Option Set	
ID Required	ID Required	Process	
<b>Customer Check In</b>	<b>new_customercheckin</b>	Entity	

6. Select the **Data** tab then click **Select view** on the right side of the screen, and then select the **All Data** view option.

The screenshot shows the 'Power Apps' interface with the 'Data' tab selected. The main area displays a table of data for the 'Customer Check In' entity. The table has columns: Name, Customer Check In, Active Stage, Active Stage Star..., Address, Automobile Com..., Automobile Make, Automobile Mila..., Automobile Model, Automobile Year, City, and Comm. The 'Select view' button is highlighted with a red box.

Name	Customer Check In	Active Stage	Active Stage Star...	Address	Automobile Com...	Automobile Make	Automobile Mila...	Automobile Model	Automobile Year	City	Comm
Joe Green	0791c997-3237-ea1...	New Stage	1/14/2020	123 West End Way	Beater	Chevrolet	676,009	Nova	1,977	San Remo	Great g
William Sonoma	30cc4856-3337-ea1...	New Stage	1/14/2020	123 Est End Drive	Wow!	Ferrari	3,600	333	2,010	San Francisco	Nice ca
William Wilson	63a169de-bb37-ea1...	New Stage	1/15/2020	5551 West End Way	Nice car	Toyota	165,779	Camry	2,008	Laguna Beach	repeat e
William Tell	3225602a-ca37-ea1...	New Stage	1/15/2020	334 Rose Lane						Huntington	
Bernie Warren	8d8e3304-cd37-ea1...	New Stage	1/15/2020	123 Test Drive Way	good shape	Ford	245,778	Escort	2,008	Los Angeles	Good C



The screenshot shows the Microsoft Power Automate interface in the 'Data' view. At the top, there are navigation links: 'Export to data lake', 'Delete entity', 'Settings', 'Switch to classic', 'Environment' (with a dropdown menu), 'Envir...', a bell icon, a gear icon, a question mark icon, and a 'DC' button. A red box highlights the 'Select view' dropdown menu. Below the header, the word 'ys' is followed by a red-bordered 'Data' button. The main area displays a table with columns: Address, Automobile Com..., Automobile Make, Automobile Mila..., Automobile Model, Automobile Year, City, and Comm. The table contains five rows of data:

Address	Automobile Com...	Automobile Make	Automobile Mila...	Automobile Model	Automobile Year	City	Comm
123 West End Way	Beater	Chevrolet	676,009	Nova	1,977	San Remo	Great g
123 Est End Drive	Wow!	Ferrari	3,600	333	2,010	San Francisco	Nice ca
5551 West End Way	Nice car	Toyota	165,779	Camry	2,008	Laguna Beach	repeat c
334 Rose Lane						Huntington	
123 Test Drive Way	good shape	Ford	245,778	Escort	2,008	Los Angeles	Good C

Now, you can view the all the data that you created with your new immersive business process flow.

In the next and final module of this learning path, you will improve your new immersive flow by customizing the form that is associated with your flow, adding logical branching, and adding instant workflow notification to make your new immersive business process flow even more powerful.

## Summary

This module explained the tools that you can use for building your first immersive business process flow. In addition, you learned about the following concepts:

- The definition of an immersive business process flow.
- The differences between an embedded and immersive business process flow.
- How to create a new immersive business process flow in Power Automate.
- How to add fields to an entity by using the business process flow designer.
- How to build and save a two-step business immersive business process flow by using stages and data steps.
- How to run an immersive business process flow.
- How to view the data that you created in the new immersive business process flow.

Though immersive business process flows are the main focus of this module, business process flows can also be created and embedded within a model-driven application. If you want to learn more about embedded business process flows, select the following links for a series of videos that demonstrate the building of an embedded business process flow within a model-driven app. The videos are informative and led by Microsoft Power Automate Product Managers.

**Introduction & Planning a Business Process Flow<sup>47</sup>**

**Building a Model Driven App<sup>48</sup>**

**Build A Business Process Flow and Add to Model Driven Apps<sup>49</sup>**

<sup>47</sup> <https://www.youtube.com/watch?v=7RiXDiPNZic>

<sup>48</sup> <https://www.youtube.com/watch?v=sslyrDVCaw8>

<sup>49</sup> [https://www.youtube.com/watch?v=e4u9fE\\_teNo](https://www.youtube.com/watch?v=e4u9fE_teNo)

**Add a Flow to help manage Stages in a Business Process Flow<sup>50</sup>**

**Watching the Business Process Flow in Action<sup>51</sup>**

---

**50** <https://www.youtube.com/watch?v=9DFgFax0lBo>

**51** <https://www.youtube.com/watch?v=X0sjXE33oGM>

## Understand advanced business process flow concepts in Power Automate

### Introduction to logical branching in business process flows

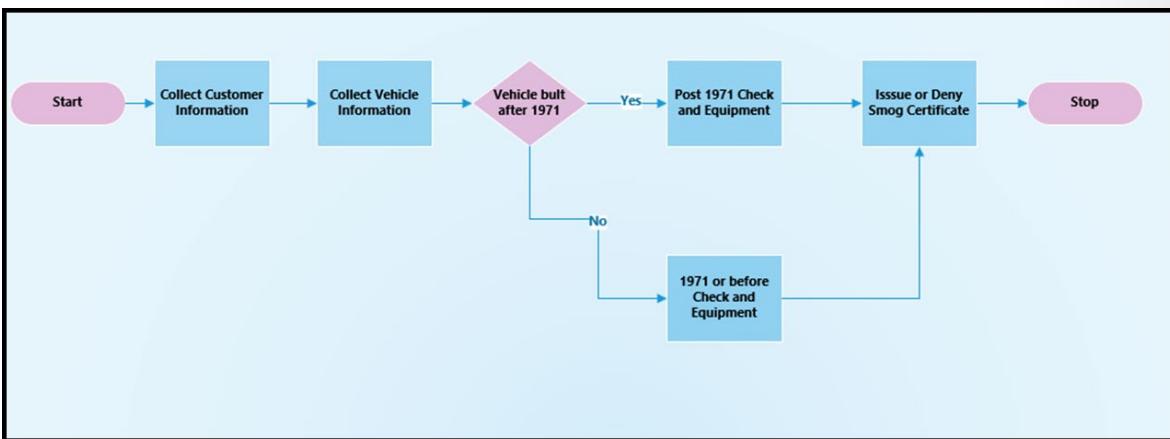
The previous module of this learning path explained how to create a simple, two-step linear business process flow. Some business processes are simple step-by-step sequential workflows, but many processes might need branching logic that requires one set of stages for one condition and another set of stages for an alternative condition. Think of this logic as a simple *if then else* statement.

Logical branching requires a decision point and a test of a value or condition. If a condition matches a certain value, then the flow goes through one branch with one or many stages; if it does not, then the flow goes through another set of one or many stages. Occasionally, the two logical paths will meet at a rendezvous point; other times, they will not. The following sections explore how you can use logical branching with business process flows.

Branching logic is useful when you are trying to model a business process to standardize data collection and improve process outcomes. Logical branching enables the creation of business process flows that can adapt to different conditions within the same flow instead of having to create and launch many different flows to handle one condition or another.

For example, consider the scenario from the previous module, where customers refer to the fictitious company SmogChecksRUs for bi-annual smog checks on their vehicles. In many jurisdictions, different requirements are in place for emission control standards and equipment based on the year of the vehicle manufacture. It would be useful to be able to model all required tests within a single business process flow. You can accomplish this task by using logical branching within a business process flow.

The following diagram models the branching business logic that you need to build into your flow.



The business process flow logical diagram shows that the smog equipment and the required checks are different for vehicles that are built before or during 1971 and after 1971. As a result, you need to add a logical branch to check the year when the vehicle was manufactured, collect different information, and then perform different checks on vehicles that are built after 1971. The exercises at the end of this module explain how to build a business process flow to manage this scenario.

Unfortunately, business process flows have some limitations on what can be modeled with logical branching. The following sections examine what types of logical branching can't be modeled and built in business process flows today.

## Unsupported logical branching and a possible work-around

Occasionally, you will want to model a business process that has multiple branches, and not all branches end up at the same rendezvous stage. Currently, this situation is not allowed in a business process flow. An example of an unsupported flow is shown in the following image.

Notice the choices that exist around the Vehicle Service decision point.

Two of the branches end up at the **Issue or Deny Smog Certificate** stage and then move into the **Collect Payment** stage, whereas the top branch skips the **Issue or Deny Smog Certificate** stage and then ends at the **Collect Payment** stage. This logical model is not supported.

All stages at a decision point must resolve to the same stage, and cannot have intermediate stages for some but not other branches.

The root of the problem in the preceding figure is that the logic is not modeled properly. The decision stage is modeling two *if then else* conditions and not one. A better way to model this logic is to add a second decision point, as shown in the following figure. The approach that is shown in the diagram is supported because the decision stages are each testing one condition and all resolve to a single stage.

Study this example so you understand how to add another logical branch and model business process flows correctly.

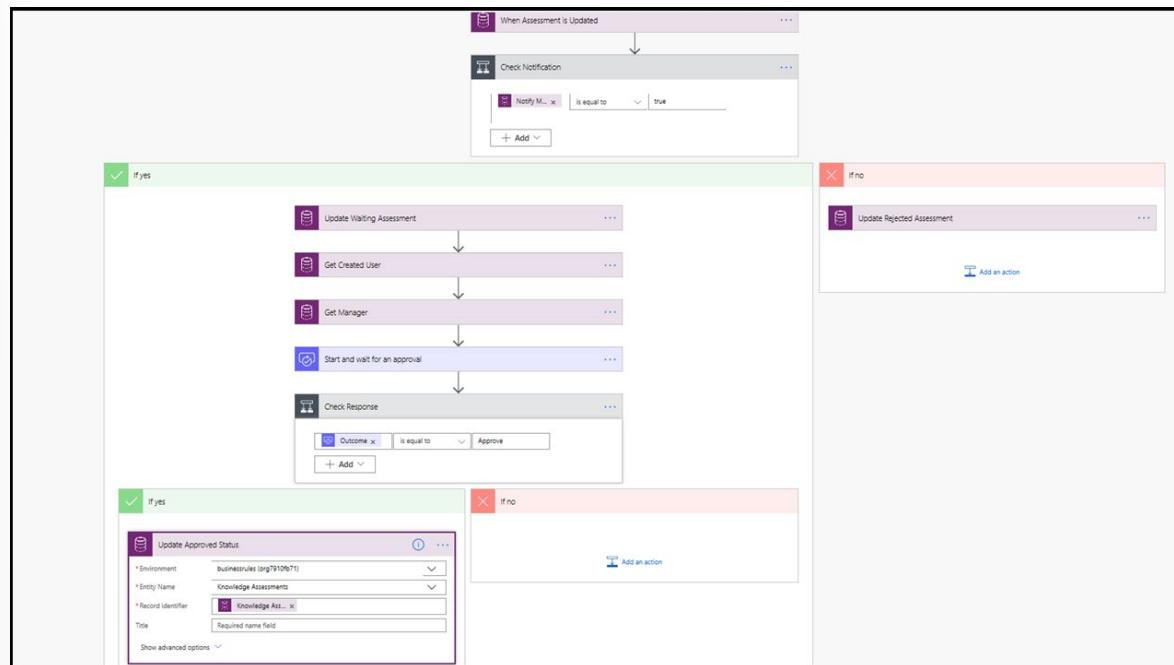
## Instant flows

Business process flows aren't the exclusive workflow automation option that is available to an entity in Common Data Service. In fact, a single business process flow can run against up to five entities in Common Data Service. Dynamics 365 workflows are also supported as part of a business process flow. This topic is beyond the scope of this learning path, but if you are interested in this topic, you can learn more about Dynamics 365 workflows by selecting the related link in the summary unit of this module.

A third workflow automation, called instant flows, in Power Automate offers a powerful capability that can supplement business process flows.

Microsoft Power Automate offers instant workflows that can run (trigger) automatically based on a schedule, time, data value, or if a record is added, selected, or modified. These workflows can work in conjunction with a business process flow to gather approvals, copy data from Common Data Service to another data source, or trigger an email notification to specific users to alert them that something has changed and needs their attention. You can make instant flows that automatically trigger based on a record that is created, deleted, or modified, or you can make the flows run (trigger) by having a user select a button within a step in a stage.

Instant flows are made with Power Automate and they start with a single trigger followed by a series of actions. The following figure shows an example of an instant flow that sends an approval request to a user when a record in an entity in Common Data Service is modified. In the exercises at the end of this module, you'll create an instant workflow in Power Automate and add it to your business process solution.



Business process flows work within Power Platform and many options are available, including instant flows and approval flows that can run in concert with a business process flow.

Business process flows don't exclude the possibility of adding more capabilities to the envisioned solution by using additional components of Power Platform like Power Apps, Power Automate instant flows, or Power BI Dashboards. You can read more about instant flows by selecting the related links in the summary unit within this module.

## Enhance the form to augment a business process flow

Examine the business process flow that was created in the exercise for the previous module, Create an immersive business process flow in Power Automate. Review the following screenshot and notice the visual representation of the process flow and a form under the business process flow called **General**, which contains only the **Name** and **Owner** fields.



If you want to add more fields to a form such as this, it would be beneficial. For example, if you want to record the amount of the invoice, you could record an autogenerated invoice number and then add general notes about the service that you completed for a customer.

Adding more fields and displaying them on a form that is associated with an immersive business process flow can be simple and quick. These flows are powerful, and you are only limited by your imagination and skill.

## Exercise – Add branching logic to a business process flow

This exercise assumes that you have completed the exercise to create an immersive business process flow in the previous module of this learning path. If you have not, return to the previous module and complete the exercises within prior to attempting the exercises in this module.

In this exercise, you'll enhance the business process flow by adding a logical branch to determine what test needs to be performed and what information is required to collect to complete a smog check on a vehicle based on the vehicle's manufacture year. You will add

logic that will provide a different set of instructions for vehicles that were manufactured prior to, or during, 1971 versus vehicles that were manufactured after 1971.

1. Sign in to **Power Automate**<sup>52</sup> and make sure that you are in the same environment that you used to create the process flow in the previous module.
2. Select **My flows** and then select **Business process flows**.
3. Select the **Edit** button (pencil icon) and open the **Customer Check In** business process flow in the **Business process flow** editor.

The screenshot shows the Microsoft Power Automate interface. On the left is a navigation sidebar with options like Home, Action items, My flows (which is selected and highlighted with a blue bar), Create, Templates, Connectors, Data, AI Builder, Solutions, and Learn. The main area is titled 'Flows' and has tabs for My flows, Team flows, Business process flows (which is selected and underlined in blue), and UI flows (preview). Below these tabs is a table listing flows. The first row, 'Customer Check In', has its entire row highlighted with a red box. The 'Edit' button (pencil icon) next to it is also highlighted with a red box. Other rows in the table include 'Translation Process', 'Expired Process', and 'New Process', each with their own set of icons for Run, Edit, and More options.

Before you create the conditions, you need to set up the data that you want to collect in each stage that you add. Select the **Customer Check In** stage and add the following fields by using the **Click to add field and forms** hyperlink on the right side of the page.

[!NOTE]

The entity window may open in the an older UI. Alternatively, you can open Power Apps, select your environment, expand **Data** and select **Entities** to see the modern UI.

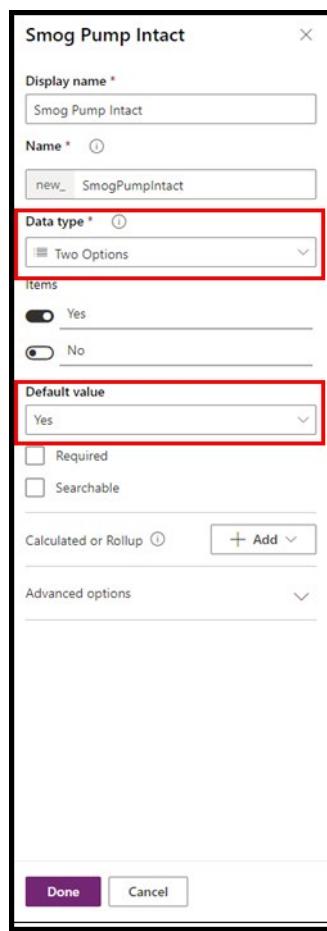
1. Select the **Add field** button in the ribbon above the fields that are shown for the **Customer Check In** entity, as shown in the following figure, and then add the following fields.

---

<sup>52</sup> <https://preview.flow.microsoft.com/?azure-portal=true>

The screenshot shows the Microsoft Power Apps interface. On the left, there's a navigation sidebar with options like Home, Learn, Apps, Create, Data, and Flows. The main area is titled 'Solutions > Default Solution > Customer Check In'. Below this, there are tabs for Fields, Relationships, Business rules, Views, Forms, Dashboards, Charts, and Keys. The Fields tab is selected. A table lists three fields: 'Active Stage' (Name: activestageid), 'Active Stage Started On' (Name: activestagestartedon), and 'Completed On' (Name: completedon). At the top of the Fields section, there's a red box highlighting the '+ Add field' button.

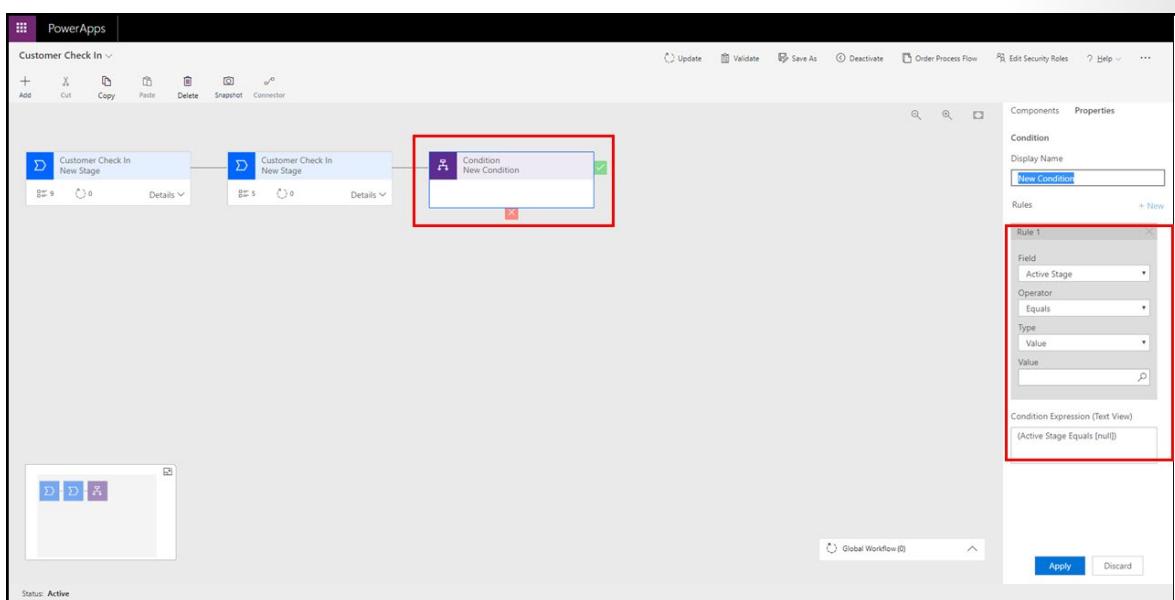
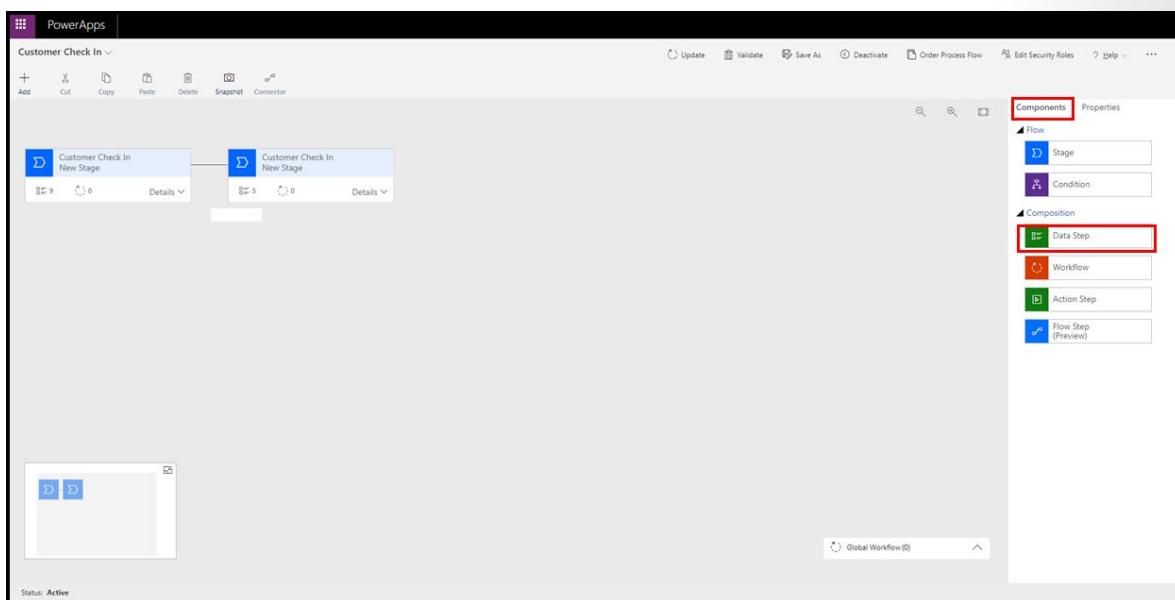
- Smog Pump Intact - Two Options - Set Yes as default
- PVC Valve Intact - Two Options - Set Yes as default
- Aftermarket Exhaust Headers - Two Options - Set No as default
- Comments Pre 1972 - Text Area
- Exhaust Test Performed with Passing Score - Two Options - Set Yes as default
- HC Reading at 2000 RPM - Floating Number
- O2 Reading 2000 RPM - Floating Number
- CO Reading 2000 RPM - Floating Number
- Original Equipment - Two Options - Set Yes as default
- Comments Post 1972 - Text Area
- Amount - Currency
- Certificate Number - Whole Number
- Payment Method - Create a new option set with the following options:
  - Cash
  - Mastercard
  - Visa
  - Discover
  - Debit Card
  - Bit Coin



After you have created the new fields, select the **Save Entity** button and then close the screen and return to the **Business process flow** designer.

Solutions > Default Solution > Customer Check In						
Fields	Relationships	Business rules	Views	Forms	Dashboards	Charts
Display name	Name	Data type	Type	Customizable	Required	Searchable
Active Stage	... activestageid	Lookup	Custom	✓	✓	
Active Stage Started On	... activestagestartedadon	Date Only	Custom	✓	✓	
Address	... new_address	Text	Custom	✓	✓	
Aftermarket Exhaust Headers	... new_aftermarketexhaustheaders	Two Options	Custom	✓		
Amount	... new_amount	Currency	Custom	✓	✓	
Amount (Base)	... new_amount_base	Currency	Custom	✓	✓	
Automobile Comments	... new_automobilecomments	Text Area	Custom	✓	✓	
Automobile Make	... new_automobilemake	Text	Custom	✓	✓	
Automobile Milage	... new_automobilemilage	Whole Number	Custom	✓	✓	
Automobile Model	... new_automobilemodel	Text	Custom	✓	✓	
Automobile Year	... new_automobileyear	Whole Number	Custom	✓	✓	
City	... new_city	Text	Custom	✓	✓	
Comments	... new_comments	Text Area	Custom	✓	✓	
Comments Post 1972	... new_commentspost1972	Text Area	Custom	✓	✓	
Comments Pre 1972	... new_commentspre1972	Text Area	Custom	✓	✓	
Completed On	... completedon	Date Only	Custom	✓	✓	
Context URL	... bpf_contexturl	URL	Custom	✓	✓	
CO Reading 2000 RPM	... new_coreding2000rpm	Floating Point Number	Custom	✓		
Created By	... createdby	Lookup	Standard	✓	✓	
Created By (Delegate)	... createdonbehalfby	Lookup	Standard	✓	✓	

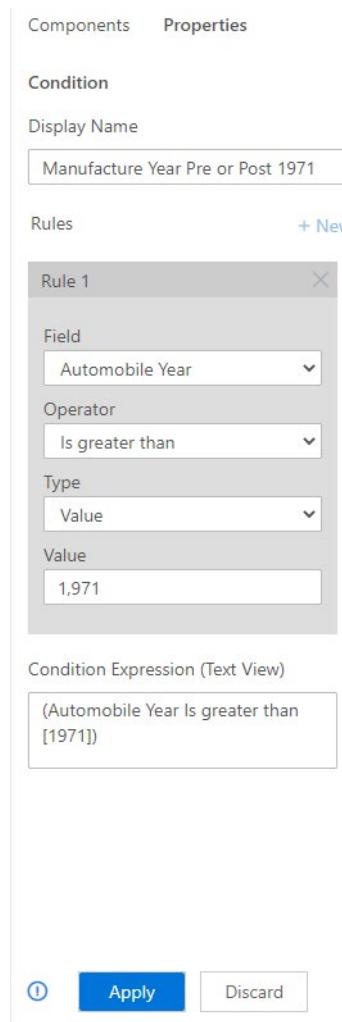
2. Select the **Condition** flow control under the **Components** tab and then drag it to the right of the **Customer Check In** stage, as shown in the following image.



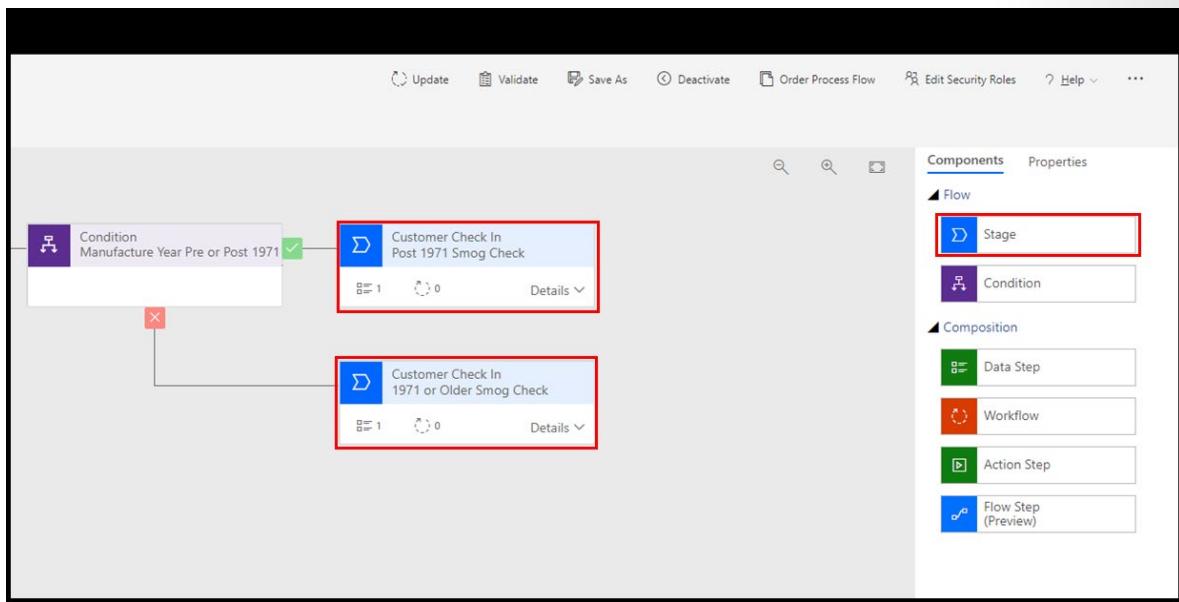
3. Select the **Condition** component within the editor and enter the following information:

- **Display Name** - Manufacture Year Pre or Post 1971
- **Field** - Automobile Year
- **Operator** - Is greater than
- **Type** - Value
- **Value** - 1971

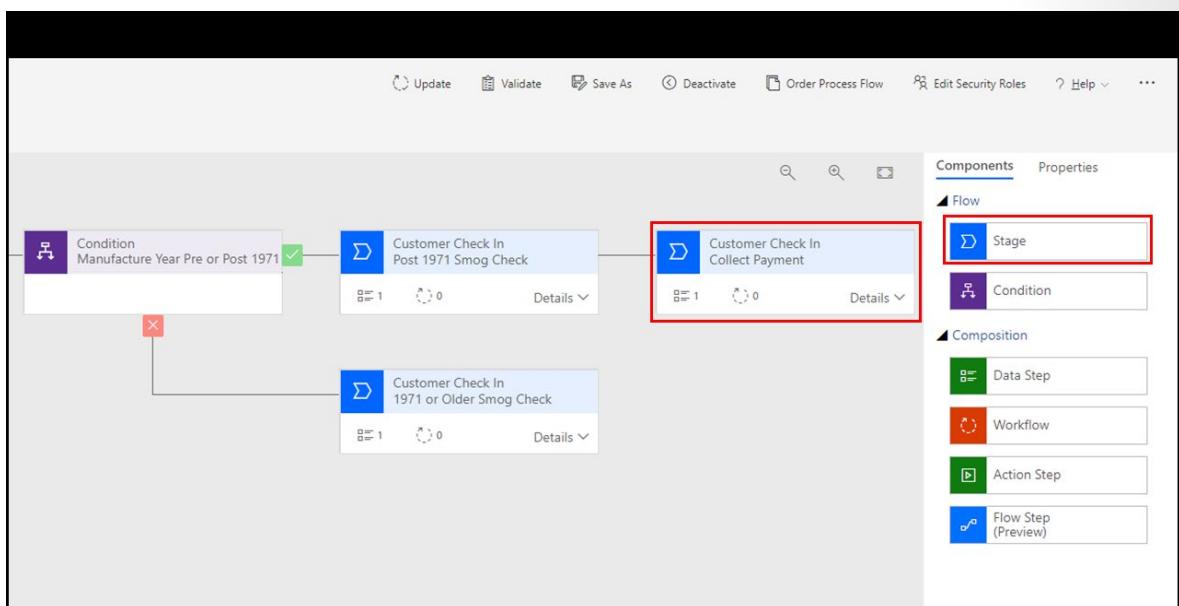
The entries should resemble the following screenshot. When finished, select the **Apply** button.



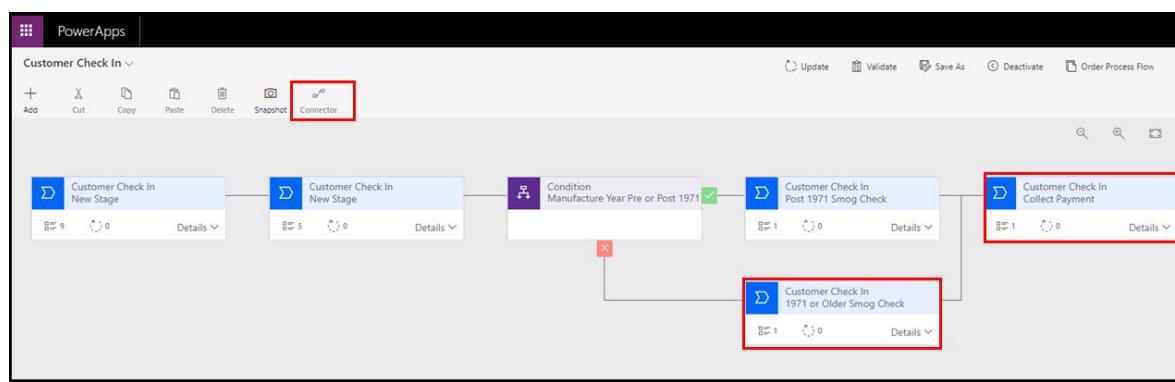
4. Select the **Components** tab and then drag a new stage to the plus (+) sign on the right of the page and another to the plus (+) sign under the **Condition** stage, as shown in the following figure.  
Name each of the new stages by selecting each new stage and entering the new name.



5. Drag another stage to the right of the **Customer Check In Post 1971 Smog Check** stage, as shown in the following screenshot. Rename this last stage as **Collect Payment**.



6. Connect the **Customer Check In 1971 or Older Smog Check** stage to the **Collect Payment Stage** by following these steps:
  1. Select the **Customer Check In 1971 or Older Smog Check** stage.
  2. Select **Connector** in the ribbon and then select the **Connect** option.
  3. Select the **Customer Check In Collect Payment** stage to connect the two stages, as shown in the following screenshot.



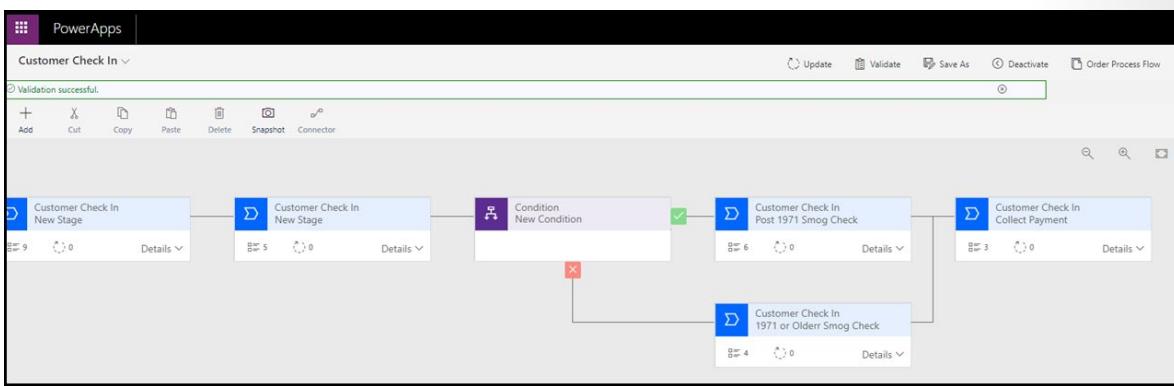
Now, you will add the fields that you created to each stage. Refresh your browser and then select the following stages. Add the fields that are noted for each stage as a step under the stage by selecting the details link drop-down menu and then adding data steps.

Add a data step for each field within each stage. When you are done, you should have a data step under each of the following stages.

Stage - Customer Check In Post 1971 - add the following fields by using the Add field option:

- Smog Pump Intact
  - PVC Valve Intact
  - Aftermarket Exhaust Headers
  - Comments Pre 1972
- Stage - Customer Check In 1971 or Older - add the following fields:
  - Exhaust Test Performed with Passing Score
  - HC Reading at 2000 RPM
  - O2 Reading 2000 RPM
  - CO Reading 2000 RPM
  - Original Equipment
  - Comments Post 1972
- Stage - Customer Check In Collect Payment - add the following fields:
  - Amount
  - Certificate Number
  - Payment Method

When you're done, select the **Update** button in the top ribbon.  
Your business process flow should look like the following screenshot.



Now, you'll test the enhanced business process flow.

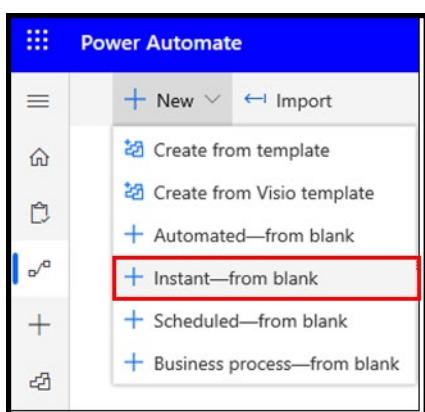
1. Select **My flows** and then **Business process flows**.
2. Run the Customer Check In flow by selecting the **run** button (the little triangle button next to the name of the flow).

You can enter a vehicle with a manufactured date of 1971 or before and another after 1971. Notice that the smog check information in Stage 3 changes based on the year of vehicle manufacture. Additionally, notice that both potential flows reconnect again at the last stage called **Collect Payment**.

## Exercise – Add an instant flow

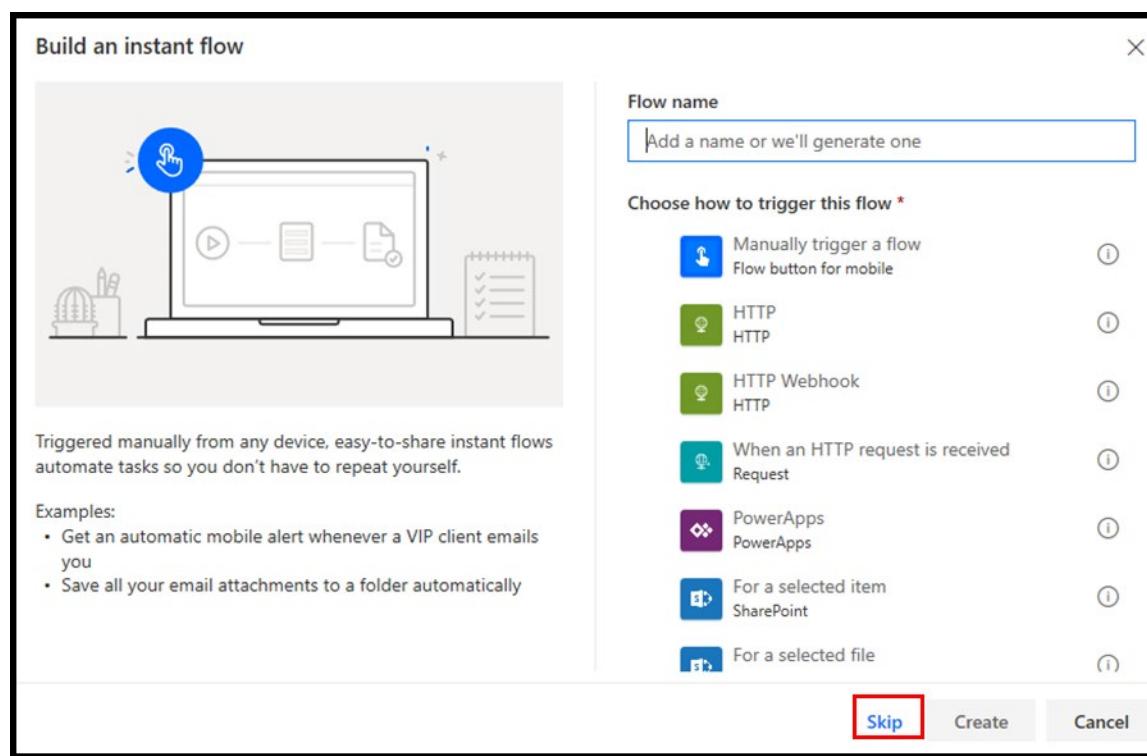
In the previous exercise, you created a business process flow that includes logical branching for vehicles that were made in 1971 or before and all others that were made in 1972 and beyond. Now, you will enhance that flow by adding a simple instant flow in Power Automate. You'll create an instant flow that sends an email to the store manager (you) when a new customer check-in record is created.

1. Sign in to **Power Automate**<sup>53</sup> and select **My flows**.
2. Select **+ New** in the ribbon in the left corner of the screen.
3. Select the **+ Instant—from blank** option, as shown in the following screenshot.

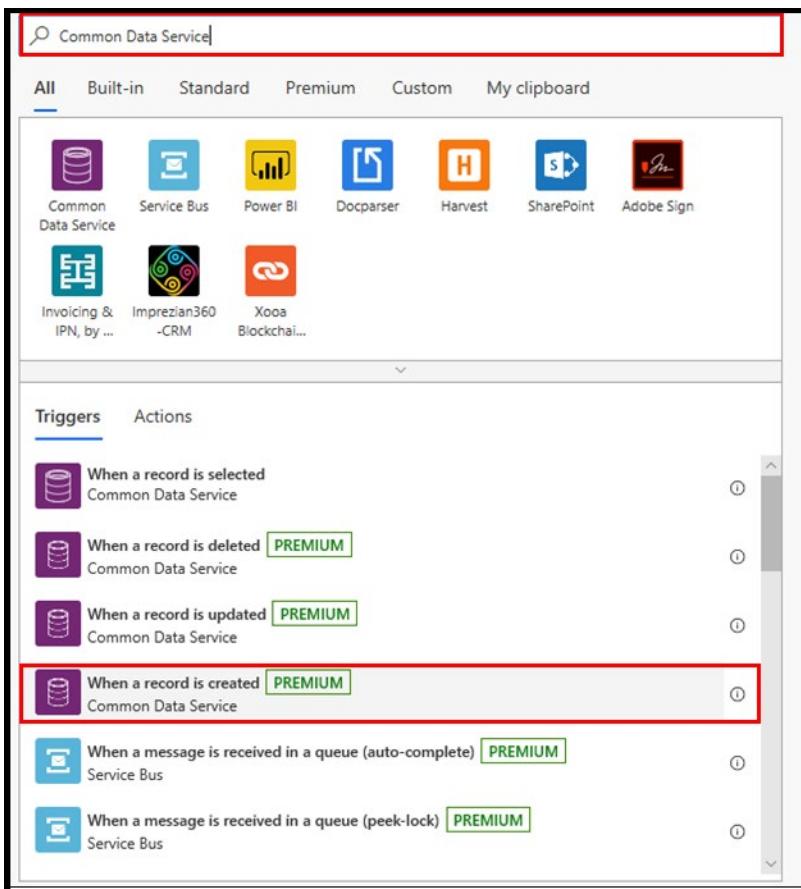


4. Select the **Skip** button to open the flow instant designer.

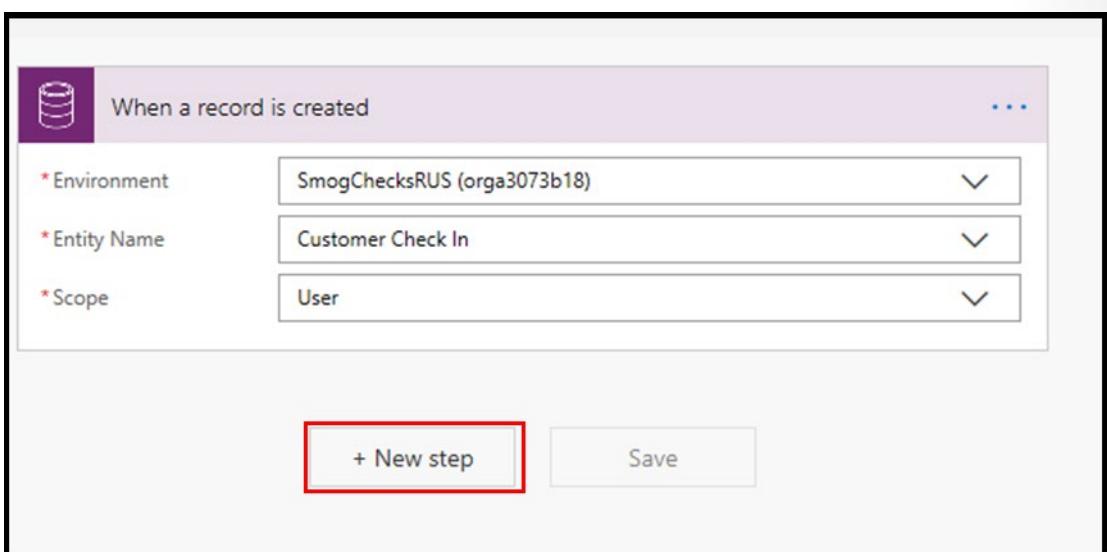
<sup>53</sup> <https://preview.flow.microsoft.com/?azure-portal=true>



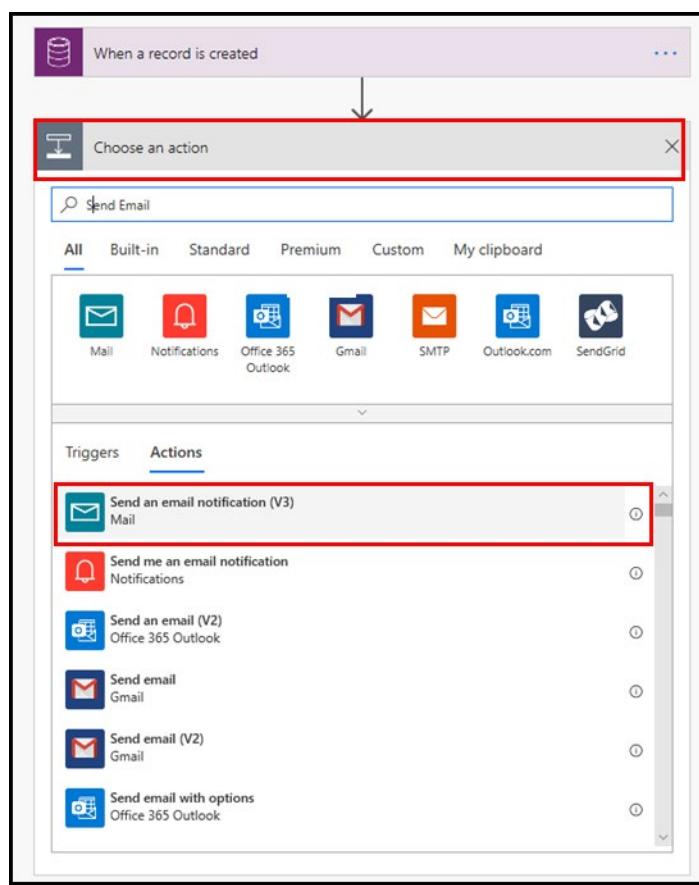
5. Enter **Common Data Service** in the connector search dialog box, as shown in the following screenshot.



6. Select the Common Data Service connector and the **When a Record is created** trigger. Select the **Environment** name that you used in the previous exercises, select **Customer Check In** from the **Entity Name** field, select the **User** in the **Scope** field, and then select **+ New step**, as shown in the following figure.



7. Enter **Send Email** in the actions search box and then select **Send an email notification (V3)**.

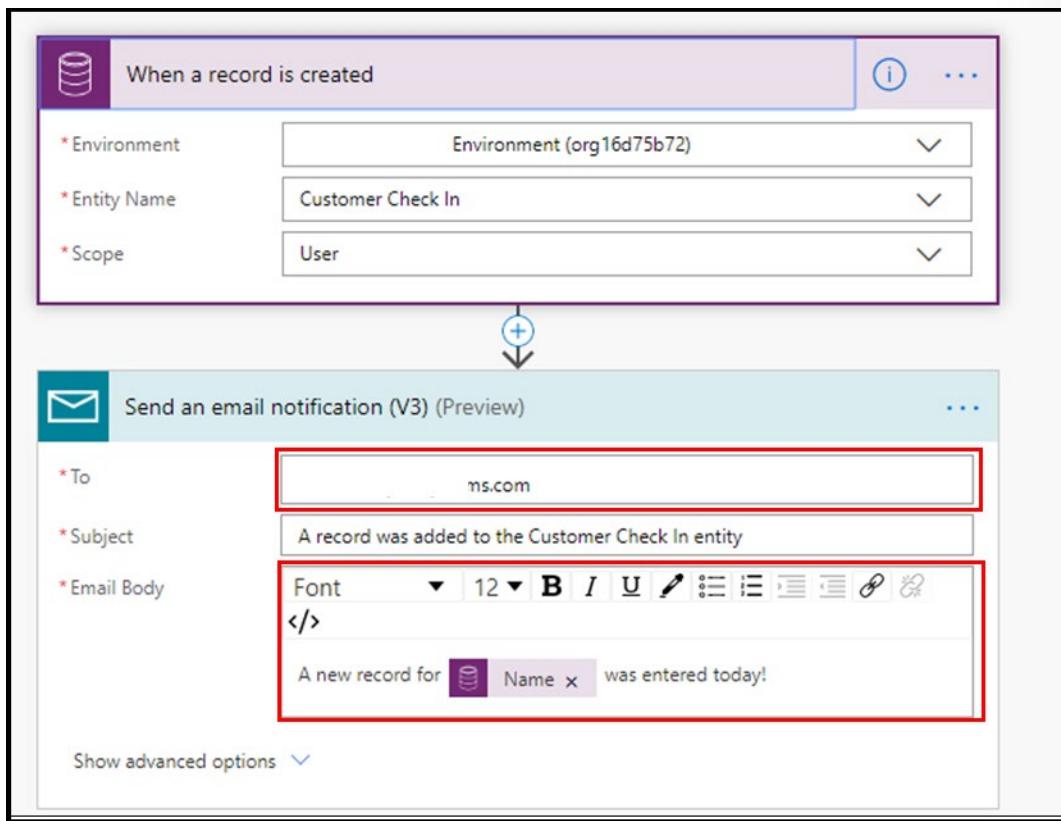


8. Enter the following information into the **Send an email notification (V3)** action:

- **To** - Enter your email address so you receive an email and can see how the instant flow works.
- **Subject** - Enter **A record was added to the Customer Check In entity**.

Select the **First Name** and **Last Name** fields from the list of fields under the Dynamic content option.

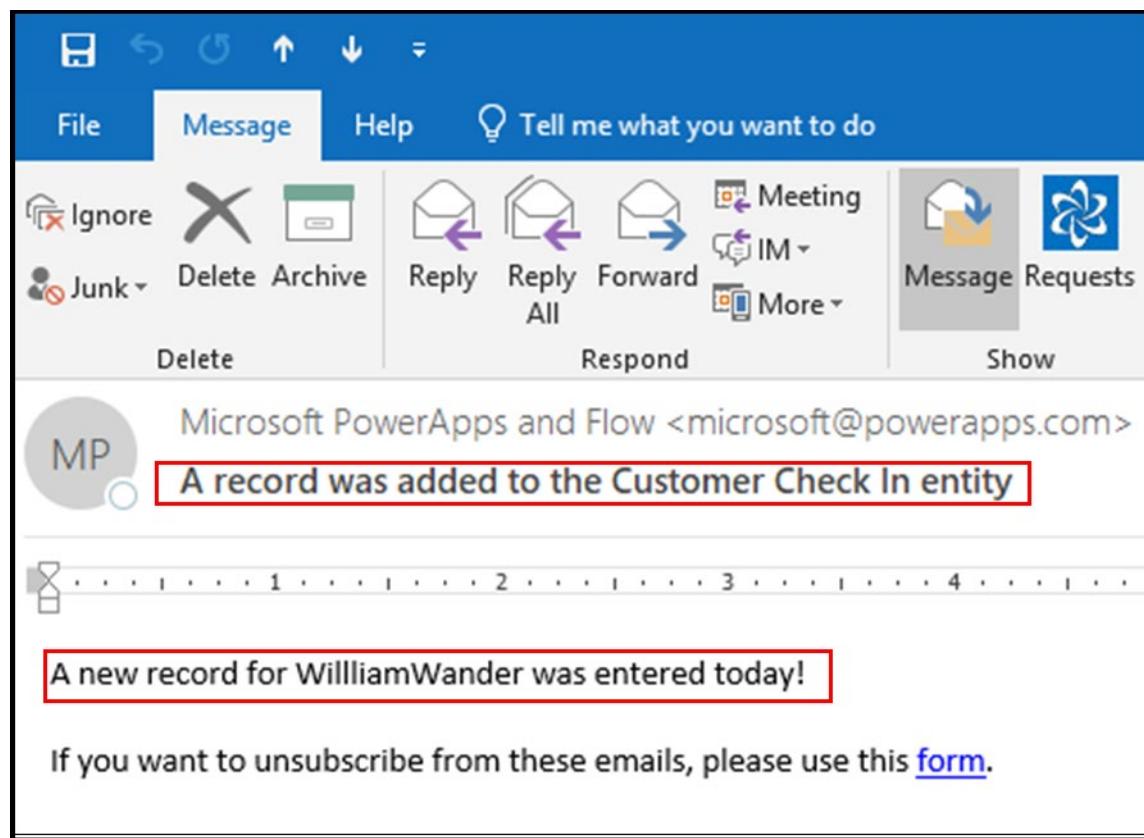
- **Email Body** - Enter **A new record for** (select the **new\_firstname** field and the **new\_lastname** field), and then type **was entered today!**



9. Save the flow by selecting **Save** in the top ribbon.

## Run the instant flow

1. Close the instant flow that you created and then select **My flows** and **Business process flows**.
2. Run the **Customer Check In** business process flow.
3. Enter some values in the first stage fields and then enter a value in the **Name** field on the main screen (the field is on the **General** form under the Business process flow diagram) so that values are in the record before you save it.
4. Save the record. An email will appear in your inbox similar to the one in the following figure.



Congratulations, you have now created an instant flow that works with your business process flow.

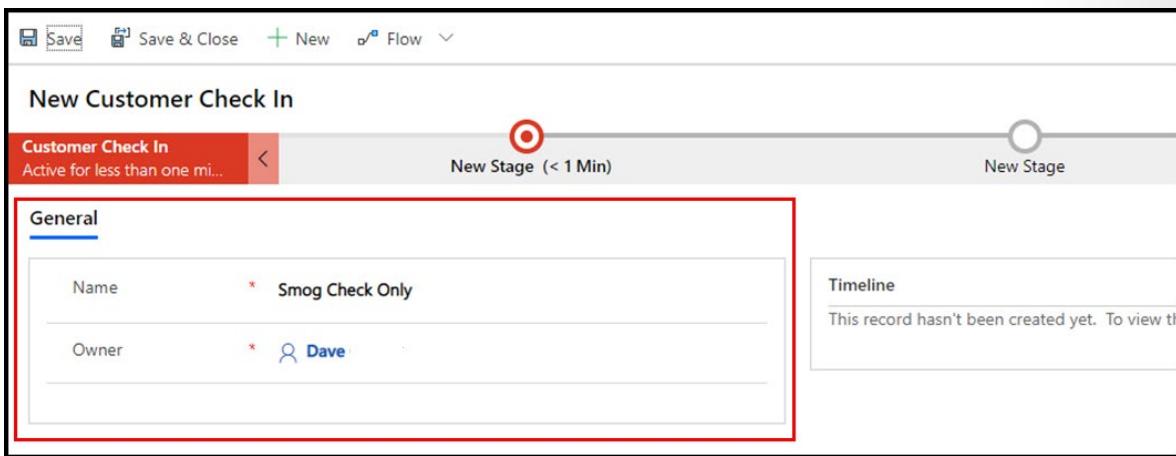
## Exercise – Enhance the main form that is associated with the business process flow

One last enhancement needs to be made before you can complete your business process flow. Before you begin, quickly review the fields on the main form under the diagram of your business process flow.

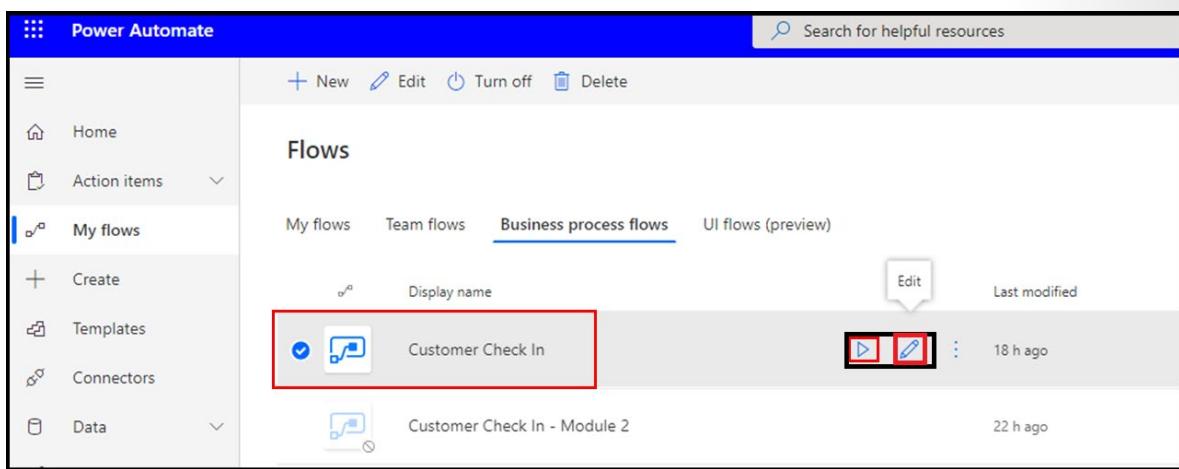
The following screenshot shows a new instance of your business process flow. Notice that only two fields are shown on the main screen (outlined in red): **Name** and **Owner**. The name of the record is similar to the title of this record, and the owner of the record is the current user who created the instance of the current business process flow. Now, you will add some fields to make this screen more useful.

### Add fields to the main form

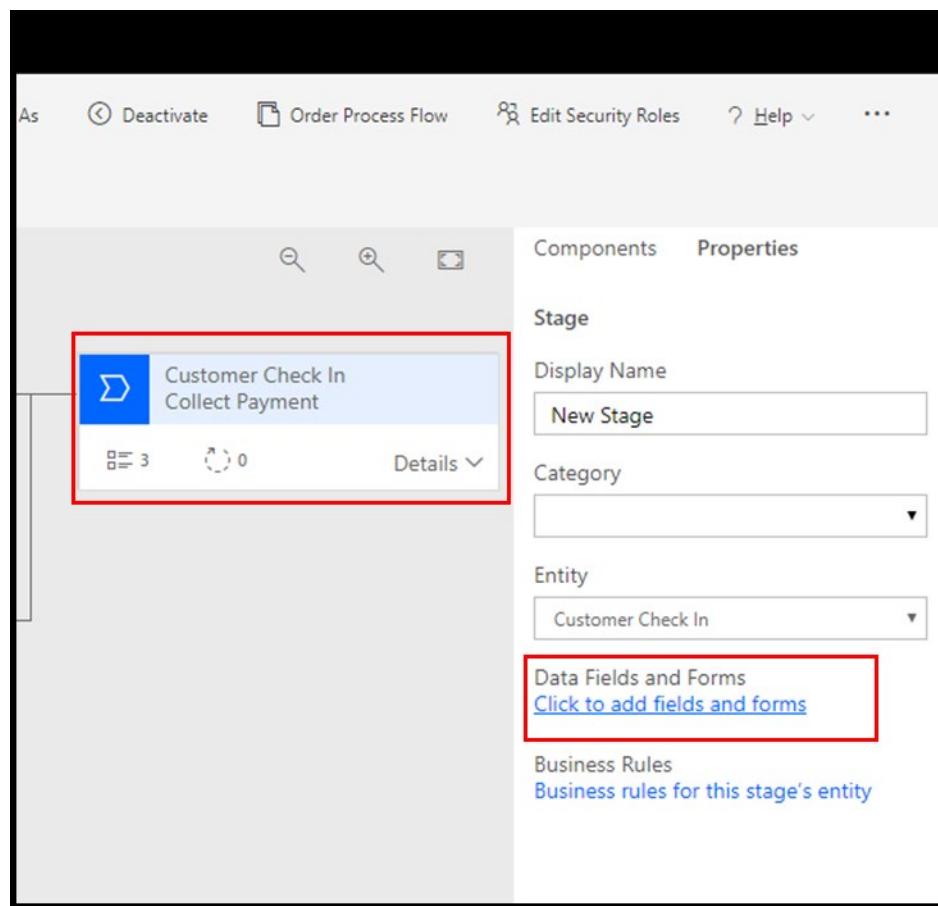
Follow these steps to add fields to the main form.



1. Sign in to Power Automate, make sure that you are in the proper environment that you have been working in, select **My flows**, and then select **Business process flows**.
2. Open the **Customer Check In** flow in design mode by selecting the pencil icon, as shown in the following screenshot.



3. Select the last stage (Collect Payment) and then select the **Data Fields and Forms** link.



4. Start your enhancements by selecting the **+Add field** button and then adding the following fields.

The screenshot shows the 'Fields' page for the 'Customer Check In' stage. A red box highlights the '+ Add field' button in the top navigation bar. The page displays a list of existing fields with columns for 'Display name', 'Name', and 'Data type'. The fields listed are: Active Stage (activestageid, Lookup), Active Stage Started On (activestagestartedon, Date Only), Address (new\_address, Text), Aftermarket Exhaust Headers (new\_aftermarketexhaustheaders, Two Options), and Amount (new\_amount, Currency). The left sidebar shows navigation options like Home, Learn, Apps, Create, Data, Flows, AI Builder, and Solutions.

Display name ↑	Name	Data type
Active Stage	activestageid	Lookup
Active Stage Started On	activestagestartedon	Date Only
Address	new_address	Text
Aftermarket Exhaust Headers	new_aftermarketexhaustheaders	Two Options
Amount	new_amount	Currency

- Clerk - Text
- Transaction Date - Date Only
- Location - New Option Set
  - Los Angeles
  - San Francisco
  - San Diego

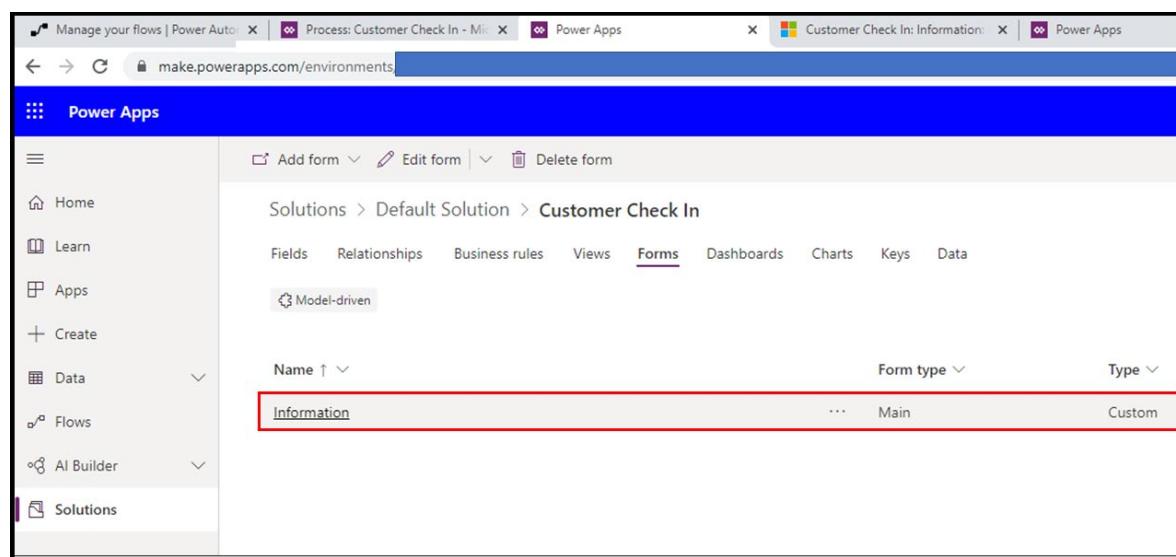
- Portland
  - Invoice Number - Autonumber - leave the defaults that are provided by Power Automate and Common Data Service
  - Service Comments - Text Area
5. After you have added the new fields, save the entity by selecting the **Save Entity** button. Don't skip saving the entity or your new fields will not be saved with the entity.

The screenshot shows the Microsoft Power Apps interface. On the left, there's a navigation sidebar with options like Home, Learn, Apps, Create, Data, Flows, AI Builder, and Solutions. The main area is titled 'Solutions > Default Solution > Customer Check In'. Below this, there's a 'Fields' tab selected, showing a list of fields with their display names, names, data types, and other properties. At the bottom right of the list, there are 'Discard' and 'Save Entity' buttons, with 'Save Entity' being highlighted by a red box.

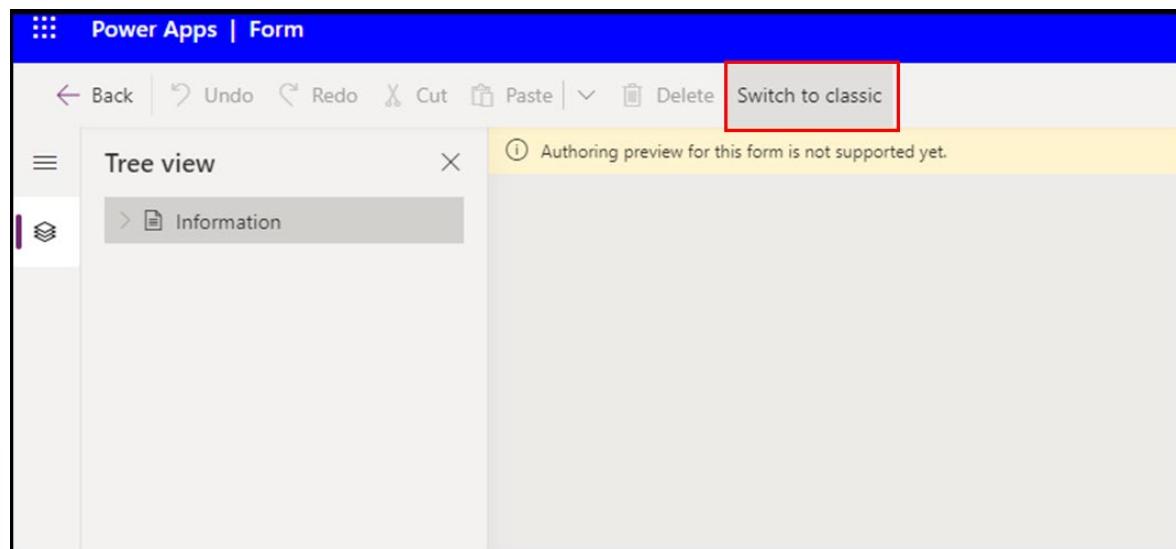
6. To enhance the main form, select the **Forms** tab on the current screen, as shown in the following figure.

The screenshot shows the Microsoft Power Apps interface again. The navigation sidebar is identical to the previous one. The main area is titled 'Solutions > Default Solution > Customer Check In'. Below this, the 'Forms' tab is selected, showing a list of forms. The first item in the list is 'Customer Check In', which is underlined and appears to be a hyperlink. The 'Discard' and 'Save Entity' buttons from the previous screen are also present here.

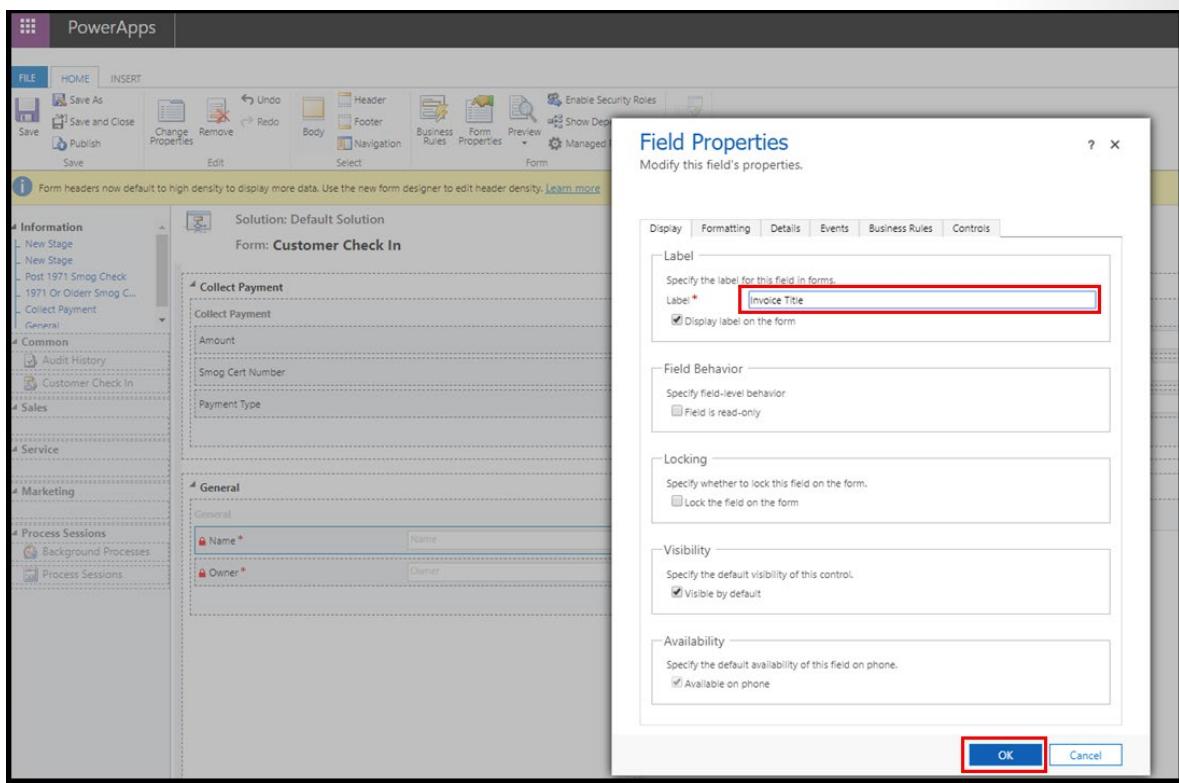
7. Select the top main form name, which is a hyperlink to the form designer (the name of your form might be different). This action will open the WYSIWYG forms designer. The WYSIWYG designer is not available for immersive business forms at this time.



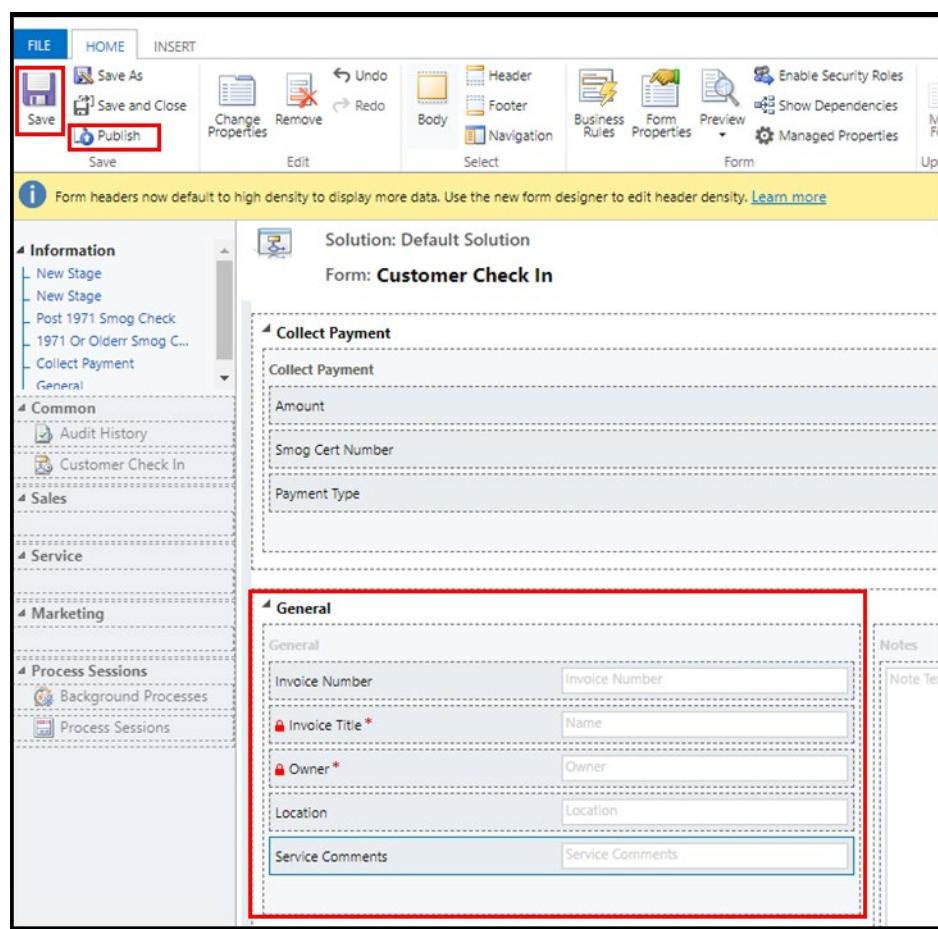
8. Select **Switch to classic** when the new screen opens.



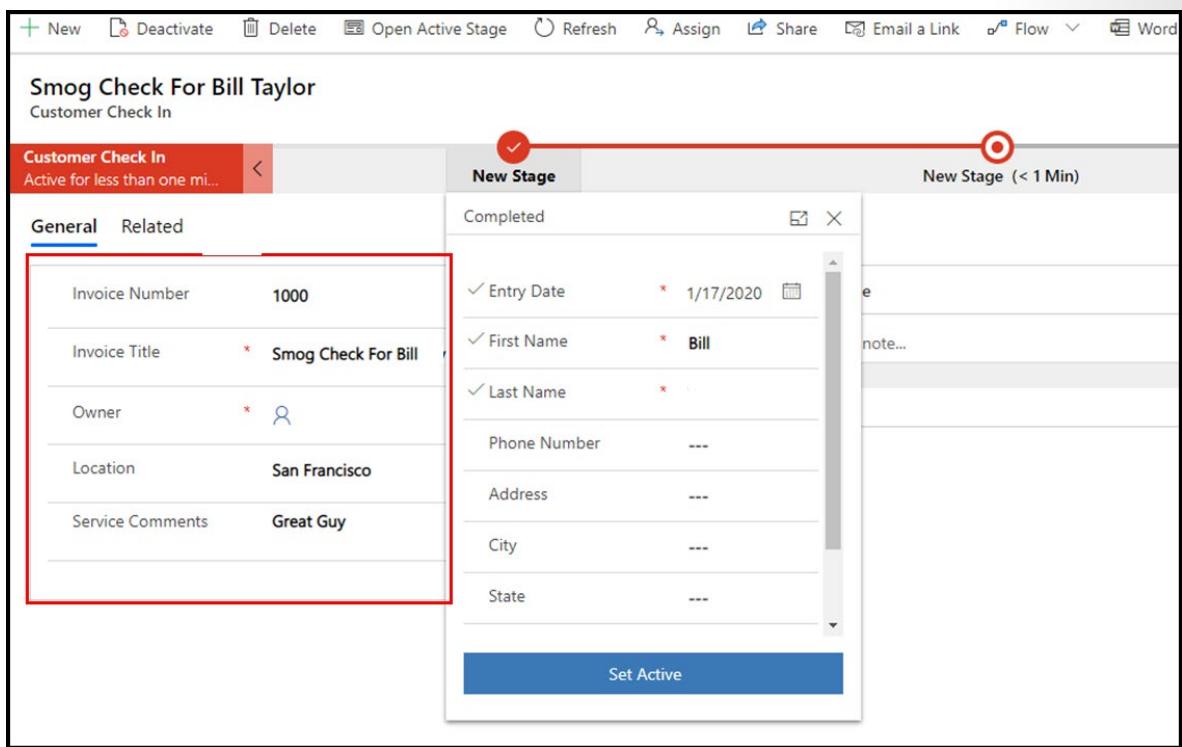
9. Scroll down to the **General** section at the bottom of the form. Double-click the **Name** field, rename the invoice title label to **Invoice Description**, and then select the **OK** button.



10. Drag the new fields that you previously added in this exercise onto the form, as shown in the following screenshot. Select the **Save** button and then select the **Publish** button in the ribbon at the top of the form designer.



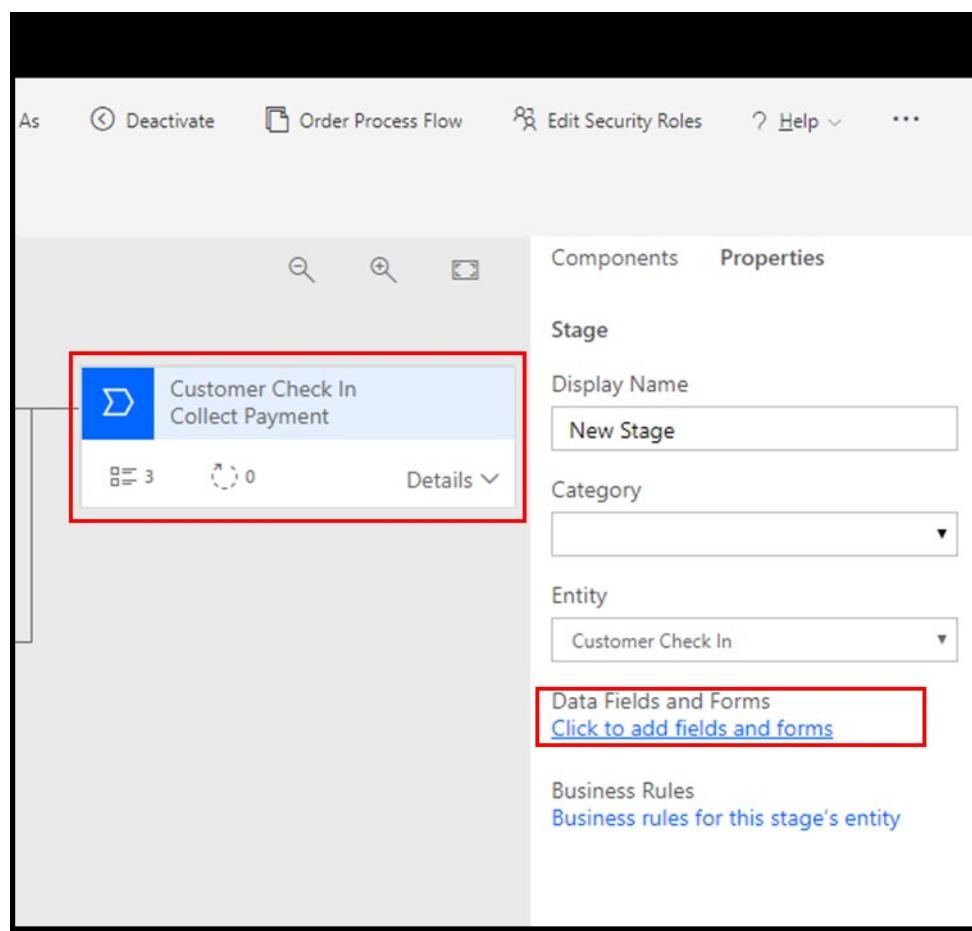
11. Close the **Designer** tab in the browser.
12. Go back to the home page of Power Automate. Select **My flows** and **Business process flows**. Select the **Customer Check In** flow and run an instance of the flow by selecting the small triangle icon next to the flow name. It should look like the following screenshot. Add data to the main form and the stages and then save the record.



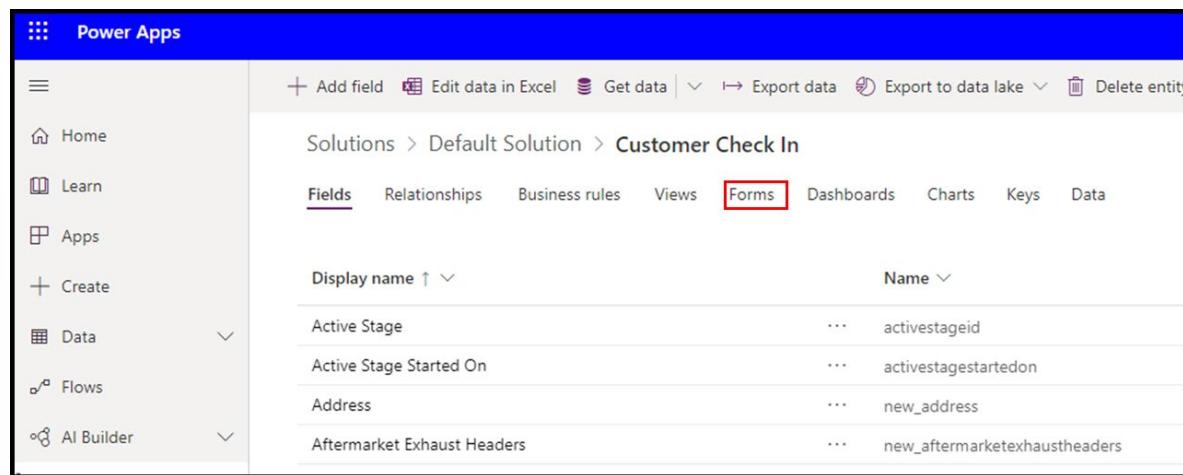
## Add read-only fields to the form

Your next task is to add a few fields to the main form from the data that you entered within a few of the stages. You will make these fields read-only.

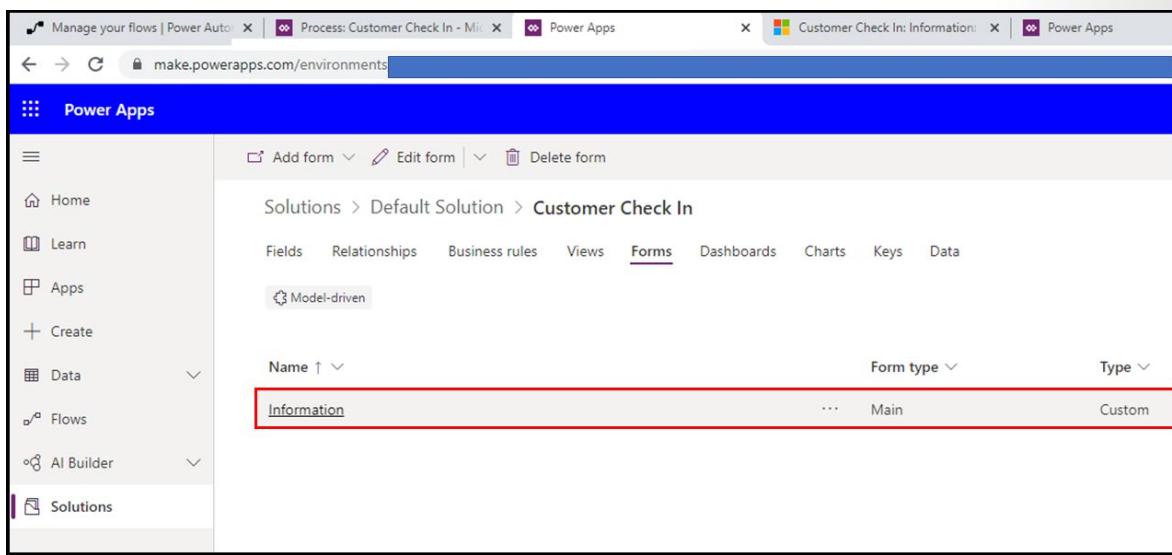
1. Go back to the list of business process flows (available from **My flows** in Power Automate).
2. Select the pencil icon next to the **Customer Check In** flow to enter the business process flow designer.
3. Select the **Collect Payment** stage and then select the **Click to add fields and forms** hyperlink, as shown in the following figure.



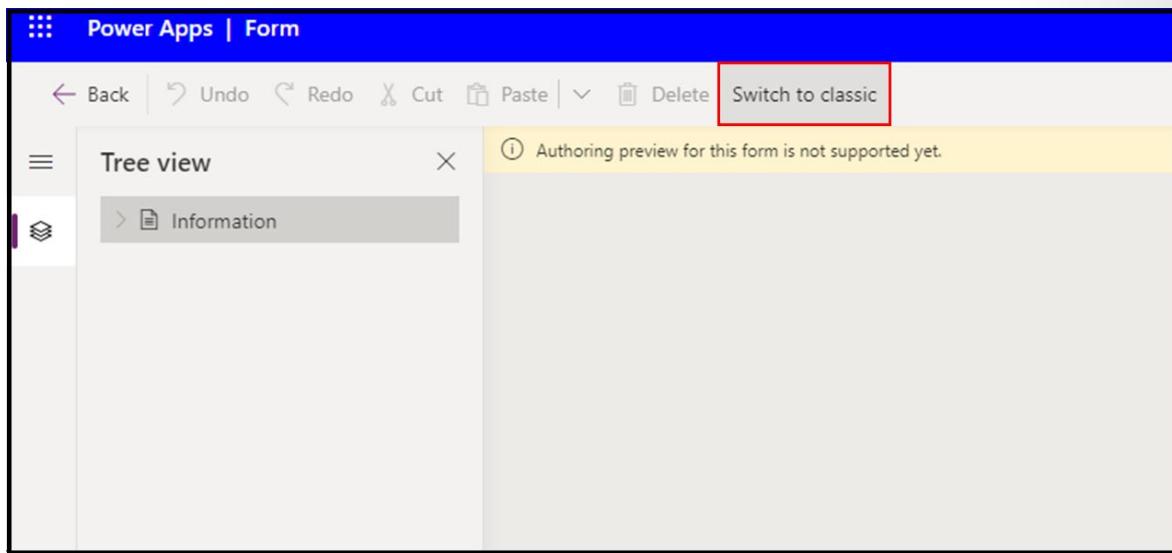
4. Select the **Forms** tab.



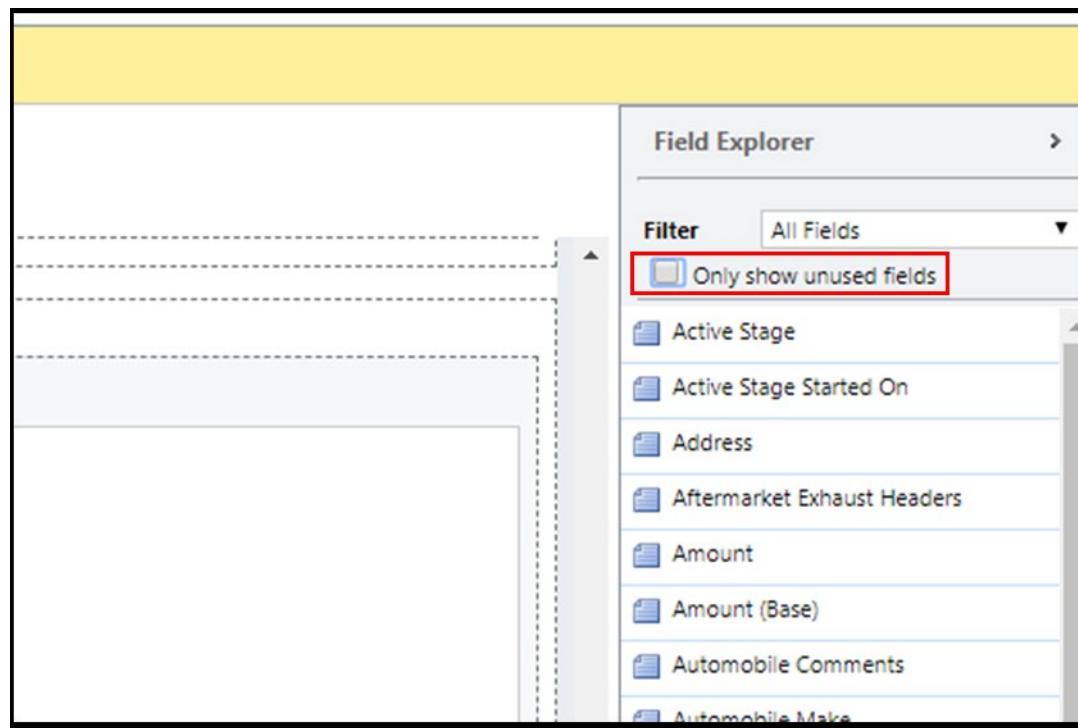
5. Select the name of the main form, which is a hyperlink to the form designer.



6. Select **Switch to classic**, as shown in the following figure. Now, you are back at the form designer, where you'll make a few more changes.

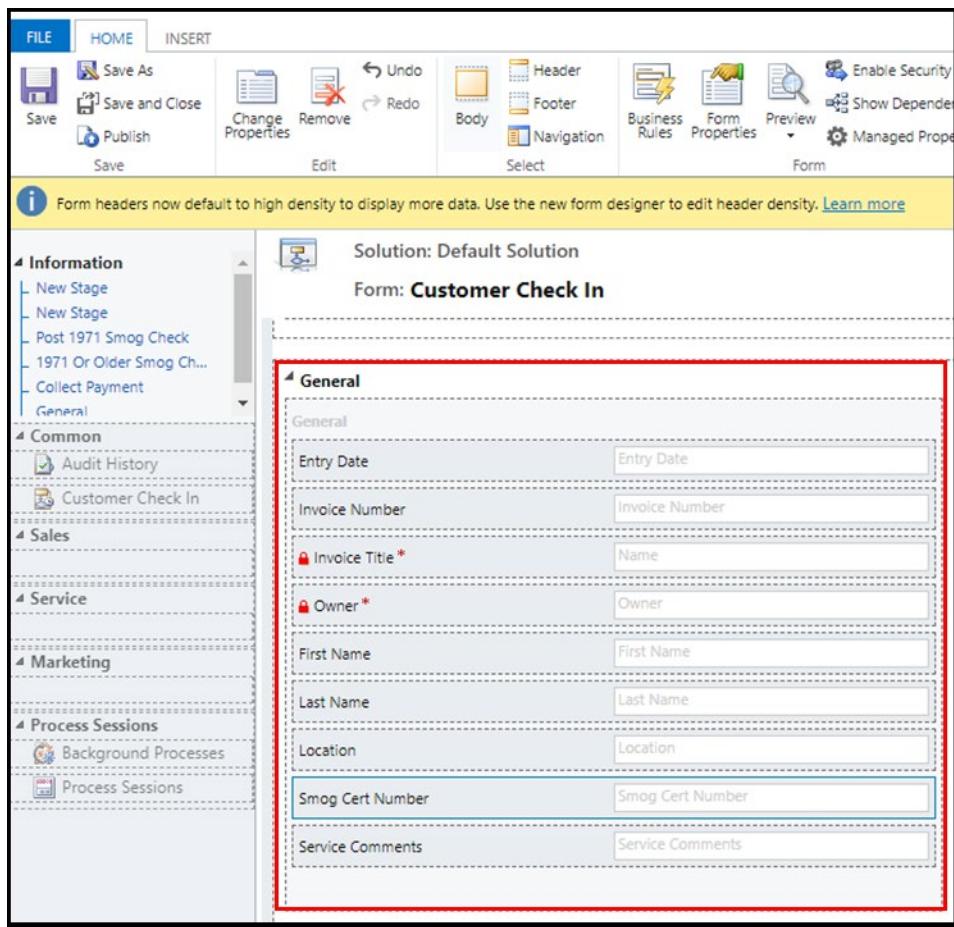


7. Enhance the main form with some of the data that you collect within the stages. Clear the **Only show unused fields** option above the list of fields.

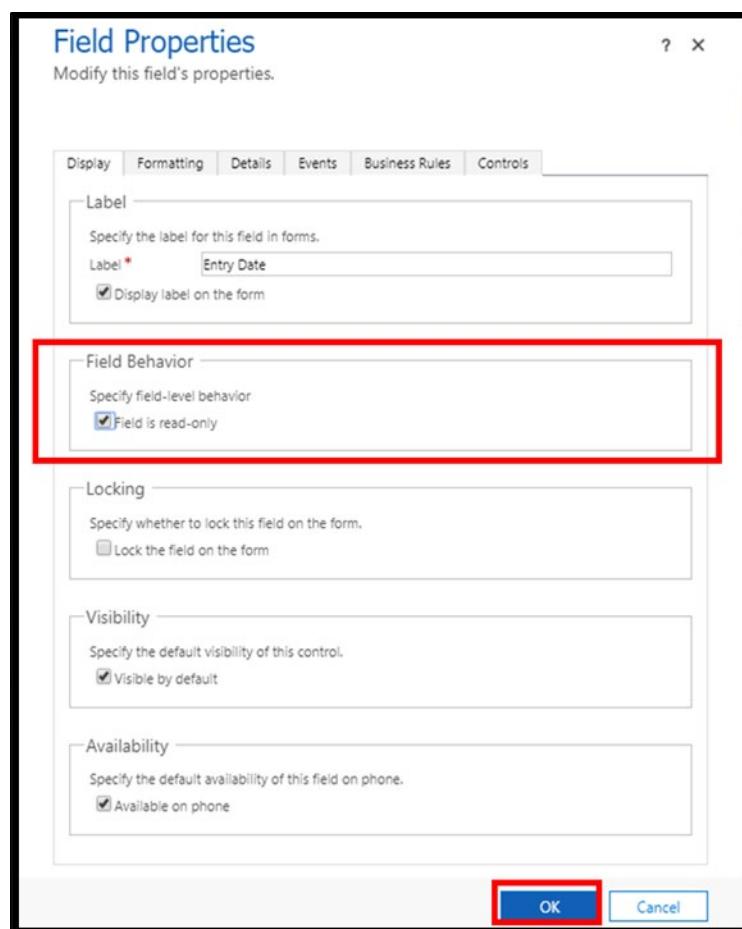


8. You should see all the fields that are available within the **Customer Check In** entity. Drag the following fields from the field panel into the General section of the form, as shown in the following screenshot.

- Entry Date
- First Name
- Last Name
- Smog Cert Number



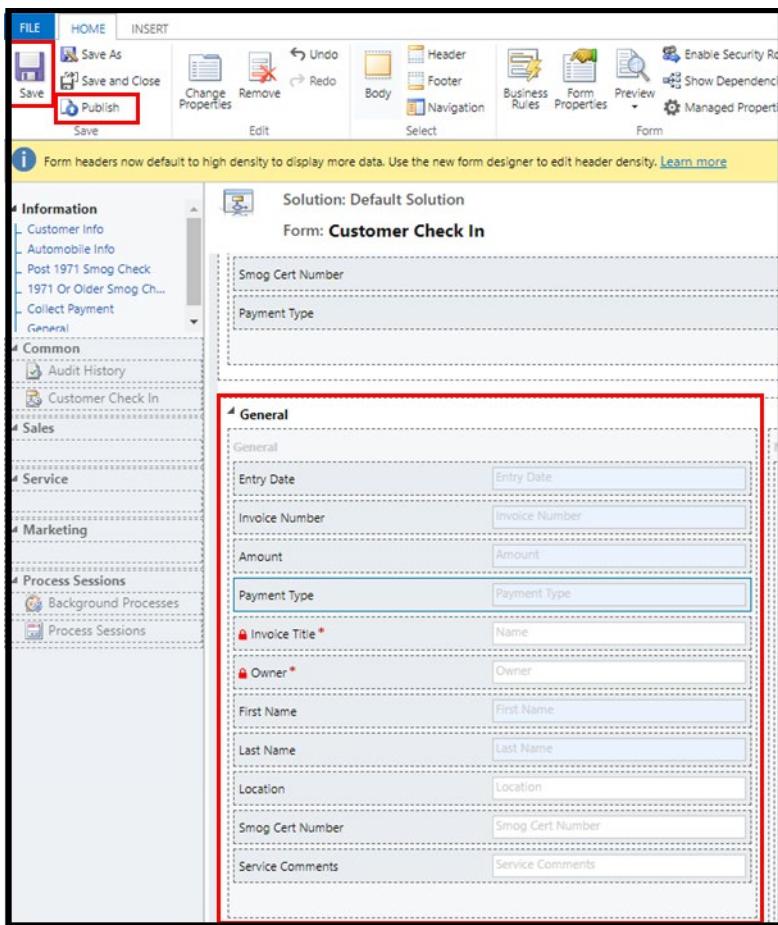
9. Double-click the **Entry Date** field that you added and make it read-only, and then select the **OK** button.



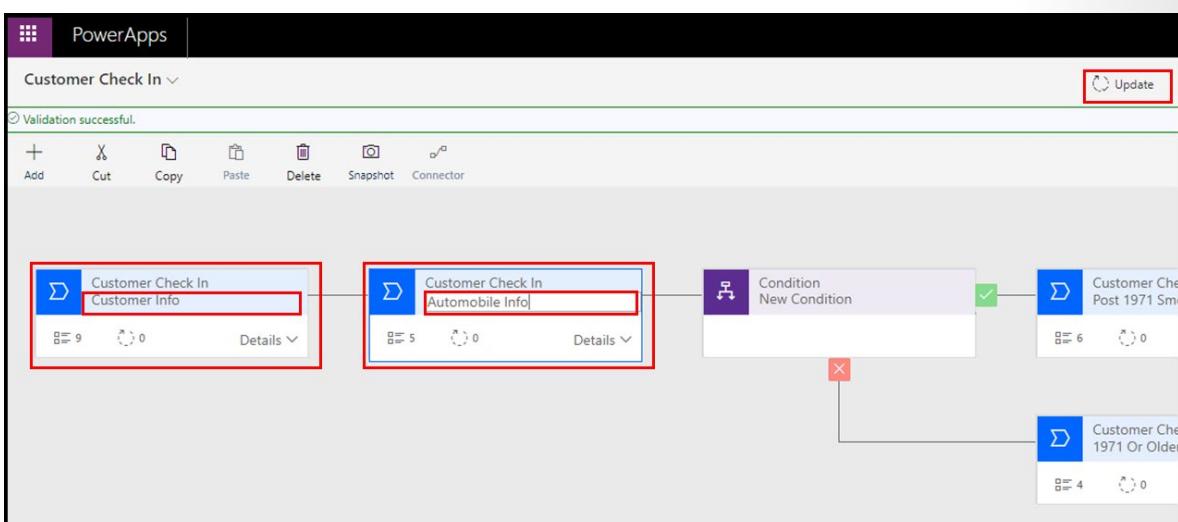
10. Do the same for the following fields to also make them read-only.

- First Name
- Last Name
- Smog Cert Number

11. Select the **Save** button in the ribbon and then select **Publish**.



12. Close the **Form Designer** tab and return to the business process flow designer. Double-click the first stage and rename it to **Customer Info**. Double-click the second stage and rename it to **Automobile Info**, as shown in the following figure. Select the **Update** button in the ribbon so all the changes are saved and ready to use.



13. Close the flow designer screen and refresh your browser. Run the **Customer Check In** flow to check that all the changes that you made are working properly. The business process flow should look like the following screenshot.

The screenshot shows the 'Customer Check In' form for 'San Remo Smog Check'. The 'General' tab is selected, displaying the following data:

Field	Value
Entry Date	1/17/2020
Invoice Number	1001
Invoice Title	* San Remo Smog Check
Owner	* <a href="#">Dave</a>
First Name	Alan
Last Name	Parsons
Location	Los Angeles
Smog Cert Number	344,990
Service Comments	---

A red arrow points from the 'Customer Info' section to the 'Automobile Info' section, which contains the following data:

Field	Value
Completed	<input checked="" type="checkbox"/> X
Note Text	✓ Automobile Make Chevy
Enter a note...	✓ Automobile Model Nova
	✓ Automobile Year 1,969
	✓ Automobile Milage 250,000
	✓ Automobile Comments Barn Find

You can create a new instance of the **Customer Check In** flow by selecting the **+ Add record** button in the ribbon.

The screenshot shows the Microsoft Power Apps portal interface. On the left is a navigation sidebar with options like Home, Learn, Apps, Create, Data, Entities (which is selected and highlighted in purple), Option Sets, Dataflows, and Export to data lake. The main area is titled 'Entities > Customer Check In'. At the top right are buttons for 'Add record', 'Edit record' (which is highlighted with a red box), 'Delete record', and a refresh icon. Below these are tabs for Fields, Relationships, Business rules, and Views. The main content area displays a table of records with columns for Name, Customer Check In, and Action. The records listed are Joe Green, William Sonoma, William Wilson, William Tell, Bernie Warren, William Brown (selected and highlighted with a purple checkmark), and Bill Bailey.

Name	Customer Check In	Action
Joe Green	0791c597-3237-ea1...	Aut
William Sonoma	30cc4656-3337-ea11...	Aut
William Wilson	63a169de-bb37-ea1...	Aut
William Tell	3225f62a-ca37-ea11...	Aut
Bernie Warren	8b8e3304-cd37-ea1...	Aut
William Brown	f2a657fe-a938-ea11...	Col
Bill Bailey	e2a2773b-aa38-ea1...	Col

Congratulations, you've created a working business process flow.

## summary

You have now learned how to improve the immersive business process flow that you created in an earlier module by using the following advanced techniques:

- Creating logical branching within a business process flow
- Using instant flows with an immersive business process flow
- Creating additional fields and enhancing the main form to improve the usefulness of a business process flow

If you want to learn more about this topic, select the following links:

- Read more about **Business process flow logical branching**<sup>54</sup>.
- Learn more about using **Instant flows**<sup>55</sup>.

<sup>54</sup> [https://docs.microsoft.com/previous-versions/dynamicscrm-2016/admins-customizers-dynamics-365/mt826751\(v=crm.8\)?redirectedfrom=MSDN/?azure-portal=true](https://docs.microsoft.com/previous-versions/dynamicscrm-2016/admins-customizers-dynamics-365/mt826751(v=crm.8)?redirectedfrom=MSDN/?azure-portal=true)

<sup>55</sup> <https://docs.microsoft.com/business-applications-release-notes/april19/microsoft-flow/instant-steps-business-process-flows/?azure-portal=true>

- Additional information about **Business process flows**<sup>56</sup>.

---

<sup>56</sup> <https://docs.microsoft.com/dynamics365/customerengagement/on-premises/customize/business-process-flows-overview/?azure-portal=true>

# Introduction to expressions in Power Automate

## Introduction to expressions

When you build a flow in Power Automate, it is easy to get started by adding a trigger and actions and then passing data between them with dynamic content. But, sometimes you need to do more. You want the ability to do a calculation or to transform data for your solution. To do these things, you can look to expressions.

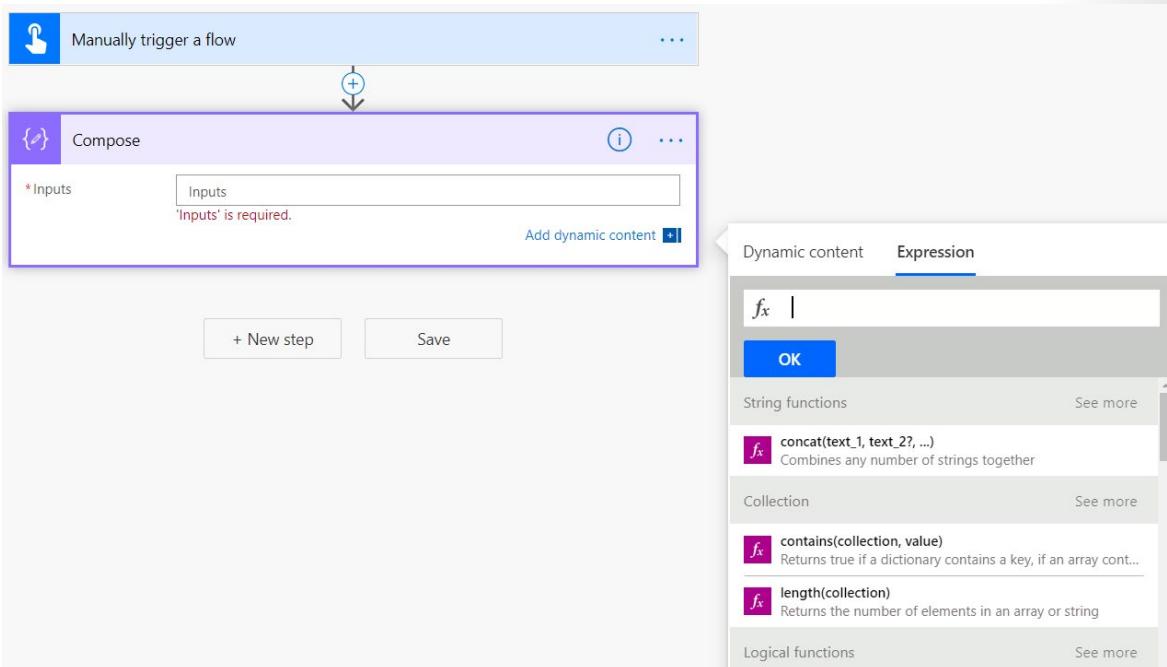
In Power Automate, expressions are a set of simple functions that enable you to return data. Put simpler, you use expressions to write a formula to get your data the way you want it. Expressions allow you to do things like convert a date to UTC, divide two numbers, create that perfect string by combining fields, and much more all by using various functions.

Flows in Power Automate run on top of Azure Logic Apps. The most important reason for you to understand that is that they both use all of the same functions. When you are searching the internet for solutions or reading documentation about flow functions, it is common to end up looking at Logic Apps documentation. For example, the **Reference Guide**<sup>57</sup> of functions for Power Automate is hosted on the Logic Apps side of the documentation.

Through the remainder of this module, you will learn about the different types of functions and the syntax so you can start to use expressions in your flows.

## Get started with expressions

To write an expression in flow, you select in a field to open the Dynamic content menu and then you select "Expression" as shown below.



<sup>57</sup> <https://docs.microsoft.com/azure/logic-apps/workflow-definition-language-functions-reference/?azure-portal=true>

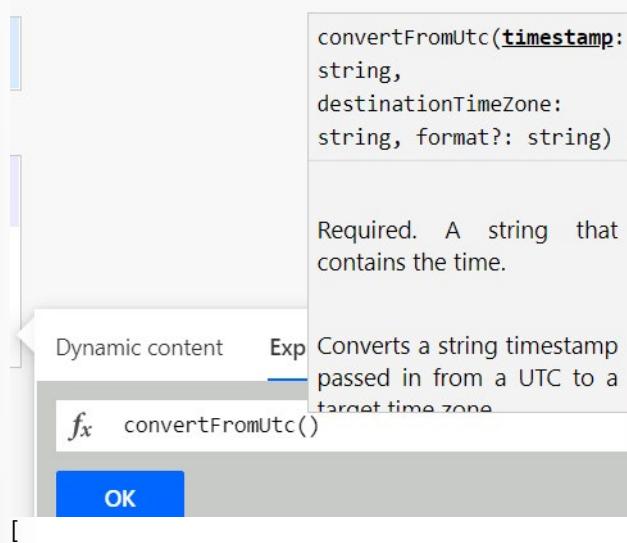
In the formula box, you will then write your expression by combining one or more functions. Before you learn about the different types of functions and their usage, there are some commonalities you should learn first. Function names are not case-sensitive.

[!TIP]

Notice in the screenshot that we used a Manually triggers a flow trigger and a *Compose* action. This allows you to have a simple setup for writing and testing your expressions without worrying about other inputs or actions. Use this setup to follow along in this app or anytime you want to try out something new.

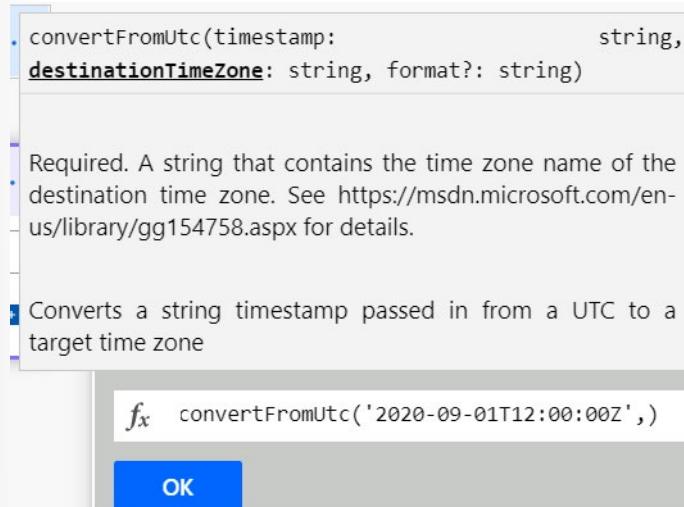
## Auto suggest, hints, and links in the formula bar

When you enter a function in the formula bar, you will see a pop-up with syntax suggestions.



Here you can see for the `convertFromUtc` function that it has two required inputs and one optional input. `Timestamp` is required and expects a string, `destinationTimeZone` is required and expects a string, and `format?` is optional and expects a string. Note the "?" at the end of `format`. That tells you it is an optional input.

After entering a Timestamp as a string typing a comma, then prompts for the destinationTimeZone:



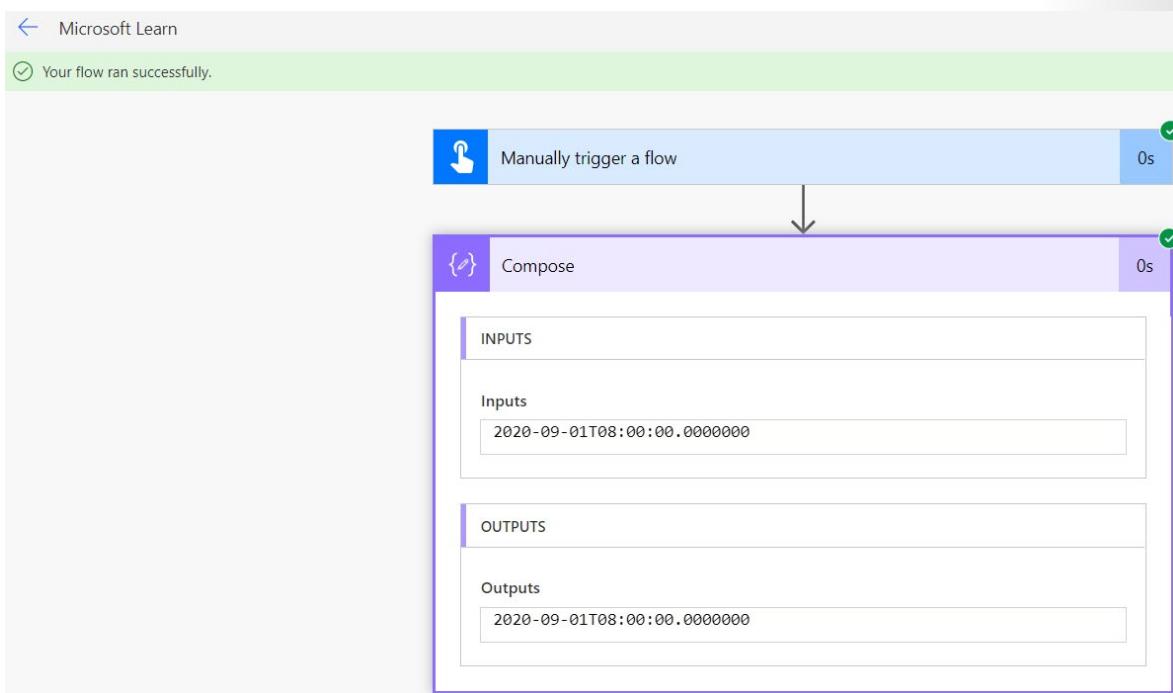
This can be daunting. What does it want for the time zone? If you look in the pop-up, you'll see that it provides a link to the **list of time zone values<sup>58</sup>**. This page provides you the string for the *destinationTimeZone*. With that information, we can now complete the expression.

```
convertFromUtc('2020-09-01T12:00:00Z', 'Eastern Standard Time')
```

With the complete formula, you can select OK to save your changes. Always be sure to select OK or UPDATE when editing an expression. If you select out of the inputs, you will lose your changes. Flow does not have autosave.

With your first expression complete, you can now select Test in the top right-hand corner. Then select Save & Test. select the Run flow button and then Done.

You should see the green bar that says "Your flow ran successfully." Expand the *Compose* action and you will see that the **OUTPUTS** is your date time converted to the new time zone.

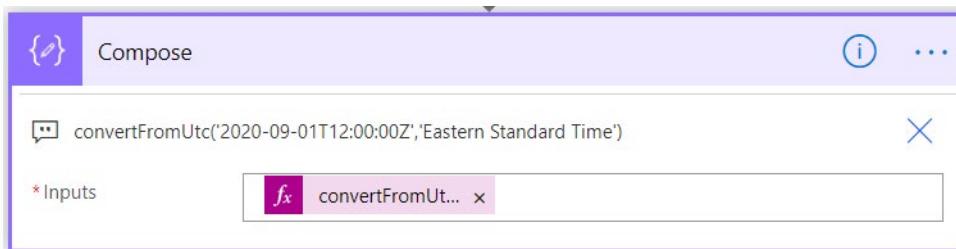


Use this same process to see the output of any test expressions you create.

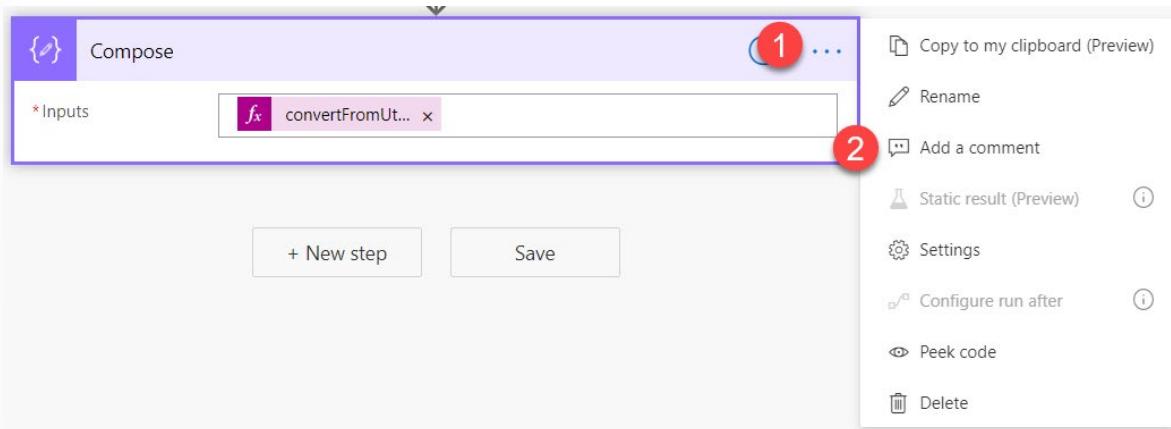
## Comments make things easier

When you look at the *convertToUtc* expression you wrote earlier, you might notice that you need to scroll to see the whole formula in the function bar. A common technique to make reviewing your expression easier is to use Comments and pasting the whole formula into the comment.

<sup>58</sup> <https://msdn.microsoft.com/library/gg154758.aspx/?azure-portal=true>



To add a comment, select the ellipse to show the menu. Then select "Add a comment".



Then you can enter any text you want. This could either be text or just a copy and paste of your expression as shown previously.

## Defining text

When you use text in a flow expression, you will use the ' (single quote) at the beginning and end of each string. For example, if you wanted to combine the string "Have a" and "good day!" in an expression, you would use the Concat function and your syntax would be:

```
Concat('Have a', ' good day!')
```



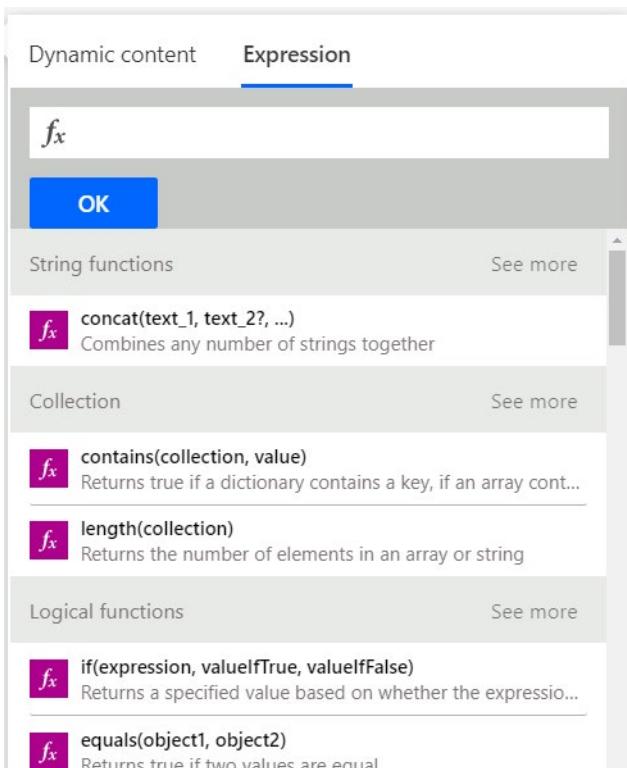
These little details will help you more effectively work with expressions. One of the great things about expressions is that they are consistent and as you learn about one function you can often apply those learnings to the next function, rapidly speeding up your learning.

## Types of functions

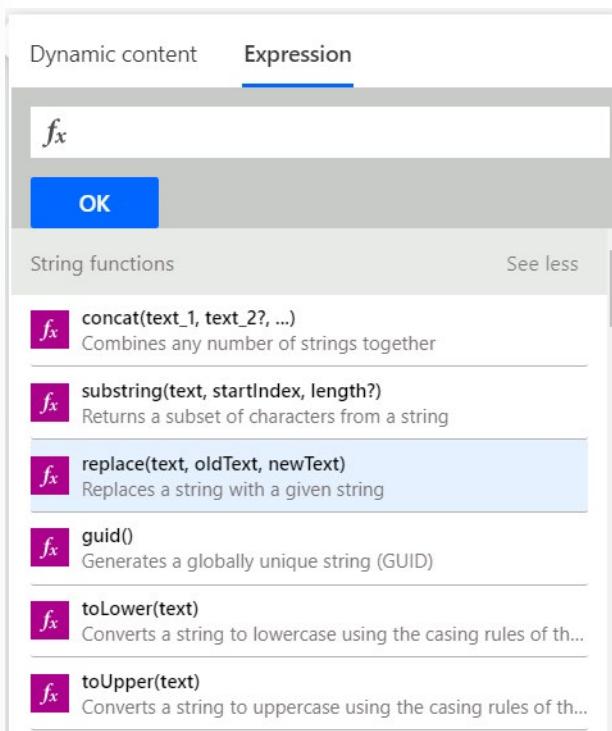
Functions are grouped into 10 different categories like math and logic. The categories are just a logical organization to making finding the function you need easier. Below you will get an overview of each category and some examples.

Keep in mind as you go through the various examples, we use static text and values. This is to allow you to test and recreate the examples as easily as possible. In your flows, you may substitute dynamic content in place of this static data. Just ensure that your dynamic data is the correct format for the function.

In the screenshot below, you will notice to the right of each category header (like String functions or Collection) the words "See more." By selecting "See more," the complete list of functions for that category will be shown.



Select **See more** beside String functions.



## String functions

String (text) functions are used to modify strings, find characters in a string, format strings, and more. Text manipulation is a core skill often used when trying to better format or modify data you received from somewhere else.

An example of a string function is the *formatNumber* function. This function can convert a number into a string in a given format. A common request is to make a string look like currency. To change the number 12.5 into \$12.50, use the following formula:

```
formatNumber(12.5, 'C')
```

The C represents the Currency numeric format string. Don't worry, a list of the other options are available at **Standard numeric format strings**<sup>59</sup>. You also might be asking what if you wanted to show the number as currency but with the Yen symbol? There is an optional parameter where you can pass the locale.

```
formatNumber(12.5, 'C', 'ja-JP')
```

This formula will return ¥13.

## Collection functions

These functions are used for arrays and strings. They may be used to check if an array is empty, to grab the first, or last item, or even for join, union, and intersection operations.

---

<sup>59</sup> <https://docs.microsoft.com/dotnet/standard/base-types/standard-numeric-format-strings/?azure-portal=true>

An example of a helpful Collection function is *length*. You can use *length* to return the number of items in a string or array. The following example would be used to count the number of characters in the string "I love Power Automate."

```
length('I love Power Automate.')
```

The output would be 22. You can use this type of function for validation or in conjunction with the String functions to manipulate strings.

## Logical functions

These functions are used to work with conditions, to compare values, and to do other logic-based evaluations. These are often thought of as If statements where you want to compare if a number is greater than another number. Flow supports all of the logical comparisons you would expect.

In the example below, an expression will compare if 12 is greater than 10 and then output the appropriate string. This will also be your first expression that uses more than one function. We will combine *if* and *greater* logical functions.

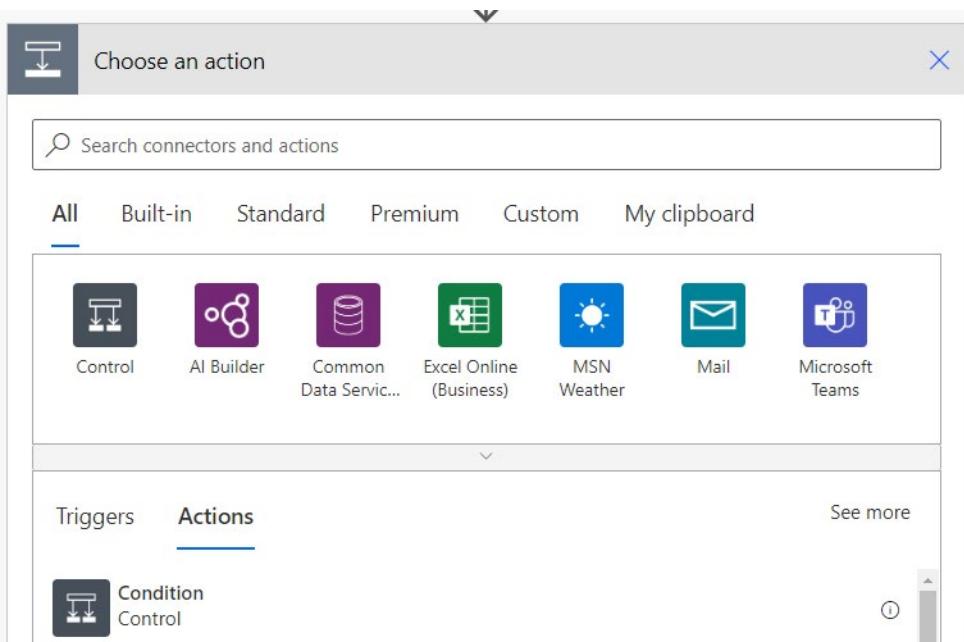
```
If(greater(12,10), 'Yes', 'No')
```

The result of this expression would be the string "Yes". To understand the expression, work from the inside out. Greater(12,10) returns true or false depending on if 12 is greater than 10. Since it is, the value returned is true.

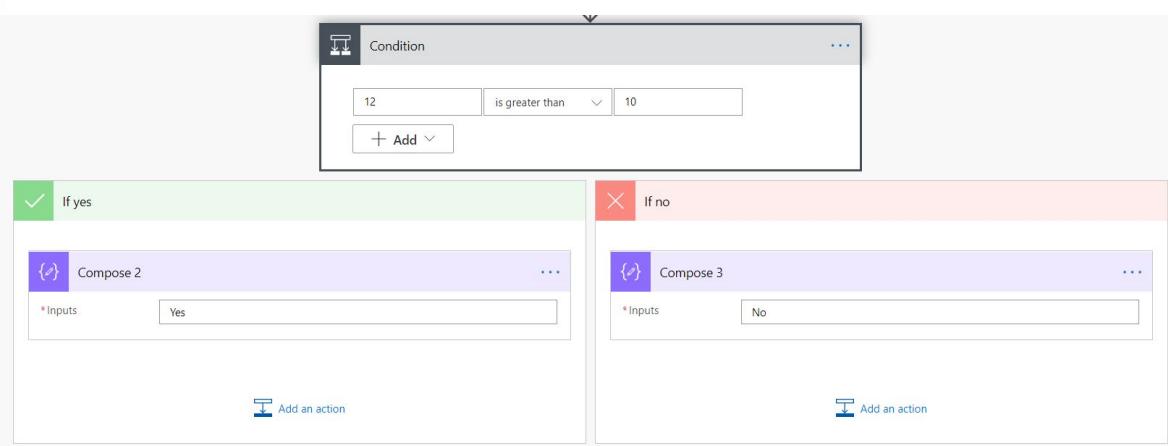
Now that you know the answer is true, you can see that the *If* function returns the data after the first comma. In this case that is the string "Yes". Had it been false then the string "No" would have been returned.

### [!IMPORTANT]

While you can write logical expressions as shown above, there is also an action called *Condition* that lets you write *If* statements without an expression.



Here is the same condition written using the action.



You will find over time that you use a combination of both methods depending on your requirements.

## Conversion functions

These functions are used to change the type of your data. This can be simple things like converting a text number into an integer, or more complex functions like changing the encoding of a file from base64 to binary. Knowing that these functions are available will help you overcome problems you encounter when getting your data shaped correctly.

A common scenario is the need to use *int* or *float* to change a text number into an actual number. This is common when importing data into your flow from a data source. The number 12 or 12.4 may be stored as text. In order to use that number in a logical function or write it to a location that expects a number, you will need to convert it. The following example changes the string "12" into the integer 12.

```
Int('12')
```

That will output the integer 12. Had it been the string "12.4" then you would need to convert it to a float because of the decimal digits. In that case you would use

```
Float('12.4')
```

Now you could use text number to do the previous example.

```
If(greater(Int('12'), Float('12.4')), 'Yes', 'No')
```

This would output the string "No" because 12 is not greater than 12.4. Combining functions like this is common and is a useful pattern to learn.

## Math functions

Math functions do exactly as you would expect. They allow you to add, subtract, multiply, and perform other similar functions. In addition, Math functions allow you to find the smallest and largest numbers of several numbers or get a random number, amongst other things.

One thing to keep in mind is that there is a different function for adding numbers (*add*) and for subtracting numbers (*sub*). Many formula languages add negative numbers to produce subtraction, but flow does not. To add two numbers together, you use the following:

```
Add(12, 13)
```

This would return 25. If you wanted to add three numbers, then you would need to add a second function like so:

```
add(add(12,13),15)
```

This would return 40. As you have seen before, you will often find yourself nesting functions to get your desired results.

## Date and time functions

These functions are used to return the current date and time, change time zones, find specific info about a date and time, and do other date/time manipulations. If you have date and time values in your data, you'll need these functions.

One important thing to remember as you explore date and time functions in flow is that they are often based on UTC. Most data sources pass data back and forth with flow in UTC format. Also, if you use the function `UTCNow()` that will return the current time in UTC format. If you want to use that to compare to user data that is currently in the Eastern Time Zone, then you will need to use the following formula to convert it:

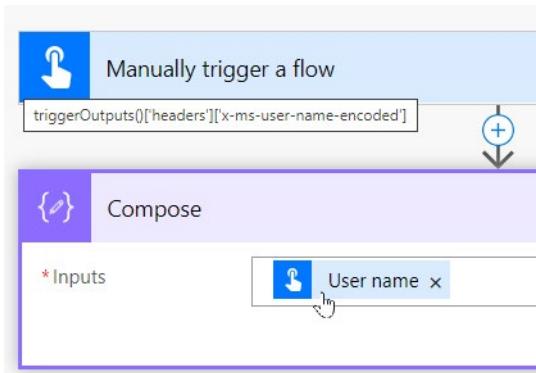
```
convertFromUtc(utcNow(), 'Eastern Standard Time', 'dd-MM-yyyy hh:mm tt' )
```

This will output 01-09-2020 05:39 PM. For a complete list of the date time format options, review **Custom date and time format strings**<sup>60</sup>.

## Referencing functions

The referencing functions are used to work with the outputs of your actions and triggers. The nice thing is that the majority of the time, flow will write these functions for you. When you add dynamic content to your flow, you are using referencing functions without knowing it. If you add dynamic content and then hover over of that content, you can see this in action.

In the screenshot below, by adding the *User name* dynamic content from my trigger to the Inputs for Compose you can see this in action.



By hovering over User name with the mouse pointer, you can see

<sup>60</sup> <https://docs.microsoft.com/dotnet/standard/base-types/custom-date-and-time-format-strings/?azure-portal=true>

```
triggerOutputs() ['headers'] ['x-ms-user-name-encoded']
```

Flow created the expression using the triggerOutputs for you. It is pulling from the Headers property the x-ms-user-name-encoded property. Most of the time in flow you will reference these properties via dynamic content. But it is possible to write your own expressions to recreate this if necessary. Each trigger and action will have different formats for how you retrieve their data.

Explore these functions by adding different triggers, actions, data sources, and apply-to-each loops in your flow. Then, use their properties as dynamic data to see more examples. The good news is writing these types of expressions is not common.

## Workflow functions

The workflow functions are used to retrieve information about your flow and are closely related to the referencing functions. One of the functions is called workflow. You can use it as shown below.

```
Workflow().run.id
```

This will return the ID of the current flow run. You could use this for error reporting or logging if needed. These functions are not commonly used.

## URI parsing functions

These functions are used to dissect a URI that is passed in as a string. You may use these functions to find the host, path, query string or other portions of the URI. The following example shows you how to use uriQuery to get the query string portion of the given URI.

```
uriQuery('https://flow.microsoft.com/fakeurl?Test=Yes')
```

This would return the string "?Test=Yes" which you could then parse with string functions to get the value passed in from the URI.

## Manipulation functions

Manipulation functions are used to work with specific objects in your flow. You can do things such as find the first non-blank value, work with properties, or find xpath matches. These functions are used typically in JSON or XML nodes evaluations.

One function you may find handy here is *coalesce*. This function will allow you to find the first non-null value from a specified set of values. You use the function as shown in the following formula:

```
Coalesce(null, 'Power Automate', 'Power Apps')
```

This formula would return the string "Power Automate". Handy when you are passing in multiple values and want to find the first one that is not null.

## Write complex expressions

Complex expressions are when you combine more than one function to get your desired result. In the Math functions and Date Time functions sections, you have already seen examples of this. You saw how in order to add three numbers, you need to combine two add functions like:

```
add(add(12,13),15)
```

Which resulted in the output of 40.

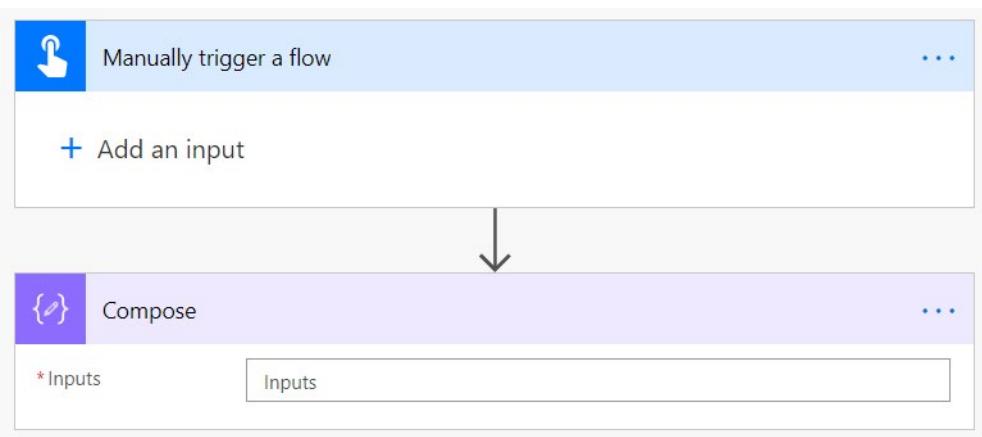
Then you saw how you use the utcNow function to get the current date and time and the use convertFromUtc to change it to the Eastern Standard time zone as shown below:

```
convertFromUtc(utcNow(), 'Eastern Standard Time', 'dd-MM-yyyy hh:mm tt' )
```

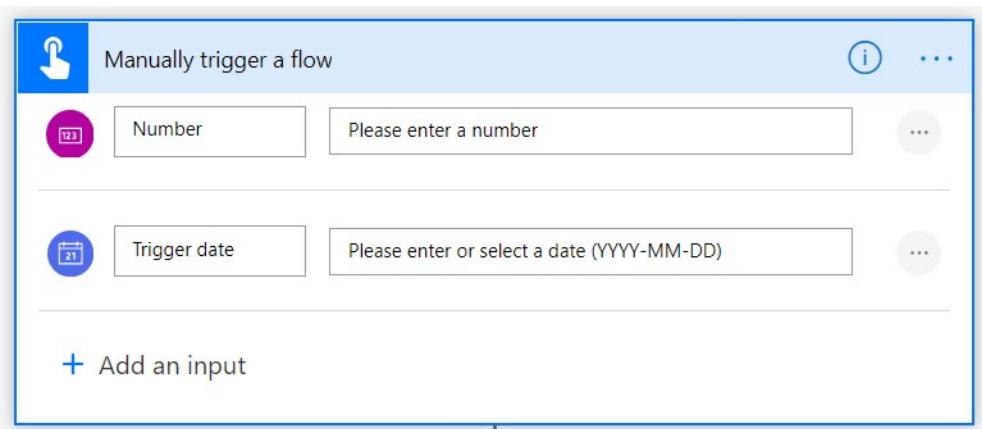
Which resulted in the output of 01-09-2020 05:39 PM.

When you think of complex expressions, that is it. More than one function in an expression where you use the output of one function as an input of another. There is no special syntax, operators, or considerations.

For a final example of a complex expression, take the scenario of having a couple of inputs as part of your "Manually trigger a flow" trigger and then using that input in a formula to compute a new time. You can use the example flow that you have used to test your expressions or to build a new flow that looks like this one below.



Now select on "Add an input" under "Manually trigger a flow" and add a Number. Then select "Add an input" again and add a Date. Your trigger should now look like this:



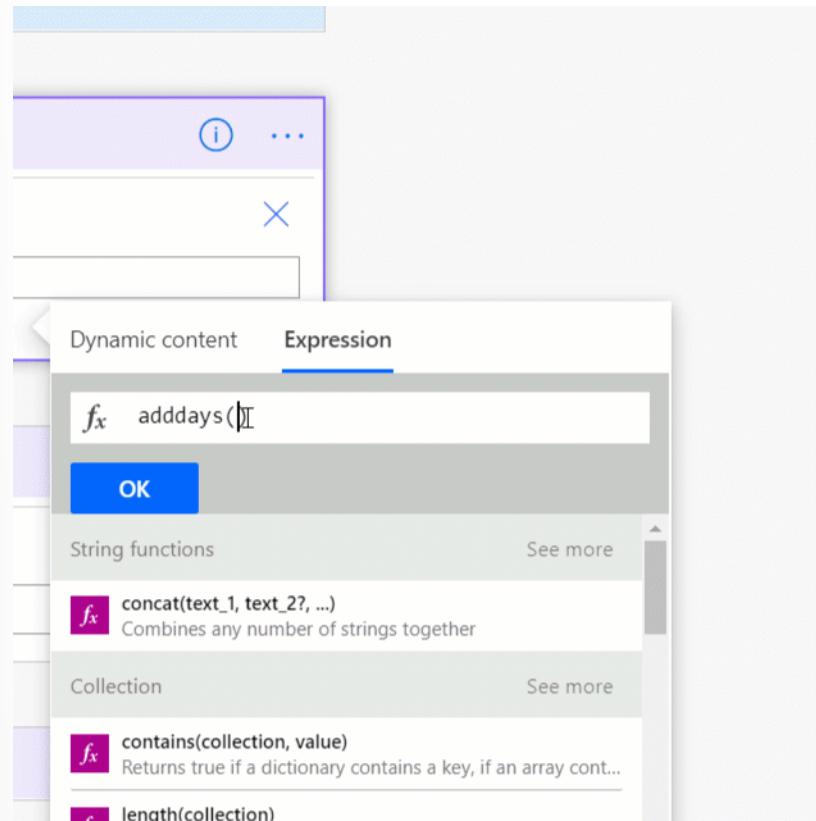
Now, in the Compose step you will add an expression to add the number of days from the trigger to the date.

```
addDays(triggerBody() ['date'], triggerBody() ['number'])
```

This is leveraging the Date Time function *addDays* and the Referencing function *triggerBody*.

[!NOTE]

If you find yourself asking “How do I know what goes in for trigger body?” then you are thinking correctly. The secret is you can combine dynamic content into your expressions. To do so, start your expression by typing *addDays()* and then with your cursor between the *()*, select Dynamic content. Then you can choose your fields as shown below. This is a great way to reference that content while letting flow do the hard work of writing the formula.

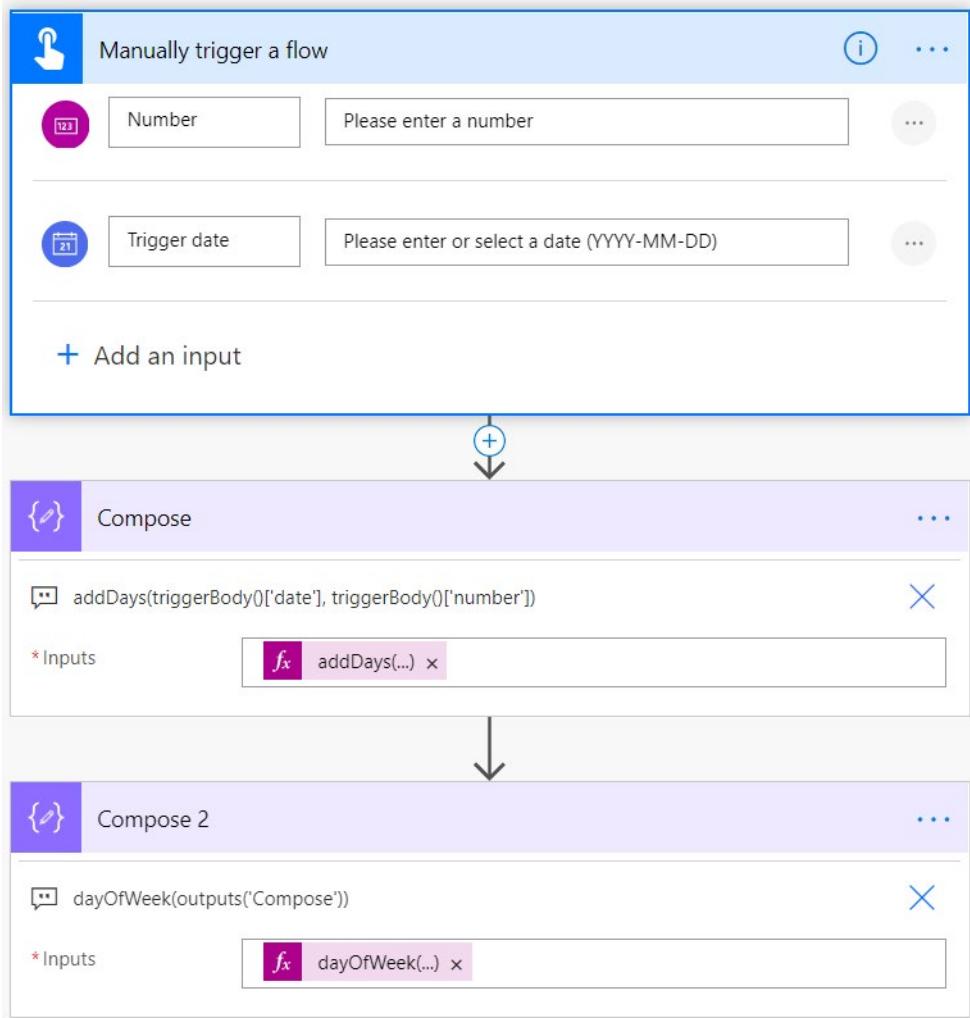


If you test your flow and enter the date 2020-09-01 and the number 2, your output will be 2020-09-03T00:00:00.0000000, which is UTC format for September 3rd, 2020.

Now you can find out what day of the week that is by using another Compose action with the following expression:

```
dayOfWeek(outputs('Compose'))
```

This returns the value of 4, which represents Thursday as it counts up from Sunday as 0. Here is a screen-shot of the current flow to validate what you have built. Note the expressions have been placed in a comment to make them easier to read.



Now add another Compose step to check to see if the date they selected is a Thursday. Do this with the following expression in Compose 3.

```
if(equals(outputs('Compose_2'), 4), 'You chose a Thursday', 'You did not choose a Thursday')
```

For September 3, 2020 this will return the string "You chose a Thursday".

This example is a typical build pattern for a complex expression, building piece by piece in separate steps. Now that you have all of the functions necessary worked out, add another Compose step. In Compose 4, write one large expression that does everything in one step. The expression will look like:

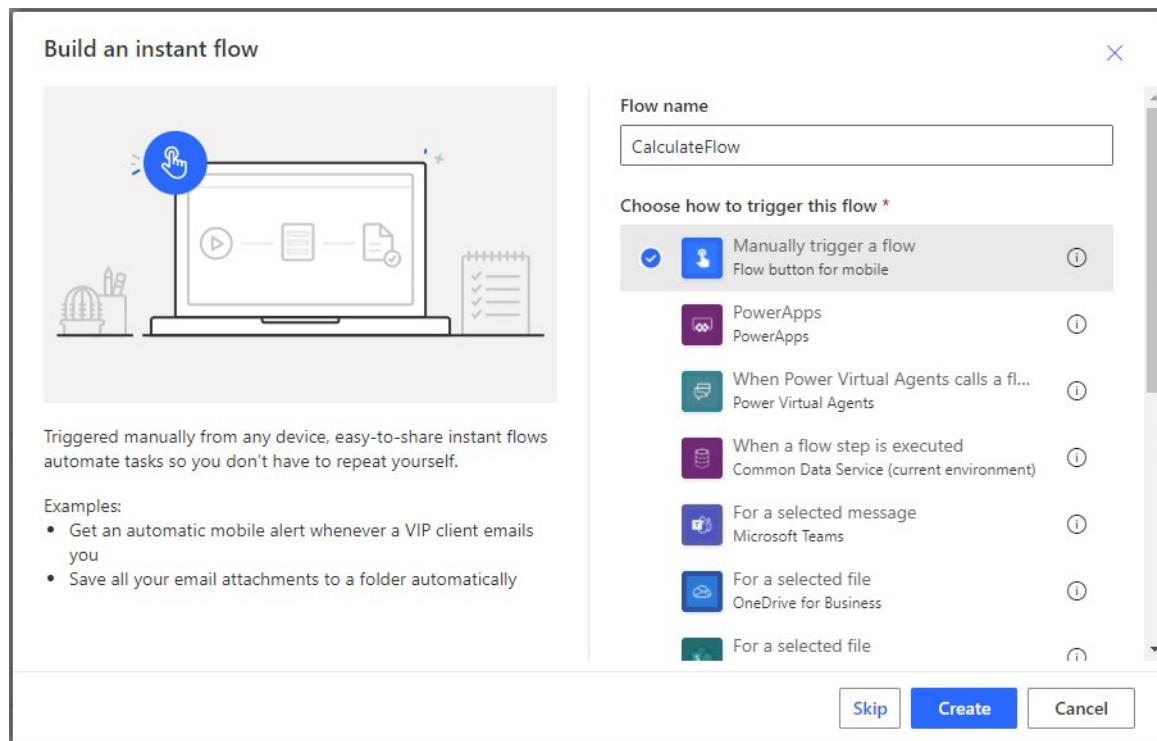
```
if(equals(dayOfWeek(addDays(triggerBody()['date'], triggerBody()['number'])), 4), 'You chose a Thursday', 'You did not choose a Thursday')
```

The output for September 3, 2020 will be the string "You chose a Thursday". Congratulations. You have written a complex expression by doing small steps and then putting it all together in the end.

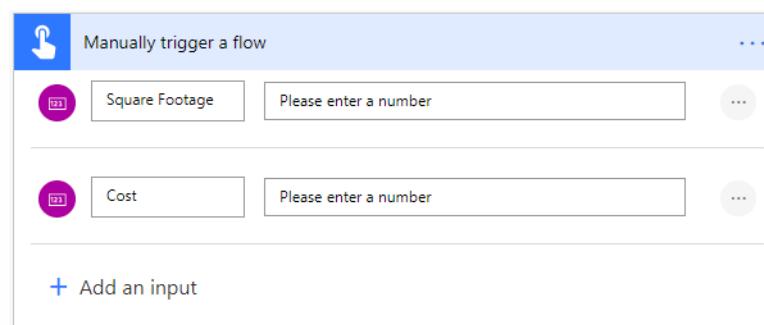
## Exercise - Creating a manual flow and using expressions

Let's say you need to find out how much it would cost to carpet a room based on its' square footage. In this exercise, you will create a manual flow that will use your input to do those calculations.

1. Sign into **Power Automate**<sup>61</sup>.
2. Create a new Instant flow from blank.
3. Name it *CalculateFlow* and select Manually trigger a flow.



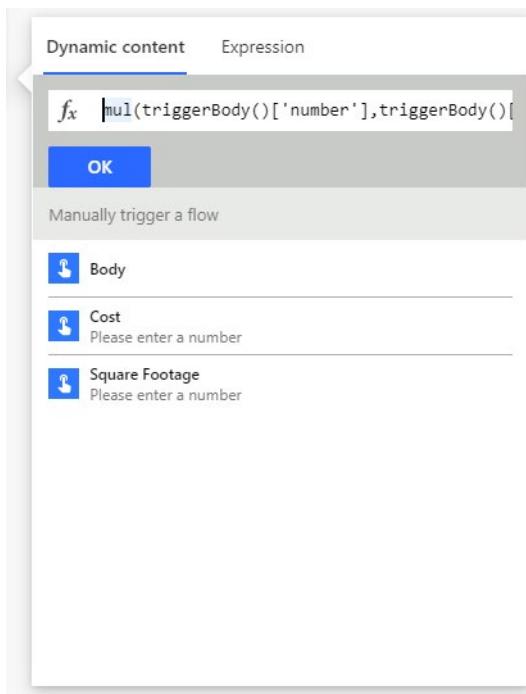
4. Select the Manually trigger a flow step and select Add an input.
5. Choose Number and name it *Square Footage*.
6. Select Add an input and choose Number again and name it *Cost*.



7. Select New Step and search for and select the Compose action.

<sup>61</sup> <https://flow.microsoft.com/?azure-portal=true>

8. Select the Inputs box and the Dynamic content window will appear.
9. Select the Expression tab and type in **mul(**. Flow will know that you are using the multiply expression and automatically add a **)** at the end for you.
10. With your cursor still between the two parentheses in the expression field, select the Dynamic content tab.
11. Select **Square Footage** in the dynamic content below.
12. Next add a **,** and choose **Cost** in the dynamic content below.
13. The complete expression should be:  
`mul(triggerBody()['number'],triggerBody()['number_1'])`
14. Select OK to add the expression into the Compose step.



15. You will know your expression is correct if it is added into the Compose step and looks like:



This compose step is now doing the math of calculating the square footage of the area multiplied by the cost per square foot based upon inputs you will provide. However, we still need to convert the final result to currency to get an accurate answer.

16. Select New Step and search for and select the Compose action again.
17. Select the Inputs box and again the Dynamic content window will appear.
18. Select the Expressions tab and type in **FormatNumber(**.

19. With your cursor in place select the Dynamic content tab and choose **Outputs** from the preview Compose step.

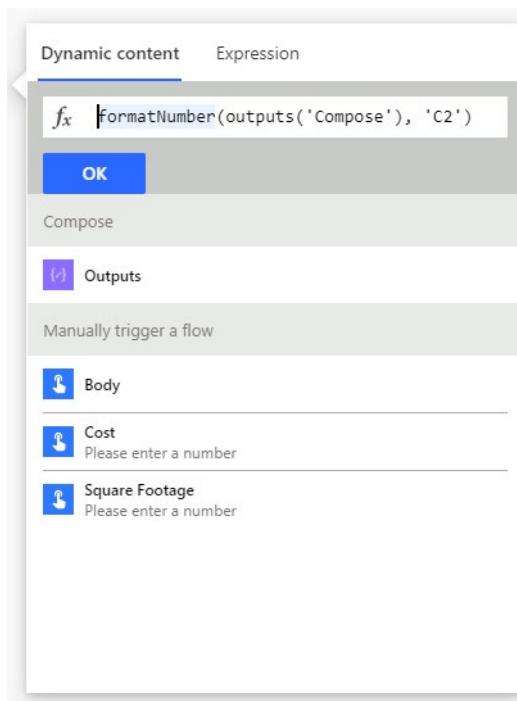
20. Next finish the expression with , 'C2'.

The C formats the number as currency, with the 2 representing how many decimal places. Refer to **Standard numeric format strings**<sup>62</sup> for more number formats.

21. The complete expression should be:

```
formatNumber(outputs('Compose'), 'C2')
```

22. Select OK to add this expression in the Compose 2 step.

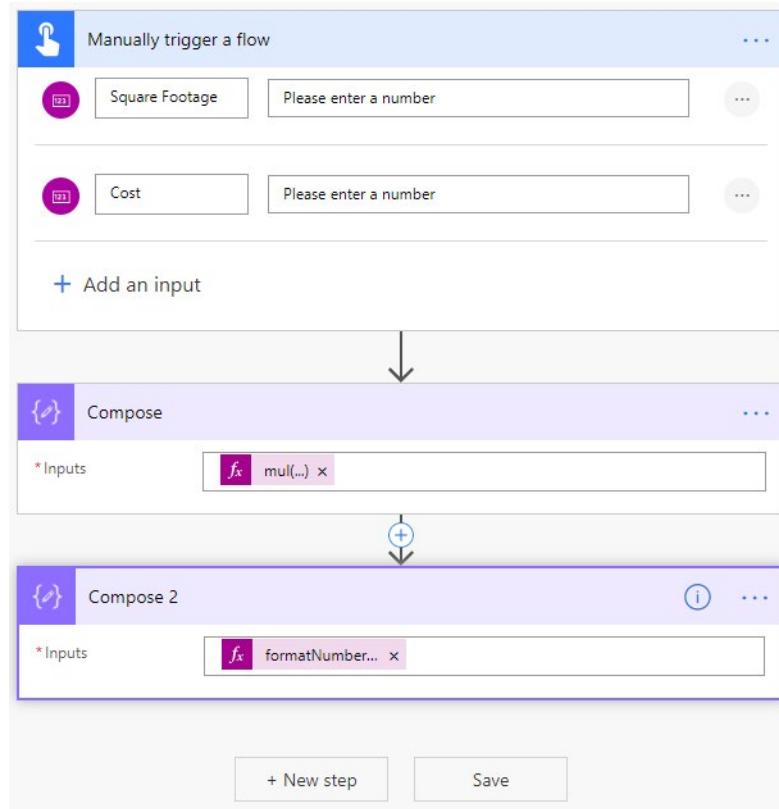


Now after our flow is triggered and multiplies the two inputs it will convert the result into currency. This will give you a quick way to get the cost associated with carpeting a room.

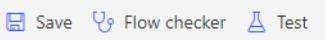
23. Your complete flow will look like this:

---

<sup>62</sup> <https://docs.microsoft.com/dotnet/standard/base-types/standard-numeric-format-strings/?azure-portal=true>



24. In the top-right corner, select the Save button, then select Test.



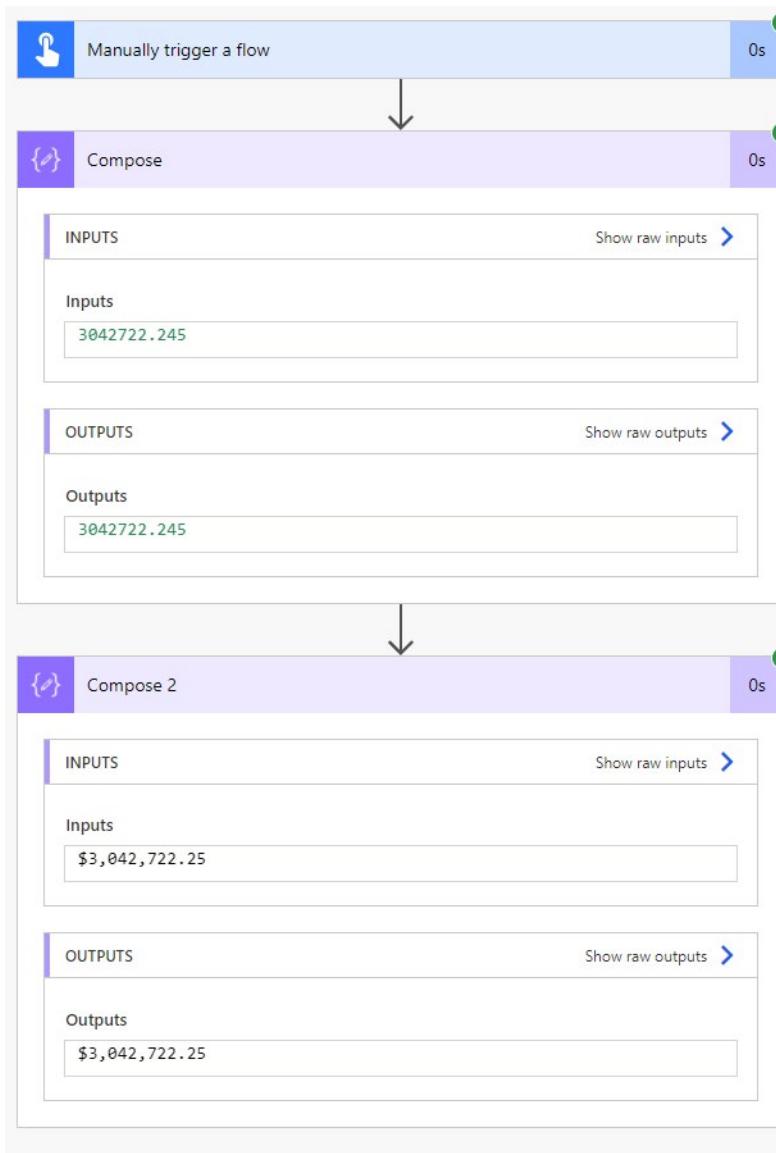
25. Choose I'll perform the trigger action and then select Test.

26. Enter the two number inputs, *Square Footage* and *Cost*, and then select Run flow at the bottom.

27. The page will reload and you will see green check marks next to each step of your flow.

28. Selecting each step will expand the details showing you the inputs and outputs of each step.

29. Selecting Compose will show the multiplied value of the two numbers you entered, while selecting Compose 2 will show the currency format of that multiplied value.



Let's recap what we've done.

**Manually trigger a flow** - Allows us to press a button to trigger a Flow and provide inputs. In this case our two inputs are *Square Footage* and *Cost*, which we are using to find out the total price to carpet a room.

**Compose** - We use this to write expressions using the data from elsewhere in the flow. In this case, we first multiplied the two inputs from the trigger. This gave us the correct number, but we still needed to convert it to currency. We then used another compose action to format the multiplied result into currency. Finally giving us the easiest answer of how much it would cost to carpet a room.

## summary

When working with flow, you will use one or more functions to create expressions to get the most out of your data. These functions can be used to retrieve data, change data, evaluate data and more. As your flow skills grow, your usage of expressions will grow. Becoming familiar with the various functions available and how to combine them will let you unlock the full power of flow.

## Module 5 Introduction to developing with Power Platform

### Introduction to Power Platform developer resources

### Introduction to the Power Platform for Developers

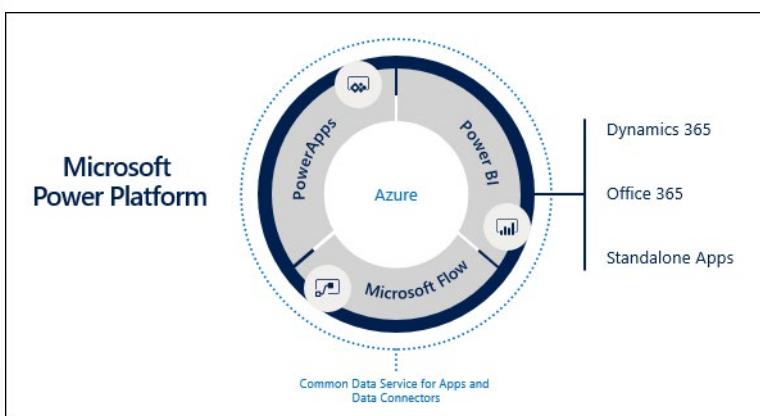
### Microsoft Power Platform overview

The Microsoft Power Platform encompasses the entire suite of Microsoft's business application products. It includes three main components: Power BI, Power Automate, and Power Apps. Each component is designed to interact with the others to achieve business objectives, be it analytics, process automation, or building data-driven productivity applications. Developers are able to take advantage of the Power Platform as a low-code framework to build robust applications and extend upon each with custom code whenever the existing framework doesn't support the desired functionality. This approach to low-code development is a transformative way to quickly achieve application development objectives within an organization.

While future units of this module will mainly be focused on Power Apps, it is important for you to know the capabilities of Power BI and Power Automate in the context of our ecosystem because they play a relevant role to building model-driven Power Apps.

Take a look at the **Introduction to the Power Platform - fundamentals<sup>1</sup>** module for a more in-depth overview of what the Power Platform has to offer.

<sup>1</sup> <https://docs.microsoft.com/learn/modules/power-plat-app-basic-fundamentals/?azure-portal=true>



## Power Apps applications

Power Apps are applications that are consumed by users through their desktop or mobile devices. These applications come in two forms: canvas and model-driven. Canvas applications can be embedded into SharePoint, Teams, Power BI, and Dynamics 365 applications. Model-driven applications are standalone, data-driven applications that are built on top of Common Data Service. To learn about creating canvas apps, see the

**Create a canvas app in Power Apps<sup>2</sup>**

learning path. For a high-level introduction on creating model-driven apps, see the **Create a model-driven application in Power Apps<sup>3</sup>** learning path.

## Power BI

Power BI is Microsoft's business analytics solution that provides interactive data visualization BI tools to help users visualize and share data and insights across their organization. For an introduction on how to create Power BI reports, see the **Create and use analytics reports with Power BI<sup>4</sup>** learning path.

## Power Automate

Power Automate is used to orchestrate activities across various services that use connectors. They can be triggered to run when specific events occur, such as when a record is created or scheduled to run at a specific time. To learn more about Power Automate, see the **Automate a business process using Power Automate<sup>5</sup>** learning path.

---

<sup>2</sup> <https://docs.microsoft.com/learn/patterns/create-powerapps/?azure-portal=true>

<sup>3</sup> <https://docs.microsoft.com/learn/patterns/create-app-models-business-processes/?azure-portal=true>

<sup>4</sup> <https://docs.microsoft.com/learn/patterns/create-use-analytics-reports-power-bi/?azure-portal=true>

<sup>5</sup> <https://docs.microsoft.com/learn/patterns/automate-process-using-flow/?azure-portal=true>

## Common Data Service

Common Data Service is a composite set of capabilities that can be used to build robust business applications. These capabilities include a cloud-scale entity framework and data store, business rules, workflows, dynamically-calculated fields, and many other capabilities. Common Data Service drives model-driven Power Apps, the primary focus in this learning path. For a more in-depth look at Common Data Service, see the [Get started with Common Data Service<sup>6</sup>](#) module.

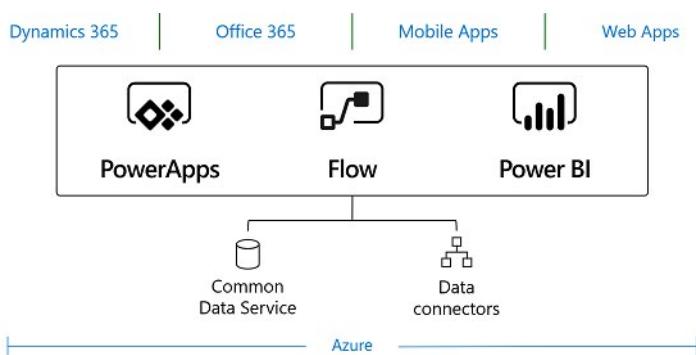
## Common Data Model

The Common Data Model is an open-sourced standard definition of entities that represent commonly used concepts and activities. When built in conjunction with a Common Data Service application, a core set of entities are provided upon which application builders can add their own custom entities to support specific business scenarios. To learn more about how to use the Common Data Model, see [The Common Data Model<sup>7</sup>](#) documentation in Microsoft Docs.

# Overview of Common Data Service and the Common Data Model

## Common Data Service overview

Common Data Service is the newest generation of a longstanding application platform that is built by Microsoft and based on the proven Dynamics 365 Customer Engagement apps platform. The extensibility model is robust and can be used to build a wealth of low-code application solutions.



## Benefits of Common Data Service

Common Data Service comes with a unified set of features that enable you to create business-focused definitions of your organization's data and business processes within a variety of applications. Because data and

<sup>6</sup> <https://docs.microsoft.com/learn/modules/get-started-with-powerapps-common-data-service/?azure-portal=true>

<sup>7</sup> <https://docs.microsoft.com/common-data-model/schema/core/applicationCommon/overview/?azure-portal=true>

metadata are both stored in the cloud, applications are simple to manage and administer. Additionally, a built-in security model allows you to control access to entities and functionality for different sets of users within your organization.

Microsoft has already built several first-party applications on Common Data Service, including Dynamics 365 for Sales, Service, and Talent. By building all apps on top of the same Common Data Service platform, you can interact with data that is constructed by other applications within Common Data Service.

## Model-driven apps vs. canvas apps

In this unit, interactions with Common Data Service generally refer to model-driven Power Apps. While canvas apps provide a robust model for building custom Power Apps, they currently don't interact with Common Data Service in the ways that are described in this unit.

## When to write code

As a PowerApp developer, you need to understand the gaps that exist between what can be accomplished through configuration versus code. Sometimes, existing features might not provide the functionality needed to meet a requirement, and Common Data Service provides a variety of points where developers can extend the common functionality by using code.

To identify gaps where features don't already exist, it's crucial that all Power Apps developers familiarize themselves with the capabilities of Common Data Service. If you aren't familiar with building model-driven applications that use Common Data Service, we recommend that you complete the [Create a model-driven application in Power Apps<sup>8</sup>](#) learning path.

## Types of extensibility

Typically, any code-customization activities are referred to as "extending" a model-driven application. Therefore, if you notice terms like extensibility, it generally means code in terms of model-driven PowerApp development. Within the extensibility model, activities are grouped into two broader buckets: extending the user experience and extending the Common Data Service platform.

## Extending the user experience of a model-driven PowerApp

To extend the user experience of a model-driven PowerApp, you can use web resources. Model-driven Power Apps expose a JavaScript client API that allows you to interact with application pages. Because TypeScript, which compiles down to JavaScript, is a recommended tooling to use in many scripting scenarios, both languages are generalized into the term of "client script."

---

<sup>8</sup> <https://docs.microsoft.com/learn/patterns/create-app-models-business-processes/?azure-portal=true>

With client script, you can configure form event handlers in the same way that you would an HTML form element. You can also call script functions from within a command bar (ribbon) and can consume client script web resources from within HTML web resources.

HTML web resources are used to display custom presentation logic within a form when more advanced requirements exist that can't be achieved by using the standard out-of-the-box forms.

## Extending the Common Data Service platform

Extending the Common Data Service platform is required whenever some level of automation needs to occur that existing features do not exist to support the current activities. These types of extensibility points run as server-side code through an asset called a plug-in. Power Automate and other options enable asynchronous automation and logic, but they haven't been able to do everything that plug-ins currently do in terms of synchronous rules. When requirements for synchronous operations exist within a model-driven application, plug-ins are still required.

Plug-in development is an essential skill and one of the most common tasks that is required of a PowerApp developer.

## Common Data Model overview

In support of Microsoft's Open Data Initiative, we now have a Common Data Model comprised of an extensible collection of schemas representing common business concepts. By using a common schema across all business applications such as Contacts, Leads, Accounts, or Products, data interoperability is greatly simplified.

The following infographic effectively represents the provided current state of the base Common Data Model schemas. This is an evolving ecosystem, and numerous others will surface as time progresses.

**INTEGRATE & DISAMBIGUATE DATA WITH THE COMMON DATA MODEL**

An **open-sourced** definition of modular and extensible **business entities** with **semantic metadata** that **simplify the challenges of application development and data integration**.

CDM unifies data in a well-known schema with semantic consistency.

App developers and backend integrators can develop **independently**.

Enables quick application deployment and development, out-of-the-box intelligence, and much more.

There's a growing collection of solutions that store data in CDM form:

- Power BI
- Dynamics 365
- PowerApps
- Azure

### CDM SCHEMA

CORE	CRM	SALES	SERVICE	SOLUTIONS
Account	Account	Competitor	Case	MARKETING
Activity	Appointment	Discount	Contract	Account
Contact	Campaign	Invoice	Entitlement	Contact
Currency	Contact	Opportunity	Resource	Event
Email	Lead	Order	Service	Marketing Email
Goal	Marketing List	Order Product	Scheduling Group	Marketing Page
Letter	Phone Call	Quote	Task	...
Note	Social Activity	...	...	...
Owner	...	...	...	...
Organization	...	...	...	...
Position	...	...	...	...
Task	...	...	...	...
...	...	...	...	...

Working with subject matter & industry experts, CDM is expanding to include additional business entities and concepts

**WEB**

- Link Clicks
- Web Interactions
- Web Page
- ...

**TALENT**

- Department
- Job Positions
- Worker
- ...

**HEALTHCARE**

- Patient
- Practitioner
- Device
- Care Plan
- Risk Assessment
- Medication
- Procedure
- Encounter
- Episode of Care
- ...

**Account**

Description: Business that represents a customer or potential customer.

Attributes:

- accountNumber
- accountRatingCode
- createdOn
- creditLimit
- openDeals
- openRevenue
- territoryid
- hotelGroup
- ...

For more details go to [aka.ms/cdmposter](http://aka.ms/cdmposter)

## Consuming the Common Data Model

The Common Data Model is maintained in [GitHub<sup>9</sup>](#). Schemas are maintained as json files. An entire entity reference can be found on [Microsoft Docs<sup>10</sup>](#). Microsoft also provides a [Visual Entity Navigator<sup>11</sup>](#) that allows users to visually navigate Common Data Model entities.

When building against the Common Data Model, you need to understand the layering approach that is taken to building out your custom application's schema. At the base of a Common Data Model application resides a core schema that contains commonly used entities such as Account and Contact. From there, you can layer function or industry-specific models to further accelerate the development of your own custom application, after which you can build your own models. We recommend that if you have a common scenario upon which data reuse is valuable, you should join the Open Data Initiative and begin contributing to the Common Data Model as well.

## Overview of Azure solutions and AI applications for Power Apps developers

### Azure solutions overview

The Azure ecosystem provides a wealth of functionality and enables Power Apps developers to harness its capabilities through various extensibility points. This unit reviews a few Azure solutions in the context of how they play, or potentially play, a role in a model-driven PowerApp. Azure offers many other solutions; therefore, all PowerApp developers should familiarize themselves with all that Azure has to offer. For an overview of all solutions, see the Azure website. For more in-depth training that will prepare you for taking the Microsoft Azure Fundamentals exam, consider completing the [Azure fundamentals<sup>12</sup>](#) learning path.

### Cognitive Services

Azure Cognitive Services includes APIs, SDKs, and services that are available to help developers add cognitive features to their applications. The Power Platform's extensibility framework makes it possible for users to incorporate these features into a PowerApp. The catalog of services within Azure Cognitive Services can be categorized into five main pillars: Vision, Speech, Language, Web Search, and Decision.

For more in-depth training on how to use some of the Cognitive Services features, review the various available [Learn modules and Learning Paths<sup>13</sup>](#) that are tagged with the Cognitive Services product. For

<sup>9</sup> <https://github.com/microsoft/CDM/?azure-portal=true>

<sup>10</sup> <https://docs.microsoft.com/common-data-model/schema/core/applicationCommon/overview/?azure-portal=true>

<sup>11</sup> <https://microsoft.github.io/CDM/?azure-portal=true>

<sup>12</sup> <https://docs.microsoft.com/learn/paths/azure-fundamentals/?azure-portal=true>

<sup>13</sup> <https://docs.microsoft.com/learn/browse/?products=azure-cognitive-services>

additional reference, see the **Azure Cognitive Services Documentation**<sup>14</sup>.

## Azure Bot Service

Azure Bot Service provides a framework in which you can build, test, deploy, and manage intelligent bots all in one place. Frequently, bots come into play with customer self-service, a common scenario in the Dynamics 365 for Customer Service application. Many other applications of Azure Bot Service are available, where you can shift simple, repetitive tasks such as gathering profile information into an interactive question and answer framework. For more information, see the **Azure Bot Service Learn modules**<sup>15</sup>

and also the **Azure Bot Service Documentation**<sup>16</sup>.  
Power Platform data seamlessly supports Azure Data Lake storage with its Common Data Service and Common Data Model framework. Building on top of the Power Platform enables you to empower complex analytics scenarios to use industrialized big data tools such as Power BI or Azure HDInsight. For more information on how the Common Data Model uses Azure Data Lake Storage, see **The Common Data Model and Microsoft Azure Data Lake Storage Gen2**<sup>17</sup>.

## Azure Functions

Azure Functions is a supported extensibility endpoint for model-driven Power Apps. Extracting custom logic into an Azure function enables you to offload complex logic outside of your transactional application, thus providing a much more stable and effective user experience.

## Service Bus

Azure Service Bus is a reliable messaging-as-a-service (MaaS) framework that enables real-time, asynchronous messaging across systems. In specific use cases, this is a valuable feature that provides the ability to integrate with both cloud and on-premise systems in a serverless, distributed fashion.

## Event Grid

Event Grid is a fully-managed event routing service.

---

<sup>14</sup> <https://docs.microsoft.com/azure/cognitive-services/?azure-portal=true>

<sup>15</sup> <https://docs.microsoft.com/learn/browse/?products=azure-bot-service>

<sup>16</sup> <https://docs.microsoft.com/azure/bot-service/?view=azure-bot-service-4.0/?azure-portal=true>

<sup>17</sup> <https://docs.microsoft.com/common-data-model/data-lake/?azure-portal=true>

## Logic Apps

Logic Apps is quickly becoming the preferred method for orchestrating integrations across the Power Platform.

## Azure SQL Database

Common Data Service provides the ability to configure near-real-time synchronization of data into an Azure SQL database by using **Data Export Service**. For more information, see the **Data export service documentation**<sup>18</sup>.

Note that the Common Data Service is built on top of Azure SQL databases.

## Dynamics 365 AI applications

Dynamics 365 provides several first-party AI applications that are worth mentioning. Among these are Customer Insights, Market Insights, AI for Sales, and AI for Customer Service Insights. The Market Insights application is currently in preview and will therefore not be covered in detail at this point.

To view currently released Dynamics 365 AI applications, see **Dynamics 365 AI apps**<sup>19</sup>.

## Customer Insights

Customer Insights is a valuable framework that enables organizations to connect to data from various transactional, behavioral, and observational sources to create a 360-degree customer view. It integrates with data sources, such as Microsoft Graph, to enable richer customer profiles, and supports embedding of customer insights within line-of-business applications that are built on the Microsoft Power Platform. For more information on how to install and/or configure this application, see **Dynamics 365 Customer Insights**<sup>20</sup> on Microsoft docs.

## Customer Service Insights

Dynamics 365 Customer Service Insights is an AI application that provides utility to a customer service organization by providing AI-driven technology for in-depth case analysis. It automatically groups cases into topics by using natural language understanding, which allows you to make informed decisions about how to improve customer service experiences. Additionally, it integrates with a variety of data sources and allows you to map custom fields into the application that might be used to drive the Customer Service Insights' machine learning models.

<sup>18</sup> <https://docs.microsoft.com/powerapps/developer/common-data-service/data-export-service/?azure-portal=true>

<sup>19</sup> <https://docs.microsoft.com/dynamics365/ai/?azure-portal=true>

<sup>20</sup> <https://docs.microsoft.com/dynamics365/ai/customer-insights/overview/?azure-portal=true>

## Sales Insights

Dynamics 365 Sales Insights has two main components as part of its application:

- Dynamics 365 Sales Insights capabilities for sellers
- Dynamics 365 Sales Insights capabilities for sales managers

Dynamics 365 Sales Insights capabilities for sellers consists of the following features:

- **Relationship analytics** - Gathers relevant information from throughout the Dynamics 365 for Sales database to create a graphical display of key performance indicators (KPIs) and activity histories. For more information, see [Use Relationship analytics to gather KPIs<sup>21</sup>](#).
- **Predictive lead scoring** - Helps you focus on revenue generation efforts by providing scores so you can prioritize efforts on quality leads. For more information, see [Work with Predictive lead scoring<sup>22</sup>](#).
- **Predictive opportunity scoring** - Helps you focus on revenue generation efforts by providing scores so you can prioritize efforts on quality opportunities. For more information, see [Convert opportunities into deals<sup>23</sup>](#).
- **Notes analysis** - Monitors notes that you enter regarding a recent meeting or discussion with your customer to provide intelligent suggestions. For more information, see [How notes analysis assists you with suggestion<sup>24</sup>](#).
- **Talking points** - Displays topics to start a conversation with your customer such as sports, vacation, family, and entertainment. For more information, see [Know conversation starters for your customers<sup>25</sup>](#).
- **Who knows whom** - Provides details such as names and email addresses of your colleagues who know a lead with whom you are going to interact. For more information, see [How to get introduced to a lead<sup>26</sup>](#).

Dynamics 365 Sales Insights capabilities for sales managers, which are a series of dashboards that provide insights into sales team performance, is currently in pre-release; therefore, it will not be covered in detail in this unit.

## Explore the Power Platform Developer Guide

A wealth of assets is available that developers can review to better acquaint themselves on developing against the Common Data

---

<sup>21</sup> <https://docs.microsoft.com/dynamics365/ai/sales/relationship-analytics/?azure-portal=true>

<sup>22</sup> <https://docs.microsoft.com/dynamics365/ai/sales/work-predictive-lead-scoring/?azure-portal=true>

<sup>23</sup> <https://docs.microsoft.com/dynamics365/ai/sales/work-predictive-opportunity-scoring/?azure-portal=true>

<sup>24</sup> <https://docs.microsoft.com/dynamics365/ai/sales/notes-analysis/?azure-portal=true>

<sup>25</sup> <https://docs.microsoft.com/dynamics365/ai/sales/talking-points/?azure-portal=true>

<sup>26</sup> <https://docs.microsoft.com/dynamics365/ai/sales/who-knows-whom/?azure-portal=true>

Service framework. Some of these are legacy assets that are branded as Dynamics 365 SDK documentation. Common Data Service is an evolution of the Dynamics 365 platform. Many of the existing materials surrounding how to extend the platform still apply. While these materials can be great resources, they're no longer current; therefore, use them with discretion.

## Sample code

Microsoft also provides a large amount of sample code that shows you a host of activities that you would commonly perform as a Common Data Service developer. These activities include interacting with the Web API or Organization Service, and also how to take advantage of a useful collection of assemblies called XrmTooling for accelerating common actions such as authentication against the Common Data Service by means of third-party applications. These codes can be found in GitHub and in various other mechanisms, such as through NuGet packages. You can find links to these in the following section.

## Useful links

The following are links that should be added as favorites to any Power Apps developer's repository.

**Common Data Service Developer Guide**<sup>27</sup>

**Model-driven App Developer Guide**<sup>28</sup>

**Power Apps Developer Documentation**<sup>29</sup>

**Common Data Service Sample and Helper Code**<sup>30</sup>

**Microsoft Power Apps NuGet Packages**<sup>31</sup>

## Summary

In this module, we reviewed the developer experience as it relates to the Power Platform. We looked at the key components that exist within the Power Platform, along with what Azure and AI solutions that are frequently used. We also reviewed where to find key developer resources for learning how to extend the Power Platform.

<sup>27</sup> <https://docs.microsoft.com/powerapps/developer/common-data-service/overview/?azure-portal=true>

<sup>28</sup> <https://docs.microsoft.com/powerapps/developer/model-driven-apps/overview/?azure-portal=true>

<sup>29</sup> <https://docs.microsoft.com/powerapps/?azure-portal=true#pivot=home&panel=developer>

<sup>30</sup> <https://github.com/microsoft/PowerApps-Samples/?azure-portal=true>

<sup>31</sup> <https://www.nuget.org/profiles/crmsdk/?azure-portal=true>

# Use developer tools to extend the Power Platform

## Explore the Power Platform Developer Guide

### Microsoft NuGet tools and other approved developer tooling

Microsoft provides a variety of tools to help support development against Common Data Service, which are available through **NuGet packages**<sup>32</sup>.

In addition, a selection of **community tools**<sup>33</sup>

can also be used to assist in your development efforts, along with numerous additional tools that you can find by searching NuGet and GitHub.

#### Microsoft NuGet tools

Tool	Executable	NuGet Package	Description
Code Generation Tool	CrmSvcUtil.exe	<b>Microsoft.CrmSdk.CoreTools</b> ( <a href="https://www.nuget.org/packages/Microsoft.CrmSdk.CoreTools/?azure-portal=true">https://www.nuget.org/packages/Microsoft.CrmSdk.CoreTools/?azure-portal=true</a> )	Generates early-bound .NET Framework classes that represent the entity data model that is used by model-driven apps.
Configuration Migration Tool	DataMigrationUtility.exe	<b>Microsoft.CrmSdk.XrmTooling.ConfigurationMigration.Wpf</b> ( <a href="https://www.nuget.org/packages/Microsoft.CrmSdk.XrmTooling.ConfigurationMigration.Wpf/?azure-portal=true">https://www.nuget.org/packages/Microsoft.CrmSdk.XrmTooling.ConfigurationMigration.Wpf/?azure-portal=true</a> )	Tool that is used to move configuration data across Common Data Service instances.
Package Deployer	PackageDeployer.exe	<b>Microsoft.CrmSdk.XrmTooling.PackageDeployment.WPF</b> ( <a href="https://www.nuget.org/packages/Microsoft.CrmSdk.XrmTooling.PackageDeployment.WPF/?azure-portal=true">https://www.nuget.org/packages/Microsoft.CrmSdk.XrmTooling.PackageDeployment.WPF/?azure-portal=true</a> )	Tool that enables administrators to deploy packages on a Common Data Service instance.

---

<sup>32</sup> <https://docs.microsoft.com/nuget/install-nuget-client-tools/?azure-portal=true>

<sup>33</sup> <https://docs.microsoft.com/nuget/install-nuget-client-tools/?azure-portal=true>

Tool	Executable	NuGet Package	Description
Plugin Registration Tool	PluginRegistration.exe	<b>Microsoft.CrmSdk.XrmTooling.PluginRegistrationTool</b> ( <a href="https://www.nuget.org/packages/Microsoft.CrmSdk.XrmTooling.PluginRegistrationTool/?azure-portal=true">https://www.nuget.org/packages/Microsoft.CrmSdk.XrmTooling.PluginRegistrationTool/?azure-portal=true</a> )	Tool that is used to register plug-ins against a Common Data Service instance.
Solution Packager Tool	SolutionPackager.exe	<b>Microsoft.CrmSdk.CoreTools</b> ( <a href="https://www.nuget.org/packages/Microsoft.CrmSdk.CoreTools/?azure-portal=true">https://www.nuget.org/packages/Microsoft.CrmSdk.CoreTools/?azure-portal=true</a> )	Tool that reversibly decomposes Common Data Service solutions into multiple XML files to be managed by a source control system.

## XrmToolbox Tools

There are a number of community developer tools available from **XrmToolBox**<sup>34</sup>. A few of the tools are listed below.

Tool	Description
WebResources Manager	Tool to install, update, delete, and preview web resources in a Model-driven app.
FetchXML Builder	Provides a UI to compose FetchXML queries and query information that is not easily found within the PowerApp user interface, and converts FetchXML into C# QueryExpression methods and OData query strings.
View Layout Replicator	Copy and apply the layout of a particular view to multiple views of the same entity.
View Designer	Tool to modify View queries that use FetchXML builder.
Early Bound Generator	Generate early-bound entities, option sets, and actions. Uses <b>CrmSvcUtil.exe</b> ( <a href="https://docs.microsoft.com/nuget/install-nuget-client-tools/?azure-portal=true">https://docs.microsoft.com/nuget/install-nuget-client-tools/?azure-portal=true</a> ).
Metadata Browser	Browse metadata that is contained within your Common Data Service
Plugin Trace Viewer	Easy-to-use tool to filter and display information within a Plug-in Trace Log.
Export to Microsoft Excel	Export records from a view or FetchXML query to Microsoft Excel.
Attribute Manager	Rename/delete/change the type of an attribute.
Iconator	Manage custom entity icons in a single screen.
Easy Translator	Export/import translations with contextual information.

<sup>34</sup> <https://www.xrmtoolbox.com/?azure-portal=true>

Tool	Description
Auto Number Manager	UI to set, update, and remove autonumber formatting on new or existing attributes.

## Deploy apps with Package Deployer

### Package Deployer overview

Package Deployer is a solution built by Microsoft that gives administrators the ability to deploy packages of solutions and data across environments. Data can be represented as flat files (in other words, csv files) or exported data from the Configuration Migration tool. You can also write custom code to run before, during, or after a package is deployed to a Common Data Service app instance.

### Configuring a Package Deployer package

The most complex element of creating a Package Deployer package is configuring its settings. These settings are maintained in an XML file (ImportConfig.xml, specifically) and can be grouped into a few main elements.

### configdatastorage Attributes

The main settings of a Package Deployer package are found at the configdatastorage root node of the ImportConfig.xml document. The following table shows the settings that are available for you to configure.

Attribute	Description
installsampledata	If set to true, installs sample data to the Common Data Service instance.
waitforsampledatatoinstall	If true, and installsampledata is also true, waits for sample data to install before deploying the package.
agentdesktopzipfile	Commonly used for creating Unified Service Desk packages. For more information about Unified Service Desk, see <b>Administration Guide for Unified Service Desk 3.0</b> ( <a href="https://docs.microsoft.com/dynamics365/customer-engagement/unified-service-desk/administration-guide-unified-service-desk-3?view=dynamics-usd-4.1?azure-portal=true">https://docs.microsoft.com/dynamics365/customer-engagement/unified-service-desk/administration-guide-unified-service-desk-3?view=dynamics-usd-4.1?azure-portal=true</a> ).

Attribute	Description
agentedesktopexename	Commonly used for creating Unified Service Desk packages. For more information about Unified Service Desk, see <b>Administration Guide for Unified Service Desk 3.0</b> ( <a href="https://docs.microsoft.com/dynamics365/customer-engagement/unified-service-desk/administration-guide-unified-service-desk-3?view=dynamics-usd-4.1?azure-portal=true">https://docs.microsoft.com/dynamics365/customer-engagement/unified-service-desk/administration-guide-unified-service-desk-3?view=dynamics-usd-4.1?azure-portal=true</a> ).
crmmigdataimportfile	File name of the default configuration data file (.zip) that is exported by using the Configuration Migration tool. For more information on the Configuration Migration tool, see <b>Move configuration data across instances and organizations with the Configuration Migration tool</b> ( <a href="https://docs.microsoft.com/dynamics365/unified-service-desk/admin/migrate-unified-service-desk-configuration-dynamics-365-server?view=dynamics-usd-4.1?azure-portal=true">https://docs.microsoft.com/dynamics365/unified-service-desk/admin/migrate-unified-service-desk-configuration-dynamics-365-server?view=dynamics-usd-4.1?azure-portal=true</a> ).

## <solutions> Node

The <solutions> node contains an array of <configsolutionfile> nodes that describe the solutions to import. The order of the solutions under this node indicates the order in which the solutions will be imported on the target model-driven app instance.

<configsolutionfile> nodes can contain the attributes that are shown in the following table.

Attribute	Description
solutionpackagefilename	Specifies .zip file name of the given solution
overwriteunmanagedcustomization	Specifies whether to overwrite any unmanaged customizations when you are importing a solution that already exists in the target Common Data Service instance. This is optional, and if you do not specify this attribute, by default the unmanaged customizations in the existing solution are not overwritten.
publishworkflowsandactivateplugins	Specifies whether to publish workflows and activate plug-ins in the target model-driven app instance after the solution is imported. This is optional. By default, workflows are published and plug-ins are activated after the solution is imported.

The following is an example of nodes that contain the attributes discussed in the previous table.

```

<solutions>
<configsolutionfile solutionpackagefilename="SampleSolutionOne_1_0_managed.zip"
overwriteunmanagedcustomizations="false"
publishworkflowsandactivateplugins="true"/>
```

```
<configsolutionfile solutionpackagefilename="SampleSolutionTwo_1_0_managed.zip"
overwriteunmanagedcustomizations="false"
publishworkflowsandactivateplugins="true"/>
<configsolutionfile solutionpackagefilename="SampleSolutionThree_1_0_managed.zip" />
</solutions>
```

## <filestoimport> Node

The <filestoimport> node contains an array of <configimportfile> and <zipimportdetails> nodes that are used to describe individual files and zip files, respectively, to be imported.

<configimportfile> nodes can contain the attributes that are outlined in the following table.

Attribute	Description
filename	Name of the file that contains the import data. If the file is a .zip file, a <zipimportdetails> node must be present with a <zipimportdetail> node for each file in the .zip file.
filetype	This can be csv, xml, or zip.
associatedmap	The name of the import data map to use with this file. If blank, the import process will attempt to use the system-determined import data map for the file that is being imported.
importtoentity	The destination entity to which this file should be imported.
datadelimiter	The name of the data delimiter that is used in the import file. Valid values are single quote or double quotes.
fielddelimiter	The name of the field delimiter that is used in the import file. Valid values are comma, colon, or single quote.
enableduplicatedetection	Indicates whether to enable duplicate detection rules on the data import. Valid values are true or false.
isfirstrowheader	Used to denote that the first row of the import file contains the field names. Valid values are true or false.
isrecordownerateam	Indicates whether the owner of the record on import should be a team. Valid values are true or false.
owneruser	Indicates the user ID that should own the records. The default value is the user who is currently logged in.
waitforimporttocomplete	If true, the system waits for the import to complete before proceeding. If false, it queues the jobs and moves on.

<zipimportdetails> nodes can contain the attributes that are shown in the following table.

Attribute	Description
filename	Name of the file that contains the import data.
filetype	This can be csv or xml.
importtoentity	The destination entity to which this file should be imported.

The following is an example of attributes within a <zipimportdetails> node.

```

<filestoimport>
<configimportfile filename="File.csv"
filetype="CSV"
associatedmap="FileMap"
importtoentity="FileEntity"
datadelimiter=""
fielddelimiter="comma"
enableduplicatedetection="true"
isfirstrowheader="true"
isrecordownerteam="false"
owneruser=""
waitforimporttocomplete="true" />
<configimportfile filename="File.zip"
filetype="ZIP"
associatedmap="FileMapName"
importtoentity="FileEntity"
datadelimiter=""
fielddelimiter="comma"
enableduplicatedetection="true"
isfirstrowheader="true"
isrecordownerteam="false"
owneruser=""
waitforimporttocomplete="true"/>
<zipimportdetails>
<zipimportdetail filename="subfile1.csv" filetype="csv" importtoentity="account" />
<zipimportdetail filename="subfile2.csv" filetype="csv" importtoentity="contact" />
</zipimportdetails>
</filestoimport>
```

## <filemapstoimport> Node

The <filemapstoimport> node contains an array of <configmapimportfile> nodes to import. The order of the map files in this node indicates the order in which they are imported. For information about data maps, see **Create Data Maps for Import**<sup>35</sup>.

The following is an example of a <filemapstoimport> node.

<sup>35</sup> <https://docs.microsoft.com/powerapps/developer/common-data-service/create-data-maps-for-import?azure-portal=true>

```
<filemapstoimport>
<configimportmapfile filename="FileMap.xml" />
</filemapstoimport>
```

## <crmdatafiles> Node

The <crmdatafiles> node contains an array of <cmtdatafile> nodes that contain localized versions of the configuration data files to be imported. You can also define custom logic to import a localized configuration data file instead of the default one that is specified in the **crmmigdataimportfile**. If you want to do this, see **Define Custom Code for your Package**<sup>36</sup>.

To import user information to a Common Data Service (on-premises) instance on a different domain, you must include the user map file (.xml) that is generated by using the Configuration Migration tool in your project, and then specify it along with the configuration data file by using the **usermapfilename** attribute.

The following is an example of importing user information to a Common Data Service (on-premises) instance.

```
<cmtdatafiles>
<cmtdatafile filename="data_1033.zip" lcid="1033" usermapfilename="UserMap.xml" />
<cmtdatafile filename="data_1041.zip" lcid="1041" usermapfilename="" />
</cmtdatafiles>
```

## Steps to create a Package Deployer package

The referenced template for creating a Package Deployer package is named Microsoft Dynamics CRM SDK Templates, and it is only supported to run through Visual Studio 2012/2013. However, it is reported to successfully run against newer Visual Studio instances such as Visual Studio 2017. Although the template is named Microsoft Dynamics CRM SDK, packages can successfully run against any model-driven PowerApp that is built against Common Data Service. Because model-driven apps are an evolution of Dynamics 365, which was previously named Dynamics CRM, some assets still contain this name throughout Microsoft's tooling. You can download the template by going to:

### **Microsoft Dynamics CRM SDK Templates<sup>37</sup>**

After downloading the template, select the CRMSDKTemplates.vsix file to install the template in Visual Studio.

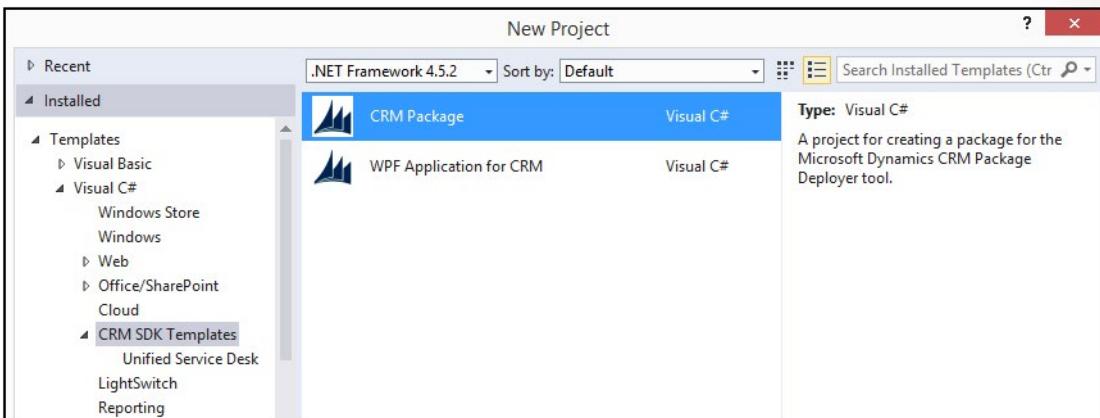
---

<sup>36</sup> <https://docs.microsoft.com/powerapps/developer/common-data-service/package-deployer/create-packages-package-deployer?azure-portal=true#Step5>

<sup>37</sup> <https://marketplace.visualstudio.com/items?itemName=DynamicsCRMPG.MicrosoftDynamicsCRMSDKTemplates>

## Step 1: Create a project by using the Visual Studio Template

1. Start Visual Studio and create a new project.
2. In the **New Project** dialog box:
  1. From the list of installed templates, expand **Visual C#** and select **CRM SDK Templates**.
  2. Ensure that **.NET Framework 4.6.2** is selected.
  3. Select **CRM Package**.
  4. Specify the name and location of the project and then select **OK**.



## Step 2: Add your files to the project

1. On the **Solutions Explorer** pane, add your solutions and files under the **PkgFolder** folder.
2. For each file that you add under the **PkgFolder** folder, on the **Properties** pane, set the **Copy to Output Directory** value to **Copy Always**. This ensures that your file is available in the generated package.

## Step 3: Update the HTML files: English and other languages

1. On the **Solution Explorer** pane, expand **PkgFolder > Content > en-us**. You'll find two folders called EndHTML and WelcomeHTML. These folders contain the HTML and associated files that enable you to display information at the end and beginning of the package deployment process. Edit the files in the HTML folder of these folders to add information for your package.
2. You can also add the HTML files in your package in other languages so that the content in the HTML appears in the language that is based on the locale settings of the user's computer.
  1. Create a copy of the **en-us** folder under **PkgFolder > Content**.
  2. Rename the copied folder to the appropriate language. For example, for the Spanish language, rename it to **es-ES**.
  3. Modify the content of the HTML files to add Spanish content.

## Step 4: Specify the configuration values for the package

1. Define the package configuration by adding information about your package in the **ImportConfig.xml** file that is available in

the **PkgFolder**. Double-click the file to open it for editing.

Configuration elements have been summarized earlier in this unit, while an example of ImportConfig.xml is as follows:

```
<?xml version="1.0" encoding="utf-16"?>
<configdatastorage xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
installsampledata="true"
waitforsampledatatoinstall="true"
agentdesktopzipfile=""
agentdesktopexename=""
crmmigdataimportfile="data_1033.zip">
<solutions>
<configsolutionfile solutionpackagefilename="SampleSolutionOne_1_0_managed.
zip"
overwriteunmanagedcustomizations="false"
publishworkflowsandactivateplugins="true"/>
<configsolutionfile solutionpackagefilename="SampleSolutionTwo_1_0_managed.
zip"
overwriteunmanagedcustomizations="false"
publishworkflowsandactivateplugins="true"/>
<configsolutionfile solutionpackagefilename="SampleSolutionThree_1_0_managed.
zip" />
</solutions>
<filestoimport>
<configimportfile filename="SampleOption.csv"
filetype="CSV"
associatedmap="SampleOption"
importtoentity="sample_option"
datadelimiter=""
fielddelimiter="comma"
enableduplicatedetection="true"
isfirstrowheader="true"
isrecordownerteam="false"
owneruser=""
waitForimporttocomplete="false"/>
<configimportfile filename="File.zip"
filetype="ZIP"
associatedmap="FileMapName"
importtoentity="FileEntity"
datadelimiter=""
fielddelimiter="comma"
enableduplicatedetection="true"
isfirstrowheader="true"
isrecordownerteam="false"
owneruser=""
waitForimporttocomplete="true"/>
<zipimportdetails>
<zipimportdetail filename="subfile1.csv"
filetype="csv"
importtoentity="account" />
<zipimportdetail filename="subfile2.csv"
```

```
filetype="csv"
importtoentity="contact" />
</zipimportdetails>
</filestoimport>
<filesmapstoimport>
<configimportmapfile filename="SampleOption.xml" />
</filesmapstoimport>
<cmtdatafiles>
<cmtdatafile filename="data_1033.zip"
lcid="1033"
usermapfilename="UserMap.xml" />
<cmtdatafile filename="data_1041.zip"
lcid="1041"
usermapfilename="" />
</cmtdatafiles>
</configdatastorage>
```

## Step 5: Build and deploy your package

1. Save your project and then build it (**Build > Build Solution**) to create the package. Your package is the following files under the `<Project>\Bin\Debug` folder.
  - **<PackageName> folder** - The folder name is the same as the one you changed for your package folder name earlier in this section (Step 5: Define custom code for your package). This folder contains all solutions, configuration data, flat files, and the contents for your package.
  - **<PackageName>.dll** - The assembly contains the custom code for your package. By default, the name of the assembly is the same as your Visual Studio project name.

2. To deploy your package, see **Deploy packages using Dynamics CRM Package Deployer and Windows PowerShell**<sup>38</sup>.

Note that while the name of the article references Dynamics CRM, the same procedure applies to any Model-driven app built on Common Data Service.

## Manage solutions with Solution Packager

### SolutionPackager tool overview

The SolutionPackager tool manifests a Common Data Service solution into a series of XML files. These XML files can be easily managed by a source control system. It's distributed as part of the NuGet package **Microsoft.CrmSdk.CoreTools**<sup>39</sup>.

<sup>38</sup> <https://docs.microsoft.com/dynamics365/customerengagement/on-premises/developer/create-packages-package-deployer>

<sup>39</sup> <https://www.nuget.org/packages/Microsoft.CrmSdk.CoreTools/?azure-portal=true>

The SolutionPackager tool is an executable that you can use to conduct the following actions:

- Extract: Extract solution .zip file to a folder
- Pack: Pack a folder into a .zip file

It comes with different kinds of command-line arguments that you can use to do specific actions as part of your solution extract/pack operation. For in-depth details on how to use the SolutionPackager tool, see the **SolutionPackager Tool**<sup>40</sup>

article on Microsoft Docs. Additionally, third-party tools are commonly used to simplify the process of extracting and/or packing solutions; these tools can be found on GitHub and NuGet.

## Extraction-based actions

The following command-line parameters are available for use with the SolutionPackager executable as it relates to extraction operations.

Argument	Description
/allowWrite{Yes No}	Optional. Default value is Yes. The /allowWrite:No parameter enables the extract option to be safely assessed without overwriting or deleting any existing files.
/allowDelete{Yes No Prompt}	Optional. Default value is Prompt. The /allowDelete:No parameter prevents any existing files within the extracted solution folder from being deleted. If /allowWrite:No is set, no deletes will occur even if /allowDelete:Yes is specified.
/clobber	Optional. When specified, the /clobber flag will overwrite or delete any file that has been set as read only. Clobbering a file is defined as overwriting its contents and is commonly used to clean up unnecessary files.
/sourceLoc: <string>	Optional. This argument generates a template resource file and is valid only on extract. Possible values are auto or an LCID/ISO code for the language that you want to export. When this argument is used, the string resources from the given locale are extracted as a neutral .resx file. If auto or just the long or short form of the switch is specified, the base locale or the solution is used. You can use the short form of the command: /src.

## Mapping files not compatible with SolutionPackager

Sometimes it's necessary to manage items, such as web resources, that might require files within a solution to be mapped from their location in the solution zip file to their location in the Package directory. To accomplish this, you can use the /map

---

<sup>40</sup> <https://docs.microsoft.com/powerapps/developer/common-data-service/compress-extract-solution-file-solutionpackager/?azure-portal=true>

argument to manage the mapping of these locations. For a detailed explanation on how to build an XML mapping document, see **Use the /map Command**

#### Argument<sup>41</sup>

At a high level, you can perform three different types of mapping activities: folder, file-to-file, and file-to-path. For folder mapping, file paths that match the specified **map** folder will be switched to the destination folder name that is specified in the **to** parameter. For file-to-file mapping, any file matching the **map** parameter will be read from the name and path that are specified in the **to** parameter. For file-to-path mapping, any file matching the **map** parameter is read from the path that is specified in the **to** parameter.

The following is a sample mapping:

```
<?xml version="1.0" encoding="utf-8"?>
<Mapping>
    <!-- Match specific named files to an alternate folder -->
    <FileToFile map="CRMDevToolkitSamplePlugins.dll"
        to=".\\.\\..\\Plugins\\bin\\**\\CRMDevToolkitSample.plugins.dll" />
    <FileToFile map="CRMDevToolkitSampleWorkflow.dll"
        to=".\\.\\..\\Workflow\\bin\\**\\CRMDevToolkitSample.Workflow.dll" />
    <!-- Match any file in and under WebResources to an alternate set of
        sub-folders -->
    <FileToPath map="WebResources\\*.*"
        to=".\\.\\..\\CrmPackage\\WebResources\\**" />
    <FileToPath map="WebResources\\**\\*.*"
        to=".\\.\\..\\CrmPackage\\WebResources\\**" />
</Mapping>
```

## Exercise - Install a Package Deployer package

In this exercise, you will create your Dynamics 365 trial organization, install a Dynamics 365 sales application, and install sample data.

Solutions used in the exercises are located in a **Git repository**<sup>42</sup>

### Task 1: Create a Dynamics 365 trial

In this task, you will create a Dynamics 365 trial organization.

1. Go to <https://trials.dynamics.com/Dynamics365/>.
2. Scroll down and select **Get Started**.

**41** <https://docs.microsoft.com/powerapps/developer/common-data-service/compress-extract-solution-file-solutionpackager?azure-portal=true#use-the-map-command-argument>

**42** <https://github.com/MicrosoftDocs/mslearn-developer-tools-power-platform/tree/master/exercise-package-deployer/?azure-portal=true>

## Try Dynamics 365

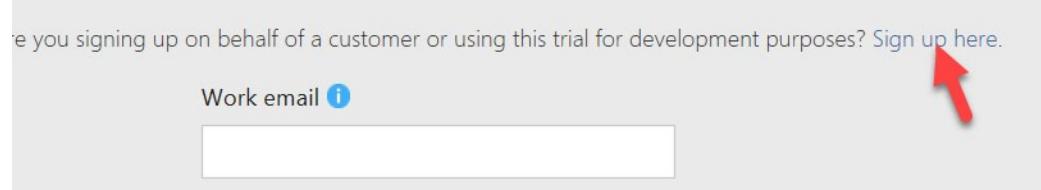
[GET STARTED >](#)

3. Select **Sign up for a free trial**.

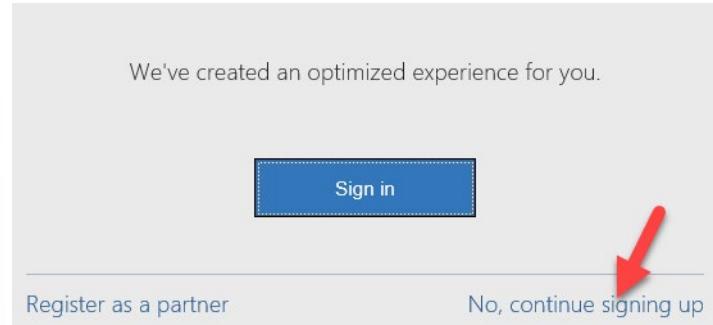
Prefer to get started on your own? [Sign up for a free trial](#). 

If you're a Microsoft partner, get access to environments through [PartnerSource](#).

4. Select **Sign up here**.



5. Select **No, continue signing up**.



6. Fill out the form and select **Next**.
7. Follow the sign-up wizard.
8. Select the **Set up** button when you get to the final step of the wizard.

Save this info. You'll need it later.

Sign-in page

<https://portal.office.com/>

Your user ID

admin@devmodules.onmicrosoft.com

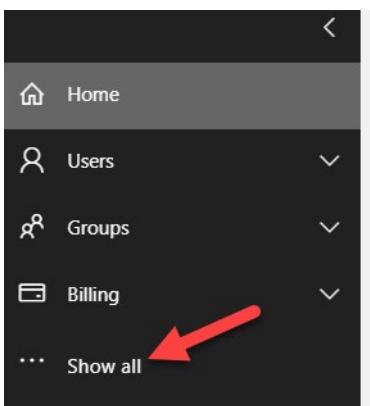
Set up

9. Wait for the setup to complete. You will receive an email when setup is completed.

## Task 2: Install the Dynamics 365 Sales application

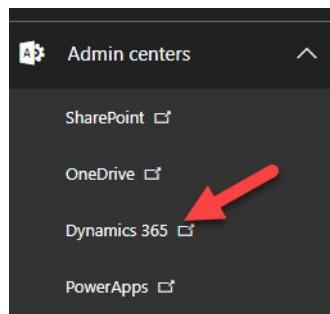
In this task, you will install the Dynamics 365 Sales Application.

1. Go to <https://admin.microsoft.com><sup>43</sup>.
2. Select **Show all**.



3. Expand **Admin centers**.
4. Select **Dynamics 365**.

<sup>43</sup> <https://admin.microsoft.com/>



5. Select your instance and then select Edit **Solutions**.

A screenshot of the 'Manage your Dynamics 365 instances' page. At the top, there are tabs for INSTANCES, UPDATES, SERVICE HEALTH, BACKUP & RESTORE, and APPLICATIONS. The INSTANCES tab is selected. Below the tabs, it says 'Manage your Dynamics 365 instances'. There are dropdown menus for Region (North America (NA)) and Instance Type (All). A table shows one instance: Dev Modules, ready, North America (NA), Production. To the right, there's a detailed view for 'Dev Modules':

- Dev Modules**: PRODUCTION Dynamics 365, version 9.0
- Purpose**: Add purpose by editing instance
- Solutions**: (with a blue edit icon) Crm Hub,OData v4 Data Provider

A red arrow points to the 'Edit' icon in the Solutions section.

6. Select **Dynamics 365 Sales Application** and then select **Install**.

7. Wait for the installation to complete.

DYNAMICS 365 PORTALS - LITTLE JAWA, LLC	1/1/2050	NOT INSTALLED
Dynamics 365 Portals - Pa... 9.0.12.11	1/1/2050	Not installed
Dynamics 365 Portals - Pa... 8.4.0.275	1/1/2050	Not installed
Dynamics 365 Portals - Pa... 8.4.0.275	1/1/2050	Not installed
Dynamics 365 Sales Application 9.0.1903.4006	1/1/2050	Installed

## Task 3: Install sample data

In this task, you will install sample data.

1. After **Dynamics 365 Sales Application** installation completes, select the back button.

Select a preferred solution to manage on selected instance: **Dev Modules**

SOLUTION NAME	VERSION	AVAILABLE UNTIL	STATUS
Dev Modules	1.0.0	2024-01-01	Active

**2. Select Open.**

**Dev Modules**  
PRODUCTION  
Dynamics 365, version 9.0  
**OPEN**

**EDIT**   **NOTIFICATIONS**

Purpose

**3. Locate and open the **Dynamics 365 - custom** application.**

**Dynamics 365 — custom**  
Provides access to the full suite of capabilities, including administration  
Microsoft Dynamics 365  
4/29/2019  
**WEB**

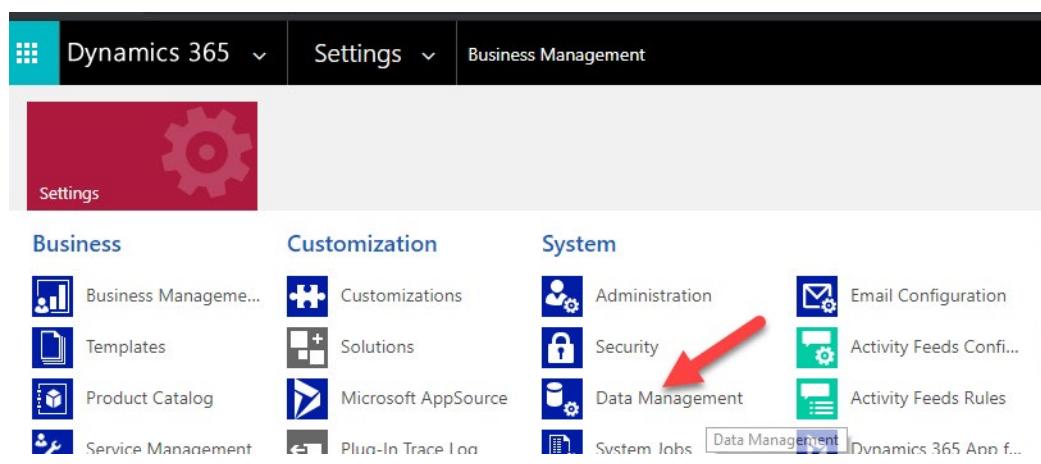
**CRM Hub**  
Mobile app that provides core CRM functionality including accounts, contacts, leads, opportunities, and more.  
Dynamics 365  
**UNIFIED INTERFACE**

**Dynamics 365**  
Provides access to the Dynamics 365 mobile app for Outlook.  
Dynamics 365  
**UNIFIED INTERFACE**

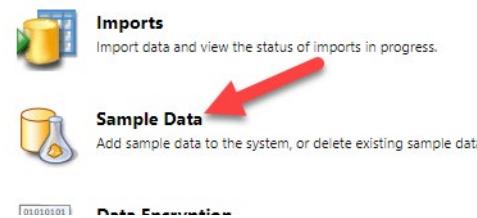
**4. Select **Settings** and then select **Advanced Settings**.**

Personalization Settings  
**Advanced Settings**  
Toast Notification Display T...  
About

**5. Select **Settings** and then select **Data Management**.**



6. Select **Sample Data**.



7. Select **Install Sample Data**.

8. Wait for the installation to start and select **Close**.

**Sample Data**

Installation of sample data is processed in the background. You can continue to use Microsoft Dynamics 365.



You are now ready to run the Tasmanian Traders package deployer and install the Scout Zone solution.

## Task 4: Run Package Deployer

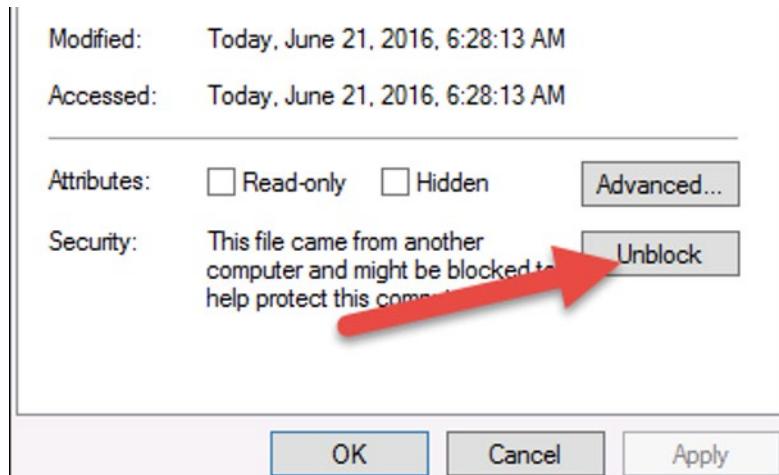
In this task, you will be deploying the Tasmanian Traders Sports Management solution. You will be using these customizations during the rest of the lab modules. Download tasmanian-traders-package-deployer.zip from **Git repository<sup>44</sup>** to complete this task.

1. Open the lab folder.
2. Right-click **TasmanianTradersPackageDeployer.zip** and then select **Properties**.

<sup>44</sup> <https://github.com/MicrosoftDocs/mslearn-developer-tools-power-platform/?azure-portal=true>

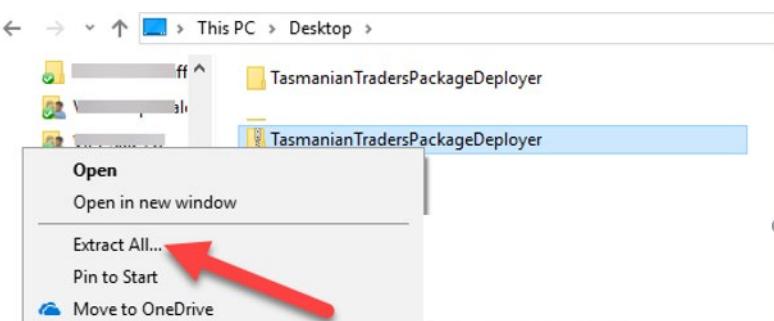


3. Select **Unblock**. This is to ensure that the file is appropriately extracted.



4. Select **Apply** and then select **Close**.

5. Right-click and select **Extract All**.



6. Select **Extract**.

7. Double-click **PackageDeployer**.

8. Select **Continue**.

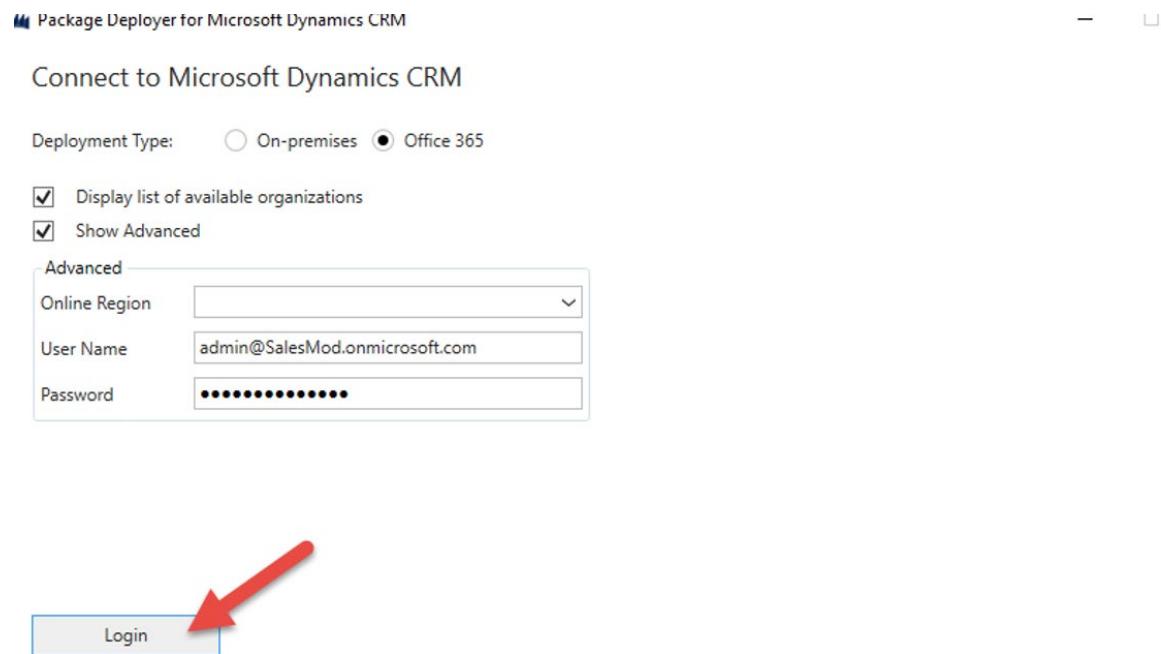
## Microsoft Dynamics CRM Package Deployer

The Package Deployer lets you package one or more CRM solutions, data, files, and custom code operations together into a "package" that can be deployed to any CRM environment. The Package Deployer also supports the upgrade of existing "package" deployments.

Please read and review the license agreement for this product.



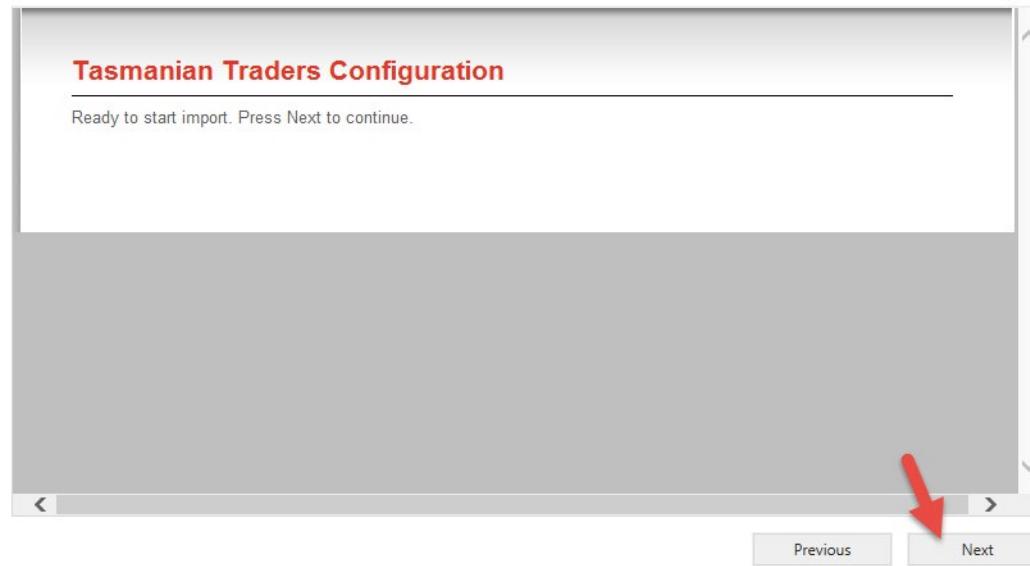
9. Select **Microsoft 365**, provide the credentials from your trial instance that you just created, and then select **Login**. Make sure to select your Region. For example, if your organization's URL is [crm.dynamics.com](http://crm.dynamics.com), then your Region is North America, if your organization's URL is [.crm4.dynamics.com](http://crm4.dynamics.com), then your Region is Europe.



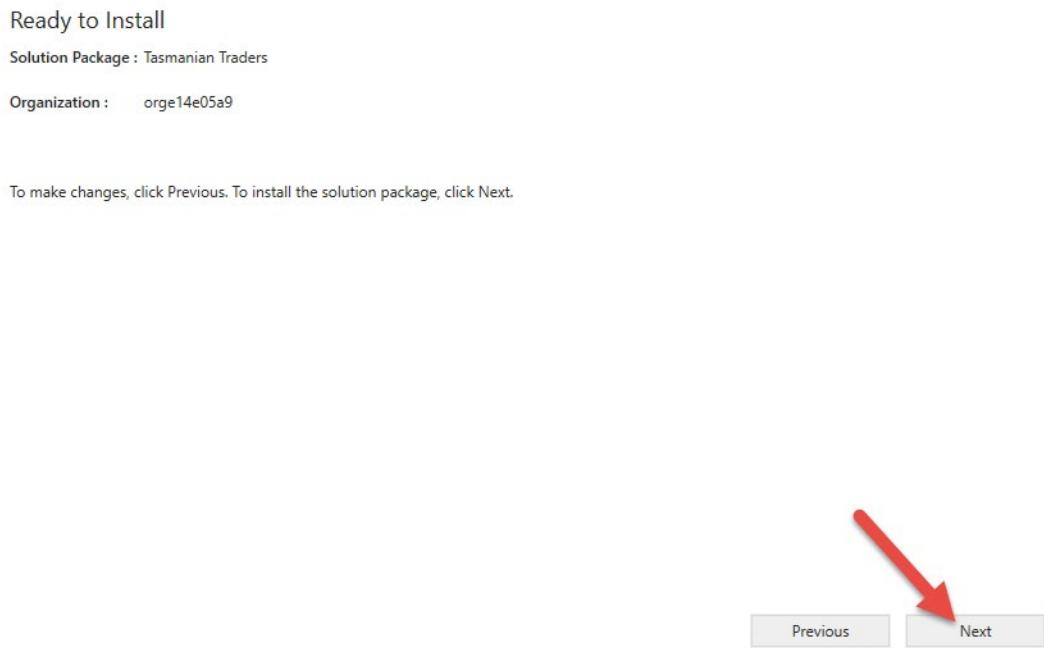
10. Select **Next**.

Welcome to the Tasmanian Traders Setup Tool

Tasmanian Traders Package

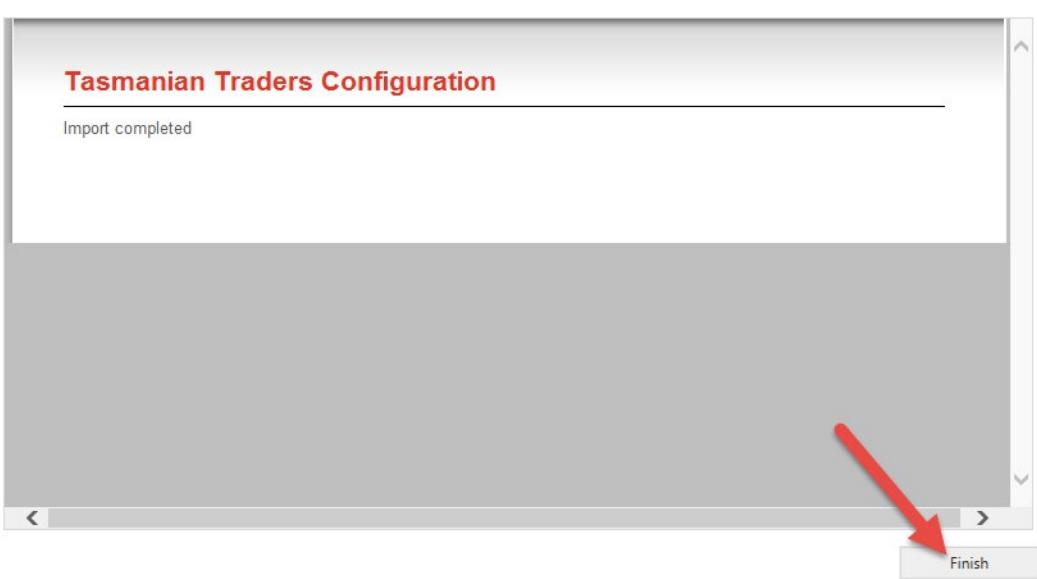


11. Select **Next**.



12. Select **Next**.
13. Wait and select **Next** again.
14. Select **Finish**.

Installation Complete



## Task 5: Import the Scout Zone solution

In this task, you will import the Scout Zone solution.

1. Go to <http://make.powerapps.com/> and make sure you are not in the default environment.

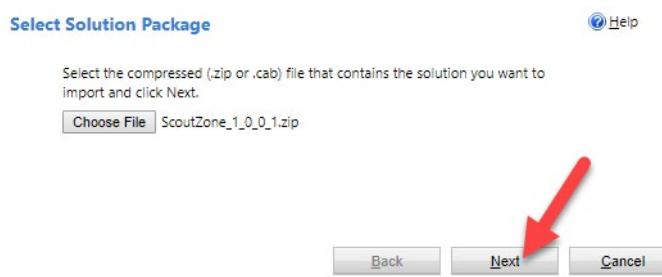
2. Select **Solutions** and then select **Import**.

The screenshot shows the 'Solutions' list page. The left sidebar includes options like Home, Learn, Apps, Create, Data, Business logic, and Solutions. The 'Solutions' option is highlighted with a red box. The top navigation bar has links for 'New solution', 'Import' (which is also highlighted with a red arrow), 'Open AppSource', 'Publish all customizations', 'Switch to classic', and a search icon.

Display name	Created	Version
Class Solution	4/29/2019	9.0.0.0
Sports Management Sitemap	4/29/2019	1.0.0.0
SportsManagementParking	4/29/2019	9.0.0.0
Sports Management Basketball	4/29/2019	9.0.0.0
Sports Management	4/29/2019	9.0.0.0
Playbook App	4/29/2019	9.0.1903.2010

3. Select **Browse**, select the **Scout Zone solution** that is located in the module resources folder, and then select **Open**.

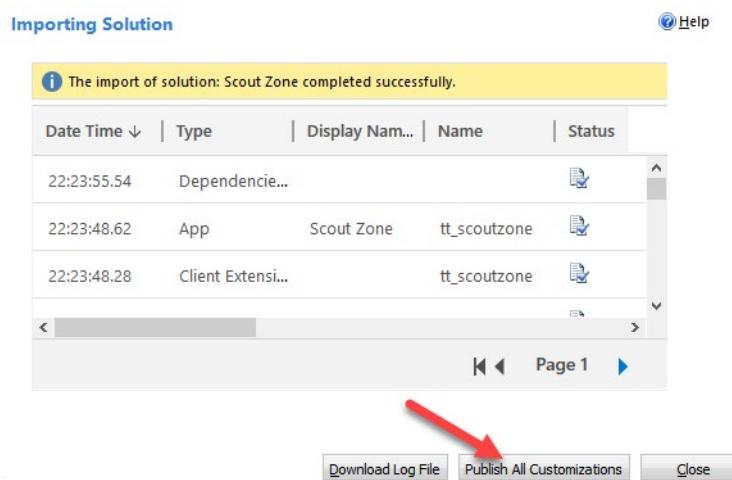
4. Select **Next**.



5. Select **Import**.

6. Wait for import to complete.

7. Select **Publish All Customizations**.

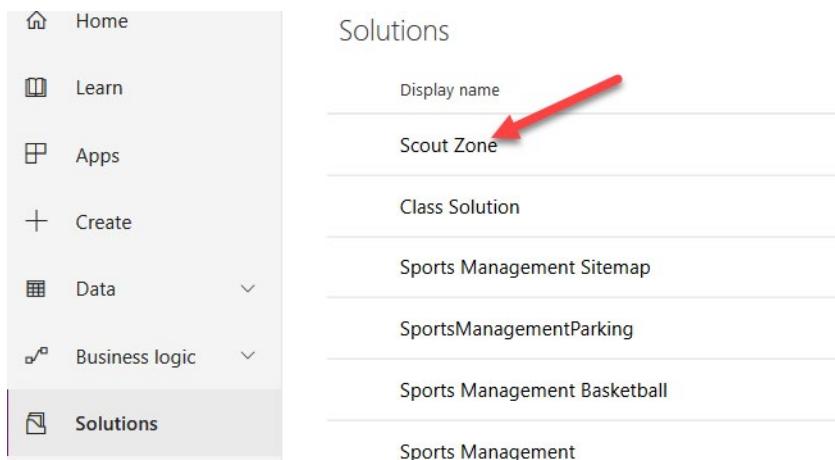


8. Select **Close**.

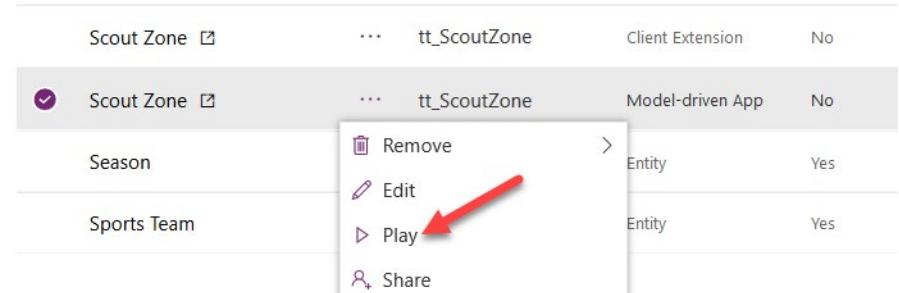
## Task 6: Tour the Scout Zone solution

In this task, you will tour the Scout Zone application to verify that it has installed successfully.

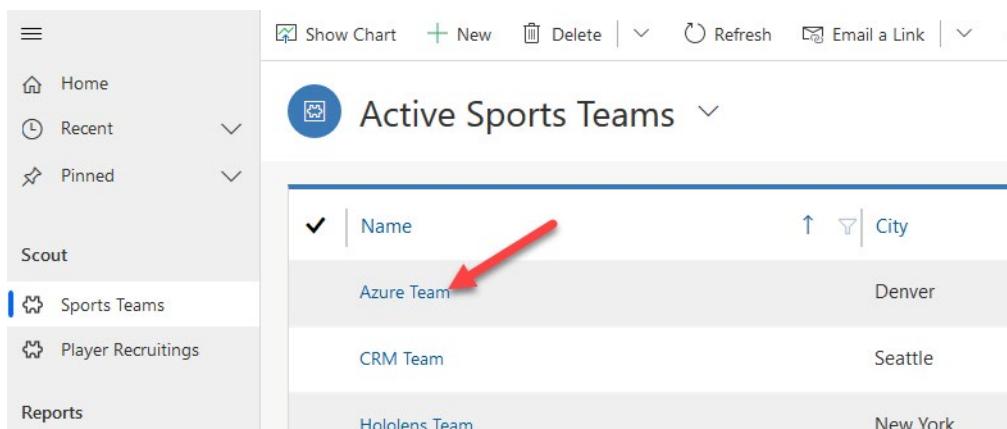
1. Go to <https://make.powerapps.com> and make sure you are not in the default environment.
2. Select **Solutions** and then select to open the **Scout Zone** solution.



3. Select the More (...) button of the Scout Zone **Model-driven app** and select **Play**. The **Scout Zone** application should load.



4. Select **Sports Teams** and then open one of the records.



5. The record should load and there should be a reference panel that shows the related **Seasons**.

The screenshot shows a Microsoft Power Platform canvas application interface. At the top, there is a circular icon with a gear and the text "SPORTS TEAM" and "Azure Team". Below this, there are two tabs: "General" (which is selected) and "Related". The main area displays a form with the following data:

Name	Azure Team
City	Denver
Division	Division 1
Home Stadium	Pepsi Center
Owner	Course User

Below the form, there is a section titled "Seasons" with a table:

Name	Created On
Azure Team - 2013	4/29/2019 4:19 PM
Azure Team - 2014	4/29/2019 4:19 PM
Azure Team - 2015	4/29/2019 4:19 PM
Azure Team - 2016	4/29/2019 4:19 PM

6. Select **Player Recruitings**. The **Active Player Recruitings** view should load.

The screenshot shows the Microsoft Power Platform canvas application interface. The left sidebar has the following navigation items:

- Home
- Recent
- Pinned
- Scout
- Sports Teams
- Player Recruitings (selected)
- Reports
- Scout Reports

The main area is titled "Active Player Recruitings" and contains a table:

Name	Created On
Maria A	4/29/2019 4:19 PM
Paul C	4/29/2019 4:19 PM
Paul G	4/29/2019 4:19 PM
Paul Z	4/29/2019 4:19 PM

7. Select **Scout Reports**. The **Active Scout Reports** view should load.

	Name	Player	Created On
Maria A	Maria A	4/29/2019 4:18 PM	
Paul C	Paul C	4/29/2019 4:18 PM	
Paul G	Paul G	4/29/2019 4:18 PM	
Paul Z	Paul Z	4/29/2019 4:18 PM	

8. Switch to the **Scout Zone Reports** view.

The **Scout Zone Reports** view should look like the following image.

	Name	Position (Player)	Recruiting Stat...	Height (Player)	Weight (Player)
Maria A	Power Forward	Pursuit	80.00	212.00	
Paul C	Power Forward	Pursuit	80.00	218.00	
Paul G	Small Foward	Pursuit	82.00	239.00	
Paul Z	Point Guard	Pursuit	77.00	208.00	
Rene T	Power Forward	Pursuit	75.00	186.00	
Rene X	Shooting Guard	Pursuit	80.00	231.00	

## Summary

In this module, we reviewed some of the developer tools available to do development activities on/with the Power Platform. There are many community tools available, along with ones provided by Microsoft. The Power Platform is continuously evolving, so we can expect new and improved tooling to continue to enter into our toolbox as time passes. Perhaps you might even come up with your own ideas on where you might implement your own tooling!

# Introduction to extending the Microsoft Power Platform

## Introduction to extending the Microsoft Power Platform

### Supported extensibility options

Various supported methods exist for extending a model-driven PowerApp.

The word *supported* refers to elements that Microsoft has documented as being an approved method for extending business apps and creating client applications that connect to Common Data Service.

### Apply business logic by using client script

Client-side scripting is one of the ways to apply custom business process logic for displaying data in a model-driven PowerApp. However, business rules are the preferred method and client script should only be used as a last resort. For more details on client scripting, refer to the Extend the user experience of a model-driven PowerApp learning path.

### Build custom UI components

You can use HTML web resources to build custom UI components within a model-driven PowerApp. The preferred method is to use the Power Apps component framework to achieve this objective.

For more details on building HTML web resources, refer to the Extend the user experience of a model-driven PowerApp learning path.

### Extend the Common Data Service platform

You can also extend the Common Data Service platform through other mechanisms such as writing plug-ins or custom workflow activities. For more details on how to develop these types of extensions, see the Extend the Common Data Service platform learning path.

### View or edit the app interface by using code (advanced)

Scenarios might occur where you want to use code to automate the customization of a model-driven app's interface. This could be an approach in certain Azure DevOps or deployment scenarios; however, it is highly uncommon. If you want to view and/or manipulate configuration at this level, we recommend that you use the Solution Packager tool. This tool extracts configuration items into their own XML files and helps make it easier to manage.

## Customize entity forms with code

All entity forms are stored in a SystemForm entity. When exporting a solution, you can view the FormXml within the form's definition. For more details, see [Customize entity forms<sup>45</sup>](#).

## Customize entity views with code

Entity views are stored in a SavedQuery entity and can be created programmatically. You can also define them as XML and import them with an unmanaged solution. Personal views are stored in the UserQuery entity and are owned by individual users. View queries are persisted by using the Common Data Service's standard query language, FetchXML. For more details on FetchXML, see [Use FetchXML to construct a query<sup>46</sup>](#).

For more details on customizing entity views, see [Customize entity views<sup>47</sup>](#).

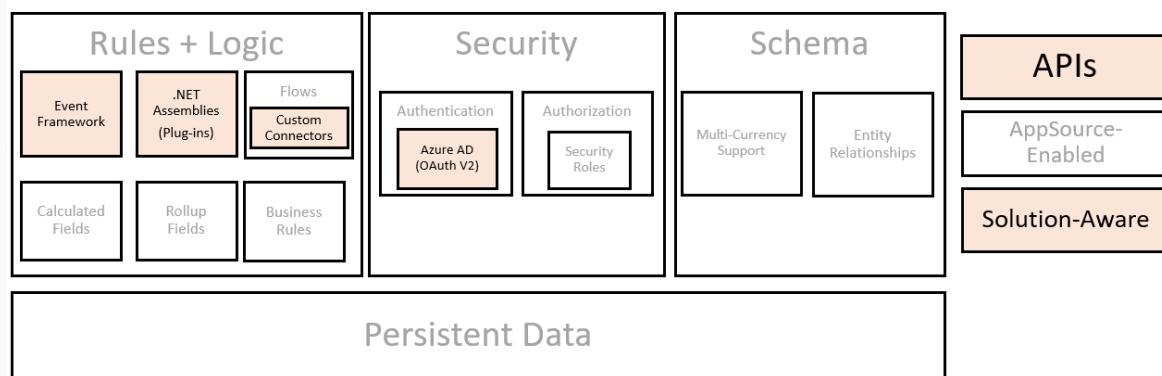
## Explore Common Data Service architecture

### Common Data Service extensibility model

To better understand how to extend Common Data Service, it is valuable to learn about its underlying architecture. Because Common Data Service is a software-as-a-service (SaaS) platform, most of these details, such as underlying data storage, are successfully abstracted from developers so they can focus on other tasks such as building custom business logic and integrating with other applications. The following graphic illustrates these underlying components, and the highlighted sections provide a summary of key areas upon which model-driven PowerApp developers should focus.

[!NOTE]

This unit is only intended to be a high-level view of the Common Data Service architecture. For more in-depth analysis of Common Data Service, refer to the [Common Data Service Developer Guide<sup>48</sup>](#).



<sup>45</sup> <https://docs.microsoft.com/powerapps/developer/model-driven-apps/customize-entity-forms/?azure-portal=true>

<sup>46</sup> <https://docs.microsoft.com/powerapps/developer/common-data-service/use-fetchxml-construct-query/?azure-portal=true>

<sup>47</sup> <https://docs.microsoft.com/powerapps/developer/model-driven-apps/customize-entity-views/?azure-portal=true>

<sup>48</sup> <https://docs.microsoft.com/powerapps/developer/common-data-service/overview/?azure-portal=true>

## Metadata and solution awareness

At its root, Common Data Service contains two core data models: metadata and application data. Additionally, Common Data Service provides a metadata-driven architecture to provide flexibility for users to create custom entities and extend existing entities. By using this approach, you can then transport your customizations across environments by means of a declarative solution that you can maintain as zip files or extract into XML files when you are using Microsoft's Solution Packager. Solution Packager is the preferred tool to compress and extract a solution file so you can check it into a source control repository. For more details, see **Use the Solution Packager tool to compress and extract a solution file<sup>49</sup>**.

When you create a custom entity, metadata about that entity, its attributes and relationships to other entities, and user interface components (such as forms and views) are stored as metadata in the system.

Common Data Service also exposes an OData V4 RESTful endpoint, known as the Web API, which can be used to view and manipulate metadata within a Common Data Service instance. One common mechanism for browsing your solution's metadata is the Metadata Browser. Microsoft provides this browser as a solution that you can install into your app. While the current links are found under **Dynamics 365 Customer Engagement apps developer documentation<sup>50</sup>**, under Microsoft Docs, this tool can be installed into any model-driven PowerApp that is built on Common Data Service. In addition, XrmToolbox has a **community metadata browser tool<sup>51</sup>** available.

## .NET Assemblies (plug-ins)

Plug-ins are managed code classes that are registered to run when specific events occur. When the event occurs, it passes through a plug-in pipeline that consists of three stages: PreValidation, PreOperation, and PostOperation. When registering a plug-in, you must choose which stage in which your code should run. The stage that you should choose depends on the purpose of the extension. You don't need to apply all your business logic within a single step.

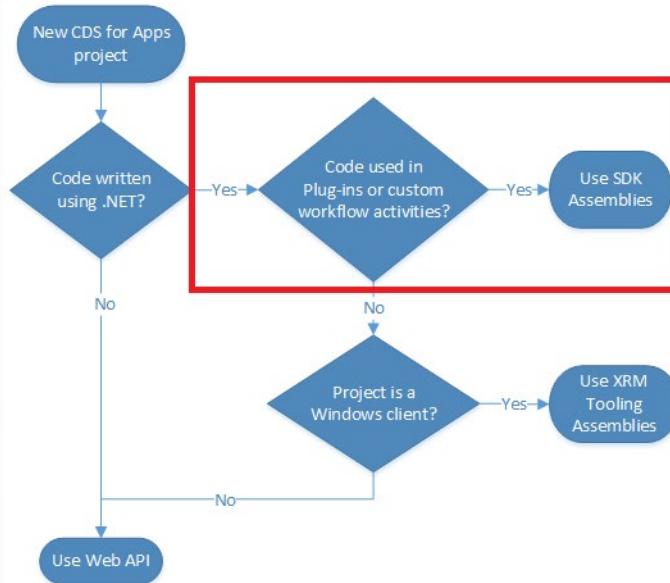
Plug-ins can also be configured to run either synchronously or asynchronously. Synchronous plug-ins run immediately according to the stage and run order, and the entire operation will wait until they complete. Asynchronous plug-ins are run through a system job after the operation completes and can only be registered for the PostOperation stage.

Plug-ins can also be developed and registered as custom workflow activities. Custom workflow activities is a tool that allows developers to produce reusable components that can be used by an administrator to perform actions not provided by the out-of-the-box workflow editor.

<sup>49</sup> <https://docs.microsoft.com/dynamics365/customerengagement/on-premises/developer/compress-extract-solution-file-solutionpackager>

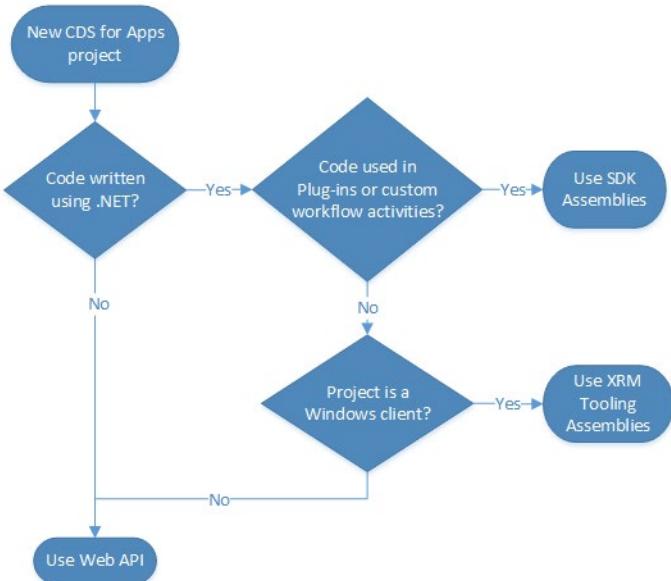
<sup>50</sup> <https://docs.microsoft.com/dynamics365/customerengagement/on-premises/developer/browse-your-metadata>

<sup>51</sup> <https://www.xrmtoolbox.com/plugins/MsCrmTools.MetadataBrowser/?azure-portal=true>



## APIs

Web services provide a programmatic interface for applications and services to access business data, organization information, and metadata that is stored in Common Data Service. The main service to access business data is called the Organization Service. Two protocols are available for application code to access this service, each with its own web endpoint: ODATA and SOAP. A RESTful Web API provides access to the ODATA V4 protocol of the web service, while an SDK provides a library of classes and methods to access the SOAP endpoint. The Web API is the more modern and preferred endpoint for new application code, while the SOAP endpoint is more for legacy applications and plug-in and/or custom workflow activity development. Both the Web API and the SOAP endpoints are functionally equivalent, except that the SOAP endpoint must be used for the development of plug-ins and custom workflow activities.



## Determine when to configure or when to code

As a best practice, you should approach customizing a model-driven PowerApp from the perspective that writing code is a last-resort method for achieving desired business application functionality. Quality areas such as maintainability, upgradability, stability, and performance should be considered when you are determining what is the best approach for a given scenario.

## Business rules vs. client script

At some point, business rules are not able to achieve certain objectives, or maybe the complexity that the rules impose cause developers to prefer writing the logic in client script. One scenario might be if you have a complex "if/then/else" situation that would be better achieved in a *switch* statement, or if you are dealing with dynamic values that aren't readily accessible by a business rule. For more practical scenarios on when client script is still a preferred and/or required method, refer to the client scripting modules of the Extend the user experience of a model-driven PowerApp learning path. For example, form notifications and filtering out option set values are not available through business rules.

## Workflows/flows vs. plug-ins/client script

In a similar manner as business rules versus client script, certain circumstances might occur upon which limitations might exist that require us to develop plug-ins to accomplish certain activities.

The following table can help you determine when it might be better to use a workflow rather than a plug-in or client script.

	Workflow	Plug-in	Client Script
Synchronous	Either	Either	Synchronous

	Workflow	Plug-in	Client Script
<b>Access External Data</b>	No	Yes	Yes (with limitations)
<b>Maintenance</b>	Business Users	Developers	Developers
<b>Can Run As</b>	User	Any licensed user, or current user	User
<b>Can Run On Demand</b>	Yes	No	No
<b>Can Nest Child Processes</b>	Yes	Yes	No
<b>Execution Stage</b>	Before/After	Before/After	Before/After
<b>Triggers</b>	Create, Field Change, Status Change, Assign to Owner, On Demand	Create, Field Change, Status Change, Assign to Owner, Delete, along with many other specialized triggers	Field Change or Form Load

## Exercise - When to configure vs. when to code

The Microsoft Power Platform is highly configurable. Sometimes, however, configuration options won't meet the system requirements, while other times, you will find the need for combining custom code with configuration.

In this exercise, you will be presented with challenges and will then determine if the requirement can be met by using configuration options or if custom code is needed.

### Challenge 1

When a record is created, child records must be created automatically.

Assume that the parent record is a sports season, and the child records would be games to be played in that season. For this challenge, you'll always want to create 10 games for each season.

Season: Summer 2019

Games: Game 1, Game 2, Game 3, and so on

This task could be accomplished through either Power Automate or a plug-in.

- With Power Automate, you would define and increment a variable for the name of each game record. The records would be created asynchronously.
- With a plug-in, you would also define a variable in the code and create each record. The records could be created either synchronously or asynchronously.

The requirement of accomplishing the task through Power Automate or a plug-in would be decided based on when you need the records created (immediately or as soon as possible) and the available resources needed to implement the requirement. In addition, if the requirement is to block the creation of the record, if for any reason any child records were unable to be created, a synchronous plug-in would be required.

## Challenge 2

The system has an option set with 17 values. You need to filter out invalid values on the form.

In this global option set, you might have values that apply across multiple entities. A small number of them could only apply on a subset of all the entities that you might use with this option set. Another possibility is that only certain values from the 17 would apply based on other available values such as a status field.

A business rule would work well in this situation because it would show and hide fields based on these criteria. However, with the values being from an option set, writing custom code is the only way to solve this requirement.

## Challenge 3

Calculate total time spent this quarter on closed activities for an account, including subaccounts and associated contacts (this does not include time spent on incomplete activities).

Calculating time-based logic with the constraints of the system is not an easy or often feasible task. If the desired functionality is to display on an account, a plug-in could be used to retrieve this data. Alternatively, a client script would likely be a better solution for exposing a Power BI element within the form in this case, where you could aggregate the data in a much cleaner manner. For more information on how to embed Power BI elements in a form, see **Embed a Power BI report in a model-driven system form**<sup>52</sup>.

## Challenge 4

A senior manager has left the company, and you need to find the 27,418 records that were owned by her and assign them to her replacement.

Before addressing this requirement, consider the following questions:

- What value do you have from historical data?
- Are you concerned about data compliance policies?
- Do you care about active records only? What about inactive records?
- What type of records do you need to reassign?
- Do you need to keep track of when the records were owned by the former manager?
- Should you treat activity records any differently?
- What about open activity records?

After having determined that these 27,418 records do, in fact, need to be assigned to another user, you can complete that action in a few different ways. The decision on how to accomplish the task usually constitutes determining what resources you have available to do the work. A functional consultant can assign the records by using an advanced find and manually reassigning in groups. You could also reas-

<sup>52</sup> <https://docs.microsoft.com/powerapps/maker/model-driven-apps/embed-powerbi-report-in-system-form/?azure-portal=true>

sign the records by using a classic workflow or Power Automate, which can still be completed by a functional consultant so you don't need to write custom code. If the highest priority is to reassign the records in the most efficient way possible, then it might make more sense to write custom code to accomplish this activity.

## Challenge 5

For this challenge, you will send birthday emails to all of your contacts.

Contact records contain a date of birth data entry. However, the date of birth and the birthday are not actually the same thing. A date of birth is a single event, whereas a birthday is the anniversary of the date of birth. Therefore, to use the birthday as a trigger, you must separate the pieces of data and take the month and day.

This task could be accomplished by using Power Automate, where you can extract the parts out of the date and store them on the record. (Likely, you would create ghost fields for these pieces of the date, meaning you would not have them exposed to the user on the form.) When you have these date parts, you could then also schedule a flow to do a daily query that searches for a match and takes any supported flow action, including sending an email.

There is a flow limitation of approximately 100,000 records for each run, which would only be a concern if you have more than 100,000 contacts with the same birthday. This might not be the most intuitive option and could be resolved with minimal amounts of code, so choosing which solution fits best depends on which requirement is most important (efficiency + intuitive versus maintaining a no-code solution).

## Challenge 6

You have an option set with three options; Gold, Silver, and Bronze. When the user selects Gold or Silver, you need to set a field on the form as required. When the user selects Bronze, you need to display a notification on the form.

This can be analyzed into two sets of requirements:

Requirement 1: When the user selects Gold or Silver, you need to set a field on the form as required.

Requirement 2: When the user selects Bronze, you need to display a notification on the form.

Requirement 1 can be accomplished quickly by using a business rule or form scripting. Requirement 2 cannot be done by using a business rule because they do not support form notifications; it must be done by using form scripting.

## Summary

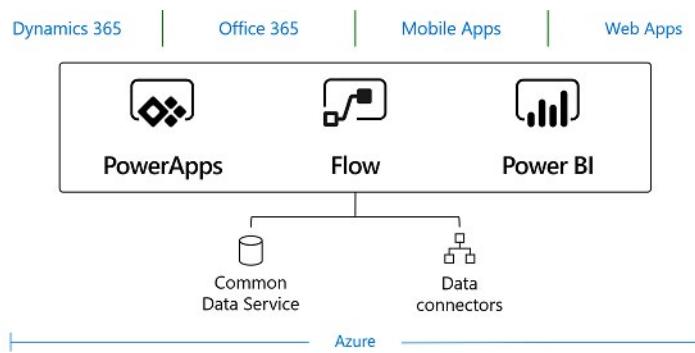
The Power Platform provides a robust extensibility framework in which developers can write code to do things like automate business processes. It is important however for developers to always use good judgment when determining when to write code versus using the existing out-of-the-box capabilities of the platform.

## Module 6 Extending the Power Platform Common Data Service

### Introduction to Common Data Service for developers

### Introduction to developing with Common Data Service

### Extending the Common Data Service platform



Extending the Common Data Service platform is required whenever some level of automation needs to occur that existing features do not support. These types of extensibility points frequently execute as server-side code via an asset called a Plug-in. Flow and other options enable asynchronous automation and logic, but they haven't been able to do all that plug-ins do today in terms of synchronous rules. When requirements for synchronous operations exist within a model-driven application, plug-ins are still required.

That being said, plug-in development is an essential skill contained in a PowerApp developer's arsenal and is one of the most common tasks required of us.

## Building model-driven apps on Common Data Service

All data and entity-level business logic that defines a model-driven app is stored within the Common Data Service. In addition, the method in which we package and distribute customizations to a Model-driven app (via Solutions) is also housed within Common Data Service.

## Building client apps on Common Data Service

Common Data Service exposes a set of APIs that can be leveraged to build any form of application, whether that is a custom web application written in ASP.Net, a Node.js application, or a Windows application (to name a few). The benefits of leveraging Common Data Service to build your application are numerous, a few of which are summarized in the next section.

## Benefits of the Common Data Service

The Common Data Service comes with a unified set of features that enable you to create business-focused definitions of your organization's data and business processes within a variety of applications. Since data and metadata are both stored in the cloud, applications are extremely easy to manage and administer. There is also a built-in security model that allows you to control access to entities and functionality for different sets of users within your organization.

Microsoft has already built a number of first-party applications on the Common Data Service, including Dynamics 365 Sales, Dynamics 365 Customer Service, and Dynamics 365 Talent. By building apps all on top of the same Common Data Service platform, you are able to easily interact with data constructed by other applications within the Common Data Service.

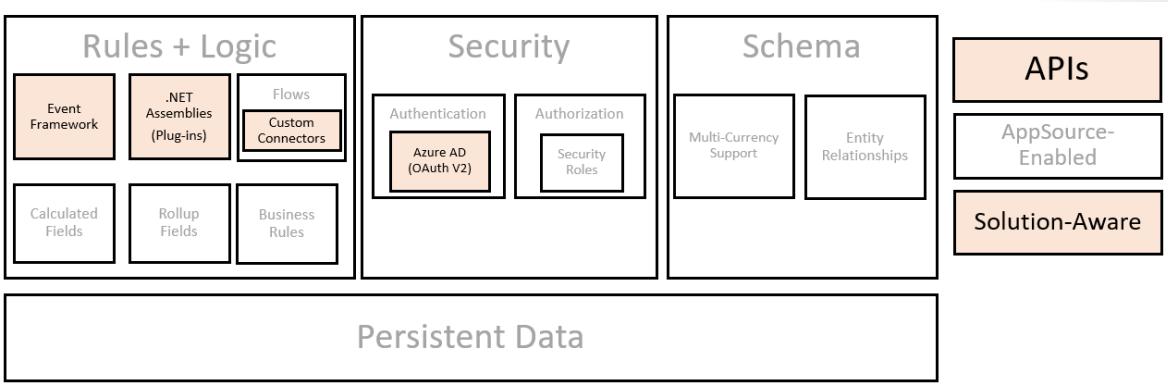
## Common Data Service extensibility model

To better understand how to extend the Common Data Service, it is valuable to learn about its underlying architecture. Since the Common Data Service is a software-as-a-service platform, most of these details such as underlying data storage, are successfully abstracted from us as developers and we can focus on more interesting items such as building custom business logic and integrating with other applications. The below graphic illustrates these underlying components, highlighted sections are areas where model-driven PowerApp developers can extend leveraging code.

[!NOTE]

This unit is only intended to be a high-level view of the Common Data Service architecture. For more in-depth analysis of the Common Data

Service, refer to the **Common Data Service Developer Guide**<sup>1</sup>.



## Metadata and solution-awareness

The Common Data Service provides a metadata-driven architecture to provide flexibility to create custom entities and extend existing entities. By leveraging this approach, we are then able to easily transport our customizations across environments via a declarative solution that we can maintain as zip files or extract into XML files when leveraging the Microsoft's Solution Packager. Solution Packager is the preferred tool to compress and extract a solution file so you can check it into source control repository. For more details, see **Use the SolutionPackager tool to compress and extract a solution file**<sup>2</sup>.

When you create a custom entity, metadata about that entity, its attributes and relationships to other entities, and user interface components (such as forms and views) are stored as metadata in the system.

The Common Data Service also exposes an OData V4 RESTful endpoint, known as the Web API, that can be leveraged to view and manipulate metadata within a Common Data Service instance. One common mechanism for browsing your solution's metadata is to leverage the Metadata Browser. Microsoft provides this as a solution that you can install into your app. The current links are found under the developer documentation under **Microsoft Docs**<sup>3</sup>. This tool can be installed into any model-driven PowerApp built on the Common Data Service. In addition, **XrmToolbox**<sup>4</sup> has a community metadata browser tool available.

## .NET assemblies (plug-ins)

Plug-ins are managed code classes that are registered to run when specific events occur. When the event occurs, they pass through a

<sup>1</sup> <https://docs.microsoft.com/powerapps/developer/common-data-service/overview/?azure-portal=true>

<sup>2</sup> <https://docs.microsoft.com/dynamics365/customerengagement/on-premises/developer/compress-extract-solution-file-solutionpackager?azure-portal=true>

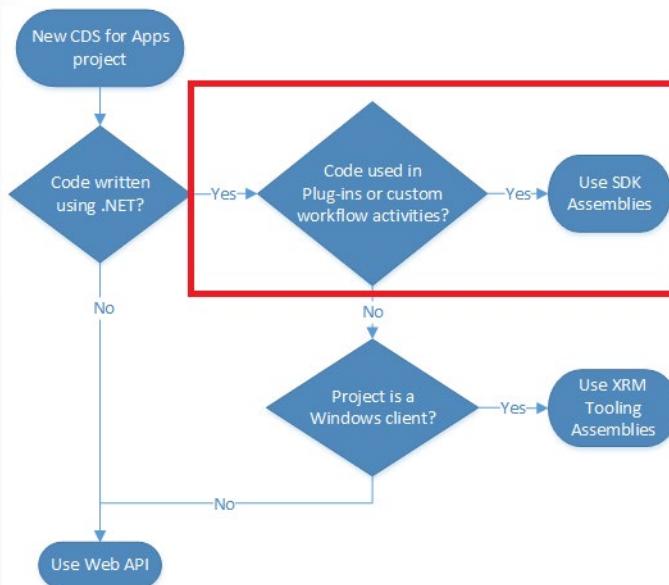
<sup>3</sup> <https://docs.microsoft.com/powerapps/developer/common-data-service/browse-your-metadata?azure-portal=true>

<sup>4</sup> <https://www.xrmtoolbox.com/plugins/MsCrmTools.MetadataBrowser/?azure-portal=true>

plug-in execution pipeline that consists of three stages: PreValidation, PreOperation, and PostOperation. When registering a plug-in, you must choose which stage in which your code should run. The stage you should choose depends on the purpose of the extension. You don't need to apply all your business logic within a single step.

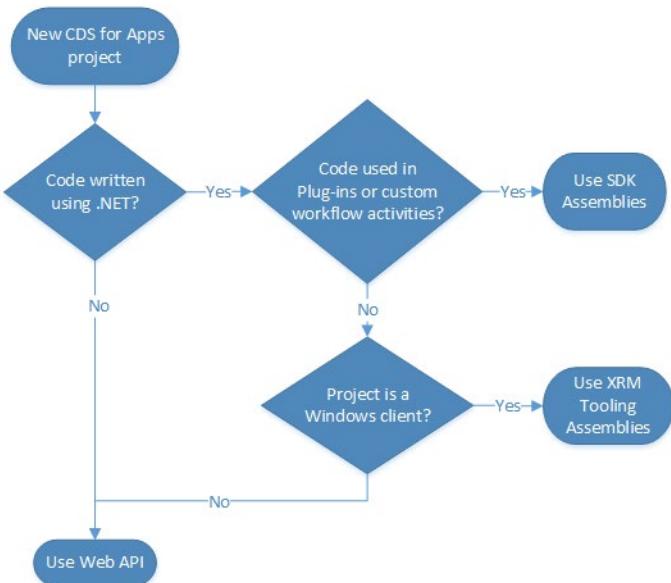
Plug-ins can also be configured to run either synchronously or asynchronously. Synchronous plugins execute immediately according to the stage and execution order, and the entire operation will wait until they complete. Asynchronous plug-ins are executed via a system job after the operation completes, and as such can only be registered for the PostOperation stage.

Plug-ins can also be developed and registered as Custom Workflow Activities. Custom Workflow Activities are a tool that allow developers to produce reusable components that can be leveraged by an administrator to conduct actions not provided by the out-of-the-box workflow editor.



## APIs

The web services provide a programmatic interface for applications and services to access business data, organization information, and metadata stored in Common Data Service. The main service to access business data is called the Organization Service. There are two protocols available for application code to access this service, each with its own web endpoint: ODATA and SOAP. A RESTful Web API provides access to the ODATA V4 protocol of the web service while an SDK provides a library of classes and methods to access the SOAP endpoint. The Web API is the more modern and preferred endpoint for new application code, while the SOAP endpoint is more for legacy applications and plug-in/custom workflow activity development. Both the Web API and the SOAP endpoints are functionally equivalent, except that the SOAP endpoint must be used for the development of plug-ins and custom workflow activities.



## Event framework

### Event framework overview

Common Data Service exposes events to indicate where the process is currently executing pipeline-wise. We can handle these events to do things like validate data, prevent transactions from completing, or automate any sort of business logic that cannot be accomplished by declarative means.

We can subscribe to these events by registering .NET assemblies (called plug-ins) to execute logic whenever the given event occurs. The method by which we actually perform the registration is by leveraging a tool called the Plugin Registration Tool. For more information on the Plugin Registration Tool, see the [Register a plug-in<sup>5</sup>](#) tutorial.

At a high level, handling events involves three things. First, we must subscribe to a specific message that represents the type of operation (or event) that is currently occurring (or about to occur) such as Create, Retrieve, Update, etc. We then must also indicate where in that event pipeline we would like our logic to execute (i.e., before or after the operation). We can also handle events before validation occurs, a convenient method that can be leveraged to perform advanced validation logic that cannot be accomplished via business rules or workflows. Lastly, we need to indicate the execution mode we would like our logic to run (synchronously or asynchronously).

Let's now go into these three areas in more detail.

<sup>5</sup> <https://docs.microsoft.com/powerapps/developer/common-data-service/register-plug-in/?azure-portal=true>

## Event messages

Common Data Service exposes a number of messages that are published when various data operations occur. For more information on these messages, see

**Use messages with the Organization service<sup>6</sup>.**

The basic data operations exposed by Common Data Service are:

- Create
- Retrieve
- RetrieveMultiple
- Update
- Delete
- Associate
- Disassociate

In addition, there are various messages that are exposed contextually, depending on what type of entity you are handling. For example if my entity has a Rollup Field, I can implement an event handler on the CalculateRollupField event message.

Generally, we can find an inventory of these custom messages made available via Common Data Service by searching through the

**Microsoft.Sdk.Messages<sup>7</sup>**

namespace for any classes whose name end in \Request. Another way to see which messages are available for a given type of entity is to navigate entity-message combinations via the Plugin Registration Tool.

In addition, we can create and expose our own messages by creating custom *Actions*. For more information on Actions, see **Create your own actions<sup>8</sup>**.

## Event pipeline

In addition to subscribing to a given message, or event type, Common Data Service also exposes a means for determining where in the pipeline that event is currently executing. For ASP.NET developers, this can be thought of similarly to how we work against page lifecycles within a web application. This is a common development pattern for publish-subscribe architectures and should feel fairly familiar to developers who are experienced with other event frameworks.

## PreValidation event

The PreValidation event occurs first in the pipeline, before any security checks are performed. It is intended for usage to ensure that the user executing the current transaction has the correct permissions needed to perform the intended operation.

---

<sup>6</sup> <https://docs.microsoft.com/powerapps/developer/common-data-service/org-service/use-messages/?azure-portal=true>

<sup>7</sup> <https://docs.microsoft.com/dotnet/api/microsoft.xrm.sdk.messages?view=dynamics-general-ce-9/?azure-portal=true>

<sup>8</sup> <https://docs.microsoft.com/powerapps/developer/common-data-service/custom-actions/?azure-portal=true>

As a developer, you can leverage this event to run validation logic and cancel the operation before the transaction occurs. For example, if configured to run whenever an entity is updated, you have the ability to cancel the operation before the update occurs by throwing an *InvalidPluginExecutionException* method within your plugin's execution logic. For more information on Execution context, see **Understand the Execution Context<sup>9</sup>**.

## PreOperation event

Use this event if you want to change any values of the entity prior to it being saved.

## PostOperation event

Use this event to modify any properties of the message before it is returned to the caller. Make sure not to apply any updates to the corresponding entity at this point as it will trigger another update event!

## Execution modes (synchronous versus asynchronous)

Plug-ins can be configured to run synchronously or asynchronously, depending on what type of pipeline operation we are handling.

### Synchronous mode

Plug-ins registered in this mode will run as soon as their execution pipeline stage is reached, and the entire operation will not proceed until the logic has completed execution. If multiple plug-ins are registered to run against the same pipeline stage, the Execution Order attribute (specified via the Plugin Registration Tool) will determine which one runs first.

### Asynchronous mode

Plug-ins registered in this mode will be dispatched as a system job to the Asynchronous service, which executes their logic after the given operation completes. For more information on how system jobs work, see **Asynchronous service<sup>10</sup>**.

[!NOTE]

Asynchronous plug-ins can only be registered for the **PostOperation** stage of the Event Pipeline.

## Summary

Common Data Service is a highly evolved platform in which we can build robust business applications. It serves as the platform for model-driven apps, first party apps such as Dynamics 365 for Sales/Customer Service

<sup>9</sup> <https://docs.microsoft.com/powerapps/developer/common-data-service/understand-the-data-context/?azure-portal=true>

<sup>10</sup> <https://docs.microsoft.com/powerapps/developer/common-data-service/asynchronous-service/?azure-portal=true>

(to name a few), and provides extensible methods for automating business processes and advanced analytics, leveraging things like Power Automate and Power BI.

For more information on data modeling, see **Common Data Service - data modeling<sup>11</sup>**.

For more information on data security, see **Common Data Service - security modeling<sup>12</sup>**.

---

<sup>11</sup> [https://youtu.be/s1Zqv\\_8QLNQ?azure-portal=true](https://youtu.be/s1Zqv_8QLNQ?azure-portal=true)

<sup>12</sup> <https://youtu.be/HHBoTNMZtsQ?azure-portal=true>

## Extend plug-ins

### Introduction to Plug-ins

A plug-in is *imperative* logic that should be used only when a *declarative* process, such as a business rule, flow, or workflow, does not meet your requirement.

Fundamentally, a plug-in is merely a .NET assembly that implements an IPlugin interface, which can be found in the Microsoft.CrmSdk.CoreAssemblies NuGet package. The IPlugin interface exposes a single method, Execute, which is where you can place any custom logic that you want to invoke based on whatever event you are handling.

Common scenarios of when to use plug-ins are:

- Canceling the event and displaying an error to the user.
- Making changes to the data in the operation.
- Initiating other actions by using the Organization Service to add automation.

### Alternatives to plug-ins

Plug-ins should be considered a last resort in many cases.

Although plug-ins are powerful and, if well written, highly performant, it's important to minimize the amount of custom/imperative logic that you place into your system, because it can affect maintainability, upgradability, and so on.

Common alternatives to plug-ins are:

- Workflows
- Power Automate
- Calculated and Rollup Fields
- Custom Actions

### Plug-in considerations

Plug-ins perform better when you consider their performance, capabilities, and ability to run synchronously.

### Performance

A well-written plug-in will always be the most efficient way to apply business logic to Common Data Service. However, a poorly written plug-in can create a significant negative impact on the performance of your environment.

### Capabilities

Plug-ins provide several capabilities that are not available with declarative business logic, such as efficiently working with external services in code. Nevertheless, Power Automate is rapidly approaching parity with plug-ins.

## Ability to run synchronously

If synchronous logic is required for your application, plug-ins might be a required option for you. However, on-demand workflows can also run synchronously and should be considered, depending on your requirements.

## Plug-ins usage scenarios

It is best practice to approach customizing a model-driven PowerApp with the idea that writing code is a last-resort method for achieving desired business application functionality. Quality areas such as maintainability, upgradability, stability, and performance should factor in when you determine the best approach for a given scenario. Considering these quality areas is one of the most important skills that any Power Apps developer can have.

## Business rules vs. plug-ins

Occasionally, business rules are not able to achieve certain objectives, or maybe their complexity causes developers to prefer writing the logic in a plug-in. One scenario could be if you have a complex "if/then/else" situation that would be more easily achieved in a *switch* statement, or when you are dealing with dynamic values that aren't readily accessible by means of a business rule. Client scripting is also an option for this scenario.

## Workflows/flows vs. plug-ins/client script

Circumstances might occur when existing limitations require you to develop plug-ins to accomplish certain activities.

The following table can help you determine when it might be more appropriate to use a workflow versus a plug-in or Client Script.

	Workflow	Plug-in	Client Script
Synchronous	Either	Either	Synchronous
Access External Data	No	Yes	Yes (with limitations)
Maintenance	Business Users	Developers	Developers
Can Run As	User	Any licensed user or current user	User
Can Run On Demand	Yes	No	No
Can Nest Child Processes	Yes	Yes	No
Execution Stage	Before/After	Before/After	Before/After
Triggers	Create, Field Change, Status Change, Assign to Owner, On Demand	Create, Field Change, Status Change, Assign to Owner, Delete, along with many other specialized triggers	Field Change or Form Load

## Custom workflow extensions

Workflow extensions are a derivative of plug-ins that enable developers to write custom activities that aren't available within the default process activities that are found within the workflow designer.

Power Automate is quickly evolving to support a wealth of operations that are not available in Common Data Service workflows and should be considered as an alternative whenever possible.

As with plug-ins, one common scenario where Common Data Service workflow assemblies might be more appropriate is if the business logic that you're creating needs to run synchronously. Currently, if you need synchronous logic to run within your Common Data Service environment, you must configure an on-demand workflow or synchronous plug-in to achieve this requirement.

## Custom workflow extension assemblies

Instead of implementing the `IPlugin` interface as is done with plug-ins, custom workflow extensions inherit a `CodeActivity` class. This class exposes an `Execute` method that provides a `CodeActivityContext` parameter.

## Input and output arguments

Custom workflow extensions allow you to define parameters that can be passed to and from your workflow extension by using `InArgument` and `OutArgument` objects in conjunction with decorating these objects with `Input` and `Output` attributes.

For example, to configure an input argument, you can use the following pattern by using **.NET Attributes**<sup>13</sup>:

```
[Input("Integer input")]

public InArgument<int> IntInput { get; set; }
```

To configure an output argument, use the following syntax:

```
[Output("Integer output")]

public OutArgument<int> IntOutput { get; set; }
```

You can also use the same property as both an input and output argument by including both attributes:

```
[Input("Int input")]

[Output("Int output")]

public InOutArgument<int> IntParameter { get; set; }
```

<sup>13</sup> <https://docs.microsoft.com/dotnet/standard/attributes/index/?azure-portal=true>

The code within your Execute method can then get or set these parameters through the provided Get and Set methods. For example, the following workflow extension increments an input parameter by 10 and returns it to the corresponding output parameter. Because math operations aren't made available through the workflow designer, this is a common scenario where a custom workflow extension might be required (Power Automate should also be considered in this case, however).

```
using Microsoft.Xrm.Sdk.Workflow;

using System.Activities;

public class IncrementByTen : CodeActivity
{
    [RequiredArgument]
    [Input("Decimal input")]
    public InArgument<decimal> DecInput { get; set; }

    [Output("Decimal output")]
    public OutArgument<decimal> DecOutput { get; set; }

    protected override void Execute(CodeActivityContext context)
    {
        decimal input = DecInput.Get(context);
        DecOutput.Set(context, input + 10);
    }
}
```

## Other property attributes

Workflow extensions also expose the following .NET attributes that can be used by your input and output parameters.

[RequiredArgument]	Used to make an input argument required.
[Default("DefaultValue")]	Used to specify default values for an input or output argument when one hasn't been set already.

[RequiredArgument]	Used to make an input argument required.
[ReferenceTarget("entityname")]	Required when you define a property for an EntityReference parameter. For more information, see <b>EntityReference parameters</b> ( <a href="https://docs.microsoft.com/powerapps/developer/common-data-service/workflow/workflow-extensions#entityreference-parameters">https://docs.microsoft.com/powerapps/developer/common-data-service/workflow/workflow-extensions#entityreference-parameters</a> ).
[AttributeTarget("entityname", "optionsetname")]	Used by OptionSetValue parameters, it defines which entity and attribute contains the valid set of values for the parameter. For more information, see <b>OptionSetValue parameters</b> ( <a href="https://docs.microsoft.com/powerapps/developer/common-data-service/workflow/workflow-extensions#option-setvalue-parameters">https://docs.microsoft.com/powerapps/developer/common-data-service/workflow/workflow-extensions#option-setvalue-parameters</a> ).

## Plug-in execution context

Whenever a plug-in (or custom workflow extension) runs, a wealth of data is made available by Common Data Service that contains information about the context in which the current operation resides. Both plug-ins and custom workflow assemblies have access to an `IWorkflowContext` class, which is accessible through differing methods.

In plug-ins, the `IPluginExecutionContext` is accessible through the `IServiceProvider` parameter of the `Execute` method by calling the `GetService` method.

```
public class SamplePlugin : IPlugin
{
    public void Execute(IServiceProvider serviceProvider)
    {
        // Obtain the execution context from the service provider.
        IPluginExecutionContext context = (IPluginExecutionContext)
            serviceProvider.GetService(typeof(IPluginExecutionContext));
    }
}
```

In custom workflow extensions, the `IPluginExecutionContext` is passed as a parameter of the `Execute` method of type `CodeActivityContext`.

```
public class SampleWorkflowExtension : CodeActivity
{
}
```

```
protected override void Execute(CodeActivityContext context)
{
    // Obtain the execution context from the code activity context.

    IWorkflowContext workflowContext = context.GetExtension();
}

}
```

Both *IPluginExecutionContext* and *IWorkflowContext* implement the *IExecutionContext* interface. Each also exposes information that is specific to their type. For more information, see **Understand the execution context**<sup>14</sup>.

## IExecutionContext properties

Two of the most remarkable properties of the *IExecutionContext* interface are *InputParameters* and *OutputParameters*. Other frequently used properties are *PreEntityImages*, *PostEntityImages*, and *SharedVariables*.

### InputParameters

Input parameters are contained in the execution context's *InputParameters* collection and are of type *Entity*. This property enables you to see what entity values were presented to the plug-in prior to the implementation of a given operation.

For example, if you want to validate your data and prevent it from being saved if validation is not successful, the steps that you take would be similar to the following:

1. Register your plug-in to run on the PreValidation stage on Create and/or Update of the entity that you want to validate.
2. Validate data in your plug-in by reading the values from the *InputParameters* collection. On Create, you'll want to retrieve the Target collection.
3. Throw an *InvalidPluginExecutionException* if the provided data is not valid.

Or, perhaps you want to update values on your entity (such as stripping special characters from a phone number) before the data is saved. You might handle this situation with the following steps:

1. Register your plug-in to run on the PreOperation stage on Create and/or Update of the entity that you want to update.
2. Update the data in your plug-in by editing the values from the *InputParameters* collection. On Create, you'll want to retrieve the Target collection.

---

<sup>14</sup> <https://docs.microsoft.com/powerapps/developer/common-data-service/understand-the-data-context/?azure-portal=true>

## OutputParameters

Output parameters are contained in the execution context's OutputParameters collection and are of the type *Entity*. Output parameters are only provided after the given operation has occurred. As such, you'll only be able to use this collection when you are handling events in the PostOperation stage.

For example, if you want to change the values that are returned by the operation, you can modify them by conducting the following steps:

1. Register your plug-in to run on the PostOperation stage on Create and/or Update of the entity that you want to update.
2. Update the data in your plug-in by editing the values from the OutputParameters collection. Note that on Create, you'll want to retrieve the Target collection.

## PreEntityImages and PostEntityImages

When registering plug-ins to run against the event framework, you can provide an immutable copy, or snapshot, of the data. This is ideal, especially when you need to reference values of the data before or after an operation has occurred, for example, for logging or custom auditing purposes.

While you can retrieve entity values by using the Organization Context from within a plug-in by means of a Retrieve request, using entity images to conduct this task is much more efficient and highly advised.

## SharedVariables

Shared variables allow you to configure data that can be passed from a plug-in to a step that occurs later in the implementation pipeline. While we highly recommend that you configure plug-ins to be stateless and not have any dependencies on external events, there are sometimes exceptions to this rule.

One exception might be when you want to ensure that you don't have any "runaway" plug-ins in your environment, that is, they are activating under an infinite-recursion scenario. You can also use the execution context to determine the depth in which your plug-in is currently running, but under certain situations, it might be appropriate to manually track. For example, perhaps your plug-in is mistakenly starting Update logic within a plug-in that is configured to run on Update of the entity, which in turn is causing it to run infinitely. To troubleshoot this, you could build a pattern to count the number of times that this plug-in is able to run.

An example might be to configure a limit of five runs of a plug-in with the following steps:

1. In the plug-in, check if the RunCount shared variable exists, and if it doesn't, create one that is initialized to zero.
2. Check if the RunCount variable's value is greater than five, and if it is, throw an InvalidPluginExecutionException.
3. At the end of your plug-in, increment the RunCount variable by 1.

## [!NOTE]

This is only an extra safeguard against a bug that accidentally causes your plug-ins to overrun your resources. You should make sure that you are building your plug-ins in such a way that this scenario won't ever occur. Common Data Service also provides safeguards to prevent "runaway" plug-ins such as these, but their recovery methods are much more resource-intensive than this method and more difficult to diagnose.

## Exercise - Write your first plug-in

In this scenario, an organization needs to ensure that phone number data is entered in a consistent format. To achieve this objective, you will create a plug-in to run on create/update that strips out all non-numeric characters from a phone number before saving to Common Data Service. You will then create another plug-in that will run on Contact retrieve/retrievemultiple to reformat the phone number to include parentheses and dashes, if the data exists.

### Exercise 1: Create/Update plug-in

In this exercise, you will create a plug-in that will run on create and update. This plug-in will strip out all non-numeric characters from a phone number.

Each exercise consists of a scenario and learning objectives. The scenario describes the purpose of the exercises, while the objectives are listed and have bullet points.

## [!NOTE]

If you don't have the Dynamics 365 SDK tools, see [Download tools from NuGet<sup>15</sup>](#) to download.

#### Task 1: Create a plug-in

1. Start Visual Studio 2019.
2. Select **File > New > Project**.
3. Select **Class Library (.NET Framework)** and select **Next**.

---

<sup>15</sup> <https://docs.microsoft.com/dynamics365/customerengagement/on-premises/developer/download-tools-nuget>

## Create a new project

### Recent project templates

Filtering by: C# [Clear filter](#)

 Class Library (.NET Framework)	C#	 Class Library (.NET Standard)	A project for creating a class library that targets .NET Standard.
		C#    Android    iOS    Linux    macOS    Windows    Library	
 Class Library (Universal Windows)		 Class Library (.NET Framework)	A project for creating a managed class library (.dll) for Universal Windows Platform (UWP) apps.
		C#    Windows    Library    UWP	
 Class Library (.NET Core)		 Android Class Library (Xamarin)	A project for creating a class library that targets .NET Core.
		C#    Windows    Linux    macOS    Library	A Xamarin.Android class library project.
		C#    Android    Mobile	
 iOS Class Library (Xamarin)			

[Next](#)

- Enter **D365PackageProject** for **Project Name**, select a location to save the project, select **.NET Framework 4.6.2** for **Framework**, and then select **Create**.

### Configure your new project

Class Library (.NET Framework) C# Windows Library

Project name

Location



Solution

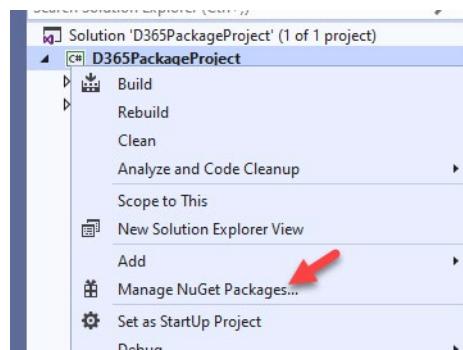
Solution name

Place solution and project in the same directory

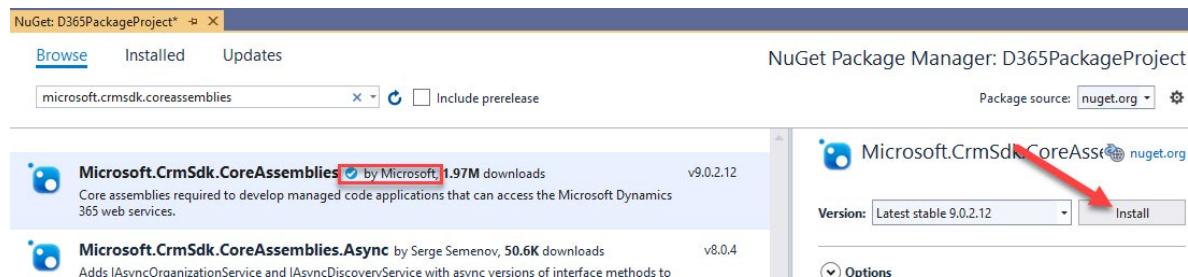
Framework

[Back](#) [Create](#)

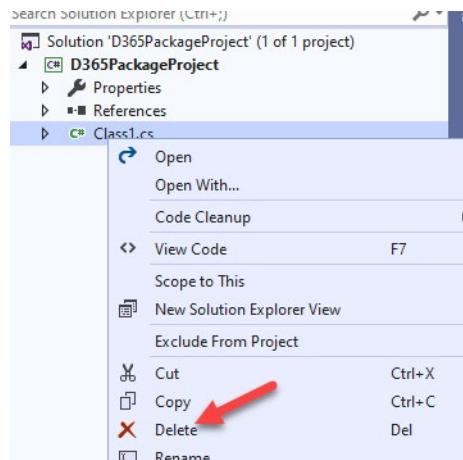
- Right-click the project and select **Manage NuGet Packages**.



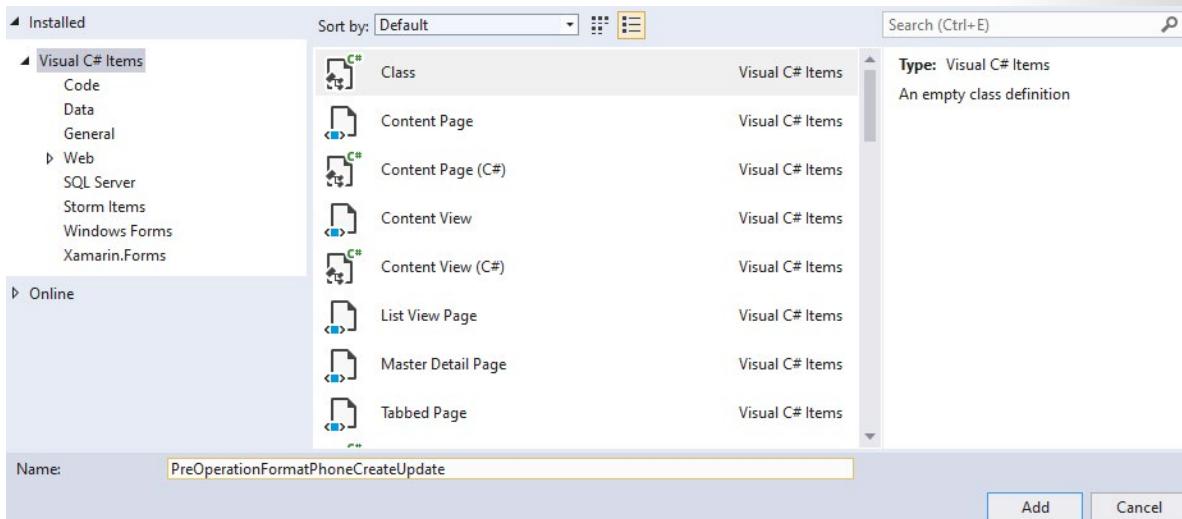
6. Select the **Browse** tab, search for and select **microsoft.crm.sdk.coreassemblies**, and then select **Install**.



7. Read the license terms and then select **Accept** if you agree.
8. Close the NuGet package manager.
9. Right-click **Class1.cs** and **Delete**.



10. Right-click the project and then select **Add > Class**.
11. Name the new class **PreOperationFormatPhoneCreateUpdate** and select **Add**.



12. Add the using statements to the new class as follows:

```
using Microsoft.Xrm.Sdk;

using System.Text.RegularExpressions;
```

13. Make the class public and implement the **IPlugin** interface.

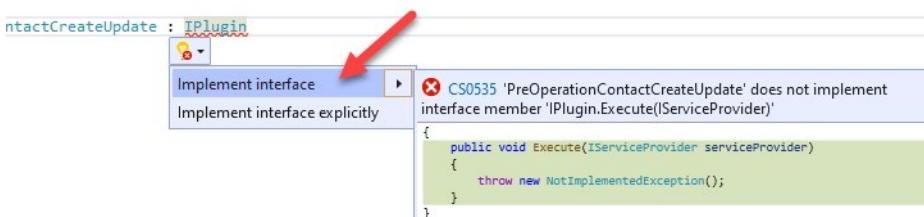
```
using System.Threading.Tasks;

using Microsoft.Xrm.Sdk;
using System.Text.RegularExpressions;

namespace D365PackageProject
{
    0 references
    public class PreOperationFormatPhoneCreateUpdate : IPlugin
    {

```

14. Implement the interface member.



Your class should now look like the following image.

```
namespace D365PackageProject
{
    public class PreOperationFormatPhoneCreateUpdate : IPlugin
    {
        public void Execute(IServiceProvider serviceProvider)
        {
            throw new NotImplementedException();
        }
    }
}
```

## Task 2: Format a phone number

1. Get the execution context from the service provider. Replace the exception in the Execute method with the following snippet.

```
IPluginExecutionContext context =
(IPluginExecutionContext)serviceProvider.GetService(typeof(IPluginExecutionContext));
```

```
public class PreOperationFormatPhoneCreateUpdate : IPlugin
{
    public void Execute(IServiceProvider serviceProvider)
    {
        IPluginExecutionContext context = (IPluginExecutionContext)serviceProvider.GetService(typeof(IPluginExecutionContext));
    }
}
```

2. Check the input parameter for Target. Add the following snippet to the Execute method.

```
if (!context.InputParameters.ContainsKey("Target"))

throw new InvalidPluginExecutionException("No target found");
```

3. Add the following snippet to the Execute method. This snippet will get the target entity from the input parameter and then check if its attributes contain telephone1 (Business Phone for Contacts, Phone for Accounts).

```
var entity = context.InputParameters["Target"] as Entity;

if (!entity.Attributes.Contains("telephone1"))

return;
```

4. Add the following snippet to the Execute function. This snippet will remove all nonnumeric characters from the user-provided phone number.

```
string phoneNumber = (string)entity["telephone1"];

var formattedNumber = Regex.Replace(phoneNumber, @"[^\\d]", "");
```

5. Set telephone1 to the formatted phone number. Add the following snippet to the Execute method.

```
entity["telephone1"] = formattedNumber;
```

The Execute method should now look like the following image.

```
0 references
public void Execute(IServiceProvider serviceProvider)
{
    IPluginExecutionContext context = (IPluginExecutionContext)serviceProvider.GetService(typeof(IPluginExecutionContext));

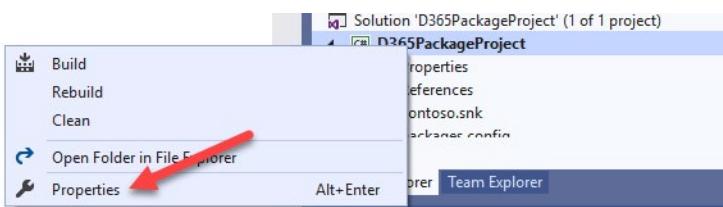
    if (!context.InputParameters.ContainsKey("Target"))
        throw new InvalidPluginExecutionException("No target found");

    var entity = context.InputParameters["Target"] as Entity;
    if (!entity.Attributes.Contains("telephone1"))
        return;

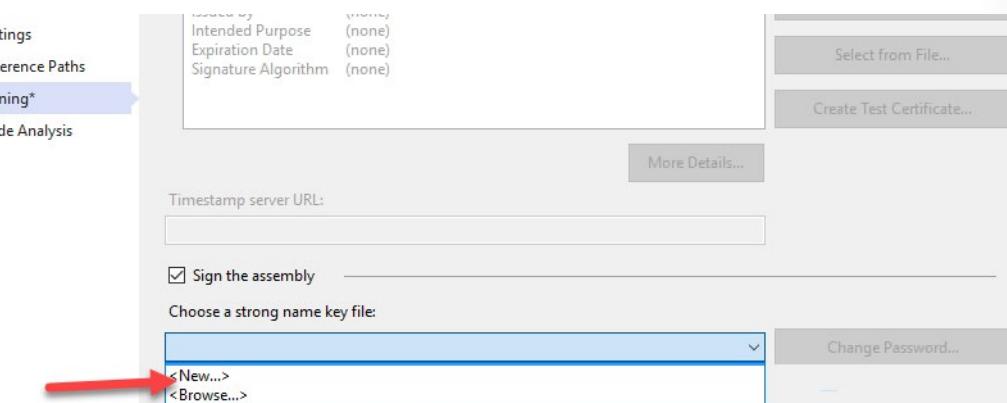
    string phoneNumber = (string)entity["telephone1"];
    var formattedNumber = Regex.Replace(phoneNumber, @"\D", "");

    entity["telephone1"] = formattedNumber;
}
```

6. Right-click the project and select **Properties**.



7. Select the **Signing** tab and select **New Key File**.



8. Enter **contoso.snk** in the **Key file name** field, clear the **Protect my key file with a password** check box, and then select **OK**.



9. Close the **Properties** tab.
10. Build the project and make sure that the build succeeds.

## Task 3: Register a plug-in and steps

[!NOTE]

If you don't have the Dynamics 365 SDK tools, see [Download tools from NuGet<sup>16</sup>](#) to download.

1. Start the plug-in registration tool that is in your Dynamics 365 SDK folder.

PluginProfiler.Library.dll	4/24/2018 6:30 PM	Application
PluginProfiler.Plugins.dll	4/24/2018 6:30 PM	Application
PluginProfiler.Solution.zip	4/24/2018 6:30 PM	Compressed (
PluginRegistration.exe	4/24/2018 6:30 PM	Application
PluginRegistration.exe.config	4/24/2018 6:30 PM	XML Configur
ServiceEndpointRegistration.dll	4/24/2018 6:30 PM	Application

2. Select **Create New Connection**.



3. Select **Microsoft 365**, select the **Show Advanced** check box, provide your credentials, and then select **Login**.

<sup>16</sup> <https://docs.microsoft.com/dynamics365/customerengagement/on-premises/developer/download-tools-nuget>

## Login

Deployment Type:  On-premises  Office 365

Display list of available organizations

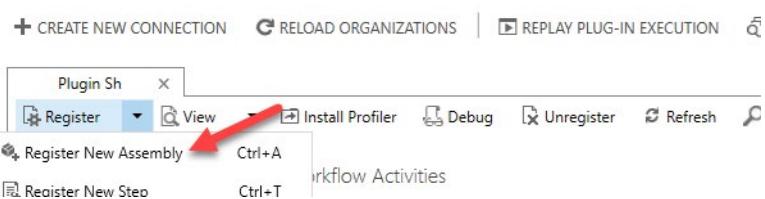
Show Advanced

Advanced

Online Region	Don't Know
User Name	admin@pluginsh.onmicrosoft.com
Password	*****

**Login** **Cancel**

4. Select **Register** and then select **Register New Assembly**.



5. Select **Browse**.

## Register New Assembly

Step 1: Specify the location of the assembly to analyze

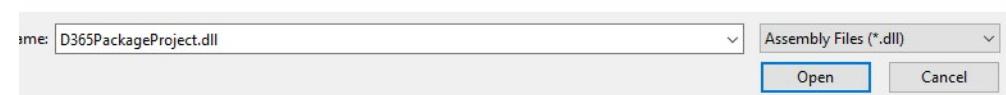
**Load Assembly**

Step 2: Select the plugin and workflow activities to register

Select All / Deselect All

6. Browse to the **Bin > Debug** folder of the class library that you created, select **D365PackageProject.dll**, and then select **Open**.

Name	Date modified	Type	Size
D365PackageProject.dll	6/15/2019 10:30 PM	Application extens...	5 KB
Microsoft.Crm.Sdk.Proxy.dll	2/14/2019 1:41 AM	Application extens...	275 KB
Microsoft.Xrm.Sdk.dll	2/14/2019 1:48 AM	Application extens...	537 KB



7. Select **Register Selected Plugins**.

## Register New Assembly

Step 1: Specify the location of the assembly to analyze

 ...  

Step 2: Select the plugin and workflow activities to register

(Assembly) D365PackageProject

(Plugin) D365PackageProject.PreOperationFormatPhoneCreateUpdate - Isolatable

Select All / Deselect All

Step 3: Specify the isolation mode

Sandbox

None

Step 4: Specify the location where the assembly should be stored

Database

Disk

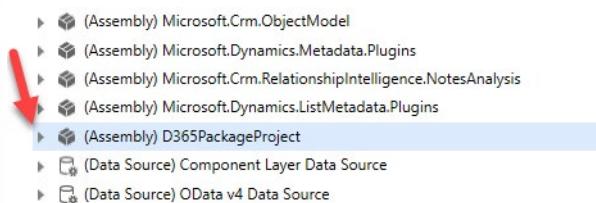
GAC

Step 5: Log

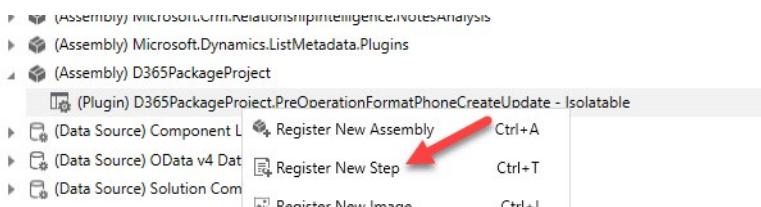
### 8. Select OK.



### 9. Expand the newly registered assembly.



### 10. Right-click the plug-in and select **Register New Step**.



11. Select **Create** for **Message** and select **contact** for **Primary Entity**.

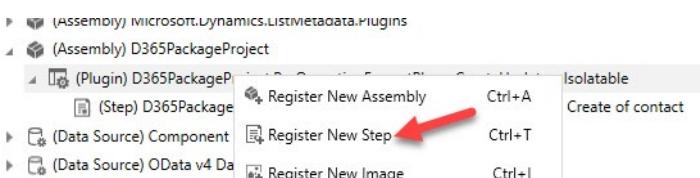
### Register New Step

General Configuration Information		Unsecure
Message	<input type="text" value="Create"/>	
Primary Entity	<input type="text" value="contact"/>	
Secondary Entity	<input type="text"/>	
Filtration Attributes	<small>Massive does not support Filtered Attributes</small>	

12. Select **PreOperation** for **Event Pipeline Stage of Execution** and then select **Register New Step**.

Run in User's Context: Calling User	Execution Order: 1	Secure Configuration
Description: D365PackageProject.PreOperationFormatPhoneCreateUpdate: Cr		
Event Pipeline Stage of Execution: <b>PreOperation</b>	Execution Mode: Asynchronous <input checked="" type="radio"/> Server <input type="checkbox"/> Offline	
<input type="checkbox"/> Delete AsyncOperation if StatusCode = Successful		<b>Register New Step</b> <input type="button" value="Close"/>

13. Right-click the plug-in and select **Register New Step** again.



14. Select **Update** for **Message**, select **contact** for **Primary Entity**, and then select the **Attributes** lookup.

## Register New Step

General Configuration Information

Message	Update
Primary Entity	contact
Secondary Entity	
Filtering Attributes	All Attributes <input type="button" value="..."/>
Event Handler	(Plugin) D365PackageProject.PreOperationFormatPhoneCreateUpd
Step Name	D365PackageProject.PreOperationFormatPhoneCreateUpdate: Up

Unsecure Configuration



15. Clear the **Select All** check box, select the **Business Phone** check box, and then select **OK**.

### Select Attributes

Select All / Deselect All

Name	Type	Type
<input type="checkbox"/> Back Office Customer	isbackofficecustomer	Boolean
<input type="checkbox"/> Birthday	birthdate	DateTime
<input checked="" type="checkbox"/> Business Phone	telephone1	String
<input type="checkbox"/> Business Phone ?	business?	String



16. Select **PreOperation** for **Event Pipeline Stage of Execution** and then select **Register New Step**.

Step Name: D365PackageProject.PreOperationFormatPhoneCreateUpdate: Up

Run in User's Context: Calling User

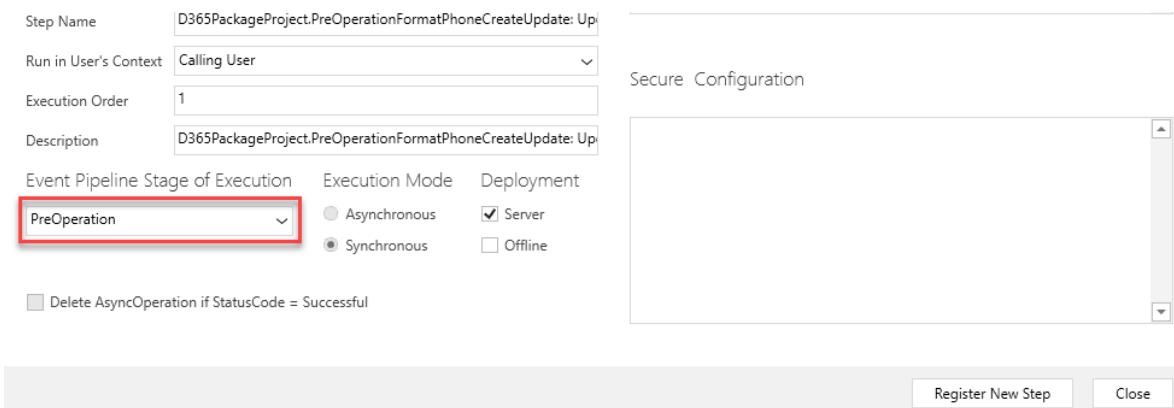
Execution Order: 1

Description: D365PackageProject.PreOperationFormatPhoneCreateUpdate: Up

Event Pipeline Stage of Execution:   Delete AsyncOperation if StatusCode = Successful

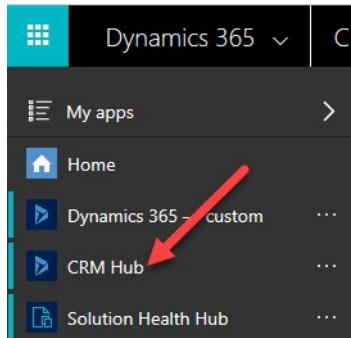
Execution Mode:  Asynchronous  Server  Synchronous  Offline

Secure Configuration



## Task 4: Test a plug-in

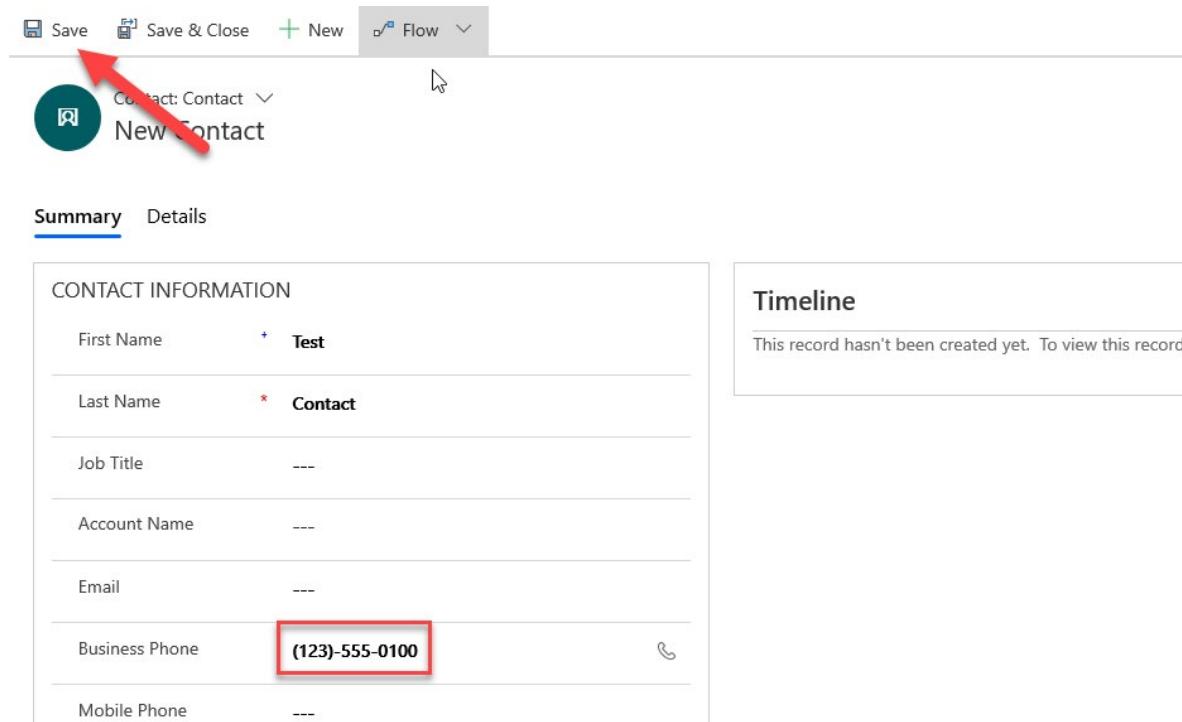
1. Go to your **Dynamics 365** folder and switch to your **CRM Hub** application.



2. Select **Contacts** and then select **+ New**.

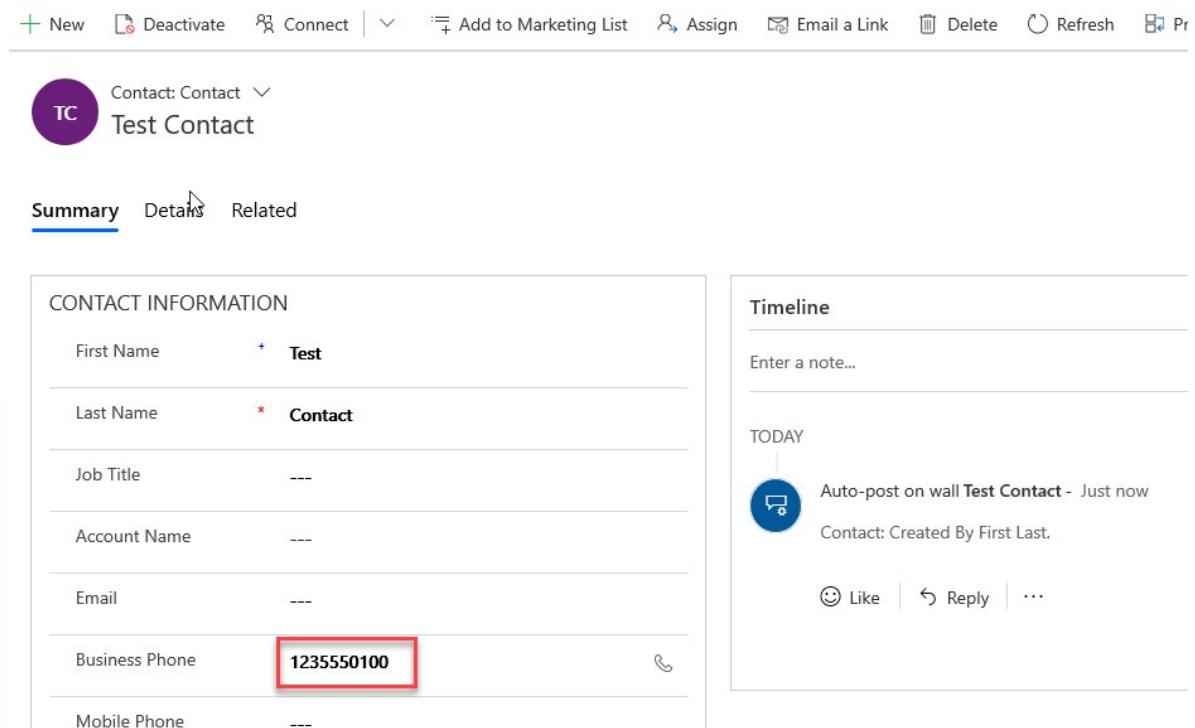
A screenshot of the Dynamics 365 Contacts list view. On the left is a sidebar with sections like "Home", "Recent", "Pinned", "My Work", "Customers", "Accounts", and "Contacts" (which is highlighted). The main area shows a title "My Active Contacts" with a red arrow pointing to the "+ New" button in the top toolbar. Below the title is a table with columns "Full Name" and "Email". The table contains five rows of sample data: Jim Glynn (sample) with email someone\_j@e...; Maria Campbell (sample) with email someone\_d@e...; Nancy Anderson (sample) with email someone\_c@e...; and Patrick Sands (sample) with email someone\_k@e....

3. Enter **Test** for **First Name**, **Contact** for **Last Name**, **(123)-555-0100** for **Business Phone**, and then select **Save**.



The screenshot shows the Microsoft Dynamics 365 Contact form. At the top, there are several buttons: 'Save' (highlighted with a red arrow), 'Save & Close', '+ New', 'Flow', and a dropdown menu. Below the buttons, the contact record is identified as 'Contact: Contact' and 'New Contact'. The main area is divided into sections: 'CONTACT INFORMATION' and 'Timeline'. In the 'CONTACT INFORMATION' section, the 'Business Phone' field contains '(123)-555-0100', which is highlighted with a red box. The 'Timeline' section indicates that the record hasn't been created yet.

The record should be saved, and the **Business Phone** should show only the numeric values.



The screenshot shows the Microsoft Dynamics 365 Contact form for the same record ('Test Contact'). The 'Business Phone' field now displays the numeric value '1235550100', also highlighted with a red box. The 'Timeline' section shows a note: 'Auto-post on wall Test Contact - Just now' and 'Contact: Created By First Last.' Below the timeline, there are 'Like', 'Reply', and '...' buttons.

4. Change the **Business Phone** to **001-123-555-0100** and wait for a few seconds.

The screenshot shows a contact record for 'Test Contact'. The contact information includes:

- First Name: Test
- Last Name: Contact
- Job Title: ---
- Account Name: ---
- Email: ---
- Business Phone: **001-123-555-0100** (highlighted with a red box)
- Mobile Phone: ---

The Timeline on the right shows an auto-post from the contact's wall: "Auto-post on wall Test Contact - Just now Contact: Created By First Last." There are Like, Reply, and More options below the post.

The record should be updated, and the **Business Phone** should show only the numeric values.

The contact information has been updated:

- First Name: Test
- Last Name: Contact
- Job Title: ---
- Account Name: ---
- Email: ---
- Business Phone: **0011235550100** (highlighted with a red box)
- Mobile Phone: ---

The Timeline on the right shows the same auto-post from the contact's wall: "Auto-post on wall Test Contact - Just now Contact: Created By First Last." There are Like, Reply, and More options below the post.

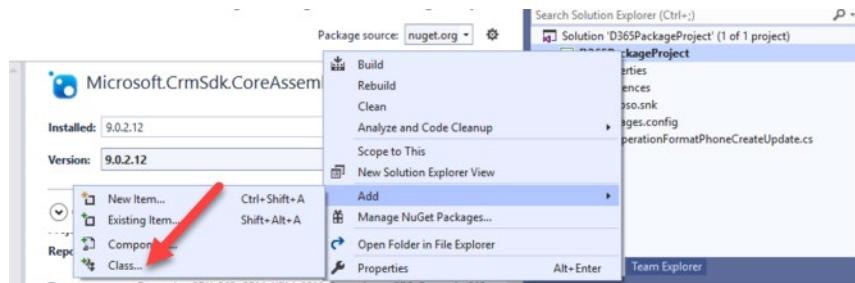
## Exercise 2: Create/Retrieve multiple plug-ins

In this exercise, you will create a plug-in that will run on retrieve and retrieve multiple. This plug-in will add parentheses and dashes to the phone numbers.

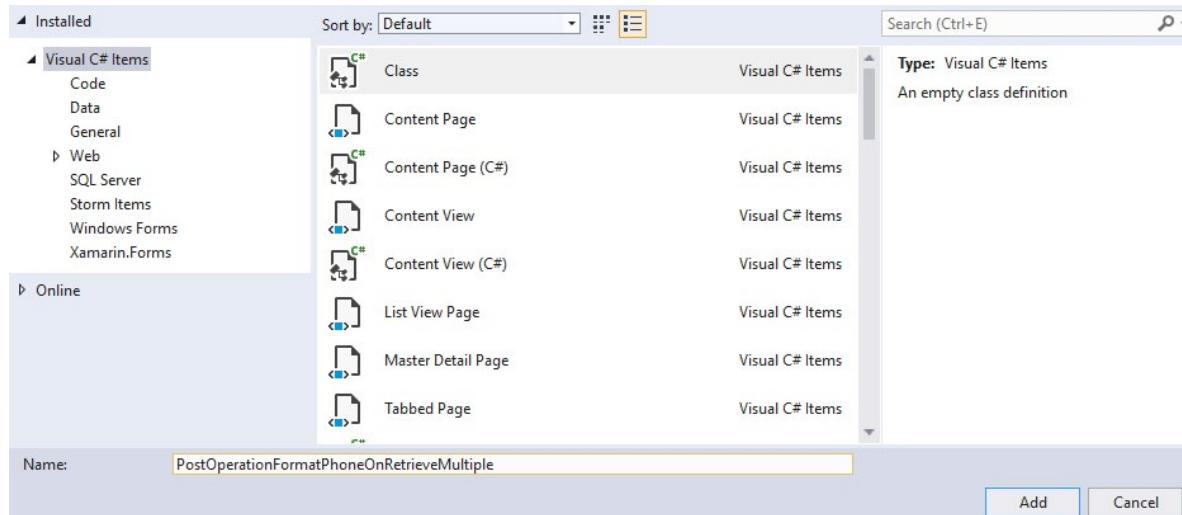
Each exercise consists of a scenario and learning objectives. The scenario describes the purpose of the exercises, while the objectives are listed and have bullet points.

### Task 1: Create a plug-in

1. Start Visual Studio.
2. Open the project that you created in Exercise 1.
3. Right-click the project and select **Add > Class**.



4. In the **Name** field, enter **PostOperationFormatPhoneOnRetrieveMultiple** for the class and then select **Add**.



5. Add the using statements to the new class as follows:

using Microsoft.Xrm.Sdk;

6. Make the class public and implement the **IPlugin** interface.

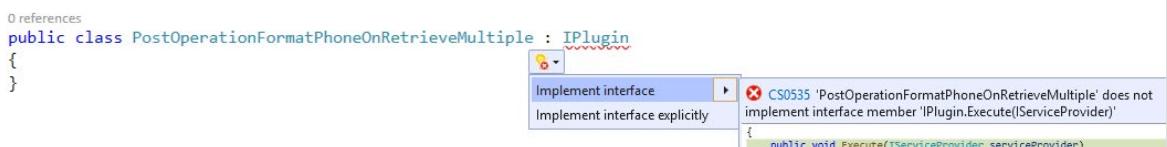
```

using Microsoft.Xrm.Sdk;

namespace D365PackageProject
{
    0 references
    public class PostOperationFormatPhoneOnRetrieveMultiple : IPlugin
    {
        {
    }
}

```

7. Implement the interface member.



Your class should now look like the following image.

```

namespace D365PackageProject
{
    0 references
    public class PostOperationFormatPhoneOnRetrieveMultiple : IPlugin
    {
        1 reference
        public void Execute(IServiceProvider serviceProvider)
        {
            throw new NotImplementedException();
        }
    }
}

```

## Task 2: Format phone number for retrieve

1. Get the execution context from the service provider. Replace the exception in the Execute method with the following snippet.

```
IPluginExecutionContext context =
(IPluginExecutionContext)serviceProvider.GetService(typeof(IPluginExecutionContext));
```

```

0 references
public class PostOperationFormatPhoneOnRetrieveMultiple : IPlugin
{
    1 reference
    public void Execute(IServiceProvider serviceProvider)
    {
        IPluginExecutionContext context = (IPluginExecutionContext)serviceProvider.GetService(typeof(IPluginExecutionContext));
    }
}

```

2. Check if the message name is Retrieve or RetrieveMultiple. Add the following snippet to the Execute method.

```

if (context.MessageName.Equals("Retrieve"))

{
}

```

```
else if(context.MessageName.Equals("RetrieveMultiple"))

{



}
```

3. Check if the output parameters contain a business entity and that it is a type of entity. Paste the following snippet inside the `if retrieve` condition.

```
if (!context.OutputParameters.Contains("BusinessEntity") && context.OutputParameters["BusinessEntity"] is Entity)
throw new InvalidPluginExecutionException("No business entity found");
```

4. Get the entity and check if it contains telephone1 field. Add the following snippet inside the `if retrieve` condition.

```
var entity = (Entity)context.OutputParameters["BusinessEntity"]

if (!entity.Attributes.Contains("telephone1"))

return;
```

5. Add the following snippet to the `if retrieve` condition. This snippet will try to parse telephone1 as `long` and will return if the parsing doesn't succeed.

```
if (!long.TryParse(entity["telephone1"].ToString(), out long phoneNumber))

return;
```

```
1 reference
public void Execute(IServiceProvider serviceProvider)
{
    IPluginExecutionContext context = (IPluginExecutionContext)serviceProvider.GetService(typeof(IPluginExecutionContext));

    if (context.MessageName.Equals("Retrieve"))
    {
        if (!context.OutputParameters.Contains("BusinessEntity") && context.OutputParameters["BusinessEntity"] is Entity)
            throw new InvalidPluginExecutionException("No business entity found");

        var entity = (Entity)context.OutputParameters["BusinessEntity"];
        if (!entity.Attributes.Contains("telephone1"))
            return;

        if (!long.TryParse(entity["telephone1"].ToString(), out long phoneNumber))
            return;
    }
    else if (context.MessageName.Equals("RetrieveMultiple"))...
```

6. Format the phone number by adding parentheses and dashes. Add the following snippet inside the `if retrieve` condition.

```
var formattedNumber = String.Format("{0:(###) ###-####}",

phoneNumber);
```

```
entity["telephone1"] = formattedNumber;
```

The retrieve part of the Execute method should now look like the following image.

```
1 reference
public void Execute(IServiceProvider serviceProvider)
{
    IPluginExecutionContext context = (IPluginExecutionContext)serviceProvider.GetService(typeof(IPluginExecutionContext));

    if (context.MessageName.Equals("Retrieve"))
    {
        if (!context.OutputParameters.Contains("BusinessEntity") && context.OutputParameters["BusinessEntity"] is Entity)
            throw new InvalidPluginExecutionException("No business entity found");

        var entity = (Entity)context.OutputParameters["BusinessEntity"];
        if (!entity.Attributes.Contains("telephone1"))
            return;

        if (!long.TryParse(entity["telephone1"].ToString(), out long phoneNumber))
            return;

        var formattedNumber = String.Format("{0:(###) ###-####}", phoneNumber);
        entity["telephone1"] = formattedNumber;
    }
    else if (context.MessageName.Equals("RetrieveMultiple")) ...
}
```

## Task 3: Format phone number for retrieve multiple

- Add the following snippet to the inside of the `retrieve multiple` condition.

This snippet will check if the output parameters contain `BusinessEntityCollection` and if it is of the type `EntityCollection`.

```
if (!(context.OutputParameters.Contains("BusinessEntityCollection") &&
    context.OutputParameters["BusinessEntityCollection"] is
    EntityCollection))
```

```
throw new InvalidPluginExecutionException("No business entity
collection found");
```

```
    entity["telephone1"] = formattedNumber;
}
else if(context.MessageName.Equals("RetrieveMultiple"))
{
    if (!context.OutputParameters.Contains("BusinessEntityCollection") && context.OutputParameters["BusinessEntityCollection"] is EntityCollection)
        throw new InvalidPluginExecutionException("No business entity collection found");
}
```

- Get the entity collection from the output parameters. Add the following snippet inside the `retrieve multiple` condition.

```
var entityCollection =
(EntityCollection)context.OutputParameters["BusinessEntityCollection"];
```

- Iterate through each entity in the entity collection.

```
foreach (var entity in entityCollection.Entities)
```

```
{  
}  
  
}  
else if (context.MessageName.Equals("RetrieveMultiple"))  
{  
    if (!context.OutputParameters.Contains("BusinessEntityCollection") && context.OutputParameters["BusinessEntityCollection"] is EntityCollection)  
        throw new InvalidPluginExecutionException("No business entity collection found");  
  
    var entityCollection = (EntityCollection)context.OutputParameters["BusinessEntityCollection"];  
  
    foreach (var entity in entityCollection.Entities)  
    {  
    }  
}
```

4. Add the following snippet inside the `foreach` condition. This snippet will do the same thing that the `retrieve` condition is doing.

```
if (!entity.Attributes.Contains("telephone1"))  
  
return;  
  
if (long.TryParse(entity["telephone1"].ToString(), out long  
phoneNumber))  
  
{  
  
var formattedNumber = String.Format("{0:(###) ###-####}",  
phoneNumber);  
  
entity["telephone1"] = formattedNumber;  
  
}
```

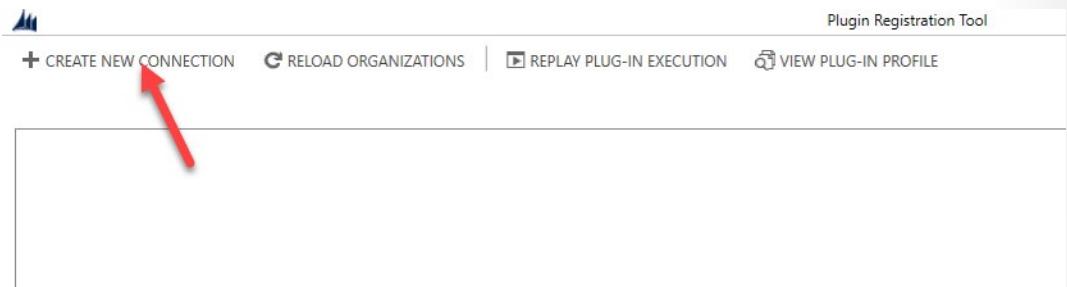
The `retrieve multiple` part of the `Execute` method should now look like the following image.

```
}  
else if (context.MessageName.Equals("RetrieveMultiple"))  
{  
    if (!context.OutputParameters.Contains("BusinessEntityCollection") && context.OutputParameters["BusinessEntityCollection"] is EntityCollection)  
        throw new InvalidPluginExecutionException("No business entity collection found");  
  
    var entityCollection = (EntityCollection)context.OutputParameters["BusinessEntityCollection"];  
  
    foreach (var entity in entityCollection.Entities)  
    {  
        if (!entity.Attributes.Contains("telephone1"))  
            return;  
  
        if (long.TryParse(entity["telephone1"].ToString(), out long phoneNumber))  
        {  
            var formattedNumber = String.Format("{0:(###) ###-####}", phoneNumber);  
            entity["telephone1"] = formattedNumber;  
        }  
    }  
}
```

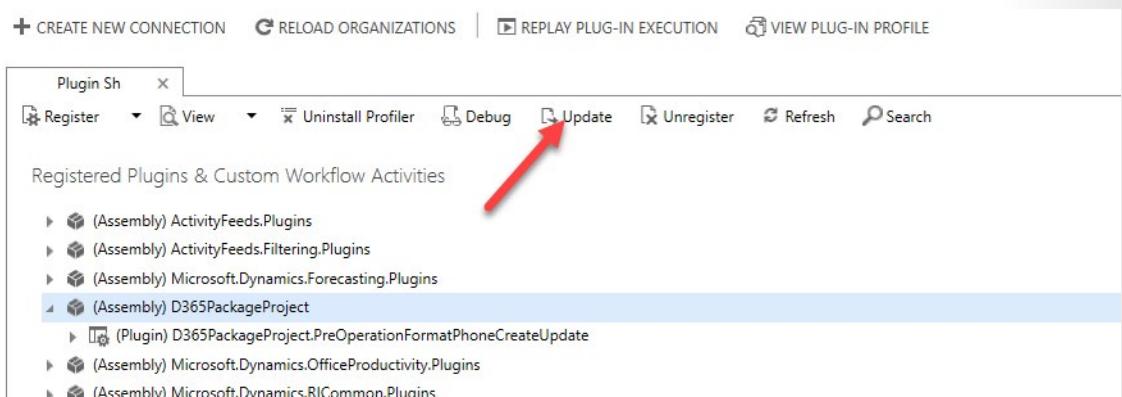
5. Rebuild the project and make sure that the build succeeds.

## Task 4: Update plug-in assembly and register steps

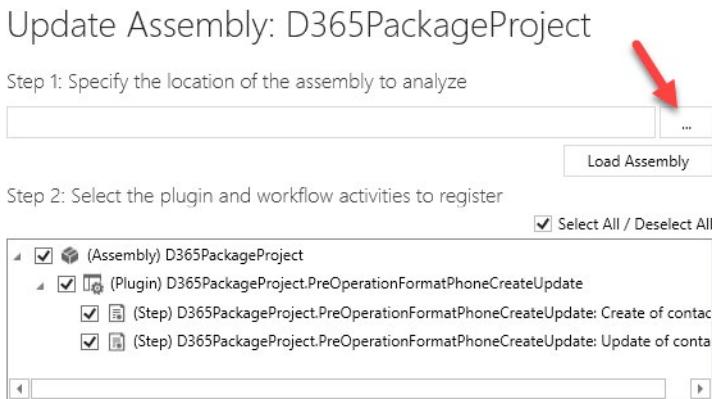
- Start the plug-in registration tool and select **Create New Connection**.



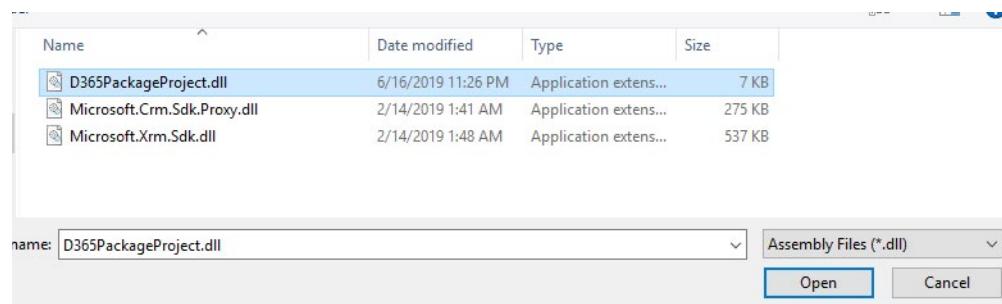
- Select **Microsoft 365**, provide your credentials, and then select **Login**.
- Select the assembly that you registered in Exercise 1 and then select **Update**.



- Select **Browse**.



- Browse to the debug folder of your project, select **D365PackageProject.dll**, and then select **Open**.



6. Select the plug-ins and then select **Update Selected Plugins**.

### Update Assembly: D365PackageProject

Step 1: Specify the location of the assembly to analyze

C:\D365PackageProject\bin\Debug\| ...

 Update Assembly Properties and Content 

Step 2: Select the plugin and workflow activities to register

Select All / Deselect All

<input checked="" type="checkbox"/> (Assembly) D365PackageProject	<input checked="" type="checkbox"/> (Plugin) D365PackageProject.PostOperationFormatPhoneOnRetrieveMultiple - Isolatable
<input checked="" type="checkbox"/> (Plugin) D365PackageProject.PreOperationFormatPhoneCreateUpdate - Isolatable	

Step 3: Specify the isolation mode

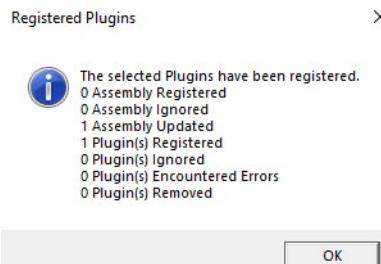
Sandbox ?  
 None

Step 4: Specify the location where the assembly should be stored

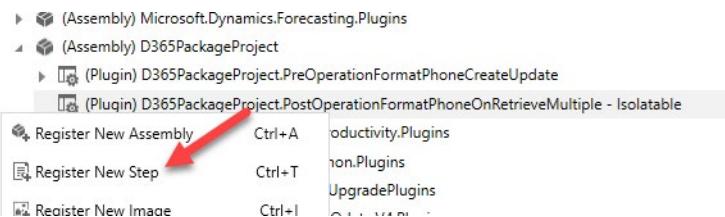
Database ?  
 Disk ?  
 GAC ?

Step 5: Log

7. Select **OK**.



8. Right-click the new plug-in and select **Register New Step**.



9. Select **Retrieve** for **Message**, select **contact** for **Primary Entity**, and then select **PostOperation** for **Event Pipeline Stage of Execution**. Make sure that you have **Synchronous** selected for **Execution Mode** and then select **Register New Step**.

### Register New Step

**General Configuration Information**

Message	Retrieve
Primary Entity	contact
Secondary Entity	
Filtering Attributes	Message does not support Filtered Attributes
Event Handler	(Plugin) D365PackageProject.PostOperationFormatPhoneOnR...
Step Name	D365PackageProject.PostOperationFormatPhoneOnRetrieveMultiple
Run in User's Context	Calling User
Execution Order	1
Description	D365PackageProject.PostOperationFormatPhoneOnRetrieveMultiple

**Event Pipeline Stage of Execution**: PostOperation (highlighted with a red box)

**Execution Mode**: Synchronous (highlighted with a red box)

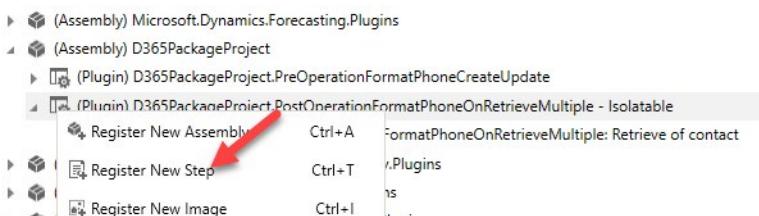
**Deployment**: Asynchronous (unchecked), Server (checked), Offline (unchecked)

Delete AsyncOperation if StatusCode = Successful

**Unsecure Configuration** and **Secure Configuration** sections are empty.

Buttons at the bottom: Register New Step (highlighted with a red box), Close

10. Right-click the plug-in and select **Register New Step** again.



11. Select **RetrieveMultiple** for **Message**, select **contact** for **Primary Entity**, and then select **PostOperation** for **Event Pipeline Stage of Execution**. Make sure that you have **Synchronous** for **Execution Mode** selected, and then select **Register New Step**.

## Register New Step

General Configuration Information

Message	RetrieveMultiple
Primary Entity	contact
Secondary Entity	
Filtering Attributes	Message does not support Filtered Attributes
Event Handler	(Plugin) D365PackageProject.PostOperationFormatPhoneOnR
Step Name	D365PackageProject.PostOperationFormatPhoneOnRetrieveMultipl
Run in User's Context	Calling User
Execution Order	1
Description	D365PackageProject.PostOperationFormatPhoneOnRetrieveMultipl

Unsecure Configuration

Secure Configuration

Event Pipeline Stage of Execution      Execution Mode      Deployment

PostOperation	<input type="radio"/> Asynchronous	<input checked="" type="radio"/> Server
	<input checked="" type="radio"/> Synchronous	<input type="checkbox"/> Offline

Delete AsyncOperation if StatusCode = Successful

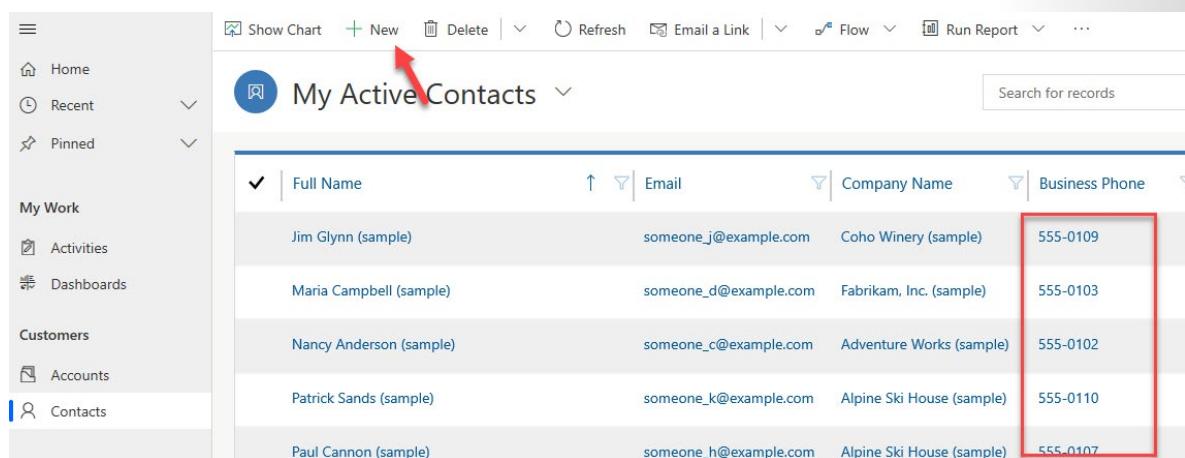
Register New Step Close

## Task 5: Test the plug-in

1. Go to your **CRM Hub** application and select **Contacts**.

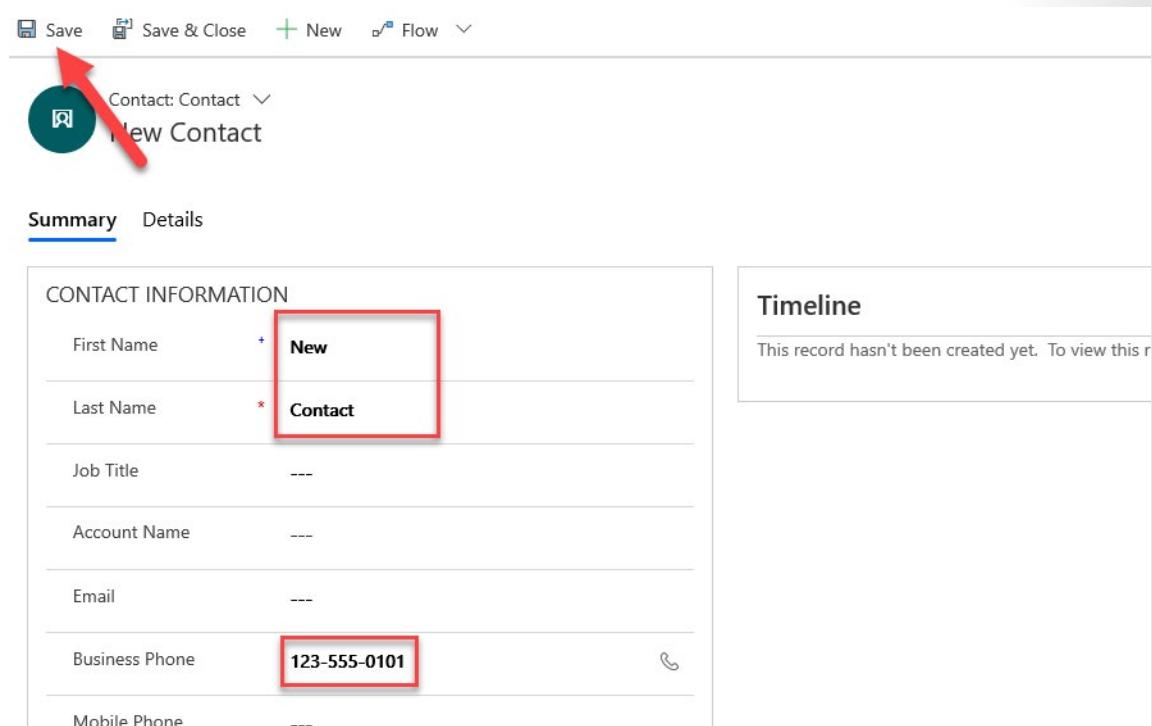
The screenshot shows the Microsoft Dynamics 365 CRM Hub interface. On the left, there is a navigation bar with links for Home, Recent, Pinned, My Work, Activities, Dashboards, Customers, Accounts, and Contacts. The Contacts link is highlighted with a blue bar. The main area is titled "My Active Contacts". It displays a list of five contacts: Jim Glynn (sample), Maria Campbell (sample), Nancy Anderson (sample), Patrick Sands (sample), and Paul Cannon (sample). Each contact entry includes their full name and an email address placeholder (someemail@sample.com). The top of the page has standard CRM navigation buttons for Show Chart, New, Delete, Refresh, Email a Link, and Flow.

2. The existing records that were not saved with the new format will not change because of the dashes. Select **New**.



Full Name	Email	Company Name	Business Phone
Jim Glynn (sample)	someone_j@example.com	Coho Winery (sample)	555-0109
Maria Campbell (sample)	someone_d@example.com	Fabrikam, Inc. (sample)	555-0103
Nancy Anderson (sample)	someone_c@example.com	Adventure Works (sample)	555-0102
Patrick Sands (sample)	someone_k@example.com	Alpine Ski House (sample)	555-0110
Paul Cannon (sample)	someone_h@example.com	Alpine Ski House (sample)	555-0107

3. Provide the following information and then select **Save**.



Contact: Contact ▾

New Contact

Summary Details

CONTACT INFORMATION

First Name	* <b>New</b>
Last Name	* <b>Contact</b>
Job Title	---
Account Name	---
Email	---
Business Phone	<b>123-555-0101</b>
Mobile Phone	---

Timeline

This record hasn't been created yet. To view this record's timeline, click the Create button.

The record should be saved, and the plug-in should apply the new format.

The screenshot shows the Microsoft Dynamics 365 Contact form. At the top, there are several action buttons: New, Deactivate, Connect, Add to Marketing List, Assign, Email a Link, Delete, Refresh, and a Print icon. Below the buttons, a purple circular icon contains the letters 'NC'. To its right, the text 'Contact: Contact' and 'New Contact' is displayed. Underneath, there are three tabs: Summary (which is selected), Details, and Related. The main area is titled 'CONTACT INFORMATION' and contains the following fields:

First Name	<b>New</b>
Last Name	<b>Contact</b>
Job Title	---
Account Name	---
Email	---
Business Phone	<b>(123) 555-0101</b>
Mobile Phone	---

To the right of the form is a 'Timeline' sidebar. It has a header 'Timeline' and a note 'Enter a note...'. Below that is a section for 'TODAY' which includes a message from a user named 'First Last' about auto-posting on the wall. There are also 'Like', 'Reply', and 'More' buttons.

4. Select **Contacts** again.

The record that you created should have the new format.

The screenshot shows the Microsoft Dynamics 365 Contacts list. On the left is a navigation bar with Home, Recent, Pinned, My Work, Activities, Dashboards, Customers, Accounts, and Contacts. The Contacts item is selected. The main area is titled 'My Active Contacts' and shows a list of contacts with columns for Full Name, Email, Company Name, and Business Phone. A new contact record, 'New Contact', is listed with a business phone number of '(123) 555-0101', which is highlighted with a red box.

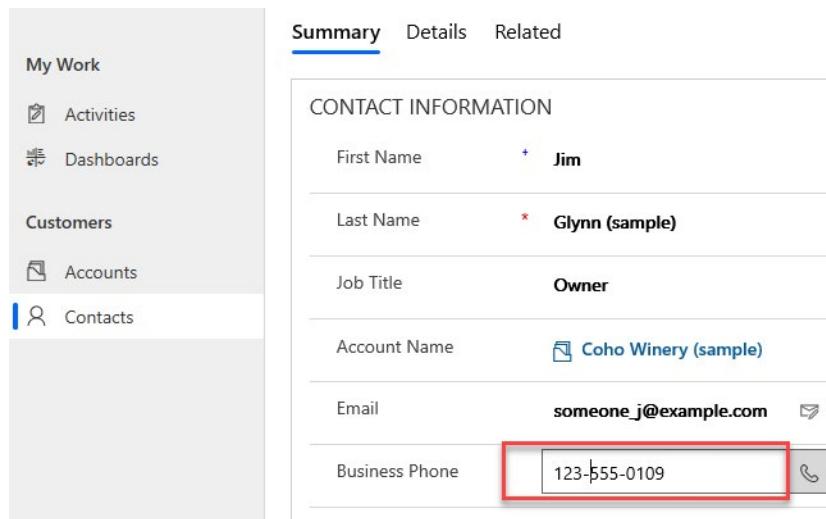
Full Name	Email	Company Name	Business Phone
Jim Glynn (sample)	someone_j@example.com	Coho Winery (sample)	(123) 555-0109
Maria Campbell (sample)	someone_d@example.com	Fabrikam, Inc. (sample)	555-0103
Nancy Anderson (sample)	someone_c@example.com	Adventure Works (sample)	555-0102
New Contact	---	---	<b>(123) 555-0101</b>
Patrick Sands (sample)	someone_k@example.com	Alpine Ski House (sample)	555-0110

5. Select one of the existing contacts to open it.

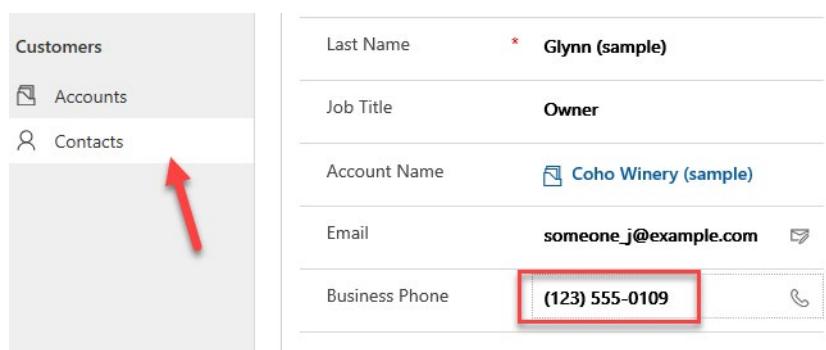
The screenshot shows the Microsoft Dynamics 365 Contacts list. The 'Contacts' item is selected in the navigation bar. The main area is titled 'My Active Contacts' and shows a list of contacts. One contact, 'Jim Glynn (sample)', is highlighted with a red arrow pointing to its name. The contact details are shown in a row: Full Name (Jim Glynn (sample)), Email (someone\_j@example.com), Company Name (Coho Winery (sample)), and Business Phone (555-0109), which is highlighted with a red box.

Full Name	Email	Company Name	Business Phone
Jim Glynn (sample)	someone_j@example.com	Coho Winery (sample)	<b>555-0109</b>
Maria Campbell (sample)	someone_d@example.com	Fabrikam, Inc. (sample)	555-0103
Nancy Anderson (sample)	someone_c@example.com	Adventure Works (sample)	555-0102

6. Edit the **Business Phone**, as follows:



7. Auto save should save the record and the new format should be applied. Select **Contacts** one more time.



The edited phone should get the new format.

My Active Contacts				Search for records
Full Name	Email	Company Name	Business Phone	
Jim Glynn (sample)	someone.j@example.com	Coho Winery (sample)	(123) 555-0109	
Maria Campbell (sample)	someone.d@example.com	Fabrikam, Inc. (sample)	555-0103	
Nancy Anderson (sample)	someone.c@example.com	Adventure Works (sample)	555-0102	
New Contact	---	---	(789) 789-7894	

## Exercise - Write a custom workflow extension

Frequently, performing date formatting on Common Data Service data is required. While Power Automate exposes some features around this process, it might occasionally be appropriate to produce custom

workflow extensions to achieve this objective (for example, if the logic needs to run synchronously within your environment).

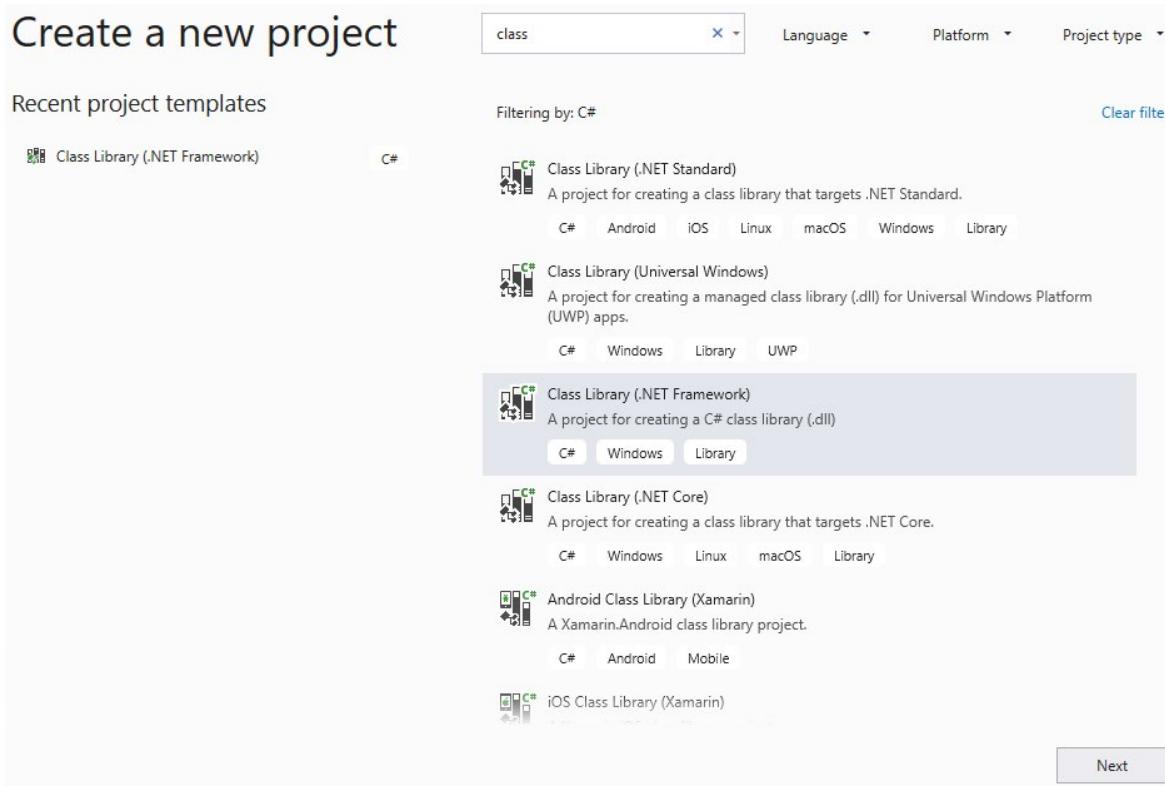
## Exercise 1: Create a custom activity

In this exercise, you will create a custom activity that will convert datetime to user local time, format it as “Monday, December 25, 2019,” and publish the formatted value as a string to the consumer of the custom activity.

Each exercise consists of a scenario and learning objectives. The scenario describes the purpose of the exercises, while the objectives are listed and have bullet points.

### Task 1: Create a custom activity

1. Start Visual Studio 2019.
2. Select **File > New > Project**.
3. Select **Class Library (.NET Framework)** and then select **Next**.



4. Enter **WorkflowActivities** in the **Project name** field, select **.NET Framework 4.6.2** in the **Framework** drop-down menu, and then select **Create**.

## Configure your new project

Class Library (.NET Framework) C# Windows Library

Project name

WorkflowActivities

Location

.Contoso Plugins\



Solution name ⓘ

WorkflowActivities

Place solution and project in the same directory

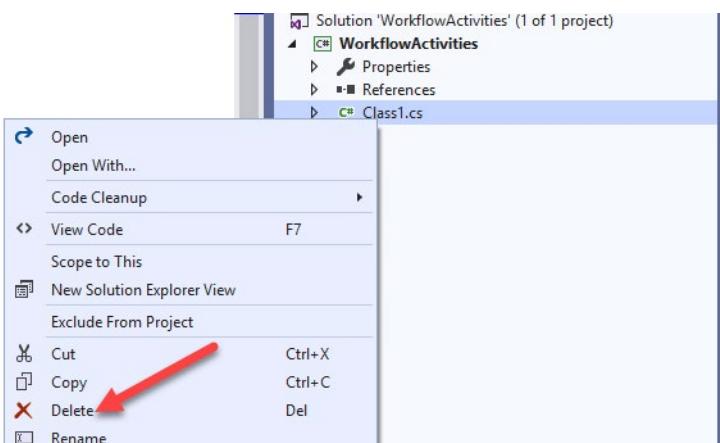
Framework

.NET Framework 4.7.1

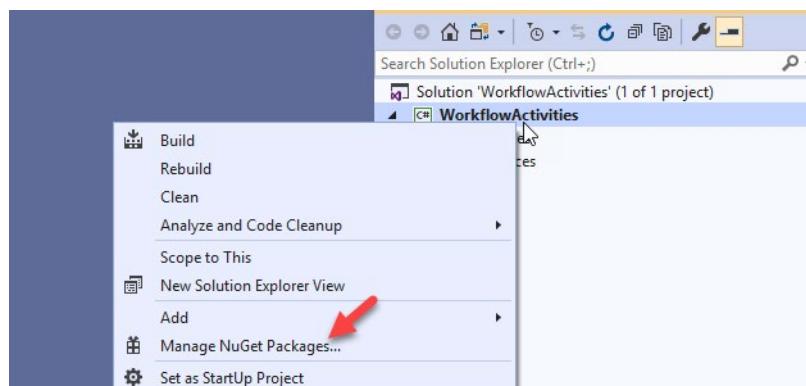
Back

Create

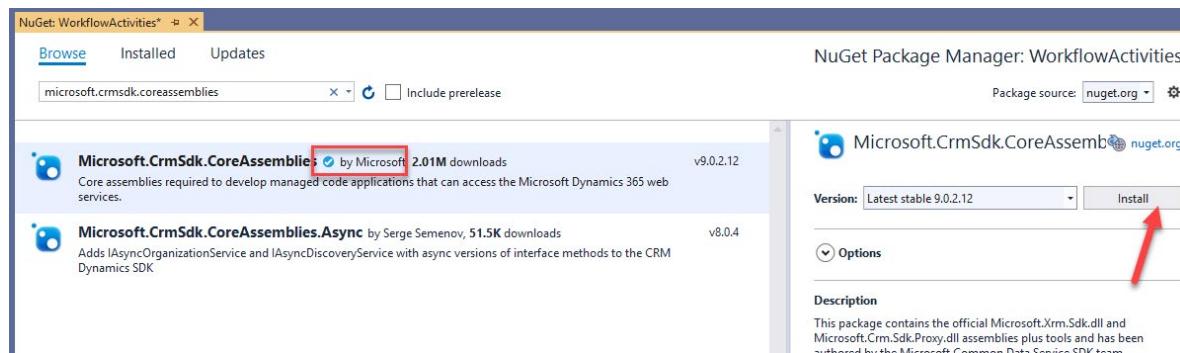
5. Right-click **Class1.cs** and then select **Delete**.



6. Right-click the project and select **Manage NuGet Packages**.



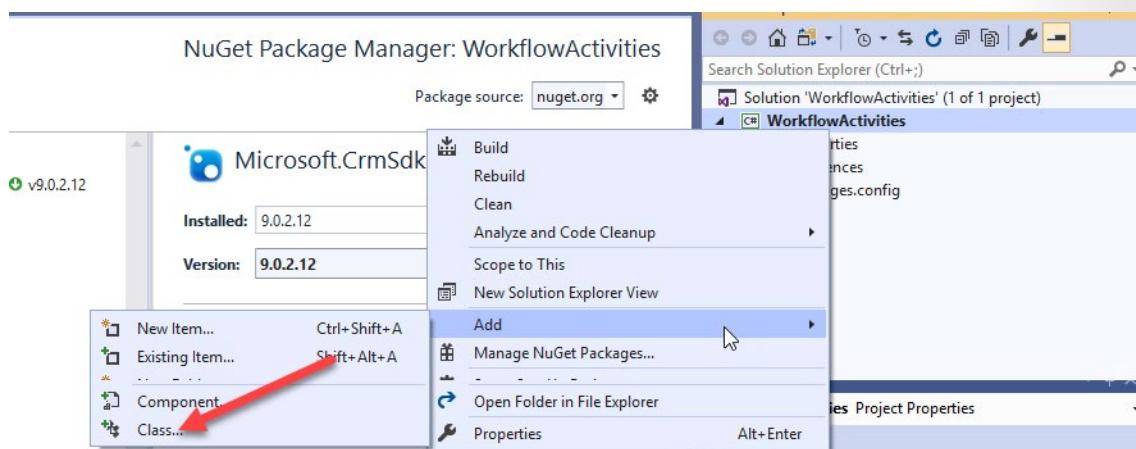
7. Select the **Browse** tab, search for and select **Microsoft.CRMSDK.CoreAssemblies**, and then select **Install**.



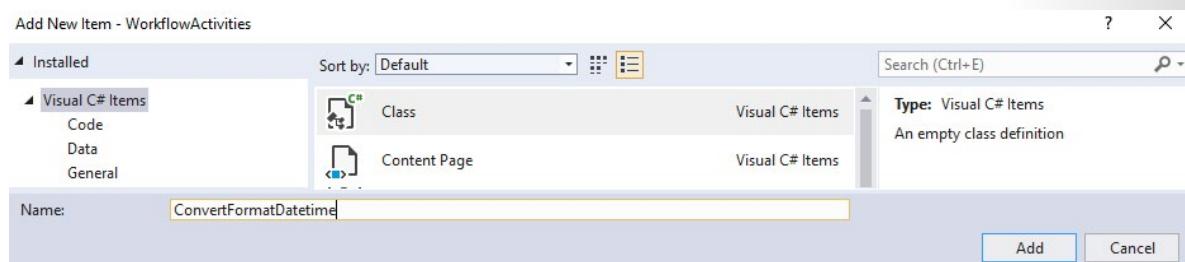
8. Select **I Accept** if you agree to the terms.
9. Search for and select **Microsoft.CRMSDK.Workflow** and then select **Install**.



10. Select **I Accept** if you agree to the terms.
11. Right-click the project, select **Add**, and then select **Class**.



12. Enter **ConvertFormatDatetime** in the **Name** field and then select **Add**.



13. Add the using statements to the class as follows and then make it public.

```
using Microsoft.Crm.Sdk.Messages;
```

```
using Microsoft.Xrm.Sdk;
```

```
using Microsoft.Xrm.Sdk.Query;
```

```
using Microsoft.Xrm.Sdk.Workflow;
```

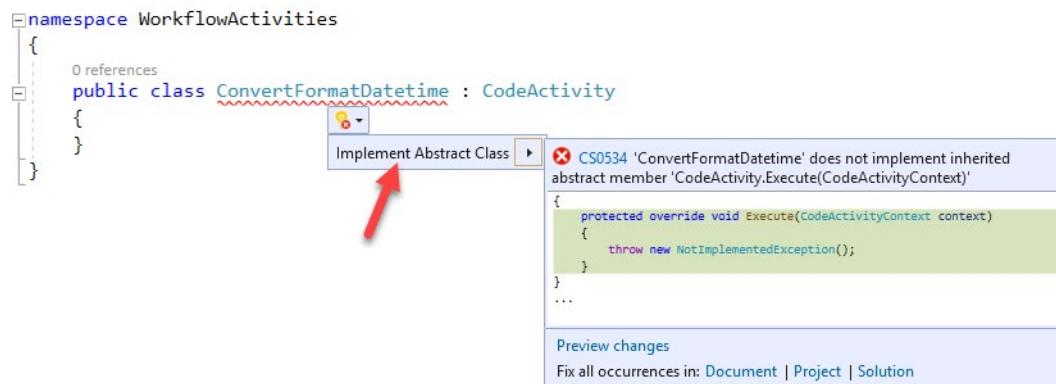
```
using System.Activities;
```

```
using System;
using Microsoft.Crm.Sdk.Messages;
using Microsoft.Xrm.Sdk;
using Microsoft.Xrm.Sdk.Query;
using Microsoft.Xrm.Sdk.Workflow;
using System.Activities;

namespace WorkflowActivities
{
    public class ConvertFormatDatetime
    {
    }
}
```

14. Inherit from **CodeActivity** and implement the abstract member.

This action will override the Execute method of the inherited class.



15. Add the following snippet to the class. This will add a required input argument of type DateTime and an output argument of type String.

```
[RequiredArgument]

[Input("DateTime input")]

public InArgument<DateTime> DateToEvaluate { get; set; }

[Output("Formatted DateTime output as string")]

public OutArgument<string> FormattedDateTimeOutput { get; set; }
```

```
0 references
public class ConvertFormatDatetime : CodeActivity
{
    [RequiredArgument]
    [Input("DateTime input")]
    0 references
    public InArgument<DateTime> DateToEvaluate { get; set; }

    [Output("Formatted DateTime output as string")]
    0 references
    public OutArgument<string> FormattedDateTimeOutput { get; set; }

    0 references
    protected override void Execute(CodeActivityContext context)
    {
        throw new NotImplementedException();
    }
}
```

16. Replace NotImplementedException with the following snippet.

```
IWorkflowContext workflowContext =
context.GetExtension<IWorkflowContext>();

IOrganizationServiceFactory serviceFactory =
context.GetExtension<IOrganizationServiceFactory>();
```

```
IOrganizationService service =
serviceFactory.CreateOrganizationService(workflowContext.UserId);
```

```
0 references
protected override void Execute(CodeActivityContext context)
{
    IWorkflowContext workflowContext = context.GetExtension<IWorkflowContext>();
    IOrganizationServiceFactory serviceFactory = context.GetExtension<IOrganizationServiceFactory>();
    IOrganizationService service = serviceFactory.CreateOrganizationService(workflowContext.UserId);
}
```

17. Get the input datetime. Add the following snippet to the Execute method.

```
DateTime utcDateTime = this.DateToEvaluate.Get(context);
```

18. Check if the input datetime is **UTC**, if not, convert it to **UTC**. Add the following snippet to the Execute method.

```
if (utcDateTime.Kind != DateTimeKind.Utc)
```

```
{
```

```
utcDateTime = utcDateTime.ToUniversalTime();
```

```
}
```

Your Execute method should now look like the following image.

```
0 references
protected override void Execute(CodeActivityContext context)
{
    IWorkflowContext workflowContext = context.GetExtension<IWorkflowContext>();
    IOrganizationServiceFactory serviceFactory = context.GetExtension<IOrganizationServiceFactory>();
    IOrganizationService service = serviceFactory.CreateOrganizationService(workflowContext.UserId);

    DateTime utcDateTime = this.DateToEvaluate.Get(context);

    if (utcDateTime.Kind != DateTimeKind.Utc)
    {
        ... utcDateTime = utcDateTime.ToUniversalTime();
    }
}
```

## Task 2: Get user settings and convert datetime to user local datetime

- Get the user TimeZoneCode of the user settings entity. Add the following snippet to the Execute method.

```
var settings = service.Retrieve("usersettings",
workflowContext.UserId, new ColumnSet("timezonecode") );
```

```
if (utcDateTime.Kind != DateTimeKind.Utc)
{
    utcDateTime = utcDateTime.ToUniversalTime();
}

var settings = service.Retrieve("usersettings", workflowContext.UserId, new ColumnSet("timezonecode"));
```

- Build a time zone change request by providing the **Utc** time that you got from the input and the **TimeZoneCode** from the user settings.

```
LocalTimeFromUtcTimeRequest timeZoneChangeRequest = new
LocalTimeFromUtcTimeRequest()
```

```
{  
  
UtcTime = utcDateTime,  
  
TimeZoneCode = int.Parse(settings["timezonecode"].ToString())  
  
};
```

```
var settings = service.Retrieve("usersettings", workflowContext.UserId, new ColumnSet("timezonecode"));

LocalTimeFromUtcTimeRequest timeZoneChangeRequest = new LocalTimeFromUtcTimeRequest()
{
    UtcTime = utcDateTime,
    TimeZoneCode = int.Parse(settings["timezonecode"].ToString())
};
```

- Run the time zone change request.

```
LocalTimeFromUtcTimeResponse timeZoneResponse =
service.Execute(timeZoneChangeRequest) as LocalTimeFromUtcTimeResponse;
```

- Format the **LocalTime** from the time zone response and set it to the output of the activity.

```
this.ForammtedDateTimeOutput.Set(context, String.Format("{0:f}",
timeZoneResponse.LocalTime));
```

Your class should now look like the following image.

```

0 references
public class ConvertFormatDatetime : CodeActivity
{
    [RequiredArgument]
    [Input("DateTime input")]
    1 reference
    public InArgument<DateTime> DateToEvaluate { get; set; }

    [Output("Formatted DateTime output as string")]
    1 reference
    public OutArgument<string> FormattedDateTimeOutput { get; set; }
    0 references
    protected override void Execute(CodeActivityContext context)
    {
        IWorkflowContext workflowContext = context.GetExtension<IWorkflowContext>();
        IOrganizationServiceFactory serviceFactory = context.GetExtension<IOrganizationServiceFactory>();
        IOrganizationService service = serviceFactory.CreateOrganizationService(workflowContext.UserId);

        DateTime utcDateTime = this.DateToEvaluate.Get(context);

        if (utcDateTime.Kind != DateTimeKind.Utc)
        {
            utcDateTime = utcDateTime.ToUniversalTime();
        }

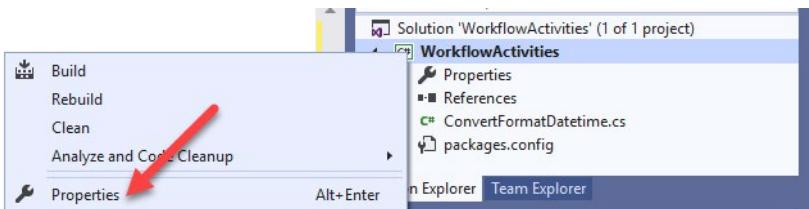
        var settings = service.Retrieve("usersettings", workflowContext.UserId, new ColumnSet("timezonecode"));

        LocalTimeFromUtcTimeRequest timeZoneChangeRequest = new LocalTimeFromUtcTimeRequest()
        {
            UtcTime = utcDateTime,
            TimeZoneCode = int.Parse(settings["timezonecode"].ToString())
        };

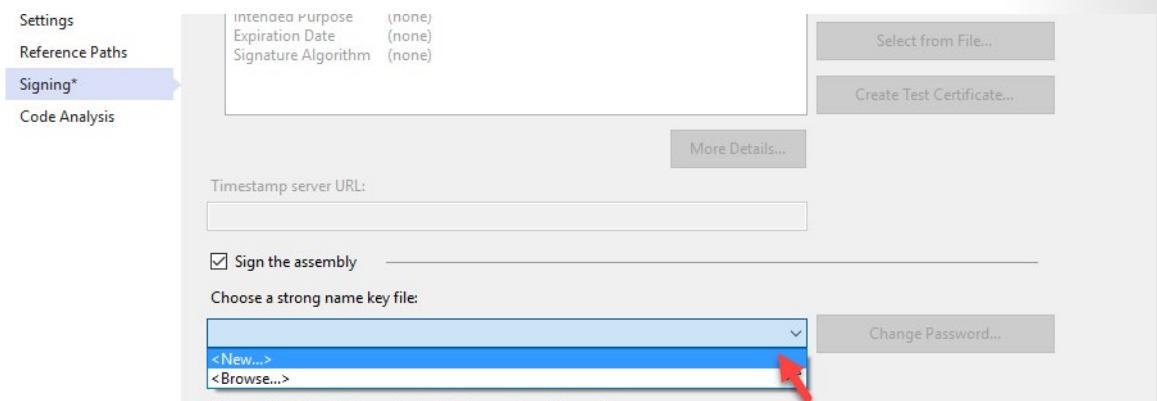
        LocalTimeFromUtcTimeResponse timeZoneResponse = service.Execute(timeZoneChangeRequest) as LocalTimeFromUtcTimeResponse;
        this.FormattedDateTimeOutput.Set(context, String.Format("{0:f}", timeZoneResponse.LocalTime));
    }
}

```

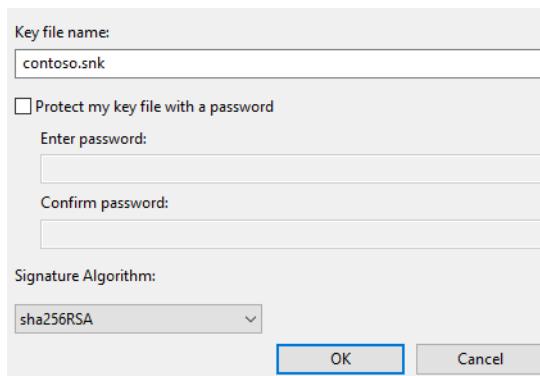
5. Right-click the project and select **Properties**.



6. Select the **Signing** tab, select the **Sign the Assembly** check box, and select **New** in the **Choose a strong name key file** drop-down menu.



7. Enter **contoso.snk** in the **Key file name** field, clear the **Protect my key file with a password** check box, and then select **OK**.



8. Build the project and make sure that the build succeeds.

## Exercise 2: Register and consume

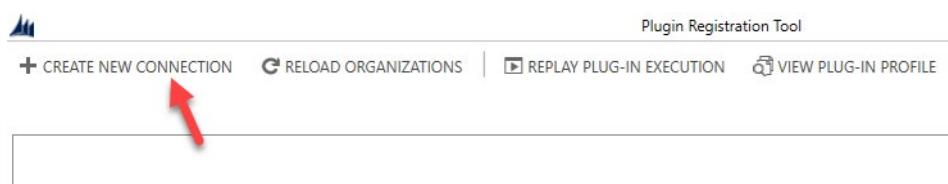
In this exercise, you will register the workflow activity, create and prepare a solution, create a workflow, and then test the workflow and custom activity.

### Task 1: Register the workflow activity

1. Go to your Dynamics 365 SDK folder and open the **PluginRegistration** folder.
2. Double-click **PluginRegistration.exe**.

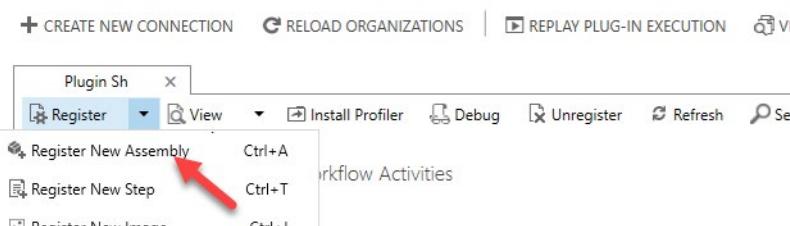


3. Select **Create New Connection**.

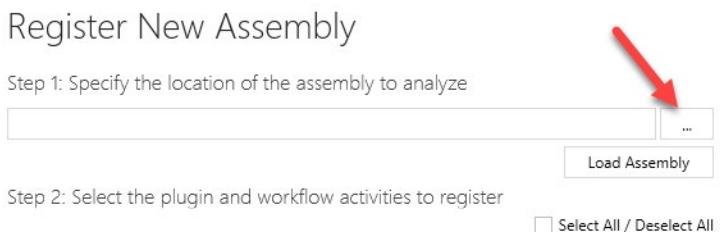


4. Select **Microsoft 365**, provide your credentials, and then select **Login**.

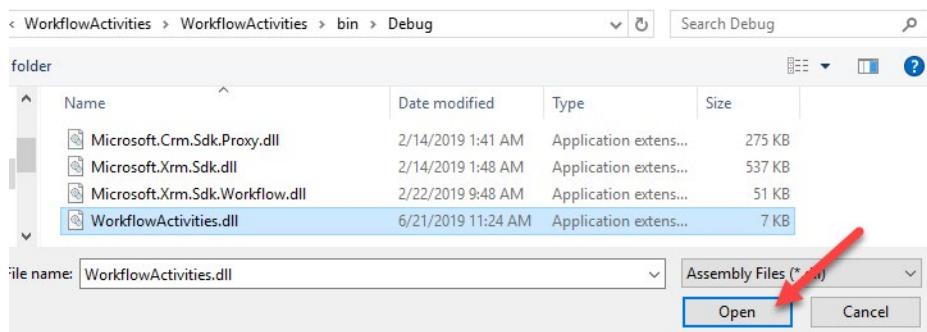
5. Select **Register** and then select **Register New Assembly**.



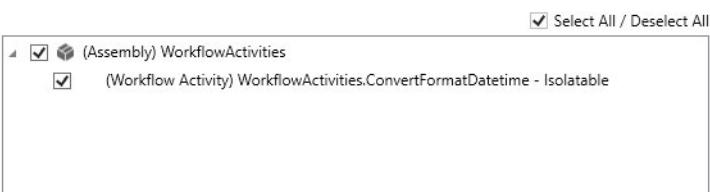
6. Select **Browse**.



7. Browse to the debug folder of your **WorkflowActivities** project, select **WorkflowActivities.dll**, and then select **Open**.



8. Select **Select Selected Plugins**.



Step 3: Specify the isolation mode

- Sandbox
- None

Step 4: Specify the location where the assembly should be stored

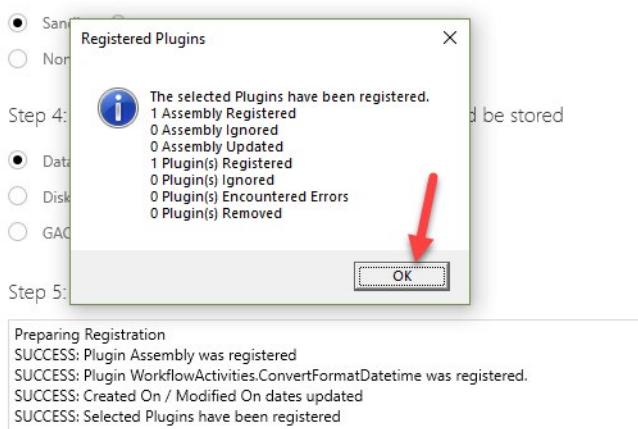
- Database
- Disk
- GAC

Step 5: Log

Register Selected Plugins

Close

9. Select **OK**.



## Task 2: Create and prepare a solution

1. Go to <https://make.powerapps.com><sup>17</sup> and make sure that you do not have the default environment selected.
2. Select **Solutions** and then select **New Solution**.

A screenshot of the Microsoft Power Apps 'Solutions' page. On the left is a navigation sidebar with options like Home, Learn, Apps, Create, Data, Flows, AI Builder (preview), and Solutions. The 'Solutions' option is currently selected and highlighted with a purple bar. The main area shows a table of existing solutions: PowerApps Checker (Created 6/14/2019), Crm Hub (Created 6/8/2019), Common Data Services Default Solution (Created 6/8/2019), and Default Solution (Created 6/8/2019). Above the table is a toolbar with buttons for 'New solution' (which has a red arrow pointing to it), Import, Open AppSource, and Publish all custom. The 'New solution' button is located at the top left of the toolbar.

3. Enter **Contoso WFA** in the **Display Name** field, enter **ContosoWFA** in the **Name** field, and then select the **Publisher** drop-down menu.

<sup>17</sup> <https://make.powerapps.com/?azure-portal=true>

New solution X

Display name \*

Name \*

Publisher \*



Version \*

**4. Select + Publisher.**



Publisher \*

+ Publisher

CDS Default Publisher

**5. Enter **Contoso** in the **Display Name** field, enter **contoso** in the **Prefix** field, and then select **Save and Close**.**

**File** Save and Close

Publisher: New Publisher

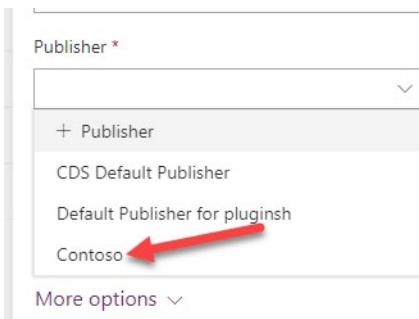
**General**

Display Name *	Contoso	Name *	contoso
Description			

Set the prefix name for custom entities and fields

Prefix *	contoso	Option Value Prefix *	46,327
Name Preview	contoso_entity		

**6. Select the **Publisher** drop-down menu again and select the publisher that you created.**



Publisher \*

+ Publisher

CDS Default Publisher

Default Publisher for pluginsh

Contoso

More options ▾

7. Enter **1.0.0.0** in the **Version** field and then select **Create**.

Publisher \*

Contoso

Version \*

1.0.0.0

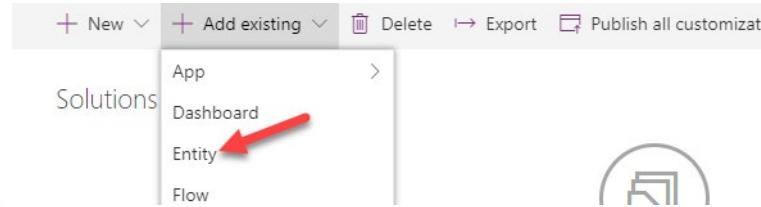
More options ▾

Create Cancel

8. Open the solution that you created.

Display name	Created	Version
Contoso WFA	6/21/2019	1.0.0.0
PowerApps Checker	6/14/2019	1.0.0.103

9. Select **Add Existing** and then select **Entity**.



10. You will use the **Task** entity to test the workflow activity. Select **Task** and then select **Next**.

Add existing entities

Select entities from other solutions or entities that aren't in solutions yet. Adding entities that aren't already in solutions will also add them to Common Data Service.

1 entity selected

Entity task

Display name	Name	Managed e...	Owner	Status
Task	task	-	-	-

Next Cancel

11. Select the **Select components** option.

[← Selected entities](#)

Select components to your selected entities.  
1 entities will be added to your project

Task	<input type="checkbox"/> Include
No components selected	
<a href="#">Select components</a>	

12. Select the **Forms** tab, select the **Task** form, and then select **Add**.

Fields Relationships Business rules Views **Forms** Dashboards Charts Messages

Display name ↴	Name ↴	Field type ↴
Information	Information	-
<input checked="" type="checkbox"/> Task	Task	-
Task Card form	Task Card form	-

**Add** **Cancel**

13. Select **Add** again.

[← Selected entities](#)

Select components to your selected entities.  
1 entities will be added to your project

Task	<input type="checkbox"/> Include all component
1 form selected	
<a href="#">Select components</a>	

**Add** **Cancel**

14. Select to open the **Task** entity that you just added to your solution.

Solutions > Contoso WFA

Display name ↴	Name	Type ↴
Task	task	Entity

15. Select the **Fields** tab and then select **+ Add field**.

Solutions > Contoso WFA > Task

Fields Relationships Business rules Views Forms Dashboards

Display name ↓ Name ↓

16. Enter **Formatted Datetime** in the **Display Name** field, select **Text** from the **Data Type** drop-down menu, and then select **Done**.

Formatted Datetime ×

Display name \*

Name \* (i)

Data type \* (i)

Required

Searchable

Calculated or Rollup (i) + Add

Done Cancel

17. Select the **Save Entity** button.

Solutions > Contoso WFA > Task

Fields Relationships Business rules Views Forms Dashboards Charts Keys Data

Display name ↓	Name ↓	Data type ↓	Type ↓	Required ↓	Searchable ↓
Formatted Datetime	contoso_form...	Add Text	Custom	✓	

Discard Save Entity

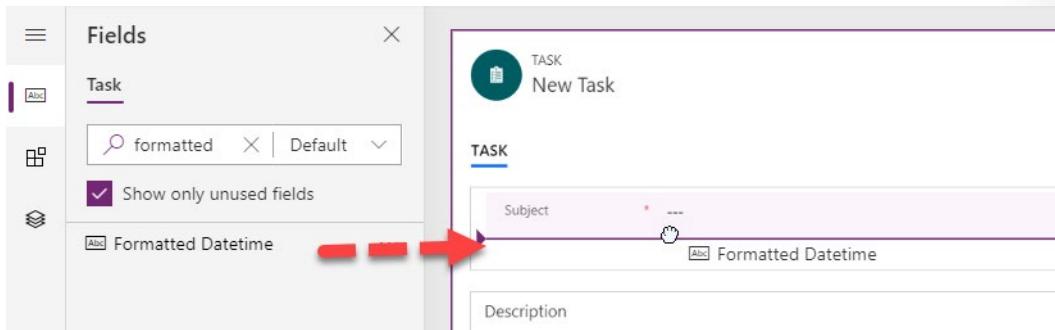
18. Select the **Forms** tab and then select to open the **Task** form.

Solutions > Contoso WFA > Task

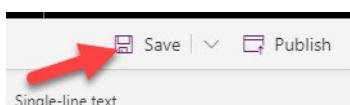
Fields Relationships Business rules Views **Forms** Dashboards Charts Keys Data

Model-driven		Form type	Type
Name	Task	Main	Standard

19. Drag the **Formatted Datetime** field to the form and then place it below the **Subject** field.



20. Select **Save**.



21. Select **Publish** and wait for the publishing to complete.

22. Close the form editor browser tab or window.

23. Select **Done**.

Currently editing a form

When you're done editing the form, click Done below to return to the entity. This will refresh the page and fetch your changes.

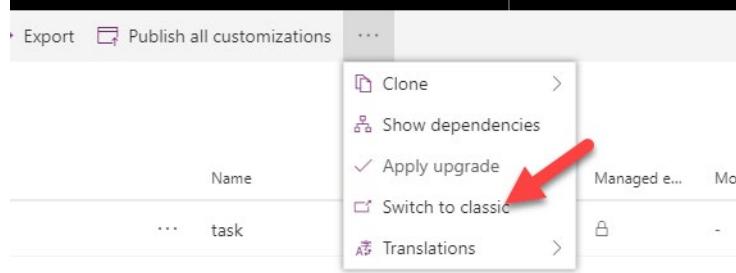
**Done**

## Task 3: Create a workflow

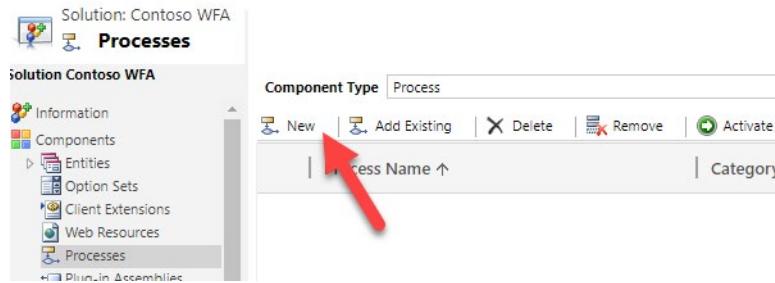
1. Select **Solutions** and open the solution that you created.

Display name	Created	Version
Contoso WFA	6/21/2019	1.0.0.0
PowerApps Checker	6/14/2019	1.0.0.103

2. Select the ellipsis (...) button and select **Switch to classic**.



3. Select **Process** and then select **New**.



4. Enter **Convert and Format Datetime** in the **Process Name** field, select **Workflow** from the **Category** drop-down menu, select **Task** from the **Entity** drop-down list, clear the **Run this workflow in the background (recommended)** check box, and then select **OK**.

A screenshot of the 'Create Process' dialog box. It has fields for 'Process name:' (set to 'Convert And Format Datetime'), 'Category:' (set to 'Workflow'), 'Entity:' (set to 'Task'), and a checkbox for 'Run this workflow in the background (recommended)' which is unchecked. Below these are 'Type:' options ('New blank process' is selected) and a 'Properties' button. At the bottom are 'OK' and 'Cancel' buttons, with a red arrow pointing to the 'OK' button.

5. Select the **As an on-demand process** check box and select **Organization** from the **Scope** drop-down menu.

Process Name\*: Convert And Format Datetime

Activate As: Process

Available to Run:

- Run this workflow in the background (recommended)
- As an on-demand process
- As a child process

Entity: Task

Category: Workflow

Workflow Log Retention:

- Keep logs for workflow job

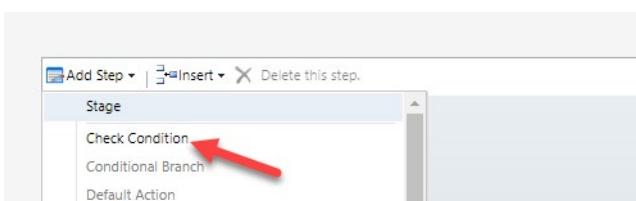
Options for Automatic Processes:

Scope: Organization

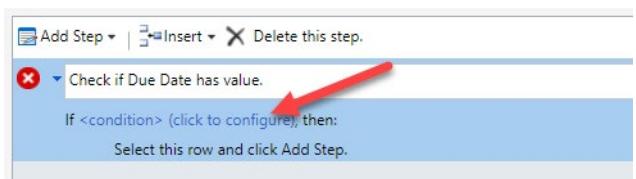
Start when:

- After Record is created
- After Record status changes
- After Record is assigned

6. Select **Add Step** and then select **Check Condition**.



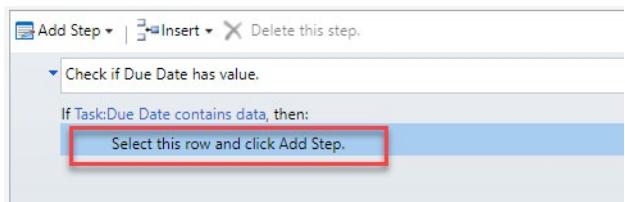
7. Enter **Check if Due Date has value** as the description and then select **Configure**.



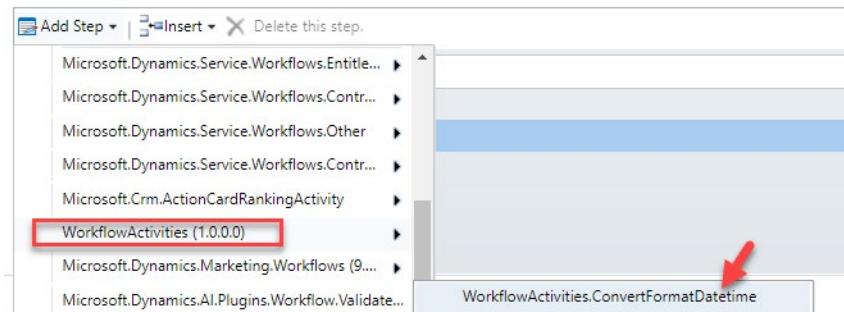
8. Set the condition as shown in the following image and then select **Save and Close**.



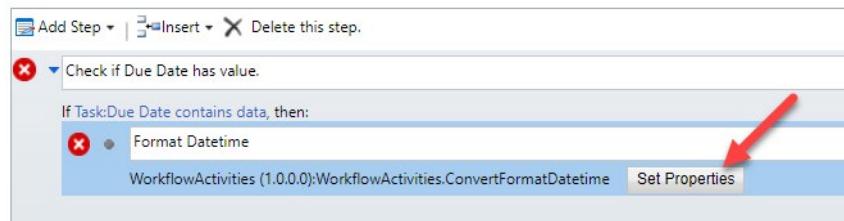
9. Select below the condition statement.



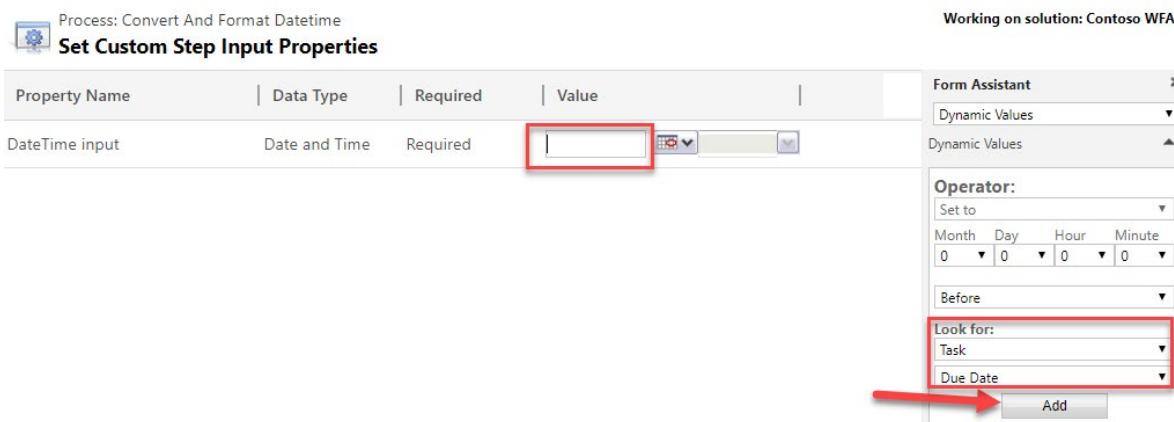
10. Select **Add Step** and then select the **Custom Workflow Activity** that you created.



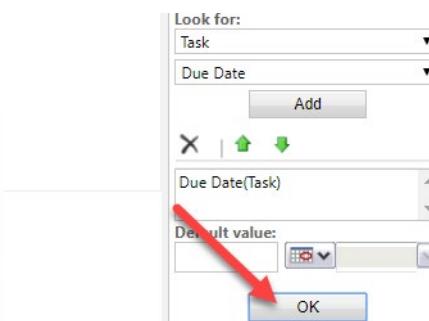
11. Provide a description and then select **Set Properties**.



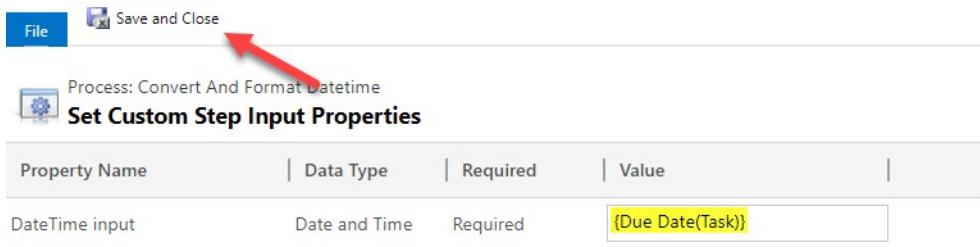
12. Select the **Value** field, select **Due Date**, and then select **Add**.



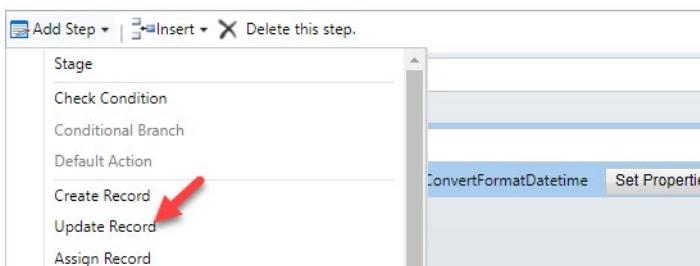
13. Select **OK**.



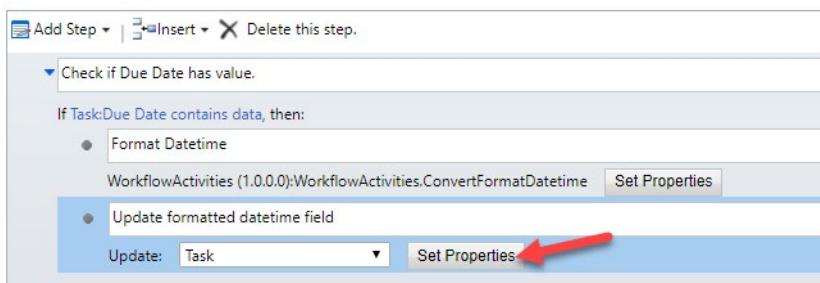
14. Select **Save and Close**.



15. Select the step that you just added, select **Add Step**, and then select **Update Record**.



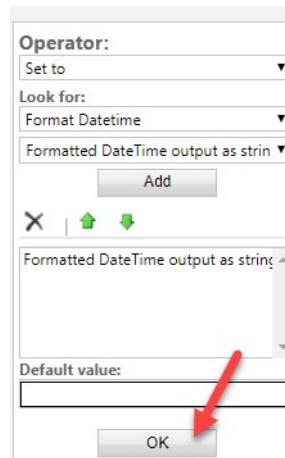
16. Provide a description, make sure that **Task** is selected for the entity in the **Update** drop-down list, and then select **Set Properties**.



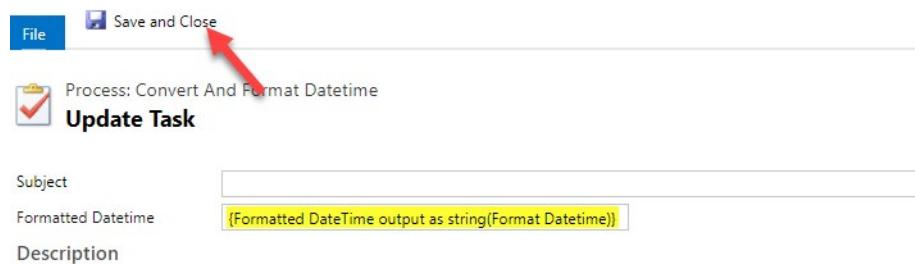
17. Select the **Formatted Datetime** field, select **Format Datetime** from the **Look for** drop-down menu, and then select **Add**.



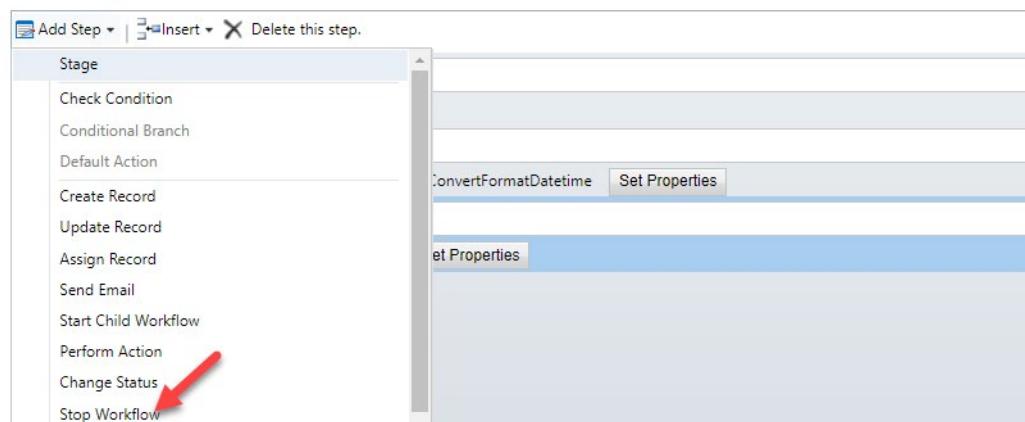
18. Select **OK**.



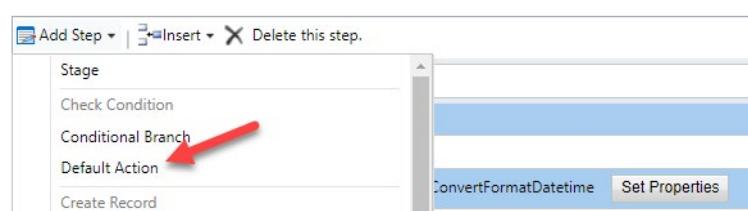
19. Select **Save and Close**.



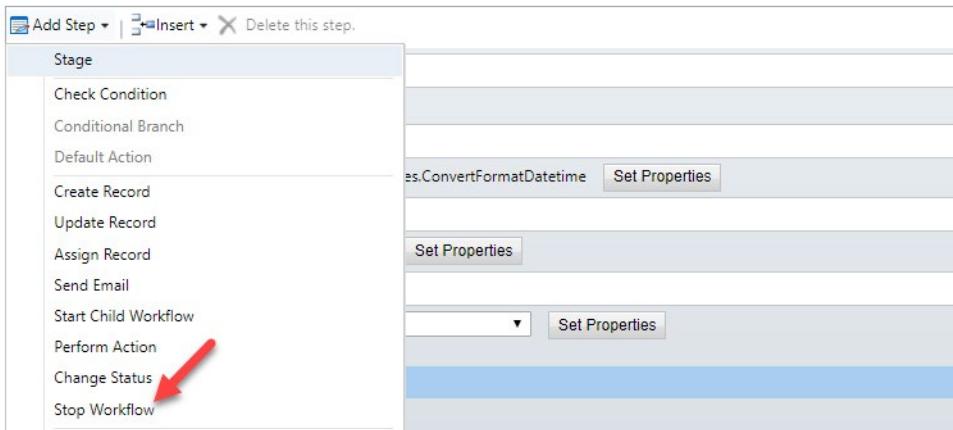
20. Select the step that you just added, select **Add Step**, and then select **Stop Workflow**.



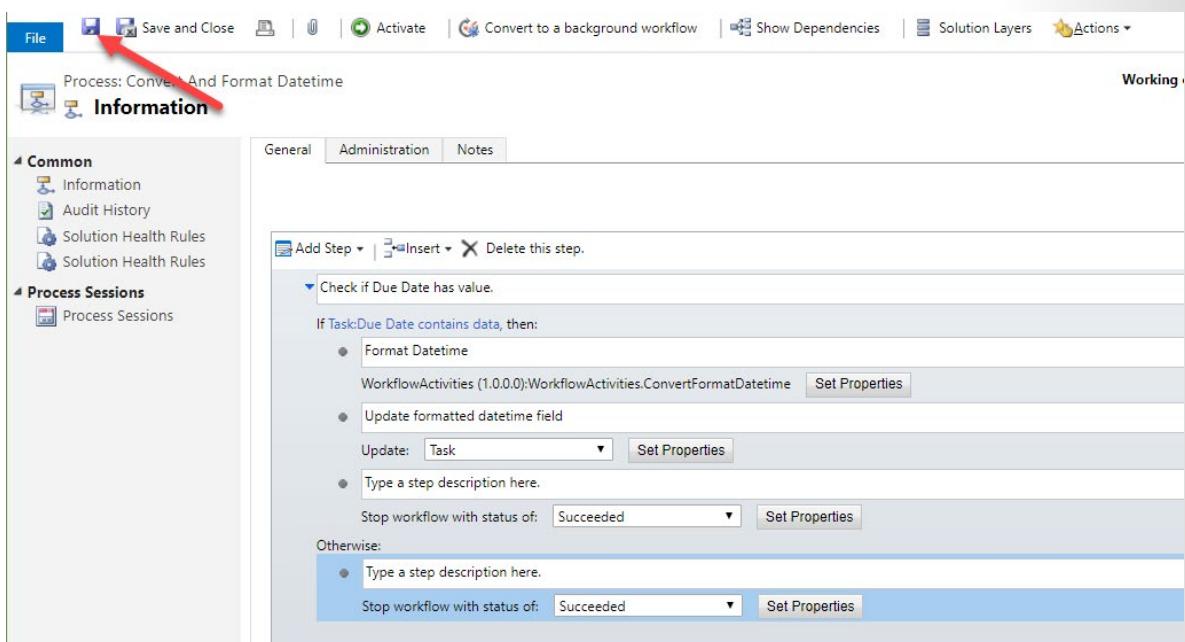
21. Select the condition step, select **Add Step**, and then select **Default Action**.



22. Select below the **Otherwise**, select **Add Step**, and then select **Stop Workflow**.



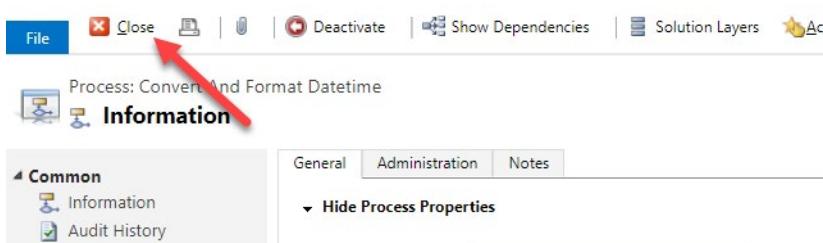
23. Your workflow should now look like the following image. Select **Save**.



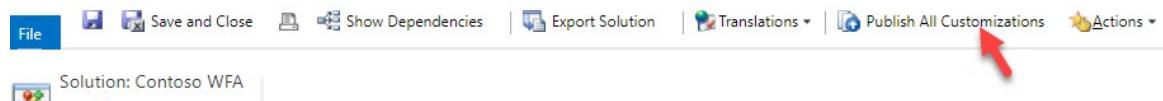
24. Select **Activate**.

25. Confirm activation.

26. Select **Close**.



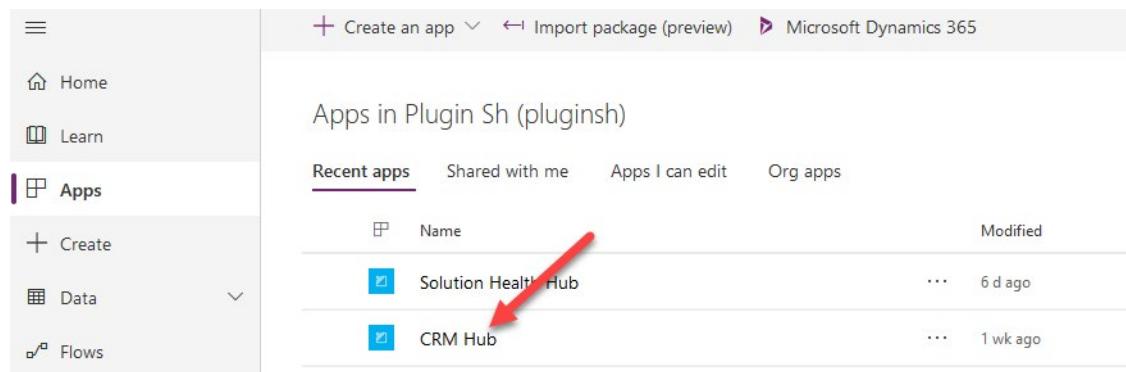
27. Select **Publish All Customizations** and wait for the publishing to complete.



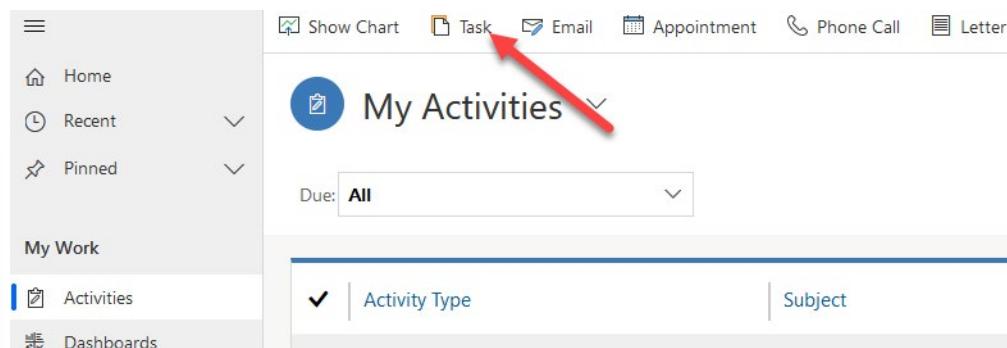
28. Close the **Solution Explorer** window.

## Task 4: Test the workflow and custom activity

1. Go to <https://make.powerapps.com><sup>18</sup> make sure that you are not in the default environment.
2. Select **Apps** and select to open the **CRM Hub** application.

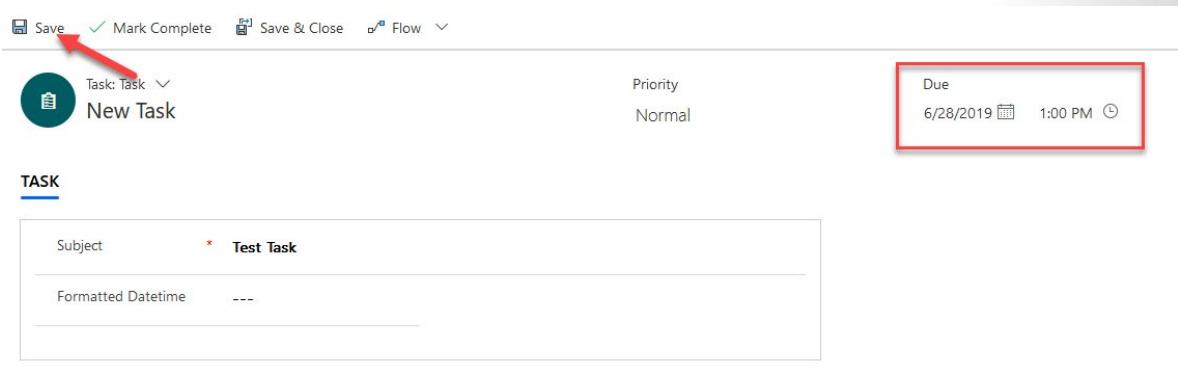


3. Select **Activities** and then select **Task**.

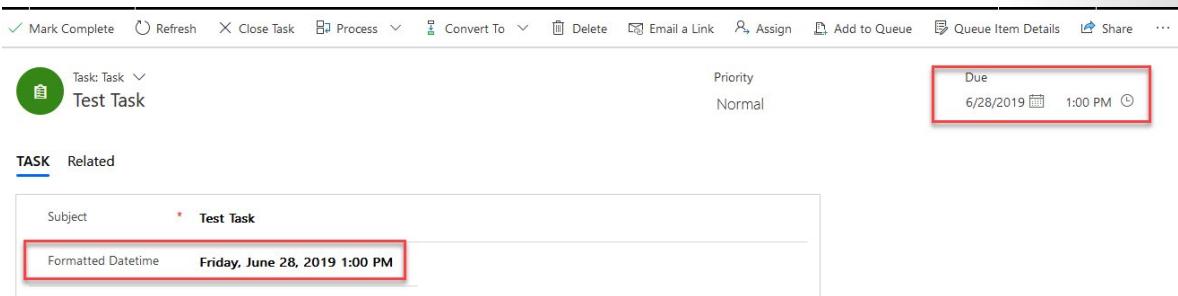


4. Enter **Test Task** for the **Subject**, select the **Due** date and time, and then select **Save**.

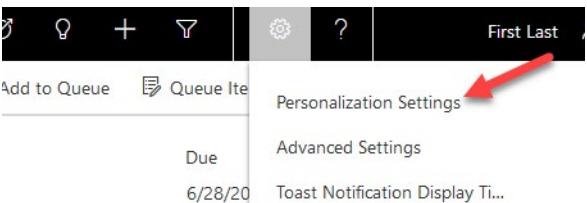
<sup>18</sup> <https://make.powerapps.com/?azure-portal=true>



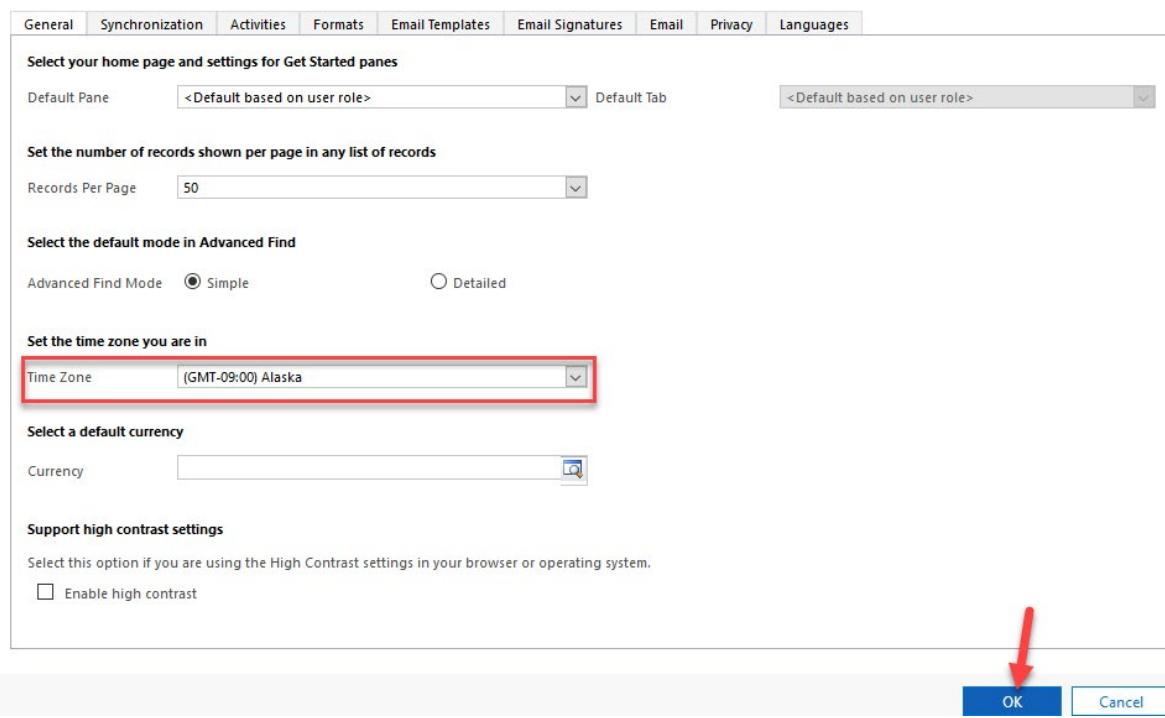
5. The custom workflow activity should run and populate the **Formatted Datetime** field. The **Due** date and **Formatted Datetime** values should be the same.



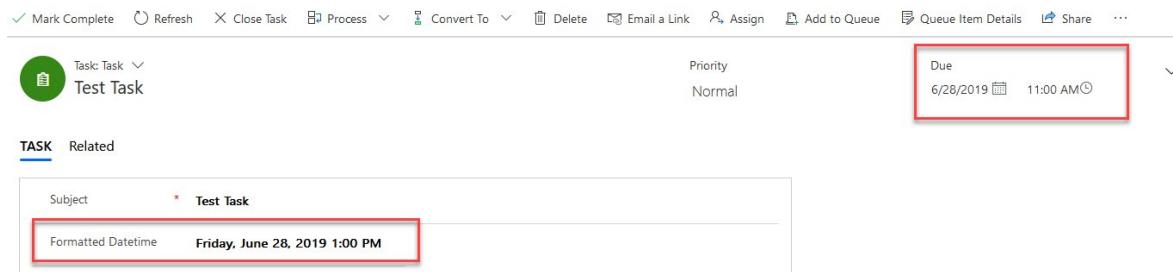
6. Select **Settings** and then select **Personalization Settings**.



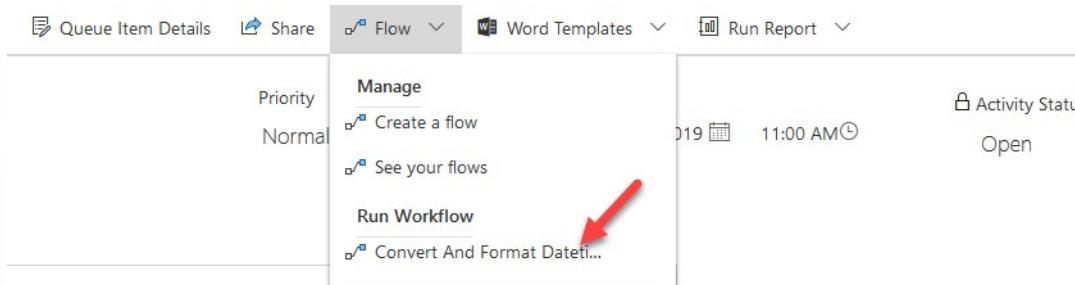
7. Change the **Time Zone** to a different one and then select **OK**.



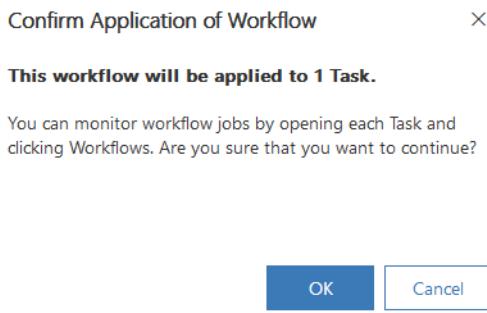
8. The Task should reload. The **Due** date field value should change to the selected time zone automatically, but the **Formatted Datetime** value should not change.



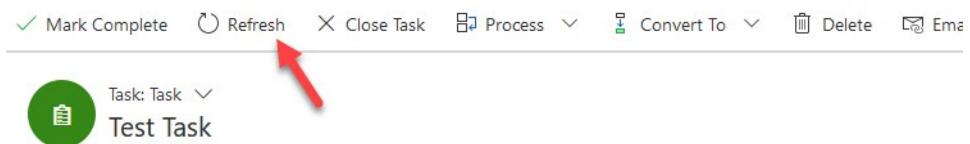
9. Select **Flow** and then select the workflow that you created.



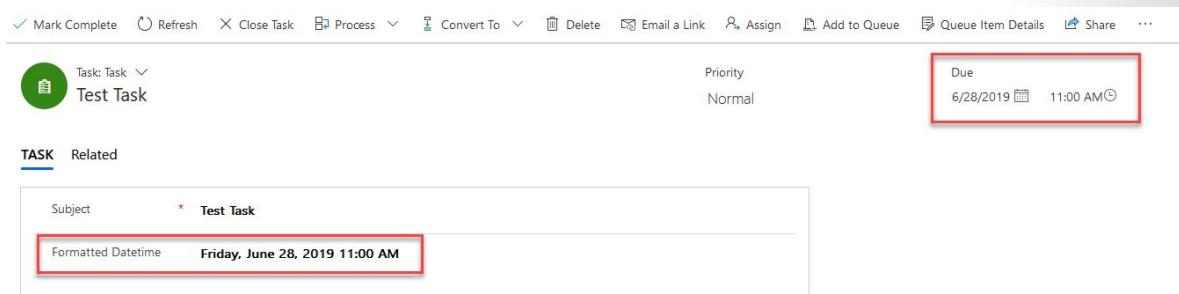
10. Select **OK**.



### 11. Select Refresh.



The **Due** date and **Formatted Datetime** values should now be the same.



## summary

Plug-ins are a powerful method that you have to extend business processes within a Common Data Service application. Plug-ins interact with the event pipeline and provide an execution context that enables you to handle seemingly endless scenarios that otherwise might not have been accomplished through declarative means.



# Module 7 Extending the Power Platform user experience Model Driven apps

## Introduction to web resources

### Web resources introduction

#### Defining web resources

Web resources are virtual, URL-addressable files that are stored in the Common Data Service database. Web resources are the methodology that the Microsoft Power Platform uses to allow for deploying custom scripts/styling/images to the client. Individual files can be uploaded to the platform and then served to the client. There are six types of web resources:

- WebPage - .html
- Style Sheet - .css / .xsl
- JavaScript - .js
- Data - .xml
- Image - .gif / .ico / .jpg / .png / .svg
- String - .resx

[!TIP]

Although these are the only web resources types that are defined in the metadata of the Microsoft Power Platform, any file extension can be used, such as *json*, *.map*, or *.ts*.

Defining client files as web resources files allows them to be referenced directly by the system, and/or included in Solutions to be propagated either manually or as part of a full ALM process through Sandbox environments and onto Production.

The default way for creating a web resource is through a Power Apps solution within the web resources component, where you can create, update, or delete web resources.

[!IMPORTANT]

Changes that are made to web resources will not take effect until customizations have been published.

When creating a new web resource, you should always use good naming conventions and end it with an appropriate extension. This makes it easier for editors outside of the Microsoft Power Platform to recognize the file type.

## Defining web resource dependencies

Web resources also allow for dependencies to be defined. This includes dependencies that a file might have on another file and attributes of particular entities.

Defining attributes on which a web resource is dependent will allow the platform to ensure that the dependencies are enforced, preventing admins from either removing an attribute that the web resource depends on, or importing it into an environment in which the attribute doesn't exist.

When a web resource has other web resources that it depends on, the platform will use this dependency to ensure that the required files are loaded before the topmost web resource is. For small-to-medium implementations, this is the recommended approach for ensuring that a JavaScript file has all of its dependencies loaded wherever it's referenced. For large projects with heavy custom JavaScript, using a tool like Gulp to remove JS dependencies by compiling them into a single file, or a dynamic JS module loading framework like RequireJS, SystemJS, or stealJS, is recommended.

[!NOTE]

The autoloading of dependencies only applies to JavaScript files that are loaded from the ribbon or from a form. Loading a JavaScript file in an Iframe by means of an HTML web resource will not auto load the dependencies.

The screenshot shows the Microsoft Power Apps interface for managing web resources. At the top, there's a toolbar with 'SAVE', 'DELETE', 'PREVIEW', 'SHOW DEPENDENCIES', and a 'More' button. Below the toolbar, it says 'Solution: Default Solution' and 'Web Resource: new\_Common.Lib.js'. There are two tabs: 'General' (selected) and 'Dependencies'. The 'General' tab contains fields for 'Name' (set to 'new\_common\_lib.js'), 'Display Name' (set to 'new\_Common.Lib.js'), and 'Description' (set to 'Common JS Library'). Under the 'Content' section, 'Type' is set to 'Script (JScript)' and 'Language' is set to 'English'. An 'Upload File' field shows 'Choose File No file chosen'. In the 'URL' section, the URL is listed as '[https://MsLearning.crm.dynamics.com//WebResources/new\\_com](https://MsLearning.crm.dynamics.com//WebResources/new_com)'. The 'Dependencies' tab is visible at the bottom.

## Community tools for creating/updating web resources

Manually adding and editing web resources directly from within the Microsoft Power Platform is a rather tedious task, but because the ability to create/update web resources is exposed by the SDK, multiple tools are available to take advantage of this to improve the developer story:

- Web Resource Manager - This tool is a part of the **XrmToolBox**<sup>1</sup>. It allows for two different workflows: working from the platform by pulling web resources down, editing them locally, and then pushing the updated files back up, or working from a local directory, loading, creating, updating, or publishing one or more files up to the platform.
- Web Resources Updated - This tool is a Visual Studio Extension, allowing for loading/creating/updating/publishing files directly into the platform from **Visual Studio**<sup>2</sup>.

### [!NOTE]

The community tools are not a product of Microsoft Dynamics apps and does not extend support to the community tools. If you have questions pertaining to the tool, contact the publisher.

## Referencing web resources

Each web resource is exposed through a URL with the following format:

<sup>1</sup> <https://www.xrmtoolbox.com/plugins/MsCrmTools.WebResourcesManager/?azure-portal=true>

<sup>2</sup> <https://marketplace.visualstudio.com/items?itemName=MaratVDeykun.MicrosoftDynamicsCRMWebResourcesUpdater>

`https://{{Domain}}/WebResources/{{SolutionProviderPrefix}}_{{WebResourcesName}}`

Organize web resources into folders by including forward slashes “/” in the web resource name (The community tools that are mentioned in this article will map the folders to the name for you). When referencing a file from another file (for example, referencing a JavaScript file from an HTML file) it is important for performance to always use relative paths and never reference anything above the **WebResources** folder. This is due to the caching strategy that is used by the Microsoft Power Platform.

Files at the default URL are instructed to never be cached by the browser to ensure that if the file is updated, the client will get the latest version. Whenever a publish occurs, web resources are also pushed to a publish cache directory of the form:

`https://{{Domain}}/{{7B6344115041100000007D}}/WebResources/{{SolutionProviderPrefix}}_{{WebResourcesName}}`

## Script web resources

Script web resources can be used to maintain libraries of client script functions that are written in JavaScript (or TypeScript) and that can be accessed from within a model-driven app form. By placing your scripts into web resources, versus directly into a form or webpage (HTML) web resource, you can more efficiently manage your code independently from where it is being consumed. This promotes better development practices and encourages code reuse.

## Using Client Script libraries

Client Script libraries can be associated with ribbon commands and form events. To review how to accomplish this, see **Apply business logic using client scripting in model-driven apps using JavaScript<sup>3</sup>**.

Additional modules in this learning path cover Client Scripting in depth.

## Referencing a script web resource from a webpage web resource

All web resources can use relative URLs to reference each other. In the following example, for the webpage web resource **new/\_content/contentpage.htm** to reference the Script web resource **new/\_scripts/myScript.js**, add the following HTML code to the head element of **new/\_content/contentpage.htm**.

```
<script type="text/jscript" src="../scripts/myScript.js"></script>
```

To reference a JavaScript from a different publisher, the path must include the customization prefix for that publisher. For example, for the **new/\_content/contentpage.htm** page to reference the **MyLsv/\_scripts/customscripts.js** page, add the following HTML code to the head element of **new/\_content/contentpage.htm**.

---

<sup>3</sup> <https://docs.microsoft.com/powerapps/developer/model-driven-apps/client-scripting/?azure-portal=true>

```
<script type="text/javascript" src=".../MyIsv_/scripts/customscripts.js"></script>
```

## Webpage web resources

Webpage (HTML) web resources enable developers to create custom user interface elements within a model-driven app's form. Webpage (HTML) web resources can include any HTML content that is able to be rendered in the user's browser; however, they cannot contain any code that must be run on the server. ASP.NET pages, for example, cannot be uploaded as webpage (HTML) web resources. For more information on creating webpage (HTML) web resources, see **Webpage (HTML) web resources**<sup>4</sup>.

## Passing parameters to webpage (HTML) web resources

HTML web resources accept a certain number of predefined parameters, along with a single custom parameter called **data**. You can use the **data** parameter to pass one or more custom values to your webpage, however you'll need to encode and decode these parameters in your page if multiple values are required. For an example on how to do this action, see **Sample: Pass multiple values to a web resource through the data parameter**<sup>5</sup>.

For a list of predefined parameters that are available for consumption in a webpage (HTML) web resource, see **Pass parameters to HTML web resources**<sup>6</sup>.

## Interacting with the parent form

You can use the Client API form context to interact with the parent form of an HTML web resource. For more information on the form context, see **Client API Form context**<sup>7</sup>.

## Create Accessible Web Resources

When developing webpage (html) web resources, you should always ensure that you consider accessibility requirements. For more information on how to create accessible HTML web resources, see **Create accessible web resources**<sup>8</sup>.

<sup>4</sup> <https://docs.microsoft.com/powerapps/developer/model-driven-apps/webpage-html-web-resources/?azure-portal=true>

<sup>5</sup> <https://docs.microsoft.com/powerapps/developer/model-driven-apps/sample-pass-multiple-values-web-resource-through-data-parameter/?azure-portal=true>

<sup>6</sup> <https://docs.microsoft.com/powerapps/developer/model-driven-apps/webpage-html-web-resources?azure-portal=true#pass-parameters-to-html-web-resources>

<sup>7</sup> <https://docs.microsoft.com/powerapps/developer/model-driven-apps/clientapi/clientapi-form-context/?azure-portal=true>

<sup>8</sup> <https://docs.microsoft.com/powerapps/developer/model-driven-apps/create-accessible-web-resources/?azure-portal=true>

## Cascading style sheet web resources

Cascading Style Sheet (CSS) web resources, when used in conjunction with webpage (HTML) web resources, are used to provide custom styling to that web resource's user interface.

### Referencing CSS web resources from a webpage resource

Web resources can use relative URLs to reference each other. We highly recommend that you follow specific naming standards when uploading your web resources to allow for intuitive file referencing. For example, an HTML web resource that is found in *sample\_content/contentpage.htm* can reference a stylesheet that is found in *sample\_styles/styles.css* by including the following code in its *head* element:

```
<link rel="stylesheet" type="text/css" href="../styles/styles.css" />
```

If you want to reference a stylesheet that was developed by a different publisher, such as a third party, you can reference it by including that publisher's solution prefix into your file reference path. For example, the HTML web resource that is found in *sample\_content/contentpage.htm* can reference a third-party stylesheet that is found in *MyISV\_styles/styles.css* by including the following code in its *head* element:

```
<link rel="stylesheet" type="text/css" href="../../MyISV_styles/styles.css" />
```

### Style guidelines

You can choose whatever styling you want within your model-driven app. If you want your app's appearance and behavior to conform to other Microsoft applications, consider adhering to styles that are found in Microsoft's **Fabric Core**<sup>9</sup> web styles.

## Image web resources

Images can be made available for usage in Model-driven apps via image web resources. Currently five types of images are supported for usage:

- PNG Format
- JPG Format
- GIF Format
- ICO Format
- Vector Format (SVG)

---

<sup>9</sup> <https://developer.microsoft.com/fabric#/styles/web/?azure-portal=true>

## Common uses for image web resources

A few scenarios that commonly require image web resources include:

- Custom entity icons (use SVG format)
- Icons for custom ribbon controls and **SiteMap** subareas
- Decorative graphics for entity forms and webpage web resources
- Background images that are used by CSS web resources

## Entity icon best practices

We highly recommend that you use the Scalable Vector Graphics (SVG) format for any icon that is presented in the application. The advantage of this format over other image types is that they support scaling. The SVG format enables you to set width and height dimensions within its file, allowing you to save 16x16 and 32x32 formats of the same image for use as icons in your application.

## Other types of web resources

### Data (XML) web resources

Data (XML) web resources can be used to cache data that you want to use in your solution (that is, configuration settings or metadata). For more information, see **Data (XML) Web resources**<sup>10</sup>.

### RESX web resources

RESX web resources are used to manage localized strings in any user interface that you define or with error messages that you'll display. Within a client script, you can use the **Xrm.Utility.getResourceString** to have the localized value of a string maintained in the web resources. For more information, see **RESX web resources**<sup>11</sup>.

### Stylesheet (XSL) web resources

Stylesheet (XSL) web resources can be used to transform XML data into other formats, such as HTML. While this is currently an uncommon practice, it might prove valuable under certain circumstances. For more information, see **Style sheet (XSL) web resources**<sup>12</sup>.

### Working with other file types

While the user interface requires you to have your files saved as one of the predefined types, you are able to store web resources of other file

<sup>10</sup> <https://docs.microsoft.com/powerapps/developer/model-driven-apps/data-xml-web-resources/?azure-portal=true>

<sup>11</sup> <https://docs.microsoft.com/powerapps/developer/model-driven-apps/resx-web-resources/?azure-portal=true>

<sup>12</sup> <https://docs.microsoft.com/powerapps/developer/model-driven-apps/stylesheets-xsl-web-resources/?azure-portal=true>

types (that is, PDF) by saving these files with an extension that matches one of the predefined types (such as, *mypdf.pdf.css*). While this is not something Microsoft encourages, it is a workaround we thought may be valuable to note.

## summary

Web resources provide a robust method for developers to customize the user experience of model-driven applications. Web resources enable you to automate business processes through client scripting and customize the user interface by using HTML web resources. As in most scenarios with making Power Apps, it is important to exhaust other non-code options before resorting to writing code.

# Performing common actions with client script

## Introduction to client-side scripting

With a platform as focused on "No-Code" and "Low-Code" as the Microsoft Power Platform, it can be as important to know what can be done without coding, as it is to know what can be done with it. Frequently, a business will have requirements that are either not a good fit for or aren't possible using the rich OOB configuration options. Specifically, with model-driven apps, Microsoft provides a robust framework for using custom scripts that are running on the client-side or users' side of the platform.

## The Microsoft Power Platform client-side scripting back-story

The Microsoft Power Platform's rich history of supporting client-side scripting dates to its Microsoft Dynamics CRM roots. Much of the client-side scripting model's core is named XRM, which stands for "Anything Relationship Management". The Microsoft Power Platform is the natural evolution of that desire to be able to model any relationships to define and solve business problems.

As with any platform that allows custom code to be run upon itself, the trickiest part is to maintain compatibility between the custom code and the platform whenever the platform is upgraded. Allowing developers full and unrestricted access to the DOM would cause extensive coupling between custom client-side code and the platform, effectively requiring most, if not all, custom code to be reassessed or rewritten each time the platform is upgraded. The Microsoft Power Platform provides the XRM library as an abstraction/contract between the platform and custom client-side code such that the changes to the underlying DOM and platform JS/HTML/CSS won't require changes to custom code.

## The Xrm client framework

Rather than using JQuery or the DOM directly to show/hide a control on a form, the Xrm framework provides a JavaScript function to show or hide a control based on the configured name of the control. You don't need to worry about adding event handlers for multiple controls for the same data attribute on a form; the Xrm framework exposes its own on-change event handler. It's this layer of abstraction that allows the platform to rapidly evolve and grow without having to require massive amounts of changes in client-side code.

## Handling events in forms and grids

You can attach client script functions to handle various events that occur in a model-driven app form. The function that is invoked is referred to as an event handler.

At a high level, events occur whenever:

- A form loads.

- Data is changed in a field or an item within the form.
- Data is saved in a form.

For more information on how to attach to events, see **Events in forms and grids in model-driven apps**<sup>13</sup>.

## Attribute OnChange event

### Event triggers

Attribute OnChange events occur in the following situations:

**From Code** - The OnChange event can be fired manually by calling the corresponding attribute's fireOnChange method.

**Form Field Changes in the UI:** - If data in an attribute's form field has been changed and focus is lost. The only exception to this is the Two-Option (Boolean) field type that is formatted as buttons or check boxes. In these cases, the event occurs immediately after the value has changed. In the event of this scenario, you'll want to use the fireOnChange event to fire the event manually.

[!NOTE]

the OnChange event does not occur if the attribute's field is changed programmatically.

**Form Refresh** - Upon form refresh, such as after a record is saved, the OnChange event will fire if data has changed on the server.

For more information on the OnChange event, see **Attribute OnChange Event (Client API reference)**<sup>14</sup>.

## Form OnLoad events

### Event triggers

Form OnLoad events occur in the following situations:

**From Code** - You can use the formContext.data.refresh method to manually trigger the form data OnLoad event. Note that this event is different from the generic Form OnLoad event in that in addition to being called on initial page load, it is called when page data is explicitly refreshed either through the formContext.data.refresh method or from a user who is saving the record, if any changes have been made. For more information on Form data OnLoad, see **Form data OnLoad Event (Client API reference)**<sup>15</sup>.

**When a Form Loads in the UI** - The OnLoad event occurs after the form has loaded.

---

<sup>13</sup> <https://docs.microsoft.com/powerapps/developer/model-driven-apps/clientapi/events-forms-grids/?azure-portal=true>

<sup>14</sup> <https://docs.microsoft.com/powerapps/developer/model-driven-apps/clientapi/reference/events/attribute-onchange/?azure-portal=true>

<sup>15</sup> <https://docs.microsoft.com/powerapps/developer/model-driven-apps/clientapi/reference/events/form-data-onload/?azure-portal=true>

## Common OnLoad event handler scenarios

OnLoad event handlers should be used for the following scenarios:

- Applying logic on how the form should be displayed
- Setting field properties
- Interacting with other page elements

[!NOTE]

You cannot use the OnLoad event to prevent the window's form from loading; any such desired behavior would require interaction with the DOM and is unsupported.

## Form OnSave event

### Event triggers

Form OnSave events occur in the following situations:

**From Code** - Calling the `formContext.data.save` and `formContext.data.refresh` (if true value is passed as the first parameter) methods will trigger the OnSave event, if there is unsaved data on the form. If you want the OnSave event to trigger regardless of whether the form has unsaved data, you can call the `formContext.data.entity.save` method.

**Save Button selected** - When the user selects any of the three save buttons: the one in the bottom-right corner when editing a record, and the **Save** and **Save & Close** buttons that are available from a new Record.

**Navigation/Auto-Save** - If a record has any unsaved data, autosave will trigger the OnSave event every 30 seconds, if enabled. If the user moves away from the form with unsaved data, the event will also trigger, because this also prompts an autosave.

### Save event arguments

When the form OnSave event occurs, various details are available that can be accessed through the save event arguments. These arguments are accessible through the `executionContext.getEventArgs()` method, which exposes the following functionality:

**Getting the Save Mode** - You can determine how a given save event was initiated by calling the `executionContext.getEventArgs().getSaveMode()` method.

**Cancelling a Save Operation**: You can also cancel the save action from occurring by calling the `executionContext.getEventArgs().preventDefault()` method. This can be helpful when you are performing custom validation activities to prevent the saving of a record if invalid data exists on the form. You can see whether the save event has been canceled by a previous event handler, for example,

by calling the `executionContext.getEventArgs().isDefaultPrevented()` method.

For more information on the Save event arguments, see **Save event arguments (Client API reference)**<sup>16</sup>.

For more information on the form OnSave event as a whole, see **Form OnSave Event (Client API reference) in model-driven apps**<sup>17</sup>.

## Form TabStateChange event

The TabStateChange event occurs whenever the display state of a tab changes to expanded or collapsed. A common technique is to use this method to reset the source of an IFRAME control because the IFrame.src property will be overwritten when a tab is expanded.

## IFRAME Control OnReadyStateComplete event

The OnReadyStateComplete event occurs when the content of an IFRAME has loaded and can be accessed through code. This event can also be used when you need to access Webpage (HTML) resource elements. It is always important to consider cross-site scripting (XSS) limitations when you are interacting with IFRAME controls that surface information from third-party sites.

## Lookup Control PreSearch event

The PreSearch event occurs just before a Lookup control launches a dialog to search for records. It can be used to change the results that are displayed in a lookup based on whatever custom conditions you might want to apply by using the `addCustomFilter` method on the control. This event is not able to be configured through the UI and must be added/removed through the `addPreSearch` and `removePreSearch` methods, respectively.

## Grid OnLoad event

The Grid OnLoad event occurs every time any subgrid (editable or non-editable) refreshes. Because grids load asynchronously from the rest of the form, it is common practice to use this event handler to perform any logic that is specific to the given grid versus the Form OnLoad event. This event not only fires on initial load, but anytime a sort is performed

Currently, there is not a supported way to register an event that handles when a record is added to/removed from a subgrid. There are various workarounds that exist that have been implemented to handle this scenario that involves maintaining the previous and current record counts of the grid in global variables that can implicitly define whether a

---

<sup>16</sup> <https://docs.microsoft.com/powerapps/developer/model-driven-apps/clientapi/reference/save-event-arguments/?azure-portal=true>

<sup>17</sup> <https://docs.microsoft.com/powerapps/developer/model-driven-apps/clientapi/reference/events/form-onsave/?azure-portal=true>

record has been added/removed. Due to the questionable nature of global variables, however, we don't recommend such activities here, but these approaches can be found by searching the web.

## Editable grid events

The following events are accessible when a grid has been shown as editable. These can be used to handle various user and code-based interactions that are conducted within the grid.

**OnChange** - Occurs when a value is changed in an editable grid and the cell loses focus, or when an attribute value has been updated from code through the `setValue` method.

**OnRecordSelect** - Occurs when a single row has been selected on an editable grid.

**OnSave** - Occurs when there is a change in record selection, a user selects the editable grid's save button, or any dynamic operation is applied while there are pending changes (such as sorting, filtering, paging, and so on). Note that this event might not fire if navigation is performed outside of the grid's context, such as when you are going to a different record. It also will not fire if the grid is refreshed because clicking the **Refresh** button on an editable grid causes it to discard any pending changes. Editable grids are not autosaved, so no such events will cause this event to fire.

## Business Process Flow events

With the ability to access the Business Process Flow entity, numerous process automations that were previously conducted through client script can now be achieved with workflows and business rules. However, there might still be scenarios where you might choose to use the client API to handle business process flow events. For more information on process events and event handler methods, see **formContext.data.process (Client API reference)**<sup>18</sup>.

## Event triggers

**OnProcessStatusChange** - Occurs whenever a process's status changes. Statuses are Active, Finished, and Aborted.

**OnStageChange** - Occurs when the active stage of a business process flow changes.

**OnStageSelected** - Occurs when a stage of a business process flow is selected.

## Knowledge Base Search Control events

The Knowledge Base Search control requires knowledge management to be enabled in your app, a feature traditionally used

<sup>18</sup> <https://docs.microsoft.com/powerapps/developer/model-driven-apps/clientapi/reference/formcontext-data-process/?azure-portal=true>

in model-driven apps for Dynamics 365. For more information on how to use this control, see **Add the Knowledge Base Search control to forms<sup>19</sup>**.

**OnResultOpened** - Occurs when a knowledge base article is opened in the knowledge base search control, either inline or through the pop-out action.

**OnSelection** - Occurs when a knowledge base article is selected in the knowledge base search control.

**PostSearch** - Occurs when a search has completed in the knowledge base search control.

## Interact with a model-driven app form with client script

### Client API execution context

Before interacting with a model-driven app's form with Client Script, you should know how the application provides context for an event handler. When an event handler is registered, you are given the option to pass execution context as its first parameter. This context provides the ability to perform various tasks such as interact with the event's formContext or gridContext. For more information on how to register event handlers to use the executionContext object, see

**Client API execution context<sup>20</sup>**.

For more details on the methods that the Client API provides, see **Execution context (Client API reference)<sup>21</sup>**.

### Client API form context

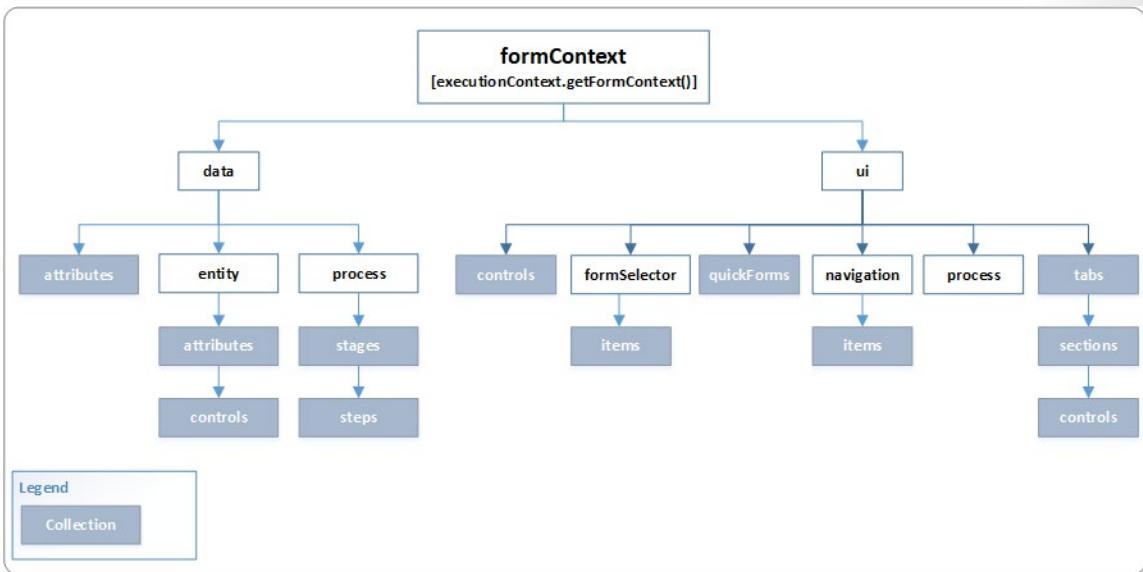
A model-driven app's form, in sum, is accessed through the executionContext.getFormContext() method. The following screenshot is a high-level overview of the properties and methods that are available within this method's returning object:

---

<sup>19</sup> <https://docs.microsoft.com/dynamics365/customer-service/add-knowledge-base-search-control-forms>

<sup>20</sup> <https://docs.microsoft.com/powerapps/developer/model-driven-apps/clientapi/clientapi-execution-context/?azure-portal=true>

<sup>21</sup> <https://docs.microsoft.com/powerapps/developer/model-driven-apps/clientapi/reference/execution-context/?azure-portal=true>



## data object

The data object is intended to be used for any entity and process data manipulation within the form. Because a business process's states and stages are maintained as entity data, automating items within a given process involves interacting with this object as opposed to the UI object.

The following is a summary of each of the data object's containing objects and collections:

Name	Description
Attributes	Collection of non-entity data on the form. Items in this collection are of the same type as the attributes collection, but it is important to note that they are not attributes of the form entity.
Entity	Provides methods to retrieve information that are specific to the record that is displayed on the page, the save method, and a collection of all the attributes that are included on the form. Attribute data is limited to attributes that are represented by fields on the form versus all fields that are available in the entity configuration. For more information, see <b>formContext.data.entity</b> ( <a href="https://docs.microsoft.com/powerapps/developer/model-driven-apps/clientapi/reference/formcontext-data-entity/?azure-portal=true">https://docs.microsoft.com/powerapps/developer/model-driven-apps/clientapi/reference/formcontext-data-entity/?azure-portal=true</a> ).
Process	Provides objects and methods to interact with the business process flow data on a form. For more information, see <b>formContext.data.process</b> ( <a href="https://docs.microsoft.com/powerapps/developer/model-driven-apps/clientapi/reference/formcontext-data-process/?azure-portal=true">https://docs.microsoft.com/powerapps/developer/model-driven-apps/clientapi/reference/formcontext-data-process/?azure-portal=true</a> ).

## UI object

The UI object is intended to be used for retrieving and/or automating any elements that are related to the form's user interface.

The following summarizes each of the UI object's containing objects and collections:

Name	Description
Controls	Collection of all the controls on the page. See <b>Collections</b> ( <a href="https://docs.microsoft.com/powerapps/developer/model-driven-apps/clientapi/reference/collections/?azure-portal=true">https://docs.microsoft.com/powerapps/developer/model-driven-apps/clientapi/reference/collections/?azure-portal=true</a> ) for information about the collections, controls, and the control objects in the collection.
FormSelector	Use the formSelector.getCurrentItem method to retrieve information about the form that is currently in use. Use the formSelector.items collection to return information about all the forms that are available for the user.
Navigation	A collection of all the navigation items on the page. See <b>formContext.ui.navigation item</b> ( <a href="https://docs.microsoft.com/powerapps/developer/model-driven-apps/clientapi/reference/formcontext-ui-navigation/?azure-portal=true">https://docs.microsoft.com/powerapps/developer/model-driven-apps/clientapi/reference/formcontext-ui-navigation/?azure-portal=true</a> ) for information about the items in the collection. Navigation is not available for Microsoft Dynamics 365 for tablets. For automating overall application navigation, we tend to use the Xrm.Navigation namespace.
process	Provides objects and methods to interact with the business process flow control on a form, such as setting its visibility. For more information, see <b>formContext.ui.process</b> ( <a href="https://docs.microsoft.com/powerapps/developer/model-driven-apps/clientapi/reference/formcontext-ui-process/?azure-portal=true">https://docs.microsoft.com/powerapps/developer/model-driven-apps/clientapi/reference/formcontext-ui-process/?azure-portal=true</a> ).
QuickForms	A collection of all the quick view controls on a form. For more information, see <b>formContext.ui.quickForms</b> ( <a href="https://docs.microsoft.com/powerapps/developer/model-driven-apps/clientapi/reference/formcontext-ui-quickforms/?azure-portal=true">https://docs.microsoft.com/powerapps/developer/model-driven-apps/clientapi/reference/formcontext-ui-quickforms/?azure-portal=true</a> ).
Tabs	A collection of all the tabs on the page. See <b>formContext.ui.tabs</b> ( <a href="https://docs.microsoft.com/powerapps/developer/model-driven-apps/clientapi/reference/formcontext-ui-tabs/?azure-portal=true">https://docs.microsoft.com/powerapps/developer/model-driven-apps/clientapi/reference/formcontext-ui-tabs/?azure-portal=true</a> ) for information about the items in the collection.

# Exercise - Write your first client script

## Exercise 1: Business rules

In this exercise, you will create business rules that will change the requirement of the observation location field based on the value of the observation type field.

Each exercise consists of a scenario and a list of learning objectives.

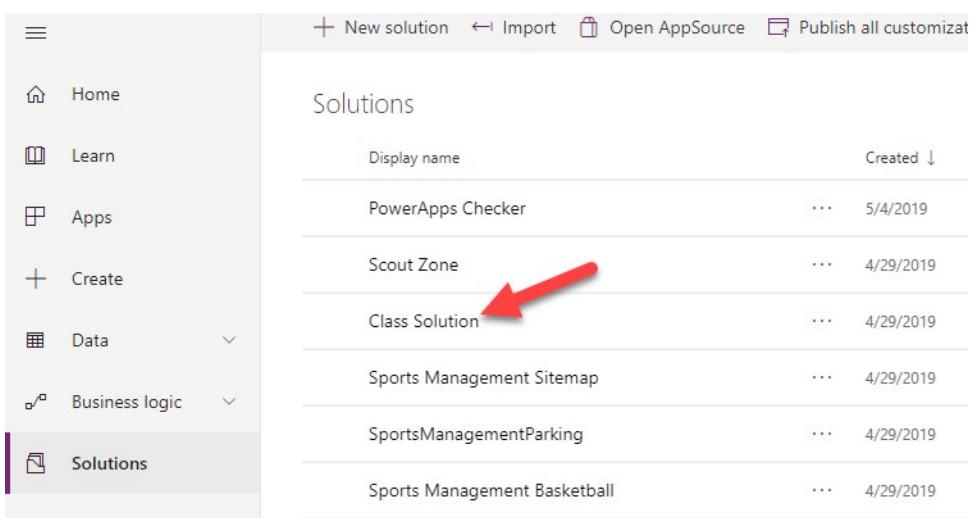
[!NOTE]

To complete this exercise, you need to have the solutions installed as described in [Install a Package Deployer package<sup>22</sup>](#) lab exercise from a previous module.

### Task 1: Create a location required business rule

In this task, you will create a business rule that will make the observation location field required if the observation type is game or practice.

1. Go to <https://make.powerapps.com/><sup>23</sup> and make sure you are not in the default environment.
2. Select **Solutions** and open the **Class Solution**.



The screenshot shows the Microsoft Power Apps Solutions interface. On the left, there's a navigation bar with options like Home, Learn, Apps, Create, Data, Business logic, and Solutions. The Solutions option is currently selected, indicated by a purple bar at the bottom of the list. The main area is titled 'Solutions' and displays a table with the following data:

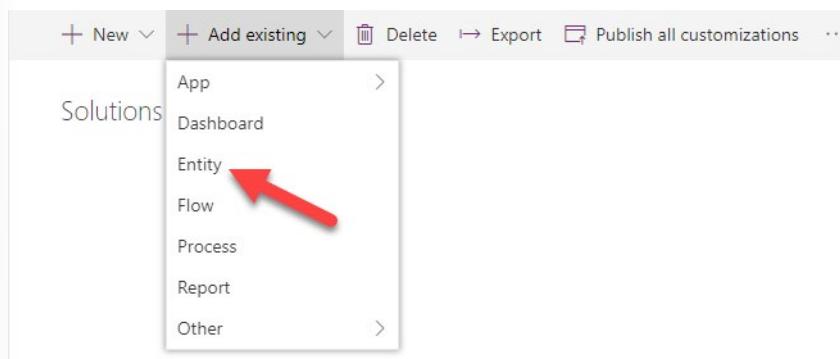
Display name	Created
PowerApps Checker	5/4/2019
Scout Zone	4/29/2019
Class Solution	4/29/2019
Sports Management Sitemap	4/29/2019
SportsManagementParking	4/29/2019
Sports Management Basketball	4/29/2019

A red arrow points to the 'Class Solution' row in the list.

3. Select **Add Existing** and select **Entity**.

<sup>22</sup> <https://docs.microsoft.com/learn/modules/use-developer-tools-extend/4-exercise/?azure-portal=true>

<sup>23</sup> <https://make.powerapps.com/?azure-portal=true>



4. Select the **Scout Report** entity and select **Next**.

1 entity selected

Display name	Name
<input checked="" type="checkbox"/> Scout Report	tt_scoutreport
Scout Report Detail	tt_scoutreportdetail

**Next** **Cancel**

5. Select the **Select Components** button.

1 entities will be added to your project

Scout Report  
No components selected  
**Select components**

**Add** **Cancel**

A screenshot of a modal dialog titled 'Select Components'. It shows a single entity named 'Scout Report' with the note 'No components selected'. Below this is a link 'Select components' with a red arrow pointing to it. At the bottom are 'Add' and 'Cancel' buttons.

6. Select the **Forms** tab, select the **Information** form, and then select **Add**.

Fields Relationships Business rules Views **Forms** Dashboards Charts Mes

Display name	Name	Field
<input checked="" type="checkbox"/> Information	Information	-

**Add** **Cancel**

A screenshot of a modal dialog titled 'Select Form'. It shows a table with one row: 'Information' under 'Display name' and 'Information' under 'Name'. The 'Forms' tab is selected at the top. At the bottom are 'Add' and 'Cancel' buttons.

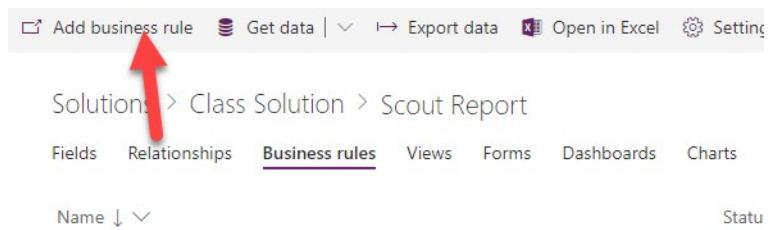
7. Select **Add** again.

8. Select to open the **Scout Report** entity.

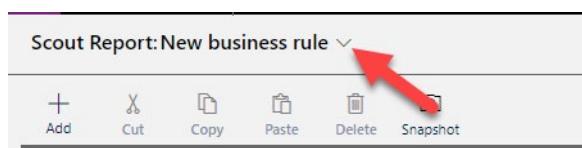
Solutions > Class Solution



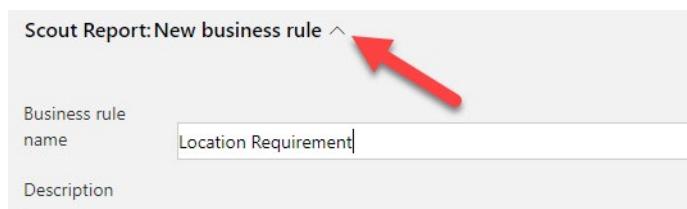
9. Select the **Business rules** tab and select **Add business rule**.



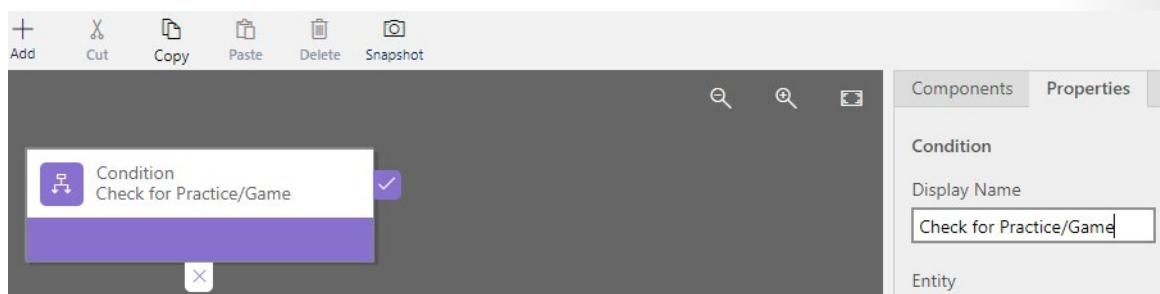
10. Select **Show Details**.



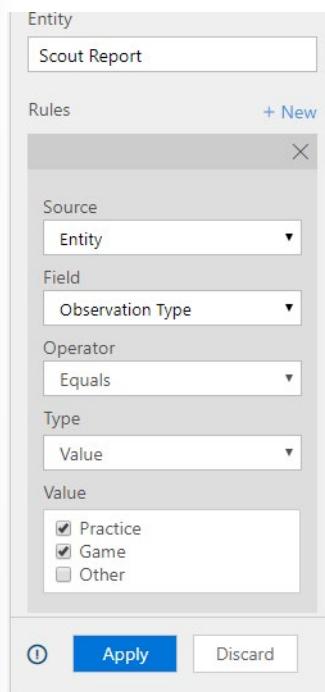
11. Enter **Location Requirement** for the **Business rule name** and then select **Hide Details**.



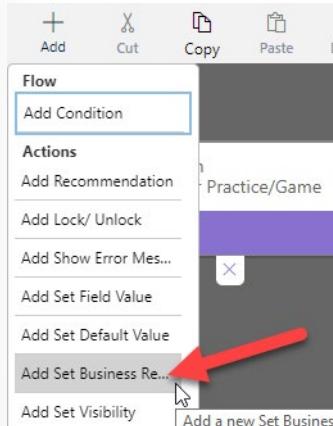
12. Select the **Condition**, and then go to the **Properties** tab and enter **Check for Practice/Game** in the **Display Name** field.



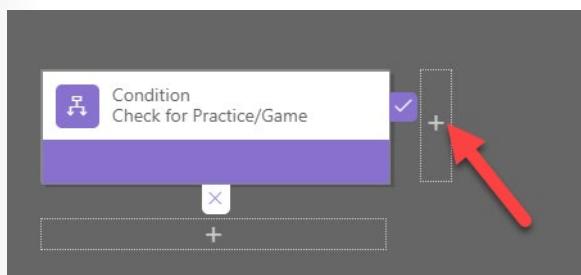
13. Set the **Rules** as shown in the following figure, and then select **Apply**.



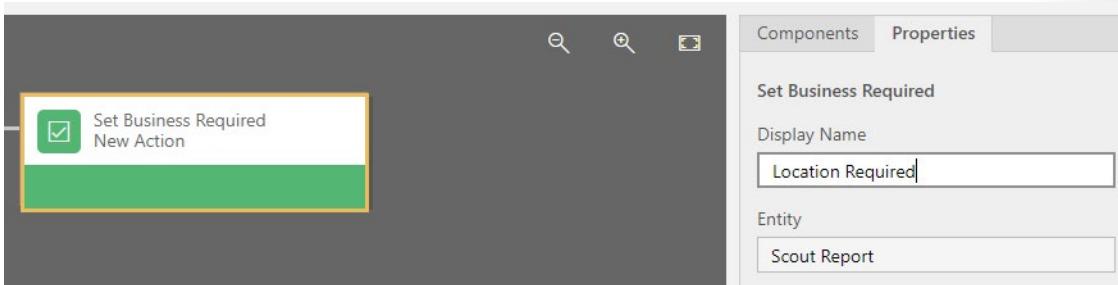
14. Select **Add** and then select **Add Set Business Required**.



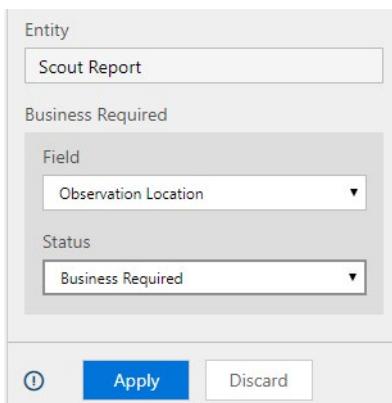
15. Add the action to the true side of the condition.



16. Select the new action and enter **Location Required** in the **Display Name** field.



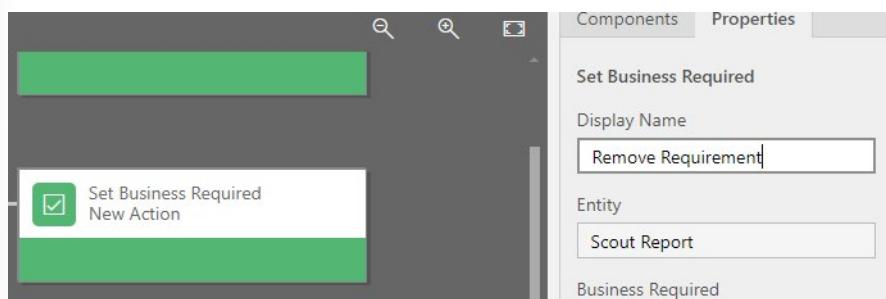
17. Select **Observation Location** for **Field**, select **Business Required** for **Status**, and then select **Apply**.



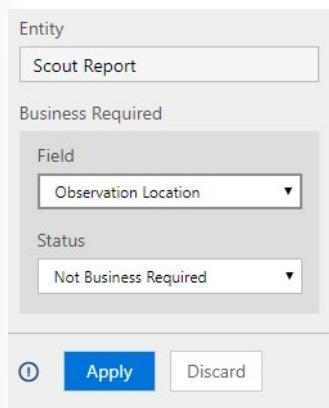
18. Select the **Components** tab, drag the **Set Business Required** action and drop it on the false side of the condition.



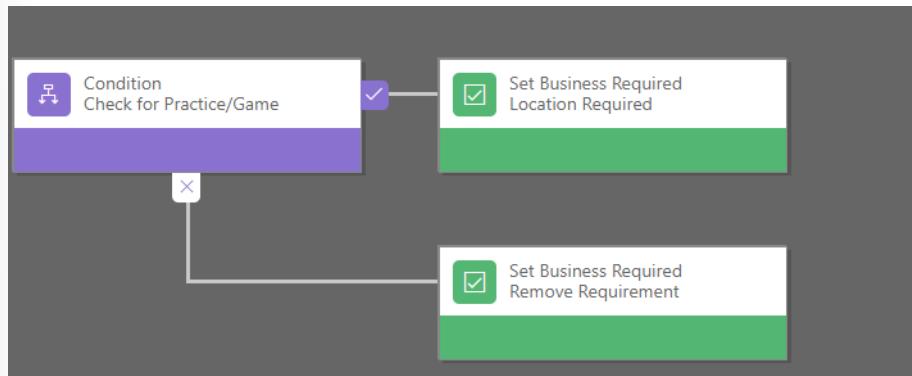
19. Select the new action and enter **Remove Requirement** in the **Display Name** field.



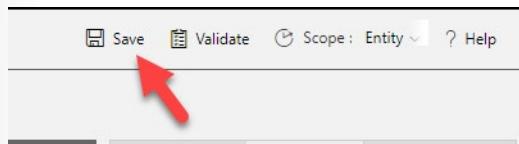
20. Select **Observation Location** for **Field**, select **Not Business Required** for **Status**, and then select **Apply**.



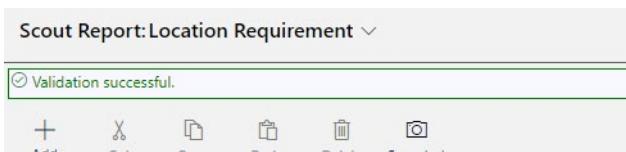
Your business rule should resemble the following image.



21. Select **Save**.



22. Your business rule will be validated and saved. Make sure the validation is successful.



23. Select **Activate**.
24. Confirm activation by selecting **Activate** again.
25. Close the business rule editor browser tab.
26. Select **Done**. Do not leave this page.

Currently creating a new  
business rule

When you're done creating the business rule, click Done below to return to the entity. This will refresh the page and fetch your changes.

**Done**

## Task 2: Test business rules

In this task, you will test the business rules that you created.

1. Select **Apps** and select to open the **Scout Zone** application.

Name
Solution Health Hub
Scout Zone
Sales Hub

2. Select **Scout Reports** and open one of the records.

Name	Player
Maria A	Maria A
Paul C	Paul C
Paul G	Paul G

3. Make sure that nothing is selected for **Observation Type** and note that the **Observation Location** field shouldn't be required.

General Interviews Observation Highlights Summary Playing Style Sportsmanship Related

Name	* Maria A
Player Recruiting	---
Observation Location	---
Owner	* <input checked="" type="radio"/> First Last
Player	 Maria A
Observation Date	9/15/2015
Observation Type	---
Recommendation	---

4. Select **Game** for **Observation Type**. The **Observation Location** should become a required field.

General Interviews Observation Highlights Summary Playing Style Sportsmanship Related

Name	* Maria A
Player Recruiting	---
Observation Location	* ---
Owner	* <input checked="" type="radio"/> First Last
Player	 Maria A
Observation Date	9/15/2015
Observation Type	<b>Game</b>
Recommendation	---

5. Select **Other** for **Observation Type**. The **Observation Location** requirement should go away.

Name	* Maria A
Player Recruiting	---
Observation Location	---
Owner	* <input checked="" type="checkbox"/> First Last
Player	 Maria A
Observation Date	9/15/2015
Observation Type	Other

6. Select **Practice** for **Observation Type**. The **Observation Location** should become required again.

Observation Location	* ---
Owner	* <input checked="" type="checkbox"/> First Last
Player	 Maria A
Observation Date	9/15/2015
Observation Type	Practice

7. Select **Save**. You should not be able to save it until you provide a value for **Observation Location**. You should receive a field notification on the required field and a form notification.

The screenshot shows a Microsoft Dynamics 365 form for a 'SCOUT REPORT' record named 'Maria A'. The form has tabs at the top: General (selected), Interviews, Observation Highlights, Summary, Playing Style, Sportsmanship, and Related. In the 'General' tab, there are three fields: 'Name' (Maria A), 'Player Recruiting' (---), and 'Observation Location' (\*). The 'Observation Location' field has a red error message box containing the text 'A required field cannot be empty.'.

## Exercise 2: Form and field notifications

In this exercise, you will create a script that will show a form and field notification if **Other** is selected for observation type.

### Task 1: Find field names and option set values

In this task, you will locate the names of the observation type and observation location fields, and then make a note of the observation type option set values.

1. Go to <https://make.powerapps.com/><sup>24</sup> and make sure you are not in the default environment.
2. Select **Solutions** and select to open the **Default Solution**.

The screenshot shows the 'Solutions' blade in the Microsoft Power Apps maker environment. The 'Solutions' section is highlighted. A red arrow points to the 'Default Solution' entry in the list, which is currently selected.

3. Select **Entity** and then search for and select to open the **Scout Report** entity.

<sup>24</sup> <https://make.powerapps.com/?azure-portal=true>

Solutions > Default Solution

Display name	Name	Type	Managed e...	Modified	Owner
Scout Report	tt_scoutreport	Entity	Open	-	-
Scout Report D...	tt_scoutreportdetail	Entity	Open	-	-

4. Select the **Fields** tab and select the **Observation Type** field.

Observation Date	...	tt_observation...	Date Only
Observation Location	...	tt_observation...	Text
Observation Type	...	tt_observation...	Option...
Player	...	tt_playerid	Lookup
Player Recruiting	...	tt_playerrecru...	Lookup

5. Go to the properties pane, make a note of the field **Name** and select **Edit option set**. This is the field that you will check in your script.

Solutions > Default Solution > Scout Report

Interviewed Player	...	tt_ir...
Interviewed Teammate	...	tt_ir...
Interviews Needed	...	tt_ir...
Name	...	tt_n...
Observation Date	...	tt_o...
Observation Location	...	tt_o...
Observation Type	...	tt_o...
Player	...	tt_p...
Player Recruiting	...	tt_p...
Playing Style	...	tt_n...

Display name \*

Name \* ⓘ

Data type \* ⓘ

Option Set

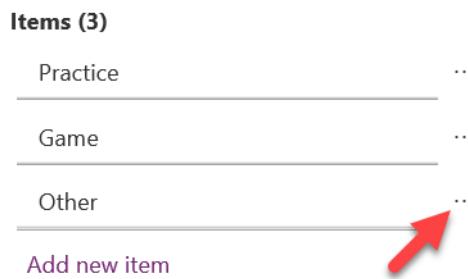
Option set \*

Sports Observation Type

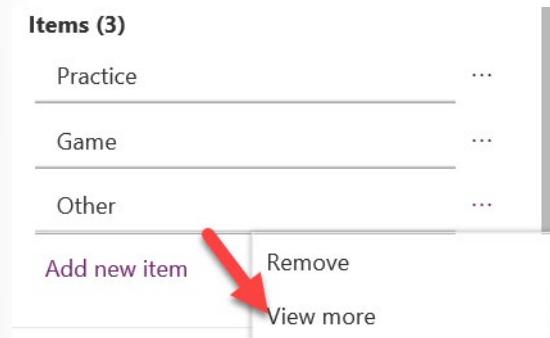
  

**Edit option set**

6. Select the **Other ...** button.



7. Select **View more**.



8. Note the **Value** and select **Cancel**. This is the value you will check for in your script.

A screenshot of an edit dialog for adding a new item. It has two fields: "Name \*" containing "Other" and "Value \*" containing "206,340,002". Below these fields are "Save" and "Cancel" buttons. A red box highlights the "Value" input field, and a red arrow points from the text "8. Note the Value and select Cancel." to the "Cancel" button.

9. Select **Cancel** again.

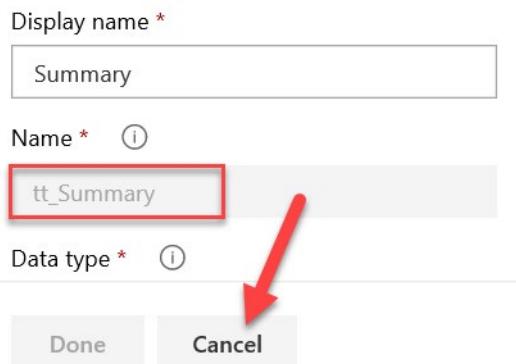
10. Enter **Summary** in the **Display name** field, note the field **Name**, and then select **Cancel**. This is the field where you will show the notification.

Display name \*

Name \* ⓘ

Data type \* ⓘ

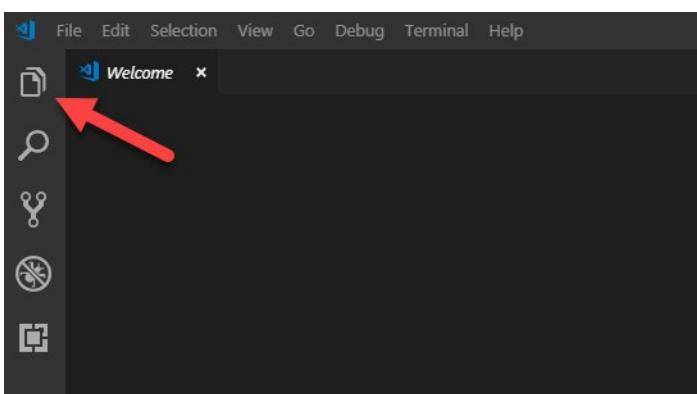
**Done**   **Cancel**



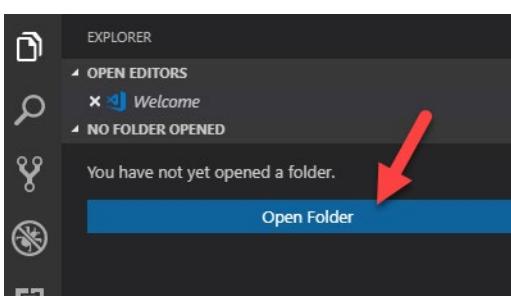
## Task 2: Create form and field notifications

In this task, you will use Visual Studio Code to create a script that will show form and field notifications if **Other** is selected for **Observation Type**.

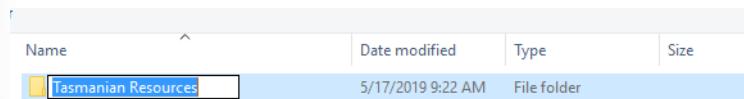
1. Start Visual Studio Code and select **Explorer**.



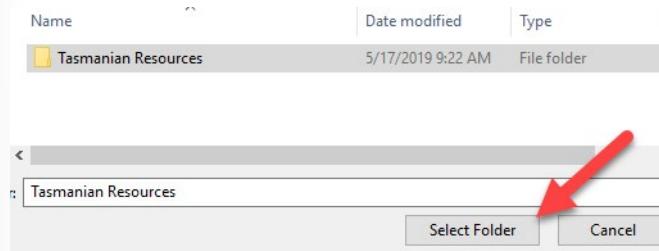
2. Select **Open Folder**.



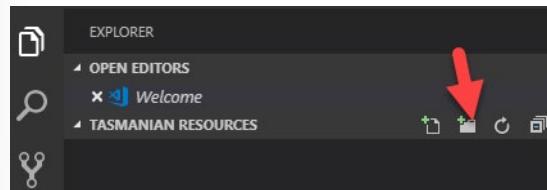
3. Create a new folder and name it **Tasmanian Resources**.



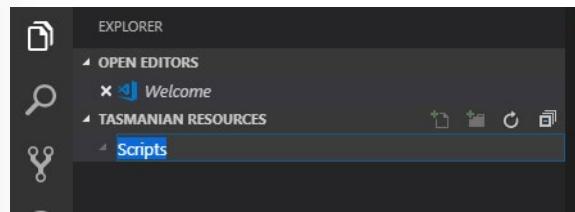
4. Select the folder and select the **Select Folder** button.



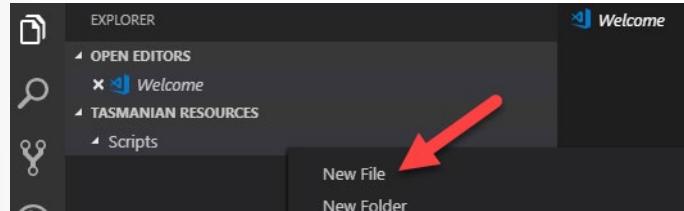
5. Select the **New Folder** icon.



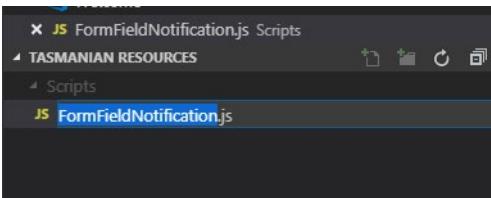
6. Name the new folder **Scripts**.



7. Right-click the **Scripts** folder and select **New File**.



8. Name the new file **FormFieldNotification.js**.



9. Add the following snippet to the file. This is the function that you will call when the observation type changes.

```
function OnObservationTypeChange(executionContext) {  
}
```

10. You will first get the form context from the execution context and then get the selected value of the observation type. Add the following snippet inside the function. The name of the observation type field is: **tt\_observationtype**.

```
var formContext = executionContext.getFormContext();  
  
var selectedObservationType =  
formContext.getAttribute("tt_observationtype").getValue();
```

```
1  function OnObservationTypeChange(executionContext) {  
2    var formContext = executionContext.getFormContext();  
3    var selectedObservationType = formContext.getAttribute("tt_observationtype").getValue();  
4  }  
5
```

11. Check the value of the selected observation type, and then add the following snippet to the function. The other option value is **206340002** (without the commas).

```
if (selectedObservationType == 206340002)  
  
{  
  
}
```

12. Show the form notification if the value evaluates to true. Add the following snippet inside the **if** statement. The three types of form notifications are INFO, WARNING, and ERROR. Each notification type shows a different icon and color.

```
formContext.ui.setFormNotification("Please provide a detailed summary  
of your findings", "INFO", "5001");
```

13. To see all the form notification types and how multiple form notifications are shown, add the snippet after the last notification. The first argument is the message, the second is the type, and the third is the notification ID, which is used to change or clear the notification.

```
formContext.ui.setFormNotification("Test for warning", "WARNING",
"5002");

formContext.ui.setFormNotification("Test for error", "ERROR",
"5003");

if (selectedObservationType == 206340002)
{
    formContext.ui.setFormNotification("Please provide a detailed summary of your findings", "INFO", "5001");
    formContext.ui.setFormNotification("Test for warning", "WARNING", "5002");
    formContext.ui.setFormNotification("Test for error", "ERROR", "5003");
}
```

14. Set the field notification for the **Summary** field, add the following snippet inside the `if` statement. The name of the **Summary** field is: `tt_summary`.

```
formContext.getControl("tt_summary").setNotification("Please provide
a detailed summary of your findings","6001")
```

Your script should now look like the following image.

```
function OnObservationTypeChange(executionContext) {
    var formContext = executionContext.getFormContext();
    var selectedObservationType = formContext.getAttribute("tt_observationtype").getValue();

    if (selectedObservationType == 206340002)
    {
        formContext.ui.setFormNotification("Please provide a detailed summary of your findings", "INFO", "5001");
        formContext.ui.setFormNotification("Test for warning", "WARNING", "5002");
        formContext.ui.setFormNotification("Test for error", "ERROR", "5003");

        formContext.getControl("tt_summary").setNotification("Please provide a detailed summary of your findings","6001")
    }
}
```

15. Select **File** and then select **Save All**.

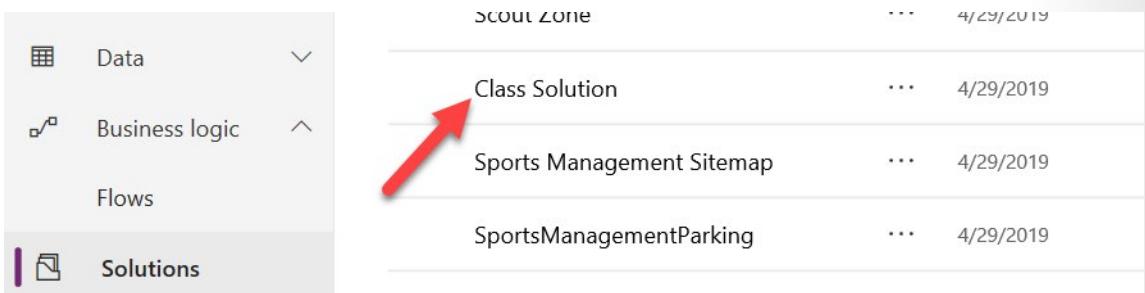
## Task 3: Add resource to solution

In this task, you will add the script that you created to the Class solution and then make sure that your function is called when the observation type changes.

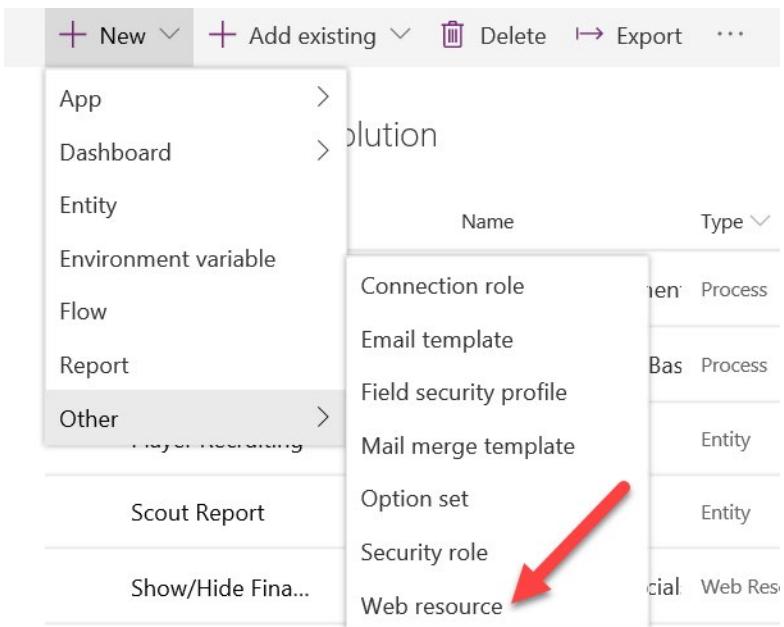
1. Go to <https://make.powerapps.com/><sup>25</sup> and make sure that you are not in the default environment.
2. Select **Solutions** and open the **Class Solution**.

---

<sup>25</sup> <https://make.powerapps.com/?azure-portal=true>



3. Select **New > Other > Web resource**.



4. Enter **FormFieldNotification** for **Name** and **Form Field Notification** for **Display Name**, select **Script (Jscript)** for **Type**, and then select **Browse**.

**Web Resource: New**

General Dependencies

**General**

Name \* tt\_ FormFieldNotification

Display Name Form Field Notification

Description

**Content**

Type \* Script (JScript) Text Editor

Language

Upload File Browse... 

**URL**

5. Select the script that you created and select **Open**.

Name	Date modified	Type	Size
FormFieldNotification.js	5/20/2019 12:29 PM	JavaScript File	1



6. Select **Save**.

7. Select **Publish** and wait for the publishing to complete.

8. Close the web resource editor browser tab.

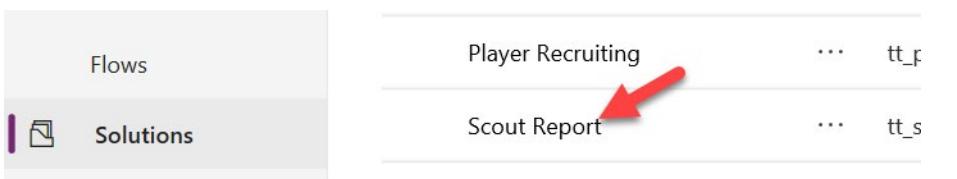
9. Select **Done**.

Currently creating a new web resource

When you're done creating the new web resource, click **Done** below to return to the page. This will refresh the page and fetch your changes.

**Done**

10. Select to open the **Scout Report** entity.



11. Select the **Forms** tab and open the **Information** form.

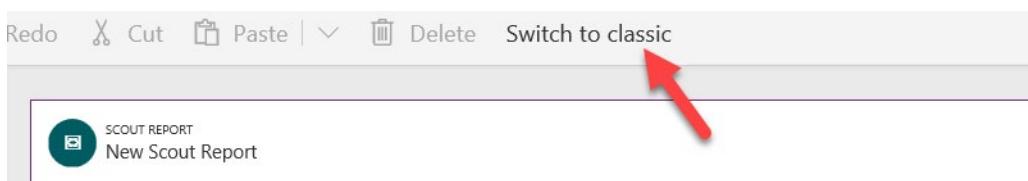
Solutions > Class Solution > Scout Report

Fields   Relationships   Business rules   Views   **Forms**   Dashboards   Charts   Keys

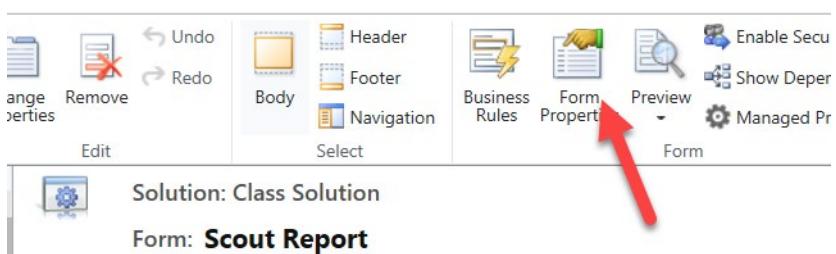
Model-driven

Name ↓	Form type ↗	Type ↗
Information	Main	Custom

12. Select **Switch to classic**.



13. Select **Form Properties**.



14. Select **Add Library**.

The screenshot shows the 'Form Libraries' section of the canvas app builder. At the top, there are tabs for 'Events', 'Display', 'Parameters', and 'Non-Event Dependencies'. Below the tabs is a 'Event List' header. Underneath it is a 'Form Libraries' header. A red arrow points to the 'Add' button in the toolbar, which has a green plus sign icon. The toolbar also includes 'Remove', 'Up', 'Down', 'Edit', and 'Edit Library' buttons. Below the toolbar is a table with columns: 'Name', 'Display Name', and 'Description'. The first row of the table is highlighted.

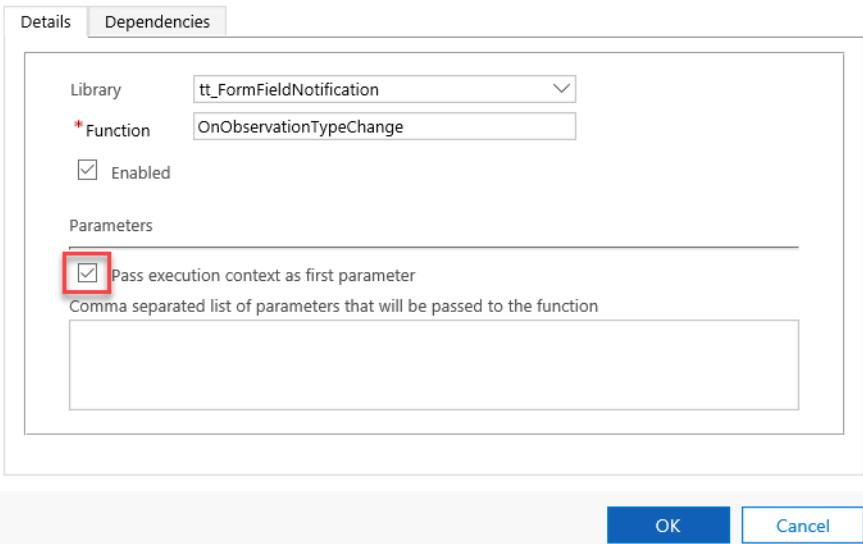
15. Search for **tt**, select **tt\_FormFieldNotification**, and then select **Add**.

The screenshot shows a search results page for 'tt'. The search bar at the top contains 'tt'. Below the search bar is a table with columns: 'Name', 'Display Name...', and 'Language'. One row is selected, showing 'tt\_FormFieldNotification' in the 'Name' column and 'Form Field N...' in the 'Display Name...' column. A red checkmark is next to the row. At the bottom of the table are buttons for 'New', 'Add', 'Cancel', and 'Remove Value'.

16. Go to the **Event Handlers** section, select **Observation Type** for **Control**, select **OnChange** for **Event**, and then select **Add**.

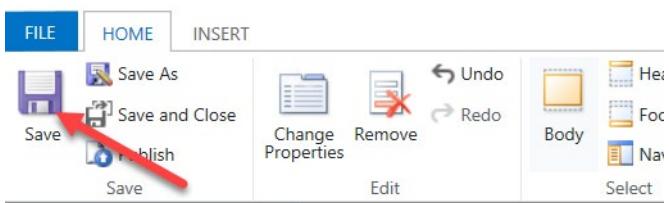
The screenshot shows the 'Event Handlers' section. At the top, there is a header for 'Event Handlers'. Below it is a table with columns: 'Control', 'Event', and 'Function'. The 'Control' dropdown is set to 'Observation Type' and the 'Event' dropdown is set to 'OnChange'. A red arrow points to the 'Add' button in the toolbar, which has a green plus sign icon. The toolbar also includes 'Remove', 'Up', 'Down', 'Edit', and 'Edit Library' buttons. Below the toolbar is a table with columns: 'Library', 'Function', and 'Enabled'. The first row of the table is highlighted.

17. Select **tt\_FormFieldNotification** for **Library**, enter **OnObservationTypeChange** for **Function**, select the **Pass execution context as first parameter** check box, and then select **OK**. This is the name of the function that you created.



18. Select **OK** again.

19. Select **Save**.



20. Select **Publish**.

21. Close the classic form editor browser tab.

22. Close the preview form editor browser tab.

23. Select **Done**.

## Task 4: Test Your Work

1. Go to <https://make.powerapps.com/><sup>26</sup> and make sure that you are not in the default environment.

2. Select **Apps** and open the **Scout Zone** application.

<sup>26</sup> <https://make.powerapps.com/?azure-portal=true>

The screenshot shows the 'Apps' section of the Power Platform interface. On the left, there's a sidebar with links like Home, Learn, Apps (which is selected), Create, Data, and Business logic. The main area is titled 'Apps in Dev Modules (devmodules)' and shows a list of recent apps: 'Solution Health Hub', 'Scout Zone' (with a red arrow pointing to it), and 'Sales Hub'. There are filter and sort options at the top of the list.

3. Select **Scout Reports** and open one of the records.

The screenshot shows the 'Scout Reports' section of the interface. On the left, there's a sidebar with Home, Recent, Pinned, Scout (selected), Sports Teams, Player Recruitings, Reports, and Scout Reports. The main area is titled 'Active Scout Reports' and shows a list of records with columns for Name and Player. The records are: Maria A (highlighted with a red arrow), Paul C, and Paul G.

4. The default value for **Observation Type** should be **Other**. The form and field notifications will not be visible because the script will only run when the **Observation Type** changes. In the next task, you will make sure that the script runs when the form loads.

SCOUT REPORT  
Maria A

**General** Interviews Observation Highlights Summary Playing Style Sportsmanship Related

Name	* <b>Maria A</b>
Player Recruiting	---
Observation Location	---
Owner	* <a href="#">First Last</a>
Player	<b>Maria A</b>
Observation Date	9/15/2015
Observation Type	<b>Other</b>

5. Change the **Observation Type** to **Game** and then change it back to **Other**. You should now see the form notification.

You have 3 notifications. Select to view.

New Deactivate Delete Refresh Assign Share Email a Link ...

SCOUT REPORT  
Maria A

6. Select the chevron button to see all notification types. You will remove the test notification in the next task.

You have 3 notifications. Select to view.

Please provide a detailed summary of your findings

Test for warning

Test for error

SCOUT REPORT  
Maria A

7. Select the **Summary** tab of the form. You should see the field notification.

General Interviews Observation Highlights **Summary** Playing S

**✖ Please provide a detailed summary of your findings**

8. Go back to the **General** tab and change **Observation Type** to **Game**. **Other** is not selected for **Observation Type**, but the notifications will not go away. You will fix this issue in the next task.

You have 4 notifications. Select to view.

+ New Deactivate Delete Refresh Assign Share Email a

 Maria A

General Interviews Observation Highlights Summary Playing Style Spor

Owner  First Last

Player  Maria A

Observation Date 9/15/2015

Observation Type **Game**

Recommendation ---

## Task 5: Update script

In this task, you will make sure that your function runs when the form loads, remove the test form notifications, and then clear the notification if **Other** is not selected for **Observation Type**.

1. Go to Visual Studio Code.
2. Remove the test form notifications. Your if statement should now look like the following image.

```

function OnObservationTypeChange(executionContext) {
    var formContext = executionContext.getFormContext();
    var selectedObservationType = formContext.getAttribute("tt_observationtype").getValue();

    if (selectedObservationType == 206340002)
    {
        formContext.ui.setFormNotification("Please provide a detailed summary of your findings", "INFO", "5001");
        formContext.getControl("tt_summary").setNotification("Please provide a detailed summary of your findings","6001")
    }
}

```

3. Add an `else` statement that will clear the notifications and then add the following snippet after the `if` statement. Make sure the IDs "5001" and "6001" match the IDs of the notifications that you created.

```

else

{
    formContext.ui.clearFormNotification("5001");

    formContext.getControl("tt_summary").clearNotification("6001");
}

```

```

function OnObservationTypeChange(executionContext) {
    var formContext = executionContext.getFormContext();
    var selectedObservationType = formContext.getAttribute("tt_observationtype").getValue();

    if (selectedObservationType == 206340002)
    {
        formContext.ui.setFormNotification("Please provide a detailed summary of your findings", "INFO", "5001");
        formContext.getControl("tt_summary").setNotification("Please provide a detailed summary of your findings","6001")
    }
    else
    {
        formContext.ui.clearFormNotification("5001");
        formContext.getControl("tt_summary").clearNotification("6001");
    }
}

```

4. Select **File > Save**.
5. Go to <https://make.powerapps.com/><sup>27</sup> and make sure that you are not in the default environment.
6. Select **Solutions** and open the **Class Solution**.
7. Select to open the **Form Field Notification** web resource.

Form Field Notification...	...	tt_FormFieldNotificati	Web Resource
Player Recruiting	...	tt_playerrecruiting	Entity

8. Select **Browse**.

<sup>27</sup> <https://make.powerapps.com/?azure-portal=true>

Content

Type \* Script (JScript) Text Editor

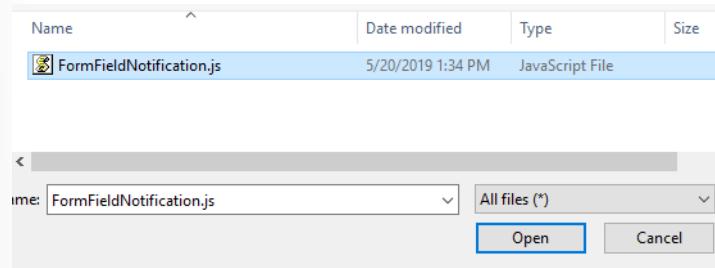
Language

Upload File Browse...

URL

URL [https://devmodules.crm.dynamics.com//WebResources/tt\\_FormFieldNotification](https://devmodules.crm.dynamics.com//WebResources/tt_FormFieldNotification)

9. Select the script file and select **Open**.



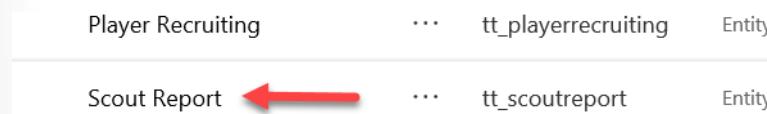
10. Select **Save**.

11. Select **Publish** and wait for the publishing to complete.

12. Close the web resource editor browser tab.

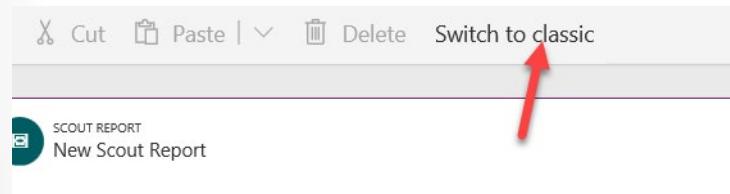
13. Select **Done**.

14. Select to open the **Scout Report** entity.

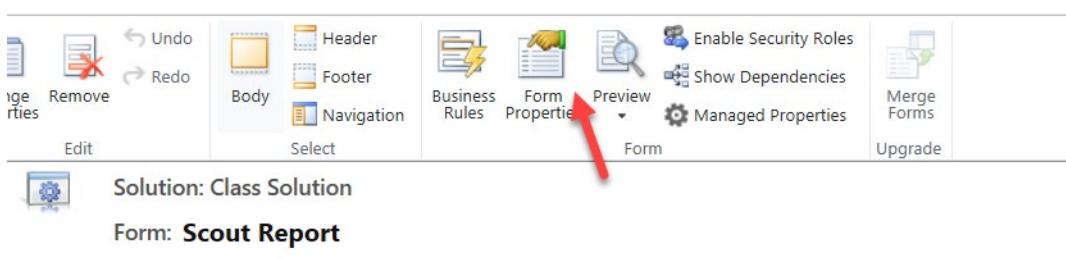


15. Select the **Forms** tab and open the **Information** form.

16. Select **Switch to classic**.



17. Select **Form Properties**.



18. Go to the Event Handlers section, make sure that **Form** is selected for **Control** and **OnLoad** is selected for **Event**, and then select **Add**.

The screenshot shows the 'Event Handlers' dialog. It has two dropdown menus: 'Control' set to 'Form' and 'Event' set to 'OnLoad'. Below these are buttons for 'Add', 'Remove', 'Up', 'Down', 'Edit', and 'Edit Library'. A red arrow points to the 'Add' button.

19. Select **tt\_FormFieldNotification** for **Library**, enter **OnObservationTypeChange** for **Function**, select the **Pass execution context as first parameter** check box, and then select **OK**.

The screenshot shows the 'Details' dialog for adding an event handler. It has tabs for 'Details' and 'Dependencies'. Under 'Details': 'Library' is set to 'tt\_FormFieldNotification', 'Function' is set to 'OnObservationTypeChange', and 'Enabled' is checked. Under 'Parameters': there is a checked checkbox labeled 'Pass execution context as first parameter'. At the bottom are 'OK' and 'Cancel' buttons. A red box highlights the 'Pass execution context as first parameter' checkbox.

20. Select **OK** again.

21. Select **Save**.

22. Select **Publish** and wait for the publishing to complete.

23. Select **Save and Close** to close the classic form editor.

24. Close the preview form editor browser tab.

25. Select **Done**.

Currently editing a form

When you're done editing the form, click Done below to return to the entity. This will refresh the page and fetch your changes.

Done

## Task 6: Test Changes

In this task, you will test the changes that you made in the previous task.

1. Select **Apps** and open the **Scout Zone** application.
2. Select **Scout Reports** and open one of the records.

The screenshot shows the 'Active Scout Reports' list view. On the left is a navigation bar with sections for Home, Recent, Pinned, Scout (Sports Teams, Player Recruitings), Reports (Scout Reports), and a search bar. The main area displays a table with three rows:

Name	Player
Maria A	Maria A
Paul C	Paul C
Paul G	Paul G

A red arrow points to the name 'Paul C' in the second row.

3. The function should now run, and the form notification should be shown. You should only have one notification.

SCOUT REPORT  
Maria A

General Interviews Observation Highlights Summary Playing Style Sportsmanship Related

Name	* Maria A
Player	Maria A
Observation Date	9/15/2015
Observation Type	Other

4. Select the **Summary** tab. The field should have a notification.

General Interviews Observation Highlights **Summary** Playing Style Sportsmanship Related

Please provide a detailed summary of your findings

5. Go back to the **General** tab and change the **Observation Type** to **Game**. The function should run again, and the form notifications should clear.

6. Select the **Summary** tab and make sure that the notification is no longer shown.

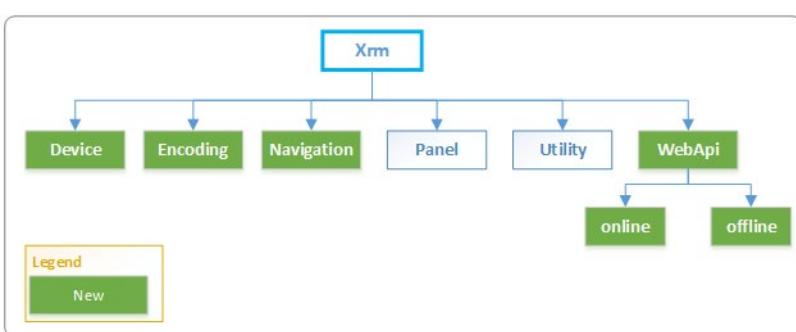
## summary

Client Scripts provide an advanced method for implementing business logic in your model-driven applications when business rules or other declarative means are not feasible. Client Scripts expose a robust API that allows you to automate numerous interactions and interact with your model-driven app's underlying Common Data Service.

# Automate business process flows with client script

## Introduction to conducting global operations with the client API Xrm object

The Client API provides a globally accessible object (Xrm), which is available for use in your code to perform a variety of activities. At a high level, the following graphic illustrates each of the properties and methods that are available. For an in-depth overview of this object, see [Client API Xrm object<sup>28</sup>](#).



### Device object

The Xrm.Device object exposes native device capabilities that are related to mobile device interactions, with the exception of the pickFile method, which is also accessible through web clients. Note that canvas apps also provide an extensible framework for mobile development and should also be considered in these scenarios. The following is a summary of the methods that are available within the Xrm.Device object.

Method	Description
<b>captureAudio</b> ( <a href="https://docs.microsoft.com/powerapps/developer/model-driven-apps/clientapi/reference/xrm-device/captureaudio/?azure-portal=true">https://docs.microsoft.com/powerapps/developer/model-driven-apps/clientapi/reference/xrm-device/captureaudio/?azure-portal=true</a> )	Invokes the device microphone to record audio.
<b>captureImage</b> ( <a href="https://docs.microsoft.com/powerapps/developer/model-driven-apps/clientapi/reference/xrm-device/captureimage/?azure-portal=true">https://docs.microsoft.com/powerapps/developer/model-driven-apps/clientapi/reference/xrm-device/captureimage/?azure-portal=true</a> )	Invokes the device camera to capture an image.
<b>captureVideo</b> ( <a href="https://docs.microsoft.com/powerapps/developer/model-driven-apps/clientapi/reference/xrm-device/capturevideo/?azure-portal=true">https://docs.microsoft.com/powerapps/developer/model-driven-apps/clientapi/reference/xrm-device/capturevideo/?azure-portal=true</a> )	Invokes the device camera to record video.

<sup>28</sup> <https://docs.microsoft.com/powerapps/developer/model-driven-apps/clientapi/clientapi-xrm/?azure-portal=true>

Method	Description
<b>getBarcodeValue</b> ( <a href="https://docs.microsoft.com/powerapps/developer/model-driven-apps/clientapi/reference/xrm-device/getbarcodevalue/?azure-portal=true">https://docs.microsoft.com/powerapps/developer/model-driven-apps/clientapi/reference/xrm-device/getbarcodevalue/?azure-portal=true</a> )	Invokes the device camera to scan the barcode information, such as a product number.
<b>getCurrentPosition</b> ( <a href="https://docs.microsoft.com/powerapps/developer/model-driven-apps/clientapi/reference/xrm-device/getcurrentposition/?azure-portal=true">https://docs.microsoft.com/powerapps/developer/model-driven-apps/clientapi/reference/xrm-device/getcurrentposition/?azure-portal=true</a> )	Returns the current location by using the device geolocation capability.
<b>pickFile</b> ( <a href="https://docs.microsoft.com/powerapps/developer/model-driven-apps/clientapi/reference/xrm-device/pickfile/?azure-portal=true">https://docs.microsoft.com/powerapps/developer/model-driven-apps/clientapi/reference/xrm-device/pickfile/?azure-portal=true</a> )	Opens a dialog box to select files from your computer (web client) or mobile device (mobile clients).

## Encoding object

The Xrm.Encoding object is used to encode and decode XML and HTML strings. This can be valuable when you are interacting with HTML web resources, encoding and decoding query string parameters, and also when you are interacting with FetchXML that contains special characters in a search string.

## Navigation object

The Navigation object provides navigation-related methods that can be used within a model-driven application. Activities such as showing alert, confirm, and error dialogs are commonly not considered to be an optimal user experience, but are available for use nonetheless. In these scenarios, we recommend that you consider form notifications and/or other mechanisms to warn users of an issue. The following is a summary of methods that are available within the Xrm.Navigation object.

Method	Description
<b>openAlertDialog</b> ( <a href="https://docs.microsoft.com/powerapps/developer/model-driven-apps/clientapi/reference/xrm-navigation/openalertdialog/?azure-portal=true">https://docs.microsoft.com/powerapps/developer/model-driven-apps/clientapi/reference/xrm-navigation/openalertdialog/?azure-portal=true</a> )	Displays an alert dialog that contains a message and a button.
<b>openConfirmDialog</b> ( <a href="https://docs.microsoft.com/powerapps/developer/model-driven-apps/clientapi/reference/xrm-navigation/openconfirmdialog/?azure-portal=true">https://docs.microsoft.com/powerapps/developer/model-driven-apps/clientapi/reference/xrm-navigation/openconfirmdialog/?azure-portal=true</a> )	Displays a confirmation dialog box that contains a message and two buttons.
<b>openErrorDialog</b> ( <a href="https://docs.microsoft.com/powerapps/developer/model-driven-apps/clientapi/reference/xrm-navigation/openerrordialog/?azure-portal=true">https://docs.microsoft.com/powerapps/developer/model-driven-apps/clientapi/reference/xrm-navigation/openerrordialog/?azure-portal=true</a> )	Displays an error dialog.
<b>openFile</b> ( <a href="https://docs.microsoft.com/powerapps/developer/model-driven-apps/clientapi/reference/xrm-navigation/openfile/?azure-portal=true">https://docs.microsoft.com/powerapps/developer/model-driven-apps/clientapi/reference/xrm-navigation/openfile/?azure-portal=true</a> )	Opens a file.
<b>openForm</b> ( <a href="https://docs.microsoft.com/powerapps/developer/model-driven-apps/clientapi/reference/xrm-navigation/openform/?azure-portal=true">https://docs.microsoft.com/powerapps/developer/model-driven-apps/clientapi/reference/xrm-navigation/openform/?azure-portal=true</a> )	Opens an entity form or a quick create form.

Method	Description
<b>openUrl</b> ( <a href="https://docs.microsoft.com/powerapps/developer/model-driven-apps/clientapi/reference/xrm-navigation/openurl/?azure-portal=true">https://docs.microsoft.com/powerapps/developer/model-driven-apps/clientapi/reference/xrm-navigation/openurl/?azure-portal=true</a> )	Opens a URL, including file URLs.
<b>openWebResource</b> ( <a href="https://docs.microsoft.com/powerapps/developer/model-driven-apps/clientapi/reference/xrm-navigation/openwebresource/?azure-portal=true">https://docs.microsoft.com/powerapps/developer/model-driven-apps/clientapi/reference/xrm-navigation/openwebresource/?azure-portal=true</a> )	Opens an HTML web resource.

## Panel object

The Xrm.Panel object provides a method to display a webpage on the side pane of a model-driven app form. Note that this feature is currently in preview and will not be covered in detail here. For more information, see [Xrm.Panel<sup>29</sup>](#).

## Utility object

The Xrm.Utility object provides a container for various useful methods. The following is a summary of the methods that are available within the Xrm.Utility object. For more information, see [Xrm.Utility \(Client API reference\)<sup>30</sup>](#)

Method	Description
<b>closeProgress</b> ( <a href="https://docs.microsoft.com/powerapps/developer/model-driven-apps/clientapi/reference/xrm-utility/closeprogressindicator/?azure-portal=true">https://docs.microsoft.com/powerapps/developer/model-driven-apps/clientapi/reference/xrm-utility/closeprogressindicator/?azure-portal=true</a> )	Indicator closes a progress dialog box.
<b>getAllowedStatusTransitions</b> ( <a href="https://docs.microsoft.com/powerapps/developer/model-driven-apps/clientapi/reference/xrm-utility/getallowedstatustransitions/?azure-portal=true">https://docs.microsoft.com/powerapps/developer/model-driven-apps/clientapi/reference/xrm-utility/getallowedstatustransitions/?azure-portal=true</a> )	Returns the valid state transitions for the specified entity type and state code.
<b>getEntityMetadata</b> ( <a href="https://docs.microsoft.com/powerapps/developer/model-driven-apps/clientapi/reference/xrm-utility/getentitymetadata/?azure-portal=true">https://docs.microsoft.com/powerapps/developer/model-driven-apps/clientapi/reference/xrm-utility/getentitymetadata/?azure-portal=true</a> )	Returns the entity metadata for the specified entity.
<b>getGlobalContext</b> ( <a href="https://docs.microsoft.com/powerapps/developer/model-driven-apps/clientapi/reference/xrm-utility/getglobalcontext/?azure-portal=true">https://docs.microsoft.com/powerapps/developer/model-driven-apps/clientapi/reference/xrm-utility/getglobalcontext/?azure-portal=true</a> )	Gets the global context.
<b>getLearningPathAttributeName</b> ( <a href="https://docs.microsoft.com/powerapps/developer/model-driven-apps/clientapi/reference/xrm-utility/getlearningpathattributename/?azure-portal=true">https://docs.microsoft.com/powerapps/developer/model-driven-apps/clientapi/reference/xrm-utility/getlearningpathattributename/?azure-portal=true</a> )	Returns the name of the DOM attribute that is expected by the learning path (guided help) content designer for identifying UI controls in the model-driven apps forms.

<sup>29</sup> <https://docs.microsoft.com/powerapps/developer/model-driven-apps/clientapi/reference/xrm-panel/?azure-portal=true>

<sup>30</sup> <https://docs.microsoft.com/powerapps/developer/model-driven-apps/clientapi/reference/xrm-utility/?azure-portal=true>

Method	Description
<b>getResourceString</b> ( <a href="https://docs.microsoft.com/powerapps/developer/model-driven-apps/clientapi/reference/xrm-utility/getresourcestring/?azure-portal=true">https://docs.microsoft.com/powerapps/developer/model-driven-apps/clientapi/reference/xrm-utility/getresourcestring/?azure-portal=true</a> )	Returns the localized string for a given key that is associated with the specified web resource.
<b>invokeProcessAction</b> ( <a href="https://docs.microsoft.com/powerapps/developer/model-driven-apps/clientapi/reference/xrm-utility/invokeprocessaction/?azure-portal=true">https://docs.microsoft.com/powerapps/developer/model-driven-apps/clientapi/reference/xrm-utility/invokeprocessaction/?azure-portal=true</a> )	Invokes an action based on the specified parameters.
<b>lookupObjects</b> ( <a href="https://docs.microsoft.com/powerapps/developer/model-driven-apps/clientapi/reference/xrm-utility/lookupobjects/?azure-portal=true">https://docs.microsoft.com/powerapps/developer/model-driven-apps/clientapi/reference/xrm-utility/lookupobjects/?azure-portal=true</a> )	Opens a lookup control to select one or more items.
<b>refreshParentGrid</b> ( <a href="https://docs.microsoft.com/powerapps/developer/model-driven-apps/clientapi/reference/xrm-utility/refreshparentgrid/?azure-portal=true">https://docs.microsoft.com/powerapps/developer/model-driven-apps/clientapi/reference/xrm-utility/refreshparentgrid/?azure-portal=true</a> )	Refreshes the parent grid that contains the specified record.
<b>showProgressIndicator</b> ( <a href="https://docs.microsoft.com/powerapps/developer/model-driven-apps/clientapi/reference/xrm-utility/showprogressindicator/?azure-portal=true">https://docs.microsoft.com/powerapps/developer/model-driven-apps/clientapi/reference/xrm-utility/showprogressindicator/?azure-portal=true</a> )	Displays a progress dialog with the specified message.

## Xrm.WebApi object

The Xrm.WebApi object provides properties and methods to use the Web API for traditional CRUD operations within a client script. The following is a summary of the methods that are available within the Xrm.WebApi object. For more information, see **Xrm.WebApi (Client API reference)**<sup>31</sup>.

Method	Description
<b>createRecord</b> ( <a href="https://docs.microsoft.com/powerapps/developer/model-driven-apps/clientapi/reference/xrm-webapi/createrecord/?azure-portal=true">https://docs.microsoft.com/powerapps/developer/model-driven-apps/clientapi/reference/xrm-webapi/createrecord/?azure-portal=true</a> )	Creates an entity record.
<b>deleteRecord</b> ( <a href="https://docs.microsoft.com/powerapps/developer/model-driven-apps/clientapi/reference/xrm-webapi/deleterecord/?azure-portal=true">https://docs.microsoft.com/powerapps/developer/model-driven-apps/clientapi/reference/xrm-webapi/deleterecord/?azure-portal=true</a> )	Deletes an entity record.
<b>retrieveRecord</b> ( <a href="https://docs.microsoft.com/powerapps/developer/model-driven-apps/clientapi/reference/xrm-webapi/retrieverecord/?azure-portal=true">https://docs.microsoft.com/powerapps/developer/model-driven-apps/clientapi/reference/xrm-webapi/retrieverecord/?azure-portal=true</a> )	Retrieves an entity record.
<b>retrieveMultipleRecords</b> ( <a href="https://docs.microsoft.com/powerapps/developer/model-driven-apps/clientapi/reference/xrm-webapi/retrievemultiplerecords/?azure-portal=true">https://docs.microsoft.com/powerapps/developer/model-driven-apps/clientapi/reference/xrm-webapi/retrievemultiplerecords/?azure-portal=true</a> )	Retrieves a collection of entity records.

<sup>31</sup> <https://docs.microsoft.com/powerapps/developer/model-driven-apps/clientapi/reference/xrm-webapi/?azure-portal=true>

Method	Description
<b>updateRecord</b> ( <a href="https://docs.microsoft.com/powerapps/developer/model-driven-apps/clientapi/reference/xrm-webapi/updaterecord/?azure-portal=true">https://docs.microsoft.com/powerapps/developer/model-driven-apps/clientapi/reference/xrm-webapi/updaterecord/?azure-portal=true</a> )	Updates an entity record.
<b>isAvailableOffline</b> ( <a href="https://docs.microsoft.com/powerapps/developer/model-driven-apps/clientapi/reference/xrm-webapi/isavailableoffline/?azure-portal=true">https://docs.microsoft.com/powerapps/developer/model-driven-apps/clientapi/reference/xrm-webapi/isavailableoffline/?azure-portal=true</a> )	Returns a Boolean value that indicates whether an entity is present in a user's profile and is currently available for use in offline mode.
<b>execute</b> ( <a href="https://docs.microsoft.com/powerapps/developer/model-driven-apps/clientapi/reference/xrm-webapi/online/execute/?azure-portal=true">https://docs.microsoft.com/powerapps/developer/model-driven-apps/clientapi/reference/xrm-webapi/online/execute/?azure-portal=true</a> )	Run a single action, function, or CRUD operation.
<b>executeMultiple</b> ( <a href="https://docs.microsoft.com/powerapps/developer/model-driven-apps/clientapi/reference/xrm-webapi/online/executemultiple/?azure-portal=true">https://docs.microsoft.com/powerapps/developer/model-driven-apps/clientapi/reference/xrm-webapi/online/executemultiple/?azure-portal=true</a> )	Run a collection of action, function, or CRUD operations.

## Client scripting best practices

While developing client scripts might be a seemingly easy task to a seasoned JavaScript developer, it is important to keep best practices in mind when considering their use.

## Business rules vs. client script

Business rules is an available feature in model-driven applications that enable users to conduct frequently required actions on a form based on certain requirements and conditions. Specifically, business rules can do the following tasks:

- Set field values
- Clear field values
- Set field requirement levels
- Show or hide fields
- Enable or disable fields
- Validate data and show error messages
- Create business recommendations based on business intelligence.

While the available framework is robust, some scenarios might exist where business rules can't fully achieve a given requirement. Luckily in these cases, client scripting can narrow the differences between business rules and a more custom requirement. This unit explores some common limitations that users experience with business rules where client script might still be a better solution.

## Referencing related data

If you need to reference data that is contained in a related entity, such as the address of the primary contact on an account, you'll need to use client script and the Web API.

## Logic needs to run on the Form Save event

Business rules only run on form load and on the change of a field. If you need business logic to run On Save, you'll need to use client script to do so.

## Complex conditions

If your condition has multiple `and` statements, or requires `or` statements, it might be more efficient to write these types of conditions in client script than through business rules.

## Clearing values of form data

Clearing data from a form field isn't easily accomplished by means of a business rule. While there are workarounds that can be achieved by assigning "dummy" fields that always contain NULL values, this is more elegantly accomplished through client script.

## Calculated fields vs. client script

Fields are calculated on retrieve, so client changes won't show until data is refreshed. If you want data to update instantly on the form, client script would be the preferred method.

## Coding Standards and Best Practices

The following sections cover the best practices and standards for coding.

### Define unique Script function names

The functions that you write will most likely be loaded into a form with many other libraries. If another library contains a function that has the same name as the one that you provide, whichever function is loaded last is the one that will be defined for that page. To avoid a potential conflict, make sure that you use unique function names. You can use the following strategies if you or your organization does not have one already established:

- **Unique Function Prefix** - Define each of your functions by using the standard syntax with a consistent name that includes a unique naming convention, as shown in the following example.

```
function MyUniqueName_performMyAction()
{
    // Code to perform your action.
```

```
}
```

- **Namespaced Library Names** - Associate each of your functions with a script object to create a kind of namespace to use when you call your functions, as shown in the following example.

```
//If the MyUniqueName namespace object isn't defined, create it.  
if (typeof (MyUniqueName) == "undefined")  
{ MyUniqueName = {}; }  
// Create Namespace container for functions in this library;  
MyUniqueName.MyFunctions = {  
    performMyAction: function(){  
        // Code to perform your action.  
        //Call another function in your library  
        this.anotherAction();  
    },  
    anotherAction: function(){  
        // Code in another function  
    }  
};
```

When you use your function, you can specify the full name, as shown in the following example.

```
MyUniqueName.MyFunctions.performMyAction();
```

If you call a function within another function, you can use the **this** keyword as a shortcut to the object that contains both functions. However, if your function is being used as an event handler, the **this** keyword will refer to the object that the event is occurring on.

## Avoid unsupported methods

While a few third-party resources on the internet might suggest or have examples of how you can use unsupported methods to perform certain actions, this is highly unadvisable. These methods cannot be guaranteed to work in future versions of your app and might contribute to instability of your implementation.

## Avoid using jQuery in forms and ribbon commands

While accessible through webpage resources, jQuery is not supported within form scripts or ribbon commands. It is highly advised that you restrict scripts to only use methods that are found in the Client API object model. For more information, see **Understand the Client API object model**<sup>32</sup>.

---

<sup>32</sup> <https://docs.microsoft.com/powerapps/developer/model-driven-apps/clientapi/understand-clientapi-object-model/?azure-portal=true>

## Write non-blocking code

When performing queries or process-intensive activities, make sure to use asynchronous patterns to ensure a non-blocking user experience.

## Write code for multiple browsers

Ensure that any scripts that you write have been tested to work against all browsers that are supported by model-driven apps. For more information, see **Supported web browsers and mobile devices**<sup>33</sup>.

## Debugging client script

Nearly every browser has some sort of debugging extension that allows for debugging of client scripts. The following toolsets are those that are frequently used to perform debugging operations:

- Microsoft Edge (through F12 Developer Tools). For more information, see Using the **F12 developer tools guide**<sup>34</sup>.
- Internet Explorer (through F12 Developer Tools)
- Google Chrome (through F12 Developer Tools)
- Mozilla Firefox (using Firebug)
- Apple Safari (using Web Inspector)

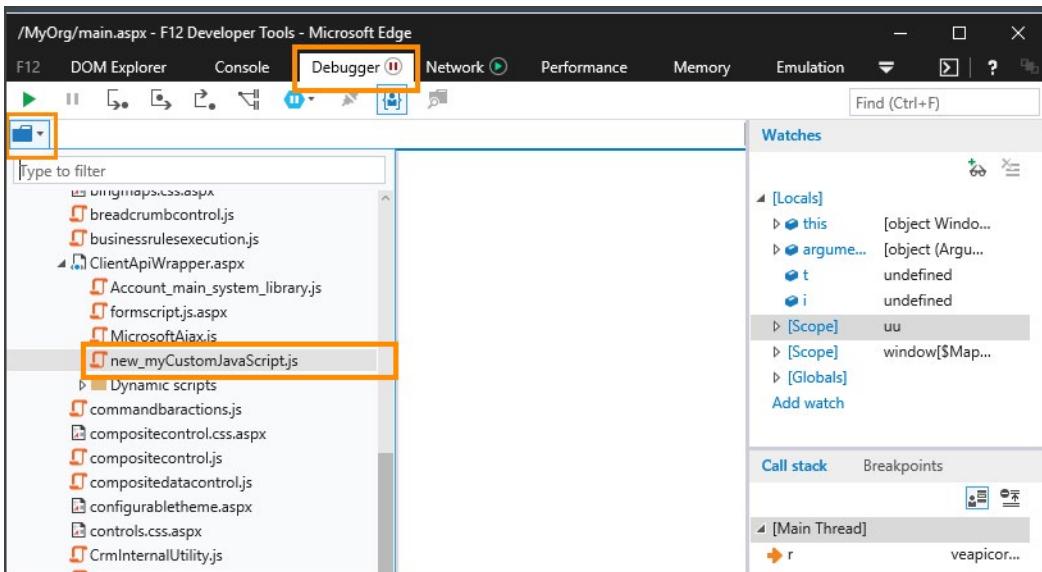
## Viewing script resources

When your model-driven app form page is loaded, all client script libraries are loaded alongside the webpage as individual script resources. Given the volume of script resources that are needed to run a model-driven app, it can be difficult to locate a given file that you might want to debug against. When using debugging tools like Microsoft Edge, we commonly recommend that you note your file name and use the tooling's search capabilities to locate your script files.

---

<sup>33</sup> <https://docs.microsoft.com/power-platform/admin/supported-web-browsers-and-mobile-devices/?azure-portal=true>

<sup>34</sup> <https://docs.microsoft.com/microsoft-edge/f12-devtools-guide/?azure-portal=true>



## Write messages to the console

Using the **window.alert** method<sup>35</sup> when debugging JavaScript is still a common way to troubleshoot code in the application. But now that all modern browsers provide easy access to debugging tools, it is not a best practice, especially when others might be using the application that you are debugging.

Consider writing your messages to the console instead. The following is a small function that you can add to your libraries to help send any messages that you want to view to the console when it is open.

```
function writeToConsole(message)
{
    if (typeof console != 'undefined') {
        console.log(message);
    }
}
```

Unlike using the alert method, if you forget to remove any code that uses this function, people who are using the application will not see your messages.

## Use Fiddler Auto-Responder to replace web resource content

Constantly editing web resources when they are under development can prove to be difficult and time-intensive when you have to republish the files within a solution upon every edit. To improve efficiency, consider using a tool like Auto-Responder in Telerik Fiddler to replace

<sup>35</sup> [https://msdn.microsoft.com/library/ie/ms535933\(v=vs.85\).aspx?azure-portal=true](https://msdn.microsoft.com/library/ie/ms535933(v=vs.85).aspx?azure-portal=true)

content of a web resource with content from a local file, rather than uploading it and republishing each time. Several other third-party tools that also support live editing can be considered. For more information on how to install and configure Fiddler Auto-Responder, see, **Script web resource development using Fiddler Auto-Responder**<sup>36</sup>.

## Exercise - Develop an HTML web resource

### Exercise 1: Building an HTML web resource

In this exercise, you will use Visual Studio Code to create a custom text box that will insert a timestamp whenever the user makes a new entry. After you create the web resource, you will deploy it to your model-driven app and then add it to the Scout Report form. Each exercise consists of a scenario and learning objectives; the scenario describes the purpose of the exercises, while the objectives are listed and have bullet points.

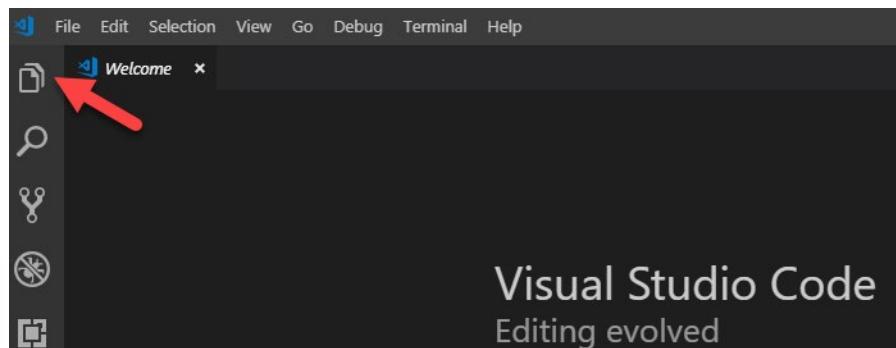
[!NOTE]

To complete this exercise, you need to have the solutions installed as described in **Install a Package Deployer package**<sup>37</sup> lab exercise from a previous module.

### Task 1: Create folders and files

In this task, you will create and deploy a web resource called Time Stamped Textbox.

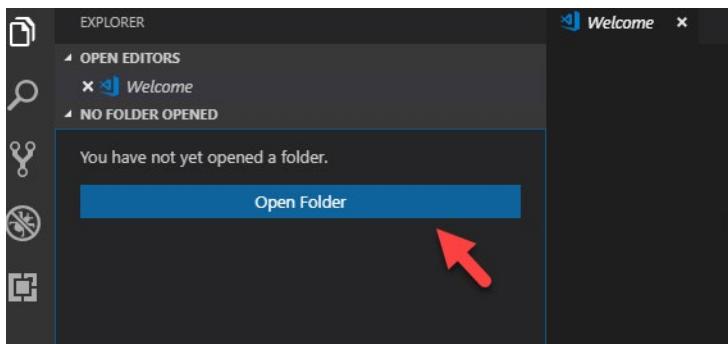
1. Start Visual Studio Code and select **Explorer**.



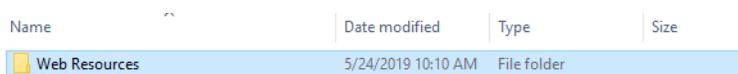
2. Select **Open Folder**.

<sup>36</sup> <https://docs.microsoft.com/powerapps/developer/model-driven-apps/streamline-javascript-development-fiddler-autoresponder/?azure-portal=true>

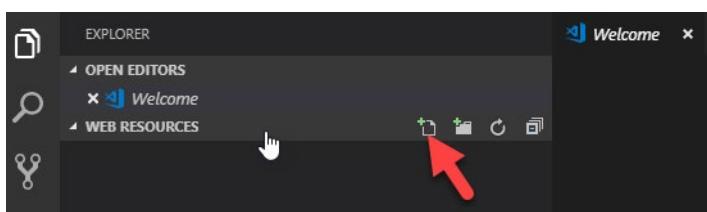
<sup>37</sup> <https://docs.microsoft.com/learn/modules/use-developer-tools-extend/4-exercise/?azure-portal=true>



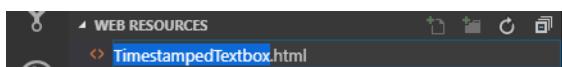
3. Right-click and create a new folder.
4. Name the folder **Web Resources** and select the **Select Folder** button.



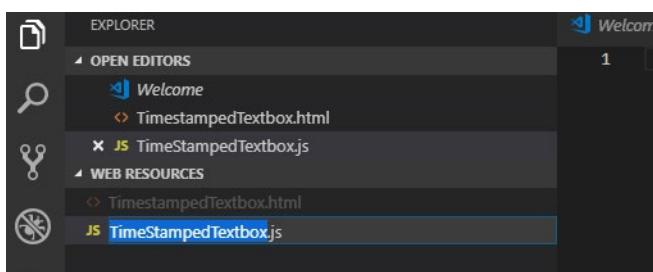
5. Hover over the folder that you created and select **New File**.



6. Name the file **TimestampedTextBox.html**.



7. Hover over the folder and select **New File** again.
8. Name the new file **TimeStampedTextBox.js**.



9. Open the **TimestampedTextBox.html** file and add the HTML markup, as follows.

[NOTE]

tt\_TimestampedTextboxScript is what you will name the script file when you upload as a web resource, and timestampedTextarea is the ID that you will refer to when you want to interact with textarea from the script.

```
<html>

<head>

<script src="tt_TimestampedTextboxScript"
type="text/javascript"></script>

</head>

<body on>

<textarea id="timestampedTextarea" style="width:100%; height:130px;
border:0 none;"></textarea>

</body>

</html>
```

```
1 <html>
2 <head>
3 | <script src="tt_TimestampedTextboxScript" type="text/javascript"></script>
4 </head>
5 <body on>
6 | <textarea id="timestampedTextarea" style="width:100%; height:130px; border:0 none;"></textarea>
7 </body>
8 </html>
```

10. Open the script file and paste the following snippet. This snippet will create a global variable for the textArea and add event listeners for the focus and changed events.

```
var textArea;

window.onload = function(){

textArea = document.getElementById("timestampedTextarea");

textArea.addEventListener("focus", onFocus);

textArea.addEventListener("change", onChange);

}
```

```
2 var textArea;
3
4 window.onload = function(){
5     textArea = document.getElementById("timestampedTextarea");
6     textArea.addEventListener("focus", onFocus);
7     textArea.addEventListener("change", onChange);
8 }
```

11. Add the event handlers for onFocus and onChange.

```
function onFocus(){

}

function onChange() {

}

2  var textArea;
3
4  window.onload = function(){
5      textArea = document.getElementById("timestampedTextarea");
6      textArea.addEventListener("focus", onFocus);
7      textArea.addEventListener("change", onChange);
8  }
9
10 function onFocus(){
11 }
12
13 function onChange() {
14 }
15
16 }
```

12. Add the function that will build the Date Time and return it as readable text. Add the following function to the script file.

```
function getTime() {

    var months = ["Jan", "Feb", "Mar", "Apr", "May", "Jun",
    "Jul", "Aug", "Sep", "Oct", "Nov", "Dec"];

    var days = ["Sunday", "Monday", "Tuesday", "Wednesday",
    "Thursday", "Friday", "Saturday"];

    var d = new Date();

    var day = days[d.getDay()];

    var hr = d.getHours();

    var min = d.getMinutes();

    var ampm = "am";

    if (min < 10) {

        min = "0" + min;

    }

    if (hr > 12) {
```

```
hr -= 12;  
  
ampm = "pm";  
  
}  
  
var date = d.getDate();  
  
var month = d.getMonth() + 1;  
  
var year = d.getFullYear();  
  
if (month < 10) {  
  
    month = "0" + month;  
  
}  
  
return month + "-" + date + "-" + year + " " + hr + ":" + min +  
ampm + " -";  
  
}
```

```
20    }  
21  
22    function getTime() {  
23        var months = ["Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul", "Aug", "Sep", "Oct", "Nov", "Dec"];  
24        var days = ["Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday"];  
25        var d = new Date();  
26        var day = days[d.getDay()];  
27        var hr = d.getHours();  
28        var min = d.getMinutes();  
29        var ampm = "am";  
30        if (min < 10) {  
31            min = "0" + min;  
32        }  
33        if (hr > 12) {  
34            hr -= 12;  
35            ampm = "pm";  
36        }  
37        var date = d.getDate();  
38        var month = d.getMonth() + 1;  
39        var year = d.getFullYear();  
40        if (month < 10) {  
41            month = "0" + month;  
42        }  
43        return month + "-" + date + "-" + year + " " + hr + ":" + min + ampm + " -";  
44    }
```

13. Add the following snippet inside the OnFocus function. This snippet will call the getDate function and add the returned value to the text area.

```
var dateTimeText = getTime();  
  
var index = dateTimeText.length;  
  
textArea.value = dateTimeText + "\n" + textArea.value;
```

```
textArea.selectionStart = index;  
  
textArea.selectionEnd = index;  
  
9  
10 function onFocus(){  
11     var dateTimeText = getDateTime();  
12     var index = dateTimeText.length;  
13     textArea.value = dateTimeText + "\n" + textArea.value;  
14     textArea.selectionStart = index;  
15     textArea.selectionEnd = index;  
16 }
```

## Task 2: Add web resources

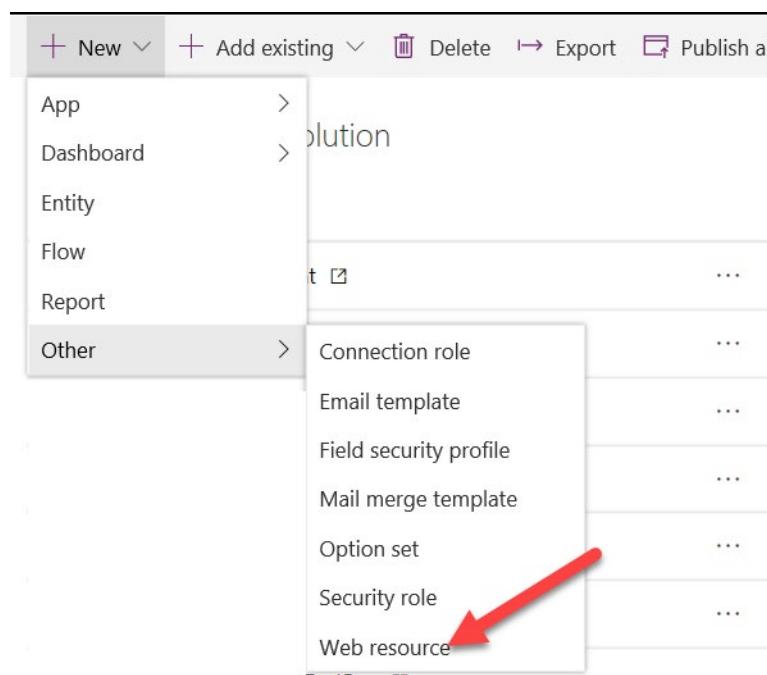
In this task, you will load the web resources to the Class.

1. Go to **Power Apps**<sup>38</sup> and make sure that you are not in the default environment.
2. Select **Solutions** and select to open **Class Solution**.

Display name	Created ↓
PowerApps Checker	... 5/4/2019
Scout Zone	... 4/29/2019
Class Solution	... 4/29/2019

3. Select **New > Other > Web Resource**.

<sup>38</sup> <https://make.powerapps.com/?azure-portal=true>



4. Enter **TimestampTextareaHTML** for **Name**, **Timestamp Text Area HTML** for **Display Name**, select **Webpage (HTML)** for **Type**, and then select **Browse**.

Solution: Class Solution  
**Web Resource: New**

General Dependencies

**General**

Name \* tt\_ TimestampTextareaHTML

Display Name Timestamp Text Area HTML

Description

**Content**

Type \* Webpage (HTML)

Language

Upload File Browse...

5. Select the HTML resource that you created and select **Open**.

Name: TimestampedTextbox.html

All files (\*)

Open Cancel

6. Select **Save**.



7. Select **Publish** and wait for the publishing to complete.  
 8. Close the web resource editor browser tab.  
 9. Select **Done**.

Currently editing a web resource

When you're done editing the web resource, click Done below to return to the page. This will refresh the page and fetch your changes.

**Done**

10. Select **New > Other > Web Resource** again.  
 11. Enter **TimestampedTextBoxScript** for **Name**, **Time Stamped Textbox Script** for **Display Name**, select **Script (Jscript)** for **Type**, and then select **Browse**. The name must match the script tag that you added to the HTML header.

Solution: Class Solution  
**Web Resource: New**

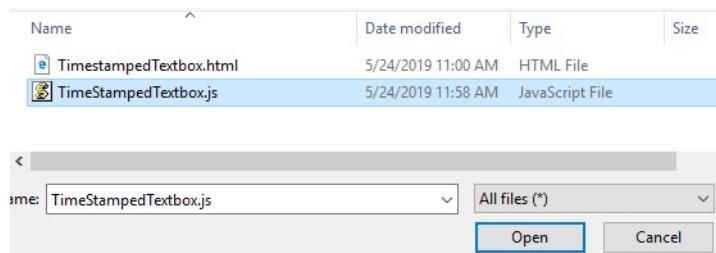
**General**

Name *	tt_ TimestampedTextBoxScript
Display Name	Time Stamped Textbox Script
Description	

**Content**

Type *	Script (JScript)	Text Editor
Language		
Upload File		Browse...

12. Select the script file that you created and select **Open**.



13. Select **Save**.
14. Select **Publish** and wait for the publishing to complete.
15. Close the web resource editor browser tab.
16. Select **Done**.

### Task 3: Add a text area to a form

In this task, you will add the text area to the **Scout Report** form.

1. While still in the **Class Solution**, locate and select to open the **Scout Report** entity.

Solutions > Class Solution

Display name	Name	Type
Player Recruiting	tt_playerrecruiting	Entity
Scout Report	tt_scoutreport	Entity

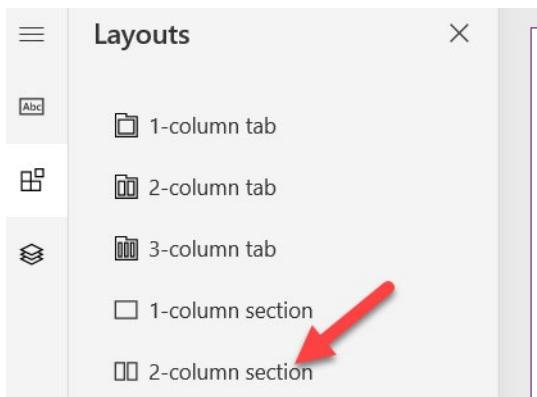
2. Select the **Forms** tab and select to open the **Information** form.

Solutions > Class Solution > Scout Report

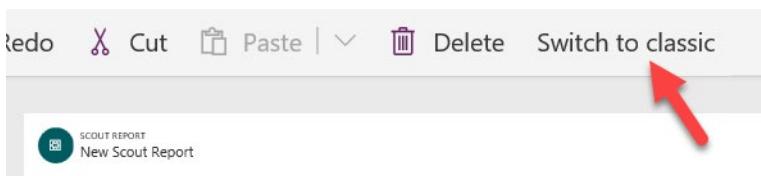
Fields   Relationships   Business rules   Views   **Forms**   Dashboards   Chart:

Model-driven
Name ↴
Information <input checked="" type="checkbox"/>

3. Select the **Layouts** tab and select **2-Column Section**.



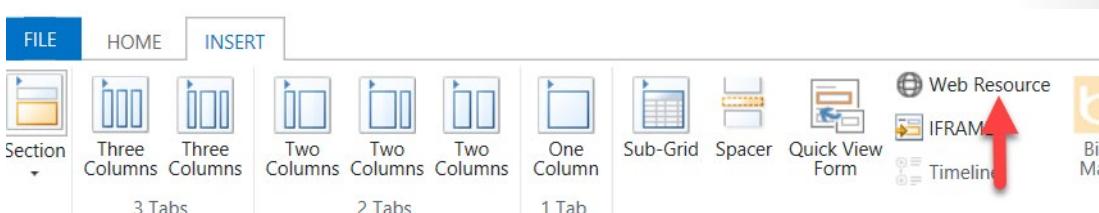
4. A new section will be added to the form. Select **Save**.
5. Select **Publish** and wait for the publishing to complete.
6. Select **Switch to classic**.



7. Scroll down and select **New Section**.



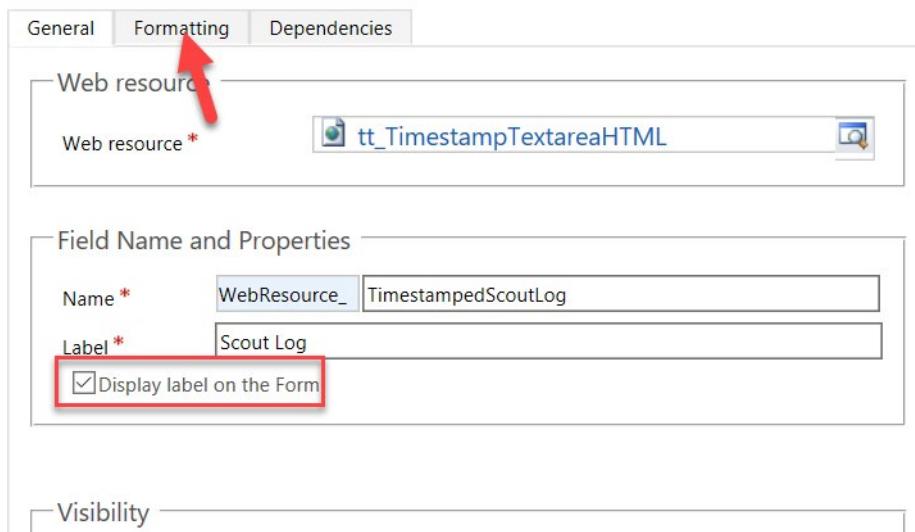
8. Select the **Insert** tab and select **Web Resource**.



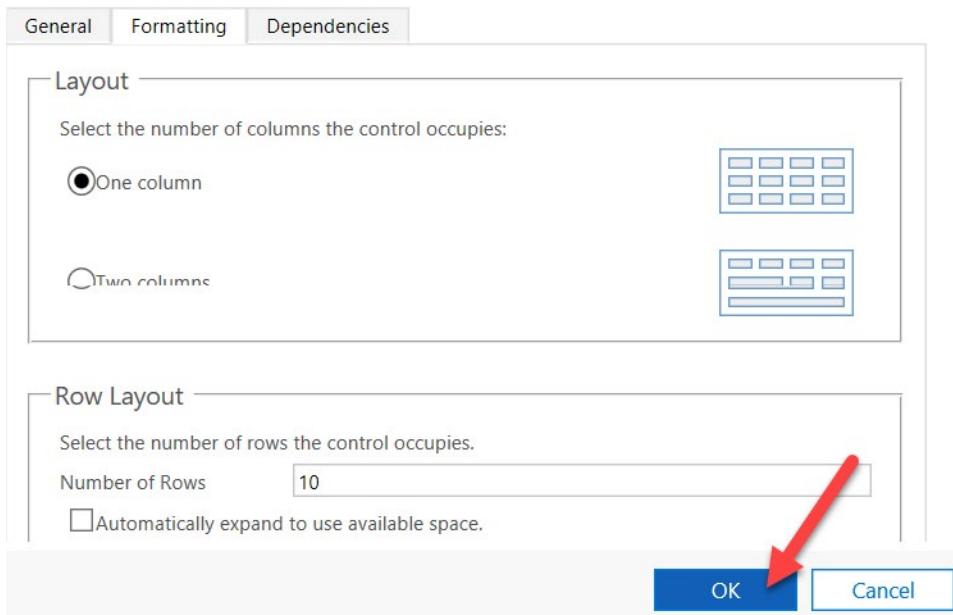
9. Enter **tt** and then select the **Lookup** button.



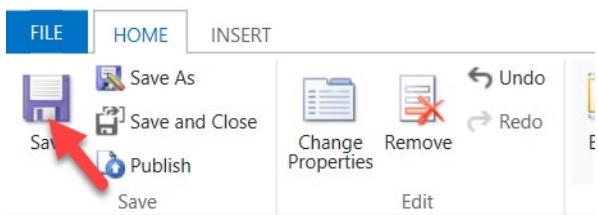
10. Select the HTML web resource that you added to the Class solution.
11. Enter **TimestampedScoutLog** for **Name, Scout Log** for **Label**, check the **Display label on the Form** check box, and select the **Formatting** tab.



12. Select **One Column**, enter **10** for **Number of Rows**, and then select **OK**.



13. Select the **Home** tab and select **Save**.



14. Select **Publish** and wait for publishing to complete.

15. Close the Classic form edition browser tab.

16. Close the Preview form editor browser tab.

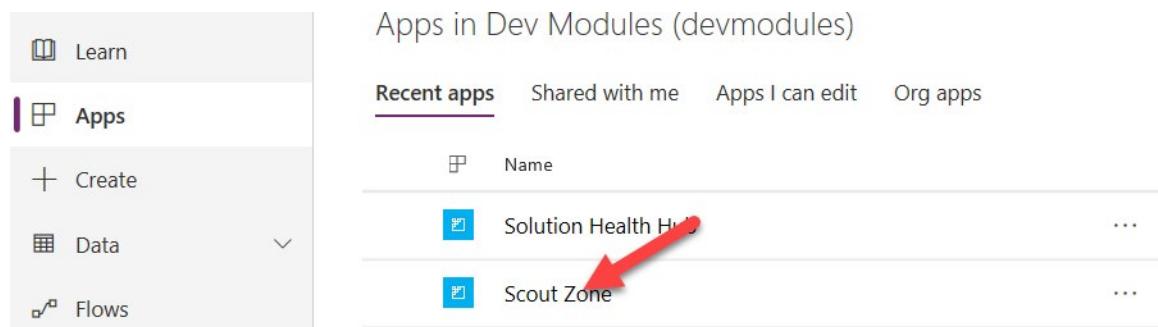
17. Select **Done**.

Currently editing a form

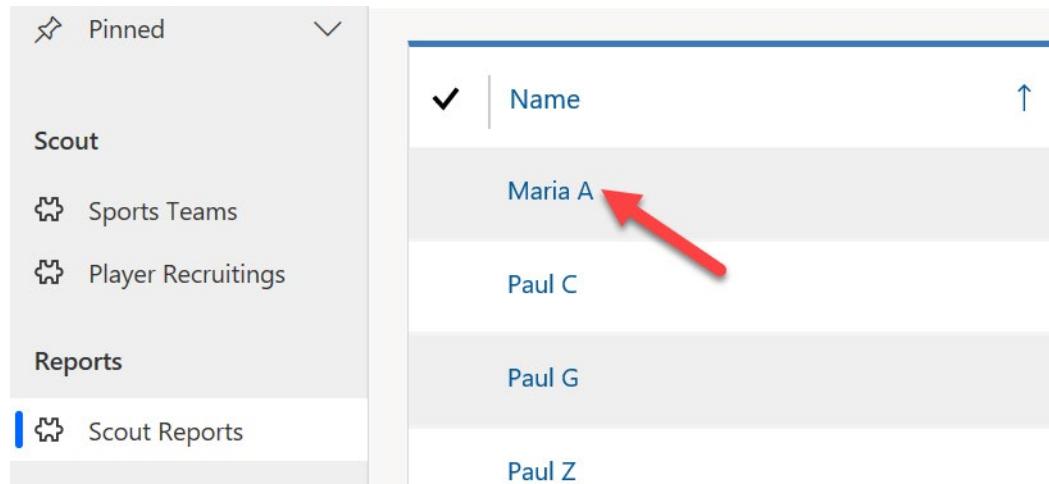
When you're done editing the form, click Done below to return to the entity. This will refresh the page and fetch your changes.

**Done**

18. Select **Apps** and select to open the **Scout Zone** application.



19. Select **Scout Reports** and open one of the records.



20. Locate the new section and select the text area that you added.



21. **Timestamped** should be added to the text area. Type some text and select outside of the text area.

### New Section

#### Scout Log

05-24-2019 1:00pm - Test



22. Select the text area again. Another timestamp should be added.

### New Section

#### Scout Log

05-24-2019 1:04pm - |  
05-24-2019 1:03pm - Test

## Exercise 2: Update the form

In this exercise, you will add a new field to the **Scout Report** entity, pass the name of this field as a parameter in the web resource, update the script to parse for the field name, and then update the field when the value of the text area changes. Each exercise consists of a scenario and learning objectives; the scenario describes the purpose of the exercises, while the objectives are listed and have bullet points.

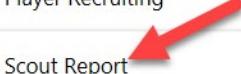
### Task 1: Add a new field

In this task, you will hide the new section label and add a new field to the **Scout Report** entity.

1. Go to **Power Apps**<sup>39</sup> and make sure that you are not in the default environment.
2. Select **Solutions** and select to open the **Class** solution.
3. Select to open the **Scout Report** entity.

Player Recruiting

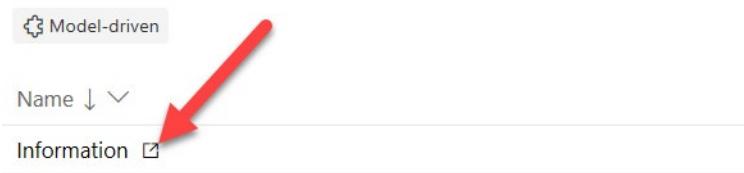
Scout Report



4. Select the **Forms** tab and open the **Information** form.

<sup>39</sup> <https://make.powerapps.com/?azure-portal=true>

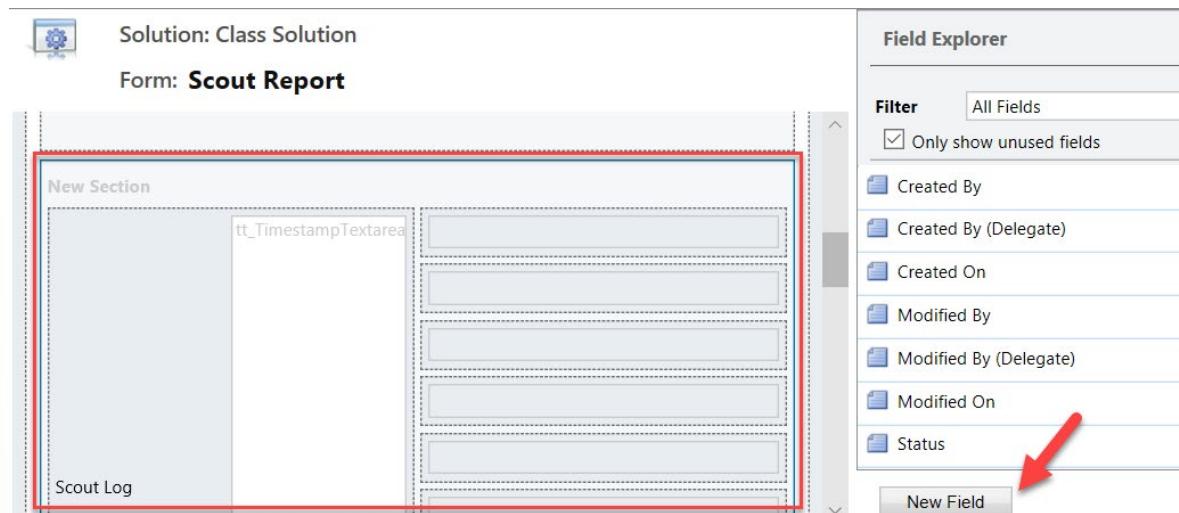
## Solutions &gt; Class Solution &gt; Scout Report

[Fields](#) [Relationships](#) [Business rules](#) [Views](#) [Forms](#) [Dashboards](#)

5. Select **New Section**, go to the **Properties** pane, and select the **Hide label** check box.



6. Select **Save**.
7. Select **Publish** and wait for the publishing to complete.
8. Select **Switch to classic**.
9. Select **New Section**, go to the **Field Explorer** pane, and select **New Field**.



10. Enter **Scout Log** for **Display Name**, select **Multiple Lines of Text** for **Data Type**, and select **Save and Close**. You will pass lowercase of this field name as a parameter.

**New for Scout Report**

**Common**

- Information
- Business Rules

**General**

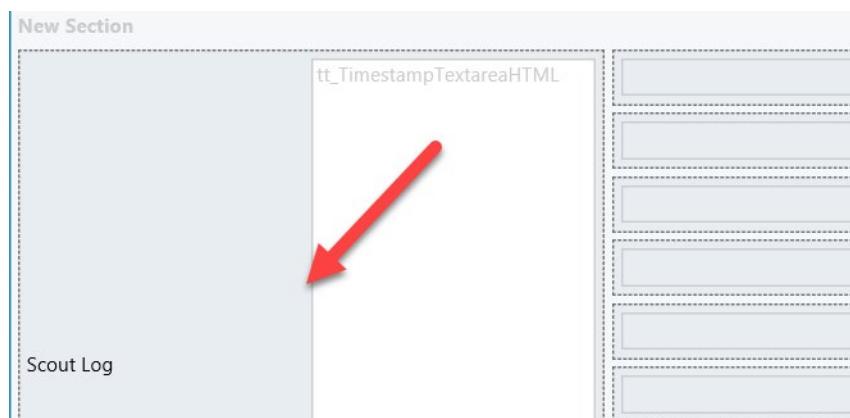
**Schema**

Display Name *	Scout Log	Field Requirement *	Optional
Name *	tt_ScoutLog	Searchable	Yes
Field Security	<input type="radio"/> Enable <input checked="" type="radio"/> Disable <small>⚠ Enabling field security? <a href="#">What you need to know</a></small>		
Auditing *	<input checked="" type="radio"/> Enable <input type="radio"/> Disable <small>⚠ This field will not be audited until you enable auditing on the entity.</small>		
Description			
Appears in global filter in interactive experience	<input type="checkbox"/>	Sortable in interactive experience dashboard	<input type="checkbox"/>
For information about how to interact with entities and fields programmatically, see the <a href="#">Microsoft Dynamics 365 SDK</a>			

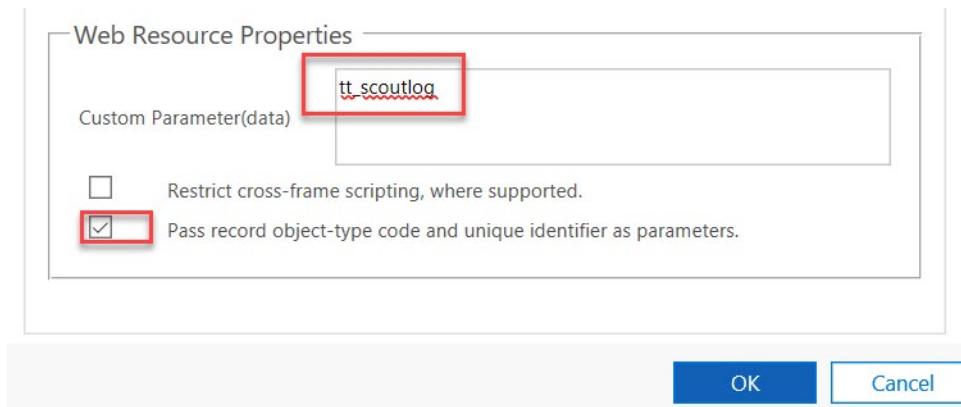
**Type**

Data Type *	Multiple Lines of Text
Maximum Length *	2000

11. Double-click the **Scout Log** web resource.



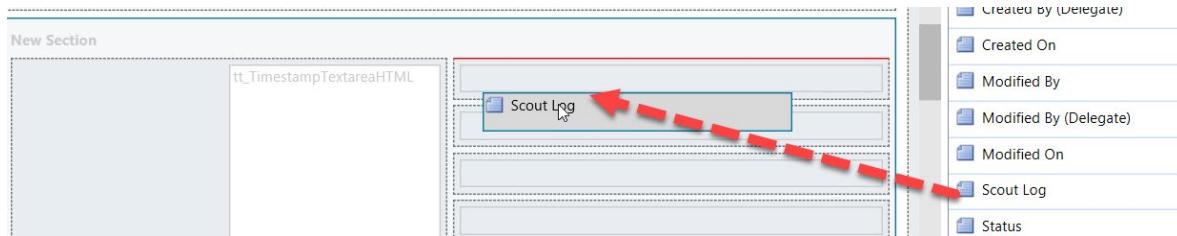
12. Scroll down to the **Web Resource Properties** section, type the name of the field that you created, select the **Pass Record ....** check box, and then select **OK**.



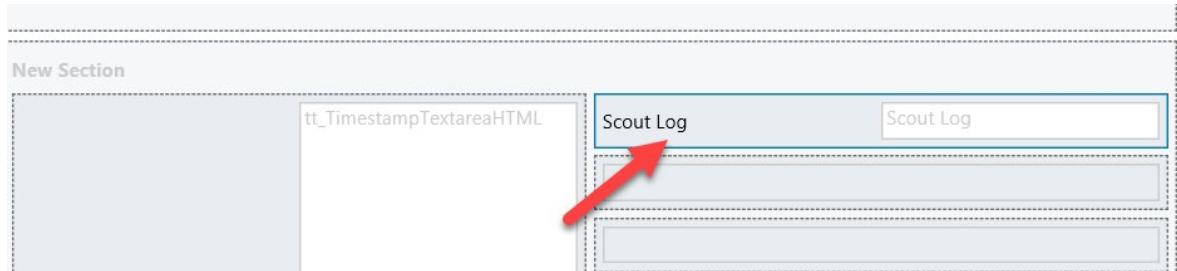
13. Select **Save**.

14. Refresh the solution explorer browser tab.

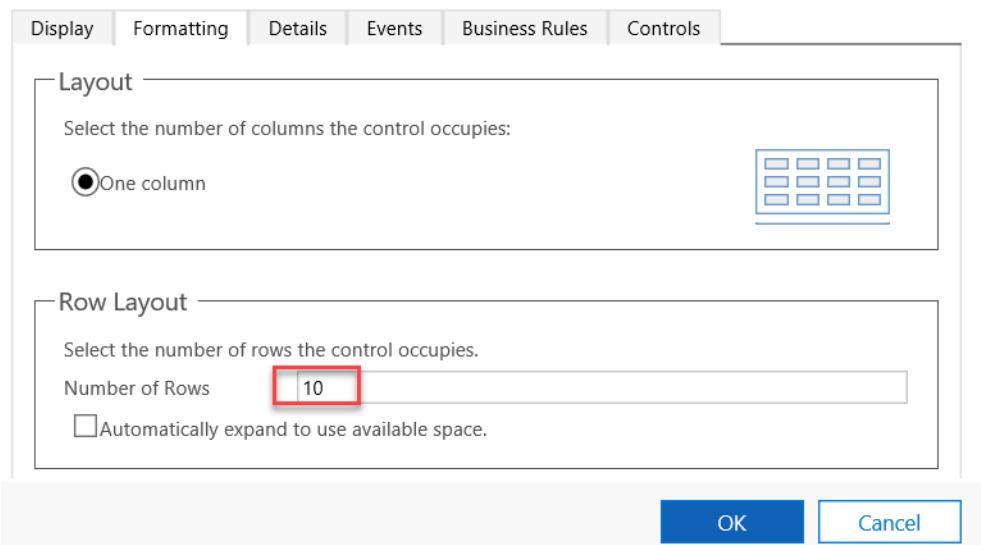
15. Go to the **Field Explorer** pane and drag the **Scout Log** field to the right column of the new section.



16. Double-click the **Scout Log** field.



17. Select the **Formatting** tab, enter **10** for **Number of Rows**, and then select **OK**.



18. Select **Save**.
19. Select **Publish** and wait for publishing to complete.
20. Close the Classic form editor browser tab.
21. Close the Preview form editor browser tab.
22. Click **Done**.

## Task 2: Update web resource

In this task, you will update the script.

1. Go back to Visual Studio Code and open the java script file.

The screenshot shows the Visual Studio Code interface with the 'TimeStampsTextbox.js' file open in the editor. The file contains a JavaScript script for a timestamped text box. The code includes functions for handling focus, change events, and getting the current date and time. The 'TimeStampsTextbox.js' file is selected in the 'EXPLORER' sidebar, indicated by a blue selection bar and a red arrow pointing to it.

```

1  var textArea;
2
3  window.onload = function(){
4      }
5
6  function onFocus() {
7      }
8
9  function onChange() {
10 }
11
12 function getDate() {
13 }
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45

```

2. Add the following variables to the top of the file.

```
var xrm;
```

```
var fieldName;
```

```
1
2  var xrm;
3  var fieldName;
4  var textArea;
5
6  window.onload = function(){...
10 }
11
```

3. Add the following function to the script file. This function will parse the query string and return the parameters.

```
function getQueryParams(qs) {
    qs = qs.split("+").join(" ");
    const params = {};
    re = /[?&]?([^\=]+)=([^\&]*)/g;
    let tokens = re.exec(qs);
    while (tokens) {
        params[decodeURIComponent(tokens[1])] =
            decodeURIComponent(tokens[2]);
        tokens = re.exec(qs);
    }
    return params;
}
```

```
23
24  function getDate() { ...
46  }
47
48  function getQueryParams(qs) {
49      qs = qs.split("+").join(" ");
50
51      const params = {};
52      re = /[?&]?([^\=]+)=([^\&]*)/g;
53      let tokens = re.exec(qs);
54
55      while (tokens) {
56          params[decodeURIComponent(tokens[1])] = decodeURIComponent(tokens[2]);
57          tokens = re.exec(qs);
58      }
59      return params;
60 }
```

4. Go to the **onload** function and add the following snippet. This snippet will call the `getQueryParams` method, check if the parameter was passed, and then set the text area field value or set the `fieldName`.

```

const params = getQueryParams(window.location.search);

if (!params.data) {

    textArea.value = "No Name for field found in parameters";

    return;

}

fieldName = params.data;

```

```

window.onload = function(){
    textArea = document.getElementById("timestampedTextarea");
    textArea.addEventListener("focus", onFocus);
    textArea.addEventListener("change", onChange);

    const params = getQueryParams(window.location.search);
    if (!params.data) {
        textArea.value = "No Name for field found in parameters";
        return;
    }
    fieldName = params.data;
}

```

5. Add the following snippet after the previous one. This snippet will set the value of the text area to the value of the field that was passed in web resource parameter.

```

xrm = window.parent.Xrm;

textArea.value = xrm.Page.getAttribute(fieldName).getValue();

```

```

    textArea.value = "No Name for field found in parameters";
    return;
}
fieldName = params.data;

xrm = window.parent.Xrm;
textArea.value = xrm.Page.getAttribute(fieldName).getValue();
}

```

6. Go to the **onChange** function and add the following snippet. This snippet will update the field that was passed in the web resource parameter to the value of the text area.

```
xrm.Page.getAttribute(fieldName).setValue(textArea.value);
```

```

function onFocus(){...}
}

function onChange() {
    xrm.Page.getAttribute(fieldName).setValue(textArea.value);
}

function getDate() {...
}

```

7. Select **File** and **Save All**.
8. Go to **Power Apps**<sup>40</sup> and make sure that you are not in the default environment.
9. Select **Solutions** and open the **Class** solution.
10. Select to open the Time Stamped Textbox Script that you added to the solution.



11. Select **Browse**.

General

Name *	tt_ TimestampedTextboxScript
Display Name	Time Stamped Textbox Script
Description	

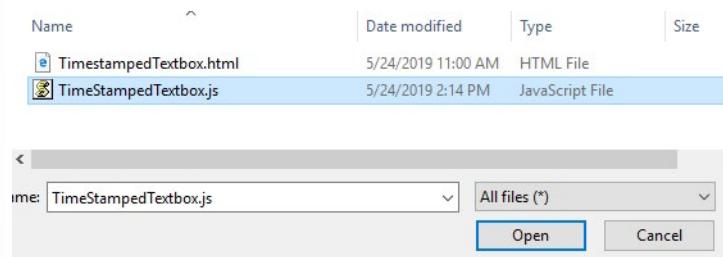
Content

Type *	Script (JScript)
Language	
Upload File	<input type="button" value="Browse..."/>

URL

URL	https://devmodules.crm.dynamics.com/WebResources/tt_TimestampedTextboxScript
-----	--

12. Select the script file and select **Open**.



13. Select **Save**.

14. Select **Publish** and wait for the publishing to complete.

15. Close the web resource editor browser tab.

16. Select **Done**.

<sup>40</sup> <https://make.powerapps.com/?azure-portal=true>

Currently editing a web resource

When you're done editing the web resource, click Done below to return to the page. This will refresh the page and fetch your changes.

**Done**

### Task 3: Test your work

In this task, you will test the timestamped text area that you created.

1. Select **Apps** and select to open the **Scout Zone** application.

Apps in Dev Modules (devmodules)			
	Name		
	Solution Health Hub		...
	Scout Zone		...

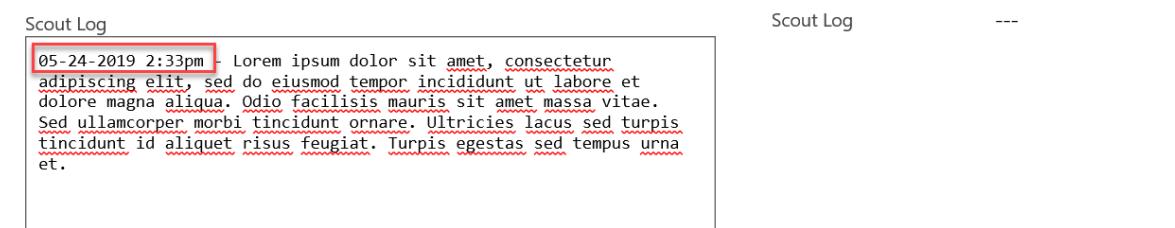
2. Select **Scout Reports** and open one of the records.

Name	Player
Maria A	Maria A
Paul C	Paul C
Paul G	Paul G
Paul Z	Paul Z

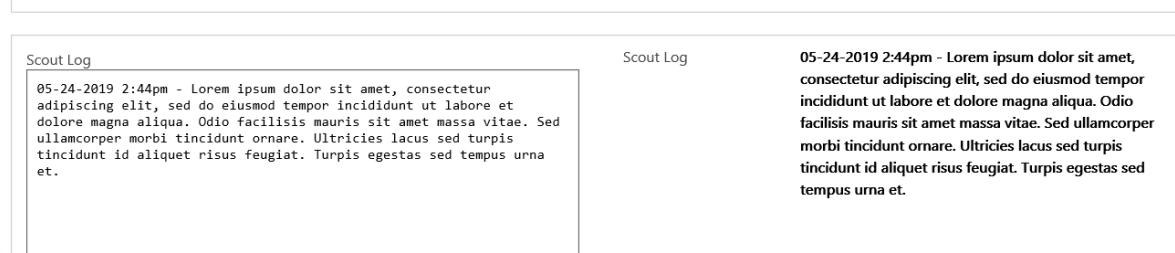
3. Scroll down to the new section. You should see both the resource and field that you added to the form. Select the web resource text area.



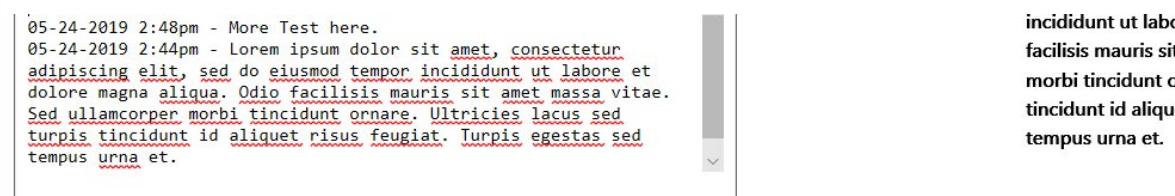
4. **Timestamp** should be added to the text area. Type something in the text area.



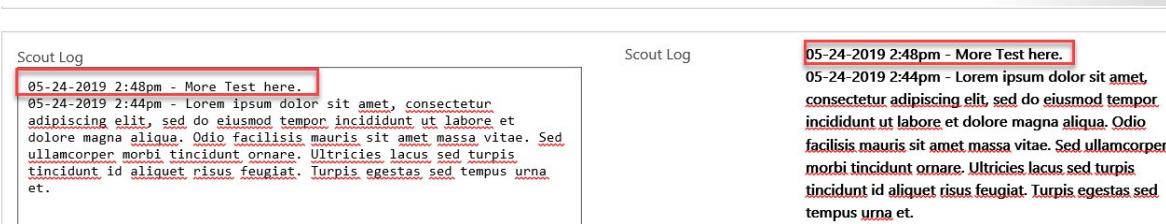
5. The value that you entered in the text area should be reflected in the **Scout Log** field.
6. Select **Save**.
7. Refresh the page. Your changes should persist.



8. Select the text area again and add more text. Another timestamp should be entered.



9. Select the **tab** key of your keyboard. The field should be updated with the data from the text area.



## Task 4: Hide a field

In this task, you will hide the **Scout Log** multiline text field.

1. Go to **Power Apps**<sup>41</sup> and make sure that you are not in the default environment.
2. Select **Solutions** and open the **Class** solution.
3. Locate and open the **Scout Report** entity.
4. Select the **Forms** tab and select to open the **Information** form.
5. Select the multiline text field and select the **Hide field** check box.



6. Select **Save**.
7. Select **Publish** and wait for the publish to complete.
8. Close the form editor browser tab.
9. Select **Done**.
10. Select **Apps** and open the **Scout Zone** application again.
11. The multiline text field should no longer be on the form.

<sup>41</sup> <https://make.powerapps.com/?azure-portal=true>

Observation Date	9/15/2015
Observation Type	---
Recommendation	---

Scout Log
05-24-2019 3:07pm - What is this 05-24-2019 2:48pm - More Test here. 05-24-2019 2:44pm - Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Odio facilisis mauris sit amet massa vitae. Sed ullamcorper morbi tincidunt ornare. Ultricies lacus sed turpis tincidunt id aliquet risus feugiat. Turpis egestas sed tempus urna et.

## Exercise - Automate business process flows with client script

### Overview

In this lab, you will create a script that will show/hide the **Financials** tab of the **Player Recruiting** form based on the current stage of the Business Process Flow.

### What you will learn:

- How to add client script in your Dynamics 365
- How to use form events
- How to hide/show form tabs

### Exercise 1: Prepare the Solution

In this exercise, you will add the Player Recruiter entity to the Class solution.

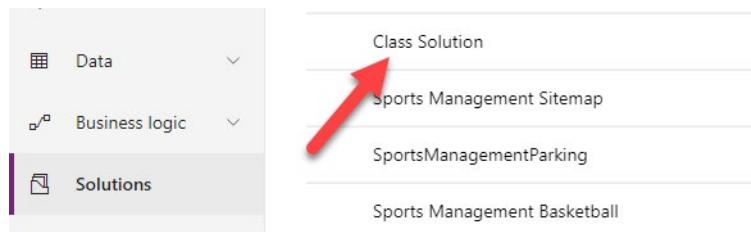
Each exercise consists of a scenario and learning objectives; the scenario describes the purpose of the exercises, while the objectives are listed and have bullet points.

### Task 1: Add an entity

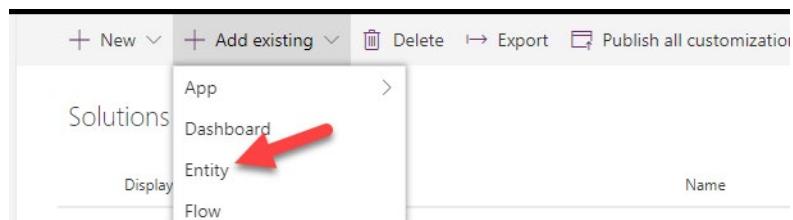
1. Go to <https://make.powerapps.com><sup>42</sup> and make sure that you are not in the default environment.
2. Select **Solutions** and open the **Class Solution**.

---

<sup>42</sup> <https://make.powerapps.com/?azure-portal=true>



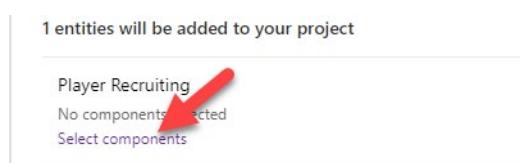
3. Select **Add existing** and select **Entity**.



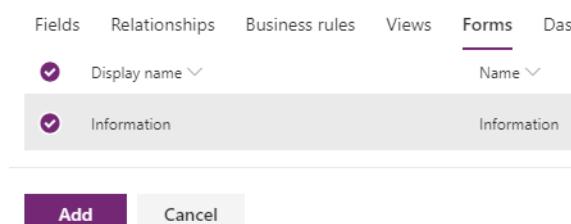
4. Select **Player Recruiting** entity and select **Next**.



5. Select the **Select Components** option.



6. Select the **Forms** tab, select the **Information** form, and then select **Add**.



7. Select **Add** again.

1 entities will be added to your project

Player Recruiting  
1 form selected  
Select components

Add Cancel

8. Select to open the **Player Recruiting** entity.

Display name	Name	Type	Manage...
Location Requirement	Location Requirement	Process	No
Observation Type Based Rec...	Observation Type Bas...	Process	No
Player Recruiting	tt_playerrecruiting	Entity	Yes
Scout Report	tt_scoutreport	Entity	Yes

9. Select the **Forms** tab and open the **Information** form.

Fields Relationships Business rules Views Forms Dashboards Charts Keys E

Model-driven

Name ↓ Form

Information Main

10. Note the Business Process Flow stage names. The script that you will create in the next exercise will verify if the current stage name is Watching or Planning; if true, it will hide the **Financials** tab, else it will make it visible.



11. Select the **Financials** tab.

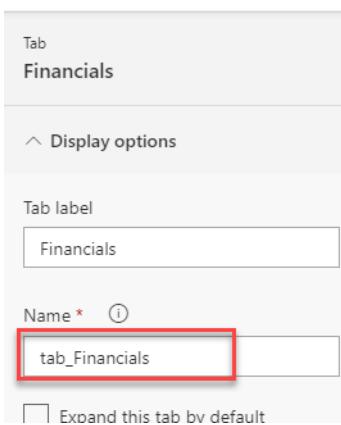
PLAYER RECRUITING  
New Player Recruiting

Player Recruiting Active for less than one mi... Watching (< 1 Min)

General Financials Scouting Reports

Budget

12. Go to the Properties pane and note the Name value "tab\_Financials." This is the tab that you will show/hide based on the selected BPF stage.



13. Close the form editor browser tab.

14. Select **Done**.

Currently editing a form

When you're done editing the form, click Done below to return to the entity. This will refresh the page and fetch your changes.

**Done**

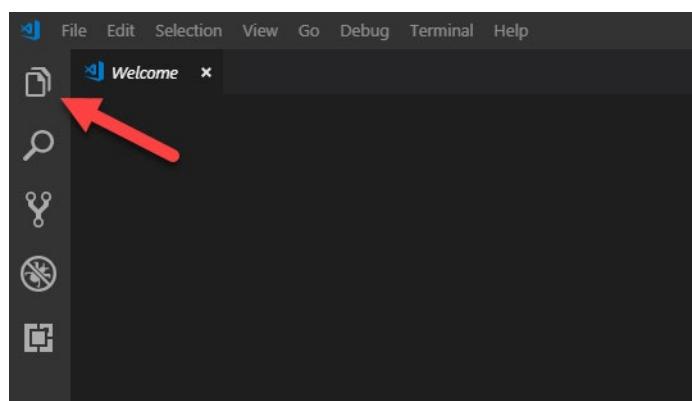
## Exercise 2: Create a script

In this exercise, you will use Visual Studio Code to create the JavaScript.

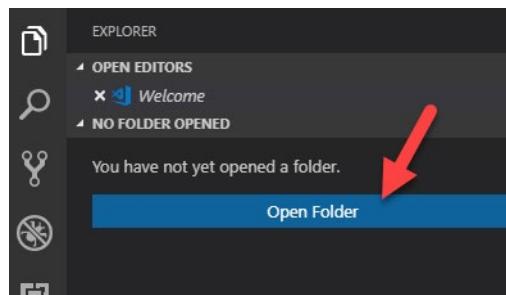
Each exercise consists of a scenario and learning objectives; the scenario describes the purpose of the exercises, while the objectives are listed and have bullet points.

### Task 1: Create the script

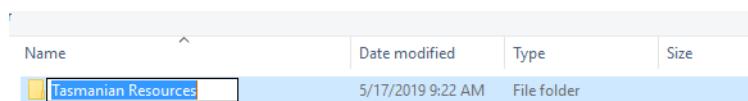
1. Start Visual Studio Code and select **Explorer**.



2. Select **Open Folder**.



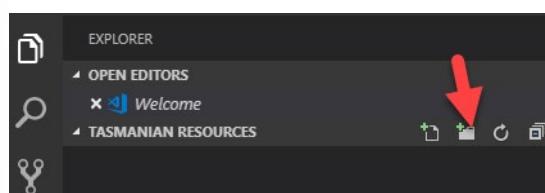
3. Create new folder and name the folder **Tasmanian Resources**.



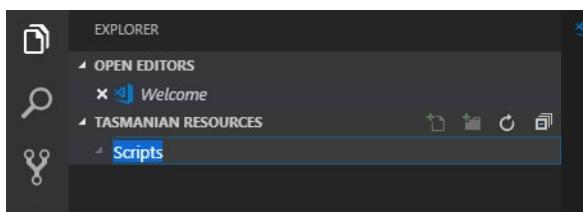
4. Select the folder and select the **Select Folder** button.



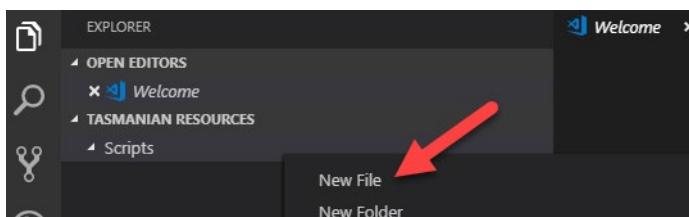
5. Select **New Folder**.



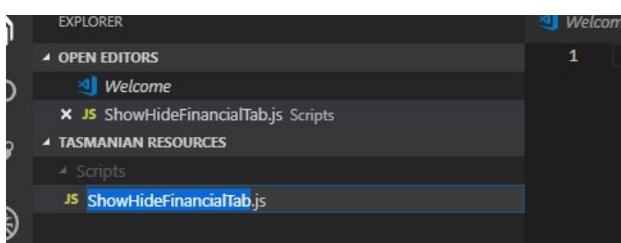
6. Name the new folder **Scripts**.



7. Right-click the **Scripts** folder and select **New File**.



8. Name the new file **ShowHideFinancialTab.js**.



9. Add the following snippet to the file. This is the function that will be called when the form loads, and it will get the form context and register the function that you want to call when the Business Process Flow stage changes.

```
function onLoad (executionContext) {
    var formContext = executionContext.getFormContext();
    formContext.data.process.addOnStageChange(onStageChange)
}

1  function onLoad (executionContext) {
2      var formContext = executionContext.getFormContext();
3      formContext.data.process.addOnStageChange(onStageChange)
4  }
5
```

10. Add the **onStageChange** function to the file. This is the function that will be called when the BPF stage changes.

```
function onStageChange(executionContext) {
```

```
}
```

```
1  function onLoad(executionContext) {
2    var formContext = executionContext.getFormContext();
3    formContext.data.process.addOnStageChange(onStageChange)
4  }
5
6  function onStageChange(executionContext) {
7
8
9}
```

11. Add the following snippet inside the **onStageChange** function. This snippet will get the formContext, selected stage, and the stage name.

```
var formContext = executionContext.getFormContext();

var stage = formContext.data.process.getSelectedStage();

var stageName = stage.getName();
```

```
5
6  function onStageChange(executionContext) {
7    var formContext = executionContext.getFormContext();
8    var stage = formContext.data.process.getSelectedStage();
9    var stageName = stage.getName();
10  }
11
```

12. Check if the selected stage name is Watching or Planning.  
Add the following snippet after the previous one.

```
if(stageName == "Watching" || stageName == "Planning")

{
```

```
}
```

```
function onStageChange(executionContext) {
  var formContext = executionContext.getFormContext();
  var stage = formContext.data.process.getSelectedStage();
  var stageName = stage.getName();

  if(stageName == "Watching" || stageName == "Planning")
  {
  }
}
```

13. Hide the **Financials** tab if the condition evaluates to true.  
Add the following snippet inside **if** condition.

```
formContext.ui.tabs.get("tab_Financials").setVisible(false);
```

14. Show the **Financials** tab if the condition evaluates to false. Add the following snippet after the `if` condition.

```
else

{

    formContext.ui.tabs.get("tab_Financials").setVisible(true);

}
```

15. Your script file should look like the following image.

```
1  function onLoad (executionContext) {
2      var formContext = executionContext.getFormContext();
3      formContext.data.process.addOnStageChange(onStageChange)
4  }

5
6  function onStageChange(executionContext) {
7      var formContext = executionContext.getFormContext();
8      var stage = formContext.data.process.getSelectedStage();
9      var stageName = stage.getName();

10
11     if(stageName == "Watching" || stageName == "Planning")
12     {
13         formContext.ui.tabs.get("tab_Financials").setVisible(false);
14     }
15     else
16     {
17         formContext.ui.tabs.get("tab_Financials").setVisible(true);
18     }
19
20 }
```

16. Select **File** and **Save All**.

## Exercise 3: Add a resource to a solution

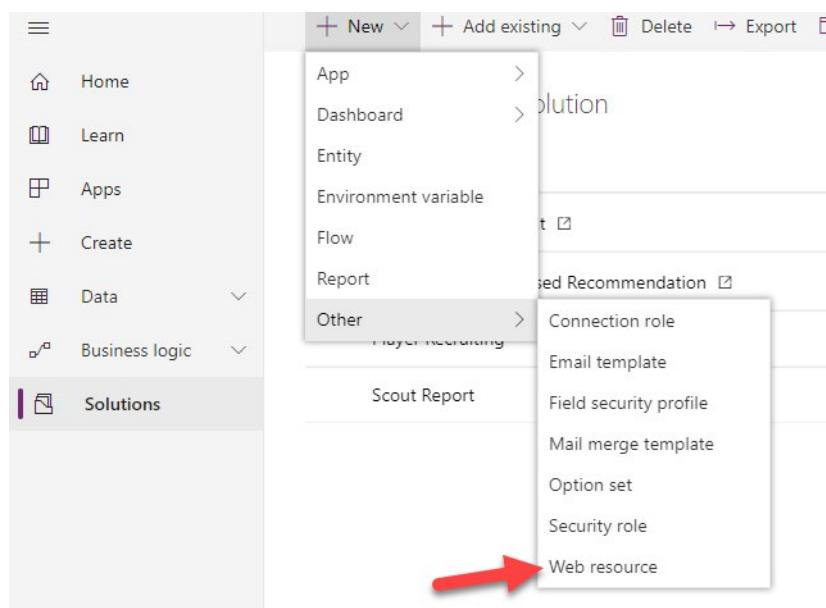
In this exercise, you will add the script that you created to the Class solution and make sure that your script runs when the form is loaded.

Each exercise consists of a scenario and learning objectives; the scenario describes the purpose of the exercises, while the objectives are listed and have bullet points.

### Task 1: Add a web resource to a solution

1. Go to <https://make.powerapps.com><sup>43</sup> and make sure that you are not in the default environment.
2. Select **Solutions** and open the **Class solution**.
3. Select **New > Other > Web Resource**.

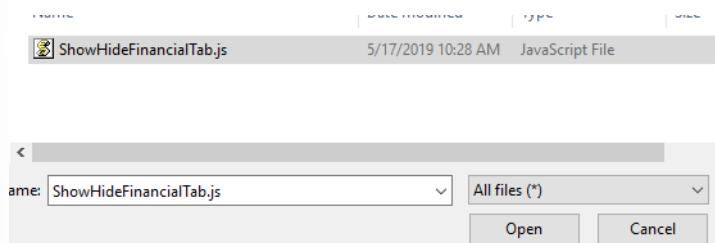
<sup>43</sup> <https://make.powerapps.com/?azure-portal=true>



4. Enter **ShowHideFinancialsTab** for **Name**, **Show/Hide Financials Tab** for **Display Name**, select **Script (Jscript)** for **Type**, select **English** for **Language**, and then select **Browse**.



5. Select the **ShowHideFinancialTab.js** file that you created and select **Open**.



6. Select **Save**.
7. Select **Publish** and wait for the publishing to complete.
8. Close the web resource editor browser tab.
9. Select **Done**. Do not leave this page.

Currently creating a new web  
resource

When you're done creating the new web resource, click Done below  
to return to the page. This will refresh the page and fetch your  
changes.

Done

## Task 2: Register OnLoad

In this task, you will add the onLoad functions to the Onload event of  
the **Player Recruiting** form.

1. Select to open the **Player Recruiting** entity.

Solutions > Class Solution

Display name	Name	Type
Location Requirements	Location Requirements	Process
Observation Type Bas...	Observation Type Bas	Process
Player Recruiting	tt_playerrecruiting	Entity
Scout Report	tt_scoutreport	Entity
Show/Hide Financials...	tt_ShowHideFinancial	Web Resource

2. Select the **Forms** tab and open the **Information** form.

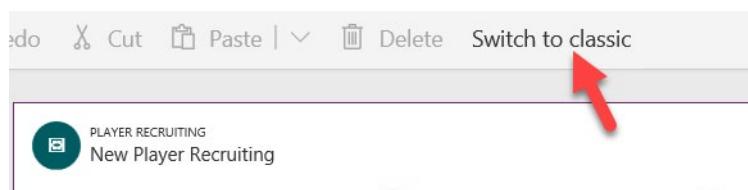
Solutions > Class Solution > Player Recruiting

Fields Relationships Business rules Views **Forms** Dashboards Charts Keys Data

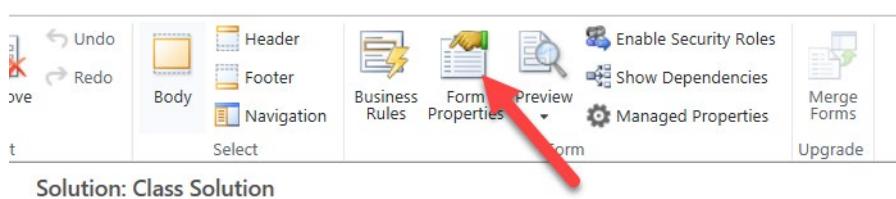
A screenshot of the Power Platform canvas interface. At the top, there's a navigation bar with 'Fields', 'Relationships', 'Business rules', 'Views', 'Forms', 'Dashboards', 'Charts', 'Keys', and 'Data'. Below this is a search bar with the placeholder 'Model-driven'. Underneath the search bar, there's a table-like view with columns for 'Name', 'Form type', and 'Type'. A red arrow points to the 'Information' checkbox in the 'Name' column.

Name	Form type	Type
Information <input checked="" type="checkbox"/>	Main	Custom

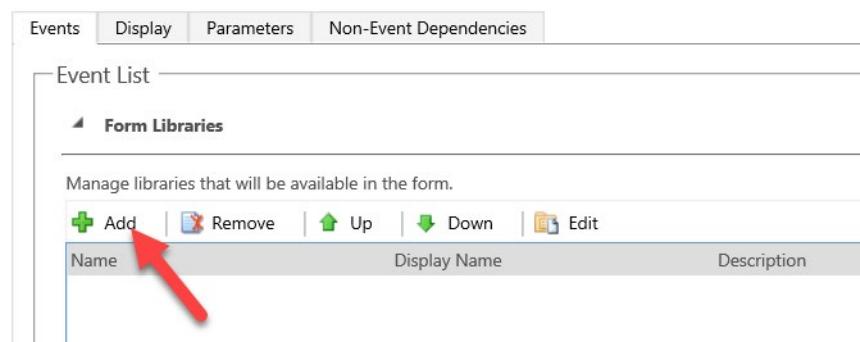
3. Select **Switch to classic**.



4. Select **Form Properties**.



5. Select **Add**.



6. Search for **tt** and select **tt\_ShowHideFinancialsTab**.

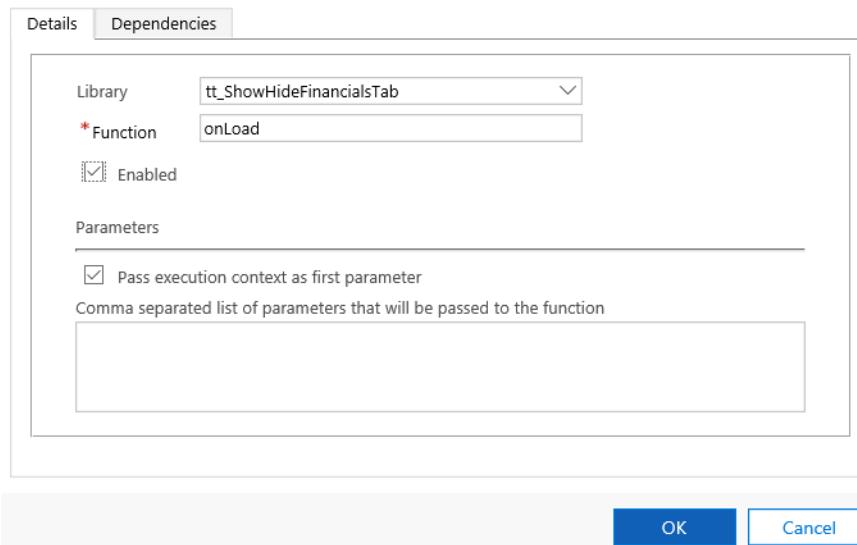
7. Select **Add**.

A screenshot of a search results page. At the top, there is a search bar with the text 'tt'. Below the search bar is a table with one row. The table has columns for 'Name', 'Display Name...', and 'Language'. The 'Name' column contains 'tt\_ShowHideFinancialsTab', which has a checked checkbox next to it. The 'Display Name...' column shows 'Show/Hide Fi...' and 'English(1033)'. At the bottom of the table, it says '1 - 1 of 1 (1 selected)'. Below the table are buttons for 'New', 'Add', 'Cancel', and 'Remove Value'. There are also navigation arrows and a 'Page 1' indicator.

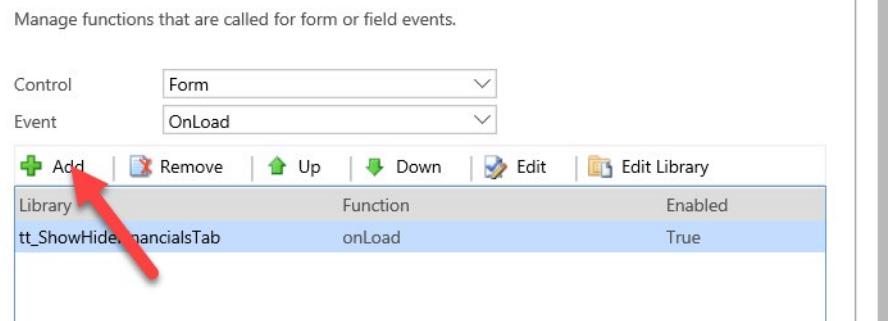
8. Go to the **Event Handlers** section, select **Form** for **Control**, select **OnLoad** for **Event**, and then select **Add**.

A screenshot of the 'Event Handlers' configuration screen. At the top, there is a header with a back arrow and the text 'Event Handlers'. Below the header, it says 'Manage functions that are called for form or field events.' There are two dropdown menus: 'Control' set to 'Form' and 'Event' set to 'OnLoad'. Below the dropdowns is a toolbar with buttons for 'Add' (highlighted with a red arrow), 'Remove', 'Up', 'Down', 'Edit', and 'Edit Library'. The 'Add' button is highlighted with a red arrow. The toolbar has tabs for 'Library', 'Function', and 'Enab'.

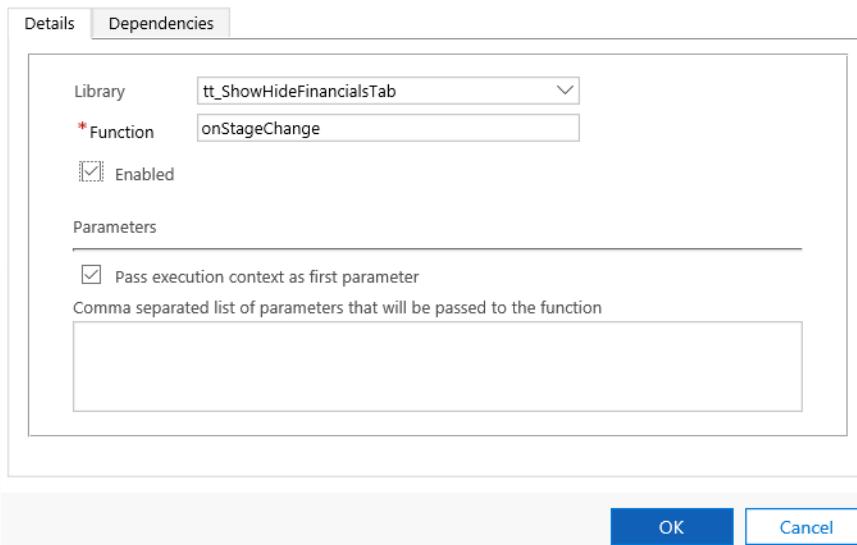
9. Make sure that **tt\_ShowHideFinancialsTab** is selected for **Library**, enter **onLoad** for **Function**, select the **Pass execution context as first parameter** check box, and then select **OK**.



10. You want the **onStageChange** function to be called when the form loads.  
Select **Add** again.



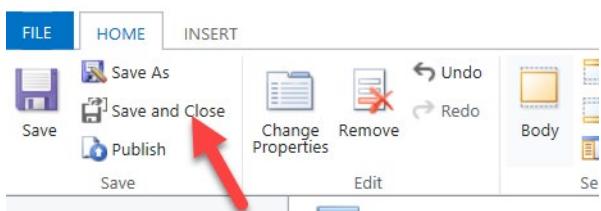
11. Make sure that **tt\_ShowHideFinancialsTab** is selected for **Library**,  
enter **onStageChange** for **Function**, select the **Pass execution context as first parameter** check box, and then select **OK**.



12. Select **OK** again.

Library	Function	Enabled
tt_ShowHideFinancialsTab	onLoad	True
tt_ShowHideFinancialsTab	onStageChange	True

13. Select **Save and Close**.



14. Close the form editor browser tab.

15. Select **Done**.

Currently editing a form

When you're done editing the form, click Done below to return to the entity. This will refresh the page and fetch your changes.

**Done**

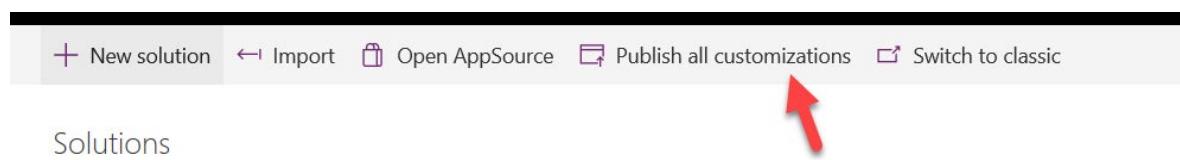
16. Select **Solutions**.

Solutions > Class Solution > Player Recruiting

Fields Relationships Business rules Views **Forms** Dashboards

Model-driven

17. Select **Publish All Customizations** and wait for the Publishing to complete. Do not leave this page.



## Task 3: Test your work

In this task, you will test your work.

1. Select **Apps** and select to open the **Scout Zone** application.

A screenshot of the Power Platform Apps screen. On the left is a sidebar with options like Home, Learn, Apps (which is selected and highlighted in purple), Create, Data, Business logic, and Settings. The main area shows 'Apps in Dev Modules (devmodules)' with tabs for Recent apps, Shared with me, Apps I can edit, and Org a. Under 'Recent apps', there is a list of applications: Solution Health Hub, Scout Zone (which has a red arrow pointing to it), and Sales Hub.

2. Select **Player Recruitings** and open one of the records.

A screenshot of the Power Platform Player Recruitings screen. On the left is a sidebar with Home, Recent, Pinned, Scout, Sports Teams, Player Recruitings (which is selected and highlighted in blue), and Reports. The main area shows the title 'Active Player Recruitings' with a dropdown arrow. Below it is a list of records: Maria A, Paul C (which has a red arrow pointing to it), and Paul G.

3. The Watching stage of the BPF should be selected and the **Financials** tab should be hidden.

PLAYER RECRUITING  
Paul C

Player Recruiting Active for 17 days < Watching (17 D) Planning Pursuing Negotiating

**General** Scouting Reports Related

Player * <b>Paul C</b>	Notes Timeline + ... Enter a note...	Current Contract Details End Date ---
Name * Paul C		

4. Select the **Watching** stage, select **College** for **Player Level**, and then select **Next Stage**.

Player Recruiting Active for 17 days < Watching (17 D) Planning

**General** Scouting Reports Related

Player * <b>Paul C</b>	Active for 17 days <input checked="" type="checkbox"/> X
Name * Paul C	✓ Player Level * <b>College</b>
Current Coach	Contract Details <input type="checkbox"/> No
---	Social Network Info <input type="checkbox"/> No

**Next Stage >**

5. The BPF should advance to the **Planning** stage and the **Financials** tab should stay hidden.

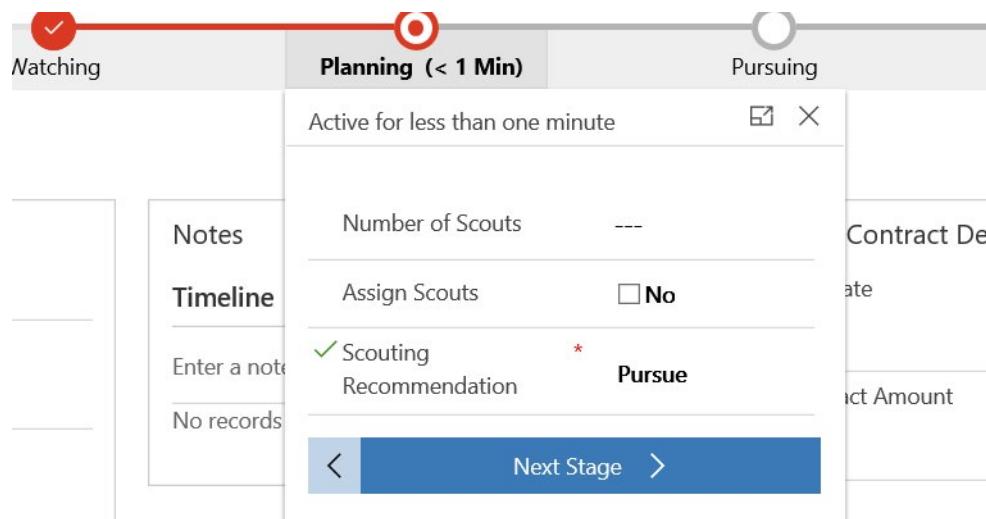
PLAYER RECRUITING  
Paul C

Player Recruiting Active for 17 days < Watching Planning (< 1 Min)

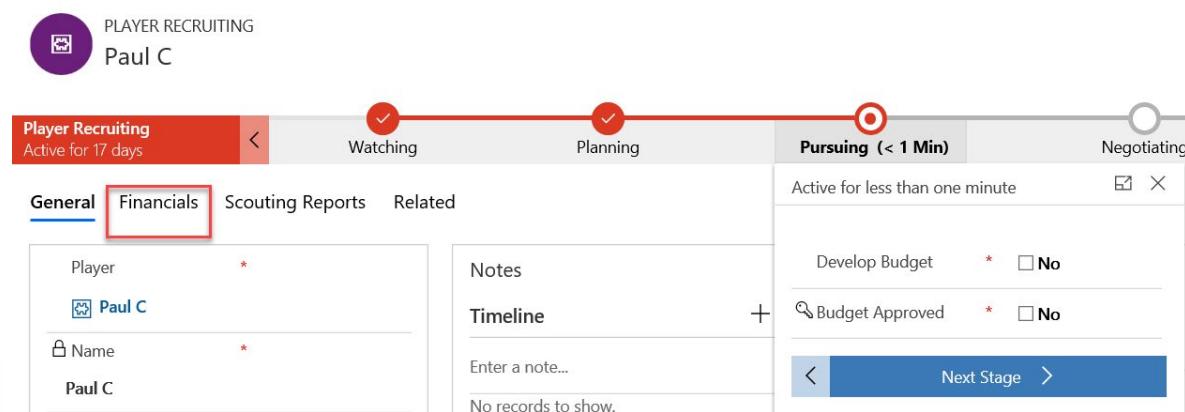
**General** Scouting Reports Related

Player * <b>Paul C</b>	Notes	Active for less than one minute
		Number of Scouts
		Assign Scouts

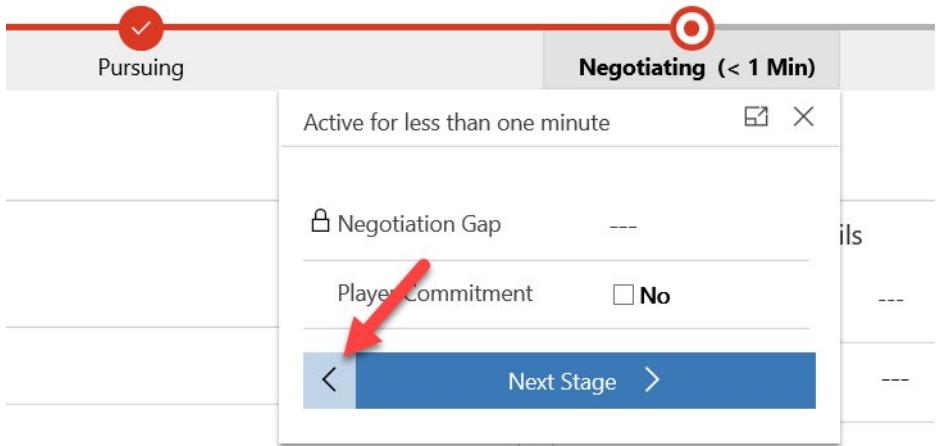
6. Select **Pursue** for **Scouting Recommendation** and then select **Next Stage**.



7. The BPF should advance to the **Pursuing** stage and the **Financials** tab should now become visible.



8. Select **Yes** for both **Develop Budget** and **Budget Approved**, and then select **Next Stage**.
9. The **Financials** tab should stay visible.
10. Select **Previous Stage**.



11. Select **Previous Stage** again.
12. The BPF should now be in the **Planning** stage and the **Financials** tab should become hidden again.

## summary

This module included more advanced topics surrounding client scripting, and contained some in-depth, hands-on exercises that allowed you to gain practical experience in building web resources and automating business process flows.



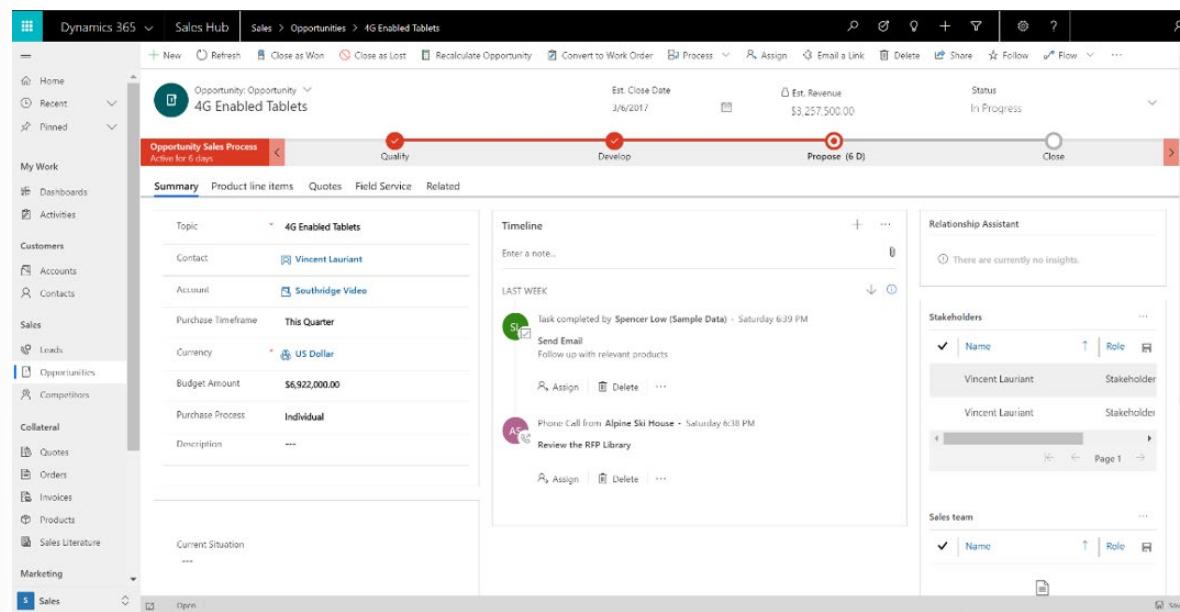
## Module 8 Create components with Power Apps Component Framework

### Get started with Power Apps component framework

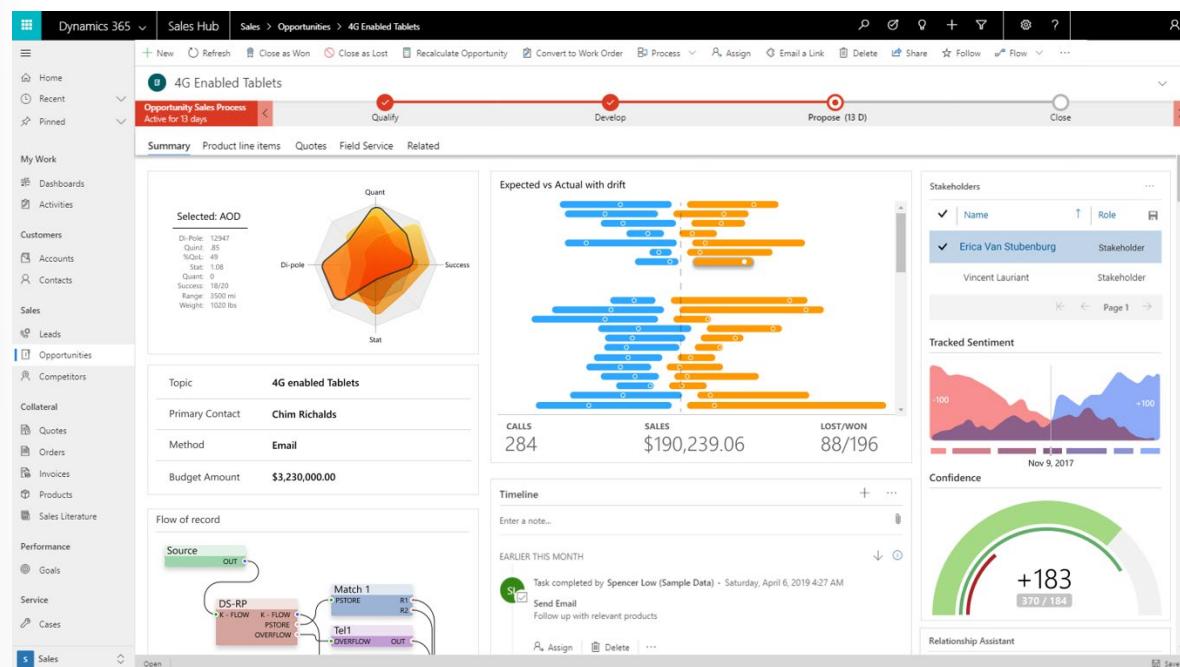
### Introduction to Power Apps component framework

Microsoft Power Apps component framework helps you create reusable components that can be used within your Power Apps applications. The component framework empowers developers and app makers to build code components when the out-of-the-box components don't fit an app maker's needs. Before the Power Apps component framework existed, people had to rely on HTML web resources to provide any form of custom presentation to a Power Apps' form. Now, you can use a more modernized framework that allows an abundance of capabilities to be exposed to your app that would otherwise be impossible to access or, even worse, be unsupported by Microsoft.

For example, the existing screen might render similar to the following image.



However, if you reconfigured your app to use custom Power Apps components, your app might look something like the following image.



## Power Apps component framework advantages

Microsoft has significant investment in ensuring that Power Apps components are built on top of a robust framework that supports modern web practices. A few of the advantages that you are afforded as a result are:

- Access to a rich set of framework APIs that expose capabilities like component lifecycle management, contextual data, and metadata

- Support of client frameworks such as React and AngularJS
- Seamless server access through Web API, utility and data formatting methods, device features like camera, location, and microphone, in addition to easy-to-invoke UX elements like dialogs, lookups, and full-page rendering
- Optimization for performance
- Reusability
- Use of responsive web design principles to provide an optimal viewing and interaction experience for any screen size, device, or orientation
- Ability to bundle all files into a single solution file

## Types of components that you can add

It might be said that if you can build it in HTML+JavaScript, you can build it as a Power Apps component. However, common types of components that you might want to use in a Power Apps application might be:

- A custom control for a field on a form
- A custom grid (or subgrid) to display data in a tabular format
- A component that displays content from external services

## Community components

The Power Apps community has been active in building out components that they have deemed valuable to share with others. For example, a common requirement is to validate user input against a regular expression. While you could write Client Script to perform this validation, or even use your own Power Apps component, it might be beneficial for you to rely on the community to determine if someone else has already solved this problem for you.

Numerous samples as such can be found at [PCF Gallery<sup>1</sup>](#). If you're interested in experimenting with pre-built Power Apps components in your solution, this documentation is a great place to start.

## Where to find help

At the time of this writing, Power Apps component framework is still a relatively new technology. While it is robust and well-tested, you might encounter areas where you'll need assistance.

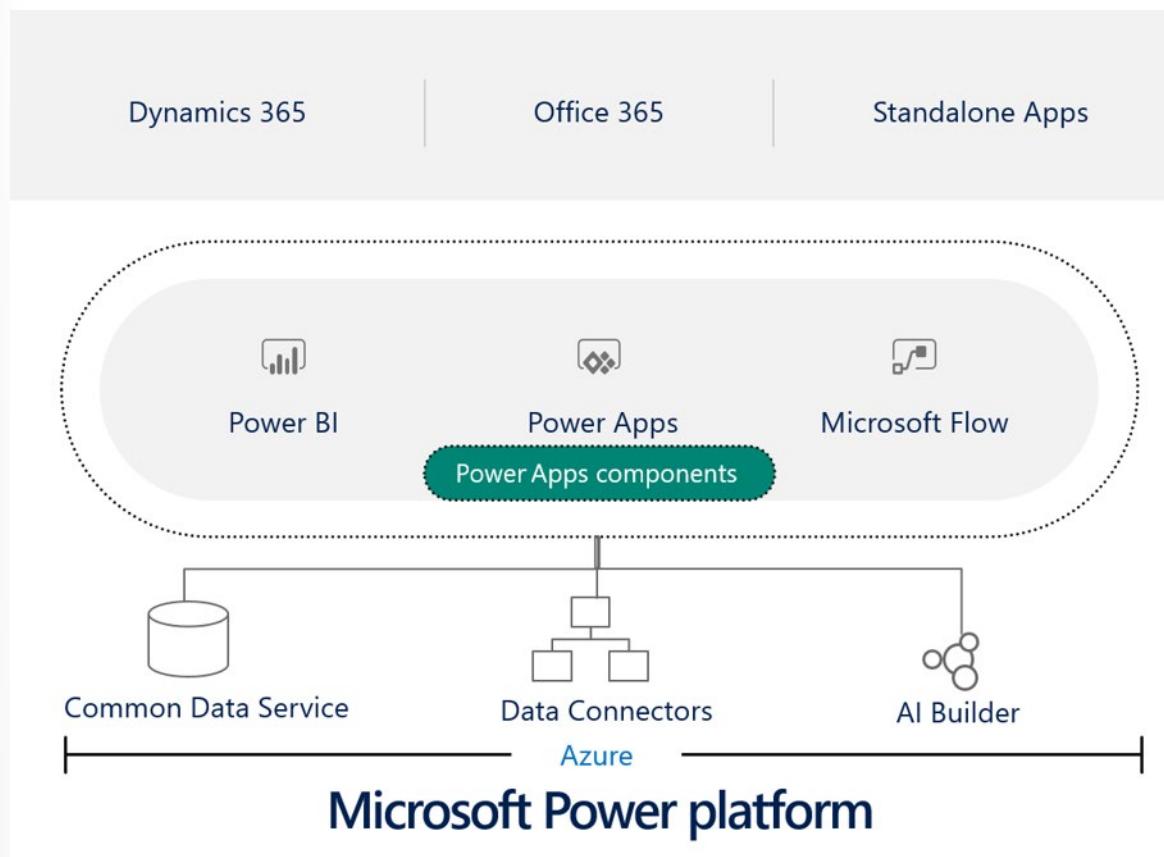
The best place to go for assistance is the [Power Apps component framework, ALM & Pro Dev community forum<sup>2</sup>](#), where you can find a wealth of questions and answers to a wide variety of topics, and where you can submit your own questions as well.

<sup>1</sup> <https://pcf.gallery/?azure-portal=true>

<sup>2</sup> [https://powerusers.microsoft.com/t5/Power-Apps-Component-Framework/bd-p/pa\\_component\\_framework/?azure-portal=true](https://powerusers.microsoft.com/t5/Power-Apps-Component-Framework/bd-p/pa_component_framework/?azure-portal=true)

## Power Apps component framework architecture

Power Apps Components are built atop of Microsoft's robust Power Platform, which means that you can seamlessly use the wealth of utilities that it provides.



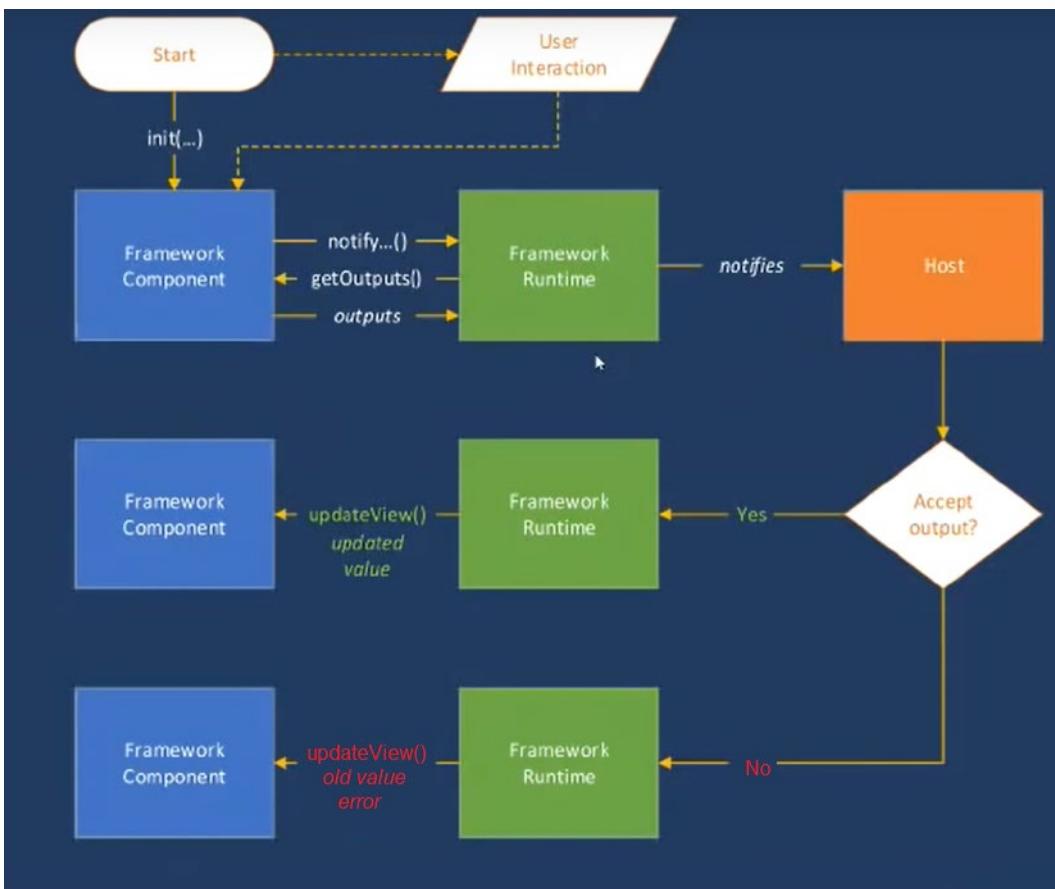
## Power Apps component life cycle

When developing a component, you are expected to implement the life cycle methods that are shown in the following table.

Method	Description
init	Required. This method is used to initialize the component instance. Components can kick off remote server calls and other initialization actions in this method. Dataset values cannot be initialized with this method; you will need to use the updateView method for this purpose.
updateView	Required. This method will be called when any value in the component's property bag has changed.

Method	Description
getOutputs	Optional. Called by the framework prior to the receipt of new data. Use this method when dynamically managing bound properties in a control.
destroy	Required. Invoked when the component is to be removed from the DOM tree. Used for cleanup and to release any memory that the component is using.

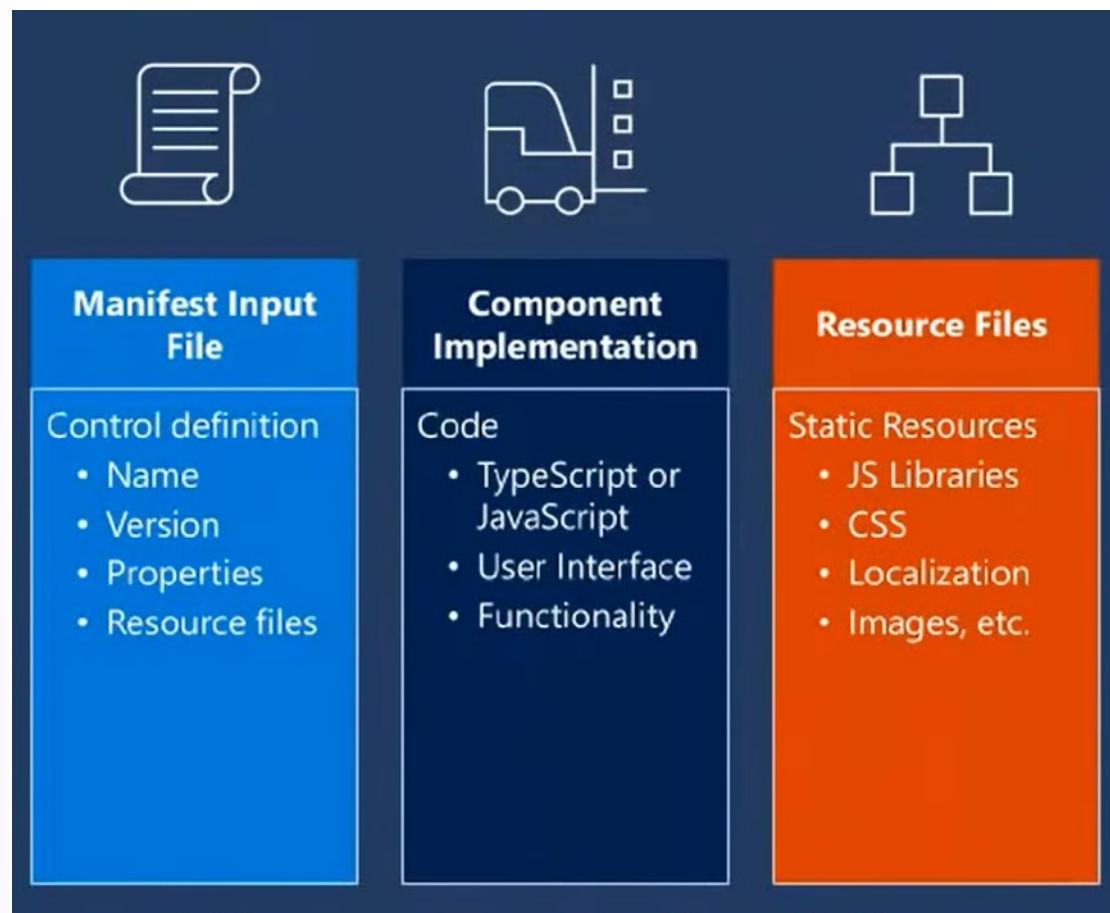
These methods are invoked through a Framework Runtime process in a standardized life cycle, as shown in the following illustration.



At the top of the image, the framework calls the `init()` function for your component. If your component is interactive, you'll also need to handle notification to the Host that the component's output has changed by calling the `notifyOutputChanged` method. The framework runtime will then call your implemented `getOutputs` function, which represents all of your component's outputs, and then return that to the runtime. The runtime will then notify the Host of that value, which will perform validation on the output. If the output is found to be valid, it will call the `updateView` method on your component. If it isn't valid for whatever reason (that is, a business rule found the new output to be invalid), it will call your `updateView` method and pass the old value along with an error message.

## Component makeup

As the following image shows, a Power Apps component is represented as three key areas: a Manifest Input File, its implementation, and any additional resource files that might be needed by the component.



For more information, see [What are code components<sup>3</sup>](#).

These concepts are covered in more detail while you're learning how to build your own custom components.

## Power Apps component tooling

### Power Apps CLI

Microsoft has built a robust CLI framework that exposes scaffolding methods that rapidly expedite your ability to build custom Power Apps components. It provides simple-to-use control templates along with built-in validations to catch issues prior to deployment of your control.

<sup>3</sup> <https://docs.microsoft.com/powerapps/developer/component-framework/custom-controls-overview/?azure-portal=true>

To use the Power Apps CLI, you'll first need to install **Npm<sup>4</sup>** (comes with Node.js) or **Node.js<sup>5</sup>** (comes with npm). We recommend that you install LTS (Long Term Support) version 10.15.3 or higher.

For additional prerequisites and a link to the CLI's installation bits, see **Install Microsoft Power Apps CLI<sup>6</sup>**.

The steps to install CLI are covered in more detail when you learn how to build a Power Apps component.

## Power Apps component CLI commands

If you want to see all of the commands that are available through the CLI, see **Common commands<sup>7</sup>**.

## Choose an IDE

While we recommend that you use Visual Studio or Visual Studio Code to write your components, you can use any IDE of your choice to build custom Power Apps components.

## summary

The Power Apps component framework is a robust framework built on modern web technologies. It enables developers to write reusable components when out-of-the-box components may not suit a solution's needs. In this module, we reviewed the overall component framework architecture, in addition to reviewing the various tooling that is available.

**4** <https://www.npmjs.com/get-npm/?azure-portal=true>

**5** <https://nodejs.org/en/?azure-portal=true>

**6** <https://docs.microsoft.com/powerapps/developer/common-data-service/powerapps-cli?azure-portal=true>

**7** <https://docs.microsoft.com/powerapps/developer/common-data-service/powerapps-cli#common-commands>

# Build a Power Apps component

## Introduction to creating a code component

Custom Power Apps components are frequently referred to as *code components* because they require custom code to implement them. They consist of three elements: a manifest, an implementation, and resources. In the following exercise, you will write your own custom code component: a linear slider control that will look like the following image.



The following steps will help you build this component.

## Install Power Apps CLI

To get Microsoft Power Apps CLI, follow these steps:

1. Install **Npm**<sup>8</sup> (comes with Node.js) or **Node.js**<sup>9</sup> (comes with npm). We recommend that you use LTS (Long Term Support) version 10.15.3 or higher.
2. Install **.NET Framework 4.6.2 Developer Pack**<sup>10</sup>.
3. If you don't already have Visual Studio 2017 or later, follow one of these options:
  - Option 1: Install **Visual Studio 2017**<sup>11</sup> or later.
  - Option 2: Install **.NET Core 3.1 SDK**<sup>12</sup> and then install **Visual Studio Code**<sup>13</sup>.
4. Install **Microsoft Power Apps CLI**<sup>14</sup>.
5. To take advantage of all the latest capabilities, update the Power Apps CLI tooling to the latest version by using this command: pac install latest

## Create a new component project

To create a new component project, follow these steps:

1. Create a directory where you'll build your component. In this sample, you'll place the component in **C:\source\pcf-samples**; however, you can create your own directory. To create your own directory, you'll use a command prompt. From your source directory, create a directory named **LinearComponent** and then go to that directory through *cd LinearComponent*:

```
c:\source\mslearn-pcf-samples>mkdir LinearComponent
c:\source\mslearn-pcf-samples>cd LinearComponent
c:\source\mslearn-pcf-samples\LinearComponent
```

<sup>8</sup> <https://www.npmjs.com/get-npm/?azure-portal=true>

<sup>9</sup> <https://nodejs.org/en/?azure-portal=true>

<sup>10</sup> <https://dotnet.microsoft.com/download/dotnet-framework/net462/?azure-portal=true>

<sup>11</sup> <https://docs.microsoft.com/visualstudio/install/install-visual-studio?view=vs-2017/?azure-portal=true>

<sup>12</sup> <https://dotnet.microsoft.com/download/dotnet/current/?azure-portal=true>

<sup>13</sup> <https://code.visualstudio.com/Download/?azure-portal=true>

<sup>14</sup> <https://aka.ms/PowerAppsCLI/?azure-portal=true>

2. Initialize your component project by using Power Apps CLI through the following command:

```
pac pcf init --namespace SampleNamespace --name TSLinearInputComponent --template field
```

The following image shows an example of the output that you should see.

```
c:\source\mslearn-pcf-samples\LinearComponent>pac pcf init --namespace SampleNamespace --name TSLinearInputComponent --template field
The PowerApps component framework project was successfully created in 'c:\source\mslearn-pcf-samples\LinearComponent'.
Be sure to run 'npm install' in this directory to install project dependencies.

c:\source\mslearn-pcf-samples\LinearComponent>
```

3. Install the project build tools by using the command `npm install`. You might see some warnings displayed; however, you can ignore them.

```
c:\source\mslearn-pcf-samples\LinearComponent>npm install
npm [WARN] deprecated opn@6.0.0: The package has been renamed to `open`
npm [notice] created a lockfile as package-lock.json. You should commit this file.
npm [WARN] pcf-project@1.0.0 No repository field.
npm [WARN] pcf-project@1.0.0 No license field.
npm [WARN] optional  SKIPPING OPTIONAL DEPENDENCY: fsevents@1.2.11 (node_modules\fsevents):
npm [WARN] notsup  SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.2.11: wanted {"os":"darwin","arch":"any"} (current: {"os":"win32","arch":"x64"})
added 657 packages from 499 contributors and audited 10365 packages in 25.96s
found 0 vulnerabilities
```

4. Open your new project in a developer environment. You can use Visual Studio, Visual Studio Code, or any other environment that you prefer. In this example, you'll use Visual Studio Code, which provides a way to open a new window from a directory in a command prompt through the command code.

```
c:\source\mslearn-pcf-samples\LinearComponent>code .
```

## Update your code component's manifest

Update the manifest file to accurately represent your control.

1. Change the *version*, *display-name-key*, and *description-key* properties that are found in the `TSLinearInputComponent` node `ControlManifest.Input.xml` file to more meaningful values. In this example, you'll change the properties to **1.0.0**, **Linear Input Component**, and **Allows you to enter the numeric values using the visual slider**, respectively:

```
<control namespace="SampleNameSpace" constructor="TSLinearInputComponent" version="1.0.0"
display-name-key="Linear Input Component" description-key="Allows you to enter the numeric
values using the visual slider." control-type="standard">
```

2. Replace the `sample` property with your own custom slider property. You will be using a custom type group called **numbers** in the property that you'll need to add.

```
<property name="sliderValue" display-name-key="sliderValue_Display_Key" description-key="slid-
erValue_Desc_Key" of-type-group="numbers" usage="bound" required="true" />
```

3. Insert the following `<type-group>` node above your property to indicate the types of attributes that can use this control:

```
<type-group name="numbers">
<type>Whole.None</type>
<type>Currency</type>
```

```
<type>FP</type>
<type>Decimal</type>
</type-group>
```

4. Update the <resources> node to include a reference to a CSS file named *TS\_LinearInputComponent.css* that you'll create. Place this file into a CSS folder. The node will look like the following example:

```
<css path="css/TS_LinearInputComponent.css" order="1" />
```

5. After you've made updates, save the changes. Your manifest file should look similar to the following example:

```
<?xml version="1.0" encoding="utf-8" ?>
<manifest>
<control namespace="SampleNamespace" constructor="TSLinearInputComponent" version="1.0.0"
display-name-key="Linear Input Component" description-key="Allows you to enter the numeric
values using the visual slider." control-type="standard">
<type-group name="numbers">
<type>Whole.None</type>
<type>Currency</type>
<type>FP</type>
<type>Decimal</type>
</type-group>
<property name="sliderValue" display-name-key="sliderValue_Display_Key" description-key="slid-
erValue_Desc_Key" of-type-group="numbers" usage="bound" required="true" />
<resources>
<code path="index.ts" order="1" />
<css path="css/TS_LinearInputComponent.css" order="1" />
</resources>
</control>
</manifest>
```

## Add styling to your code component

To add styling to your code component, follow these steps:

1. Create a new CSS subfolder under the *TSLinearInputComponent* folder.
2. Create a new *TS\_LinearInputComponent.css* file inside the CSS subfolder.
3. Add the following style content to the *TS\_LinearInputComponent.css* file:

```
.SampleNamespace\TSLinearInputComponent input[type=range].linearslider {
margin: 1px 0;
background: transparent;
-webkit-appearance: none;
width: 100%;
padding: 0;
height: 24px;
-webkit-tap-highlight-color: transparent
}
```

```
.SampleNamespace\TSLinearInputComponent input[type=range].linearslider:focus {  
    outline: none;  
}  
  
.SampleNamespace\TSLinearInputComponent input[type=range].linearslider::-webkit-slider-runnable-track {  
    background: #666;  
    height: 2px;  
    cursor: pointer  
}  
  
.SampleNamespace\TSLinearInputComponent input[type=range].linearslider::-webkit-slider-thumb {  
    background: #666;  
    border: 0 solid #f00;  
    height: 24px;  
    width: 10px;  
    border-radius: 48px;  
    cursor: pointer;  
    opacity: 1;  
    -webkit-appearance: none;  
    margin-top: -12px  
}  
  
.SampleNamespace\TSLinearInputComponent input[type=range].linearslider::-moz-range-track {  
    background: #666;  
    height: 2px;  
    cursor: pointer  
}  
  
.SampleNamespace\TSLinearInputComponent input[type=range].linearslider::-moz-range-thumb {  
    background: #666;  
    border: 0 solid #f00;  
    height: 24px;  
    width: 10px;  
    border-radius: 48px;  
    cursor: pointer;  
    opacity: 1;  
    -webkit-appearance: none;  
    margin-top: -12px  
}  
  
.SampleNamespace\TSLinearInputComponent input[type=range].linearslider::-ms-track {  
    background: #666;  
    height: 2px;  
    cursor: pointer  
}  
  
.SampleNamespace\TSLinearInputComponent input[type=range].linearslider::-ms-thumb {  
    background: #666;  
    border: 0 solid #f00;  
    height: 24px;
```

```
width: 10px;  
border-radius: 48px;  
cursor: pointer;  
opacity: 1;  
-webkit-appearance: none;  
}
```

4. Save the TS\_LinearInputComponent.css file.

## Build your code component

Before you can implement your component logic, build your component so that the appropriate types are generated, as specified in your ControlManifest.xml document.

Go back to your command prompt and build your project by using the

```
npm run build
```

command.

```
c:\source\mslearn-pcf-samples\LinearComponent>npm run build  
> pcf-project@1.0.0 build c:\source\mslearn-pcf-samples\LinearComponent  
> pcf-scripts build  
[12:36:59] [build] Initializing...  
[12:36:59] [build] Validating manifest...  
[12:36:59] [build] Validating control...  
[12:37:0] [build] Generating manifest types...  
[12:37:0] [build] Compiling and bundling control...  
[Webpack stats]:  
Hash: 65eeb4042383bed4ff6f  
Version: webpack 4.28.4  
Time: 1420ms  
Built at: 01/18/2020 12:37:02 PM  
    Asset      Size  Chunks      Chunk Names  
bundle.js  6.51 KiB  main  [emitted]  main  
Entrypoint main = bundle.js  
[./TSLinearInputComponent/index.ts] 2.45 KiB {main} [built]  
[12:37:2] [build] Generating build outputs...  
[12:37:2] [build] Succeeded  
c:\source\mslearn-pcf-samples\LinearComponent>
```

The component is compiled into the out/controls/TSLinearInputComponent folder. The build artifacts include:

- **bundle.js** - Bundled component source code.
- **ControlManifest.xml** - Actual component manifest file that is uploaded to the Common Data Service organization.

## Implement your code component's logic

To implement your code component's logic, follow these steps:

1. Open index.ts in Visual Studio or something similar.
2. Above the *constructor* method, insert the following private variables:

```
// Value of the field is stored and used inside the component
private _value: number;
// Power Apps component framework delegate which will be assigned to this object which would be
// called whenever any update happens.
private _notifyOutputChanged: () => void;
// label element created as part of this component
private labelElement: HTMLLabelElement;
// input element that is used to create the range slider
private inputElement: HTMLInputElement;
// reference to the component container HTMLDivElement
// This element contains all elements of our code component example
private _container: HTMLDivElement;
// reference to Power Apps component framework Context object
private _context: ComponentFramework.Context<IInputs>;
// Event Handler 'refreshData' reference
private _refreshData: EventListenerOrEventListenerObject;
```

3. Replace the *init* method with the following logic:

```
public init(
    context: ComponentFramework.Context<IInputs>,
    notifyOutputChanged: () => void,
    state: ComponentFramework.Dictionary,
    container: HTMLDivElement
) {
    this._context = context;
    this._container = document.createElement("div");
    this._notifyOutputChanged = notifyOutputChanged;
    this._refreshData = this.refreshData.bind(this);
    // creating HTML elements for the input type range and binding it to the function which refreshes
    // the component data
    this.inputElement = document.createElement("input");
    this.inputElement.setAttribute("type", "range");
    this.inputElement.addEventListener("input", this._refreshData);
    //setting the max and min values for the component.
    this.inputElement.setAttribute("min", "1");
    this.inputElement.setAttribute("max", "1000");
    this.inputElement.setAttribute("class", "linearslider");
    this.inputElement.setAttribute("id", "linarrangeinput");
    // creating a HTML label element that shows the value that is set on the linear range component
    this.labelElement = document.createElement("label");
    this.labelElement.setAttribute("class", "TS_LinearRangeLabel");
    this.labelElement.setAttribute("id", "lrclabel");
    // retrieving the latest value from the component and setting it to the HTML elements.
    this._value = context.parameters.sliderValue.raw
        ? context.parameters.sliderValue.raw
        : 0;
    this.inputElement.value =
        context.parameters.sliderValue.formatted
        ? context.parameters.sliderValue.formatted
        : "0";
```

```
this.labelElement.innerHTML = context.parameters.sliderValue.formatted  
    ? context.parameters.sliderValue.formatted  
    : "0";  
// appending the HTML elements to the component's HTML container element.  
this._container.appendChild(this.inputElement);  
this._container.appendChild(this.labelElement);  
container.appendChild(this._container);  
}
```

4. Add the *refreshData* code below the previous code:

```
/**  
 * Updates the values to the internal value variable we are storing and also updates the html label that  
 displays the value  
 * @param context : The "Input Properties" containing the parameters, component metadata and  
 interface functions  
 */  
  
public refreshData(evt: Event): void {  
this._value = (this.inputElement.value as any) as number;  
this.labelElement.innerHTML = this.inputElement.value;  
this._notifyOutputChanged();  
}
```

5. Replace the *updateView* method with the following logic:

```
public updateView(context: ComponentFramework.Context<IInputs>): void {  
// storing the latest context from the control.  
this._value = context.parameters.sliderValue.raw  
    ? context.parameters.sliderValue.raw  
    : 0;  
this._context = context;  
this.inputElement.value =  
  
    context.parameters.sliderValue.formatted  
    ? context.parameters.sliderValue.formatted  
    : "";  
  
this.labelElement.innerHTML = context.parameters.sliderValue.formatted  
    ? context.parameters.sliderValue.formatted  
    : "";  
}
```

6. Replace *getOutputs* with the following method:

```
public getOutputs(): IOutputs {  
return {  
    sliderValue: this._value  
};
```

```
}
```

7. Replace the *destroy* method with the following logic:

```
public destroy() {
    this.inputElement.removeEventListener("input", this._refreshData);
}
```

8. After you've made the updates, your *index.ts* file should look similar to the following example:

```
import { IInputs, IOutputs } from "./generated/ManifestTypes";
export class TSLinearInputComponent
    implements ComponentFramework.StandardControl<IInputs, IOutputs> {
    // Value of the field is stored and used inside the component
    private _value: number;
    // Power Apps component framework delegate which will be assigned to this object which would be
    // called whenever any update happens.
    private _notifyOutputChanged: () => void;
    // label element created as part of this component
    private labelElement: HTMLLabelElement;
    // input element that is used to create the range slider
    private inputElement: HTMLInputElement;
    // reference to the component container HTMLDivElement
    // This element contains all elements of our code component example
    private _container: HTMLDivElement;
    // reference to Power Apps component framework Context object
    private _context: ComponentFramework.Context<IInputs>;
    // Event Handler 'refreshData' reference
    private _refreshData: EventListenerOrEventListenerObject;

    constructor() {}

    public init(
        context: ComponentFramework.Context<IInputs>,
        notifyOutputChanged: () => void,
        state: ComponentFramework.Dictionary,
        container: HTMLDivElement
    ) {
        this._context = context;
        this._container = document.createElement("div");
        this._notifyOutputChanged = notifyOutputChanged;
        this._refreshData = this.refreshData.bind(this);
        // creating HTML elements for the input type range and binding it to the function which refreshes
        // the component data
        this.inputElement = document.createElement("input");
        this.inputElement.setAttribute("type", "range");
        this.inputElement.addEventListener("input", this._refreshData);
        //setting the max and min values for the component.
        this.inputElement.setAttribute("min", "1");
        this.inputElement.setAttribute("max", "1000");
        this.inputElement.setAttribute("class", "linearslider");
    }
}
```

```
thisInputElement.setAttribute("id", "linearrangeinput");
// creating a HTML label element that shows the value that is set on the linear range component
this.labelElement = document.createElement("label");
this.labelElement.setAttribute("class", "TS_LinearRangeLabel");
this.labelElement.setAttribute("id", "Irclabel");
// retrieving the latest value from the component and setting it to the HTML elements.
this._value = context.parameters.sliderValue.raw
? context.parameters.sliderValue.raw
: 0;
thisInputElement.value =
context.parameters.sliderValue.formatted
? context.parameters.sliderValue.formatted
: "0";

this.labelElement.innerHTML = context.parameters.sliderValue.formatted
? context.parameters.sliderValue.formatted
: "0";
// appending the HTML elements to the component's HTML container element.
this._container.appendChild(thisInputElement);
this._container.appendChild(this.labelElement);
container.appendChild(this._container);
}

/**
 * Updates the values to the internal value variable we are storing and also updates the html label
that displays the value
 * @param context : The "Input Properties" containing the parameters, component metadata and
interface functions
*/
public reloadData(evt: Event): void {
this._value = (thisInputElement.value as any) as number;
this.labelElement.innerHTML = thisInputElement.value;
this._notifyOutputChanged();
}

public updateView(context: ComponentFramework.Context<IInputs>): void {
// storing the latest context from the control.
this._value = context.parameters.sliderValue.raw
? context.parameters.sliderValue.raw
: 0;
this._context = context;
thisInputElement.value =

context.parameters.sliderValue.formatted
? context.parameters.sliderValue.formatted
: "";

this.labelElement.innerHTML = context.parameters.sliderValue.formatted
? context.parameters.sliderValue.formatted
: "";
```

```
}

public getOutputs(): IOoutputs {
    return {
        sliderValue: this._value
    };
}

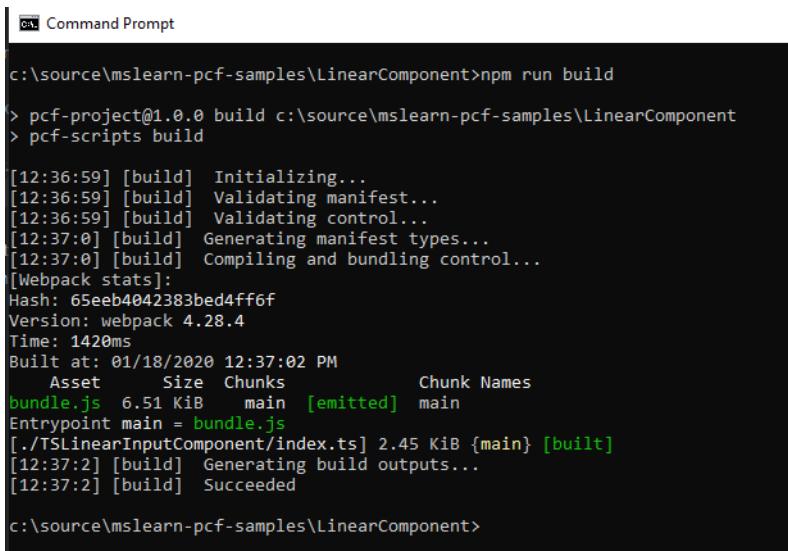
public destroy() {
    this.inputElement.removeEventListener("input", this._refreshData);
}
}
```

## Rebuild and run your code component

To rebuild and run your code component, follow these steps:

1. Now that your component's logic is implemented, go back to your command prompt to rebuild it by using this command:

```
npm run build
```



The screenshot shows a Windows Command Prompt window titled "Command Prompt". The command "npm run build" was entered, and the output shows the build process for a PCF project. The output includes logs from PCF scripts, Webpack, and the build process itself, indicating the creation of a bundle.js file and the entrypoint main = bundle.js. The build completed successfully at 12:37:02 PM on 01/18/2020.

```
c:\source\mslearn-pcf-samples\LinearComponent>npm run build
> pcf-project@1.0.0 build c:\source\mslearn-pcf-samples\LinearComponent
> pcf-scripts build

[12:36:59] [build]  Initializing...
[12:36:59] [build]  Validating manifest...
[12:36:59] [build]  Validating control...
[12:37:0] [build]  Generating manifest types...
[12:37:0] [build]  Compiling and bundling control...
[Webpack stats]:
Hash: 65eeb4042383bed4ff6f
Version: webpack 4.28.4
Time: 1420ms
Built at: 01/18/2020 12:37:02 PM
Asset      Size  Chunks      Chunk Names
bundle.js  6.51 KiB  main  [emitted]  main
Entrypoint main = bundle.js
[./TSLinearInputComponent/index.ts] 2.45 KiB {main}  [built]
[12:37:2] [build]  Generating build outputs...
[12:37:2] [build]  Succeeded

c:\source\mslearn-pcf-samples\LinearComponent>
```

2. Run your component in Node's test harness by running `npm start`. You can also enable watch mode to ensure that any changes to the following assets are made automatically without having to restart the test harness by using the `npm start --watch` command.
  - index.ts file.
  - ControlManifest.Input.xml file.
  - Imported libraries in index.ts.
  - All resources listed in the manifest file

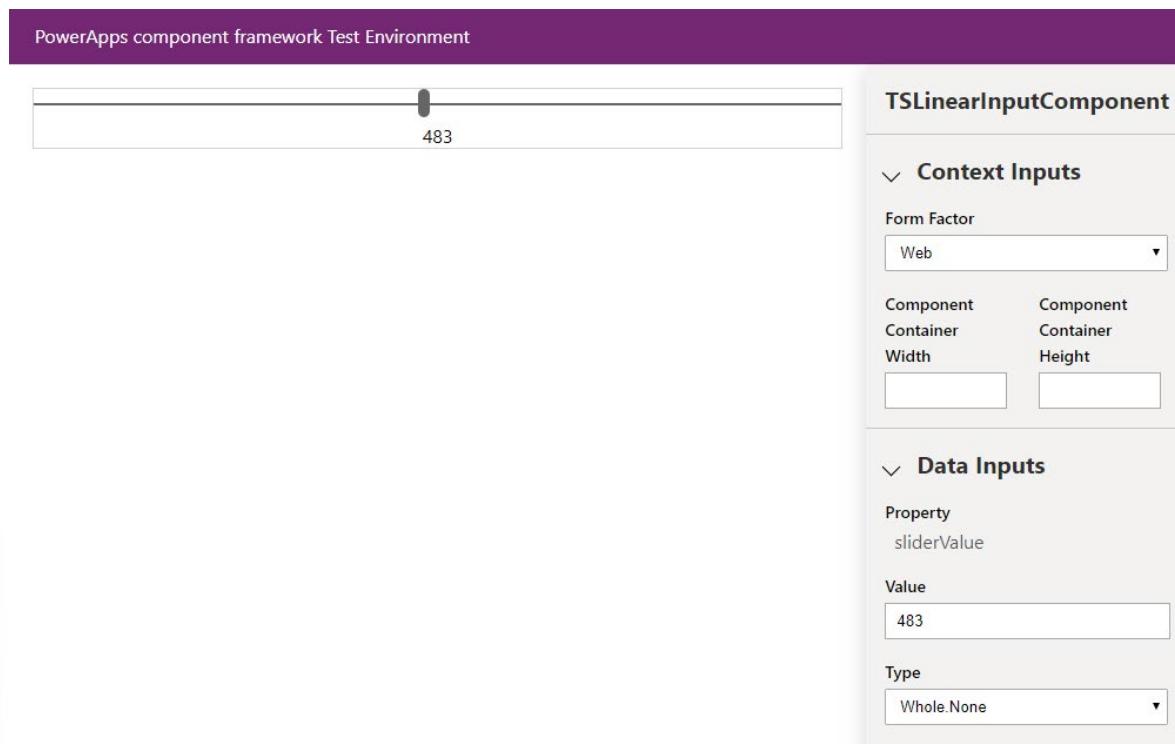
```
c:\source\mslearn-pcf-samples\LinearComponent>npm start
> pcf-project@1.0.0 start c:\source\mslearn-pcf-samples\LinearComponent
> pcf-scripts start

[13:0:29] [start] Initializing...
[13:0:29] [start] Validating manifest...
[13:0:29] [start] Validating control...
[13:0:30] [start] Generating manifest types...
[13:0:30] [start] Compiling and bundling control...
[Webpack stats]:
Hash: 12fba58e6603049e1ccc
Version: webpack 4.28.4
Time: 1595ms
Built at: 01/18/2020 1:00:32 PM
    Asset      Size  Chunks             Chunk Names
bundle.js   7.32 KiB  main  [emitted]  main
Entrypoint main = bundle.js
[./TSLinearInputComponent/index.ts] 3.21 KiB {main} [built]
[13:0:32] [start] Generating build outputs...
[13:0:32] [start] Starting control harness...

Starting control harness...

Mapping / to "c:\source\mslearn-pcf-samples\LinearComponent\node_modules\pcf-start"
Serving "c:\source\mslearn-pcf-samples\LinearComponent\out\controls\TSLinearInputComponent" at http://127.0.0.1:8181
Ready for changes
```

3. Observe your control in the test harness by going to the hosting address in a browser window (the window likely popped up for you automatically, but you can also reference the address as found in the command window, too).



## Create a code component solution package

### [!NOTE]

To run MSBuild, you need to have Visual Studio installed. (You can optionally install without Visual Studio:

**download MSBuild without Visual Studio<sup>15</sup>**). To access MSBuild, you

might need to add it to the path directory of your Windows environment variables. For example, Visual

<sup>15</sup> <https://visualstudio.microsoft.com/downloads/?q=build+tools>

Studio 2019 stores MSBuild

at C:\Program Files (x86)\Microsoft Visual Studio\2019\Enterprise\MSBuild\Current\Bin. You can also use the Visual Studio

Developer Command Prompt to access MSBuild, or run it by using the full qualified path ("C:\Program Files (x86)\Microsoft

Visual Studio\2019\Enterprise\MSBuild\Current\Bin\MSBuild.exe"/t:build /restore).

To use a code component in Power Apps, it must be deployed to an underlying Common Data Service instance. To deploy the component, your first task is to package your code component into a solution file that can then be distributed for import.

## Prerequisites

You must have an existing code component project that is already built. You can follow these steps for any other code component project, such as the **Sample components**<sup>16</sup>.

## Create a solution file with Power Apps CLI

To create a solution file with Power Apps CLI, follow these steps:

1. In the command prompt, go to your Power Apps component's project directory, which is the LinearComponent project that you built in the previous exercise. Create a new folder and name it **Solutions** (or any name of your choice) by using the command **mkdir Solutions**. Go to the directory by using the command **cd Solutions**.

```
c:\source\mslearn-pcf-samples\LinearComponent>mkdir Solutions
c:\source\mslearn-pcf-samples\LinearComponent>cd Solutions
c:\source\mslearn-pcf-samples\LinearComponent\Solutions>
```

2. Initialize your Common Data Service solution project with the following command:

```
pac solution init --publisher-name mslearn --publisher-prefix msl
```

The following image shows an example of the results that you should see.

```
c:\source\mslearn-pcf-samples\LinearComponent\Solutions>pac solution init --publisher-name mslearn --publisher-prefix msl
CDS solution project with name 'Solutions' created successfully in current directory.
CDS solution files were successfully created for this project in the sub-directory Other, using solution name Solutions, publisher name mslearn, and customization prefix msl.
Please verify the publisher information and solution name found in the Solution.xml file.
```

3. Inform your solution project. Its components will be added during the build. To accomplish this task, use the following command:

```
pac solution add-reference --path <path to your Power Apps component framework project>
```

```
c:\source\mslearn-pcf-samples\LinearComponent\Solutions> pac solution add-reference --path c:\source\mslearn-pcf-samples\LinearComponent\Project reference successfully added to CDS solution project.
```

<sup>16</sup> <https://docs.microsoft.com/powerapps/developer/component-framework/use-sample-components>

4. To generate your solution's zip file, use Microsoft Build Engine, or *MSBuild* for short. You'll only need to use the */restore* flag the first time that the solution project is built. In every subsequent build, you will need to run *msbuild* only.

```
msbuild /t:build /restore
```

The default package type is a Managed solution. If you want to export as Unmanaged (or Both), you can clear (or uncomment) the comment in the following section from your Solutions.cdsproj and edit the SolutionPackageType node accordingly:

```
<!-- Solution Packager overrides un-comment to use: SolutionPackagerType Managed, Unmanaged, Both-->
<PropertyGroup>
<SolutionPackageType>Unmanaged</SolutionPackageType>
</PropertyGroup>
```

You can set the msbuild configuration to **Release** to issue a production build, for example, `msbuild /p:configuration=Release`.

5. The generated solution files are found inside of the \bin\debug\ folder after the build is successful. You can manually import the containing zip file that is found in bin\Debug or bin\Release, depending on your release configuration. You can also programmatically deploy your code components by using the Power Apps CLI. For more information, see [Connecting to your environment<sup>17</sup>](#) and [Deploying code components<sup>18</sup>](#) sections of the [Package a code component<sup>19</sup>](#) documentation.

## Test and debug code components

As you begin to develop more complex components, you might need to test and debug their logic in interactive ways. One useful utility that you have is the Power Apps Component Test Harness, which allows you to test different data and context inputs to ensure that your component is functioning properly. Also, because your components are built with standard web technologies like TypeScript, CSS, and HTML, you have numerous utilities that are provided through that ecosystem, such as client debugging capabilities that are available in most modern browsers.

### Test and debug your code component in the Power Apps Component Test Harness

The Power Apps Component Test Harness is a utility built by Microsoft that helps you quickly render a Power Apps component in a localized environment. If you have gone through the exercise in this module on writing a code component, you have already seen this utility in action. You can start a local test harness from within a Power Apps component project by using the *npm start* command.

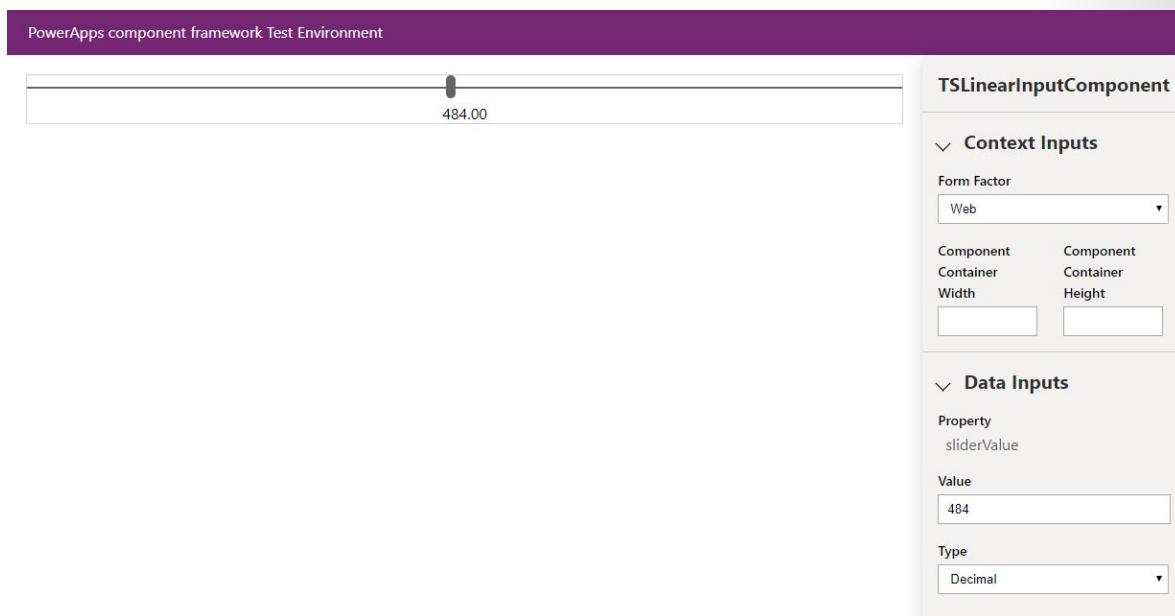
After the Test Harness has started, your component will display in a browser window, like the one that you built in the previous exercise.

---

<sup>17</sup> <https://docs.microsoft.com/powerapps/developer/component-framework/import-custom-controls#connecting-to-your-environment>

<sup>18</sup> <https://docs.microsoft.com/powerapps/developer/component-framework/import-custom-controls#deploying-code-components>

<sup>19</sup> <https://docs.microsoft.com/powerapps/developer/component-framework/import-custom-controls>



## Context Inputs

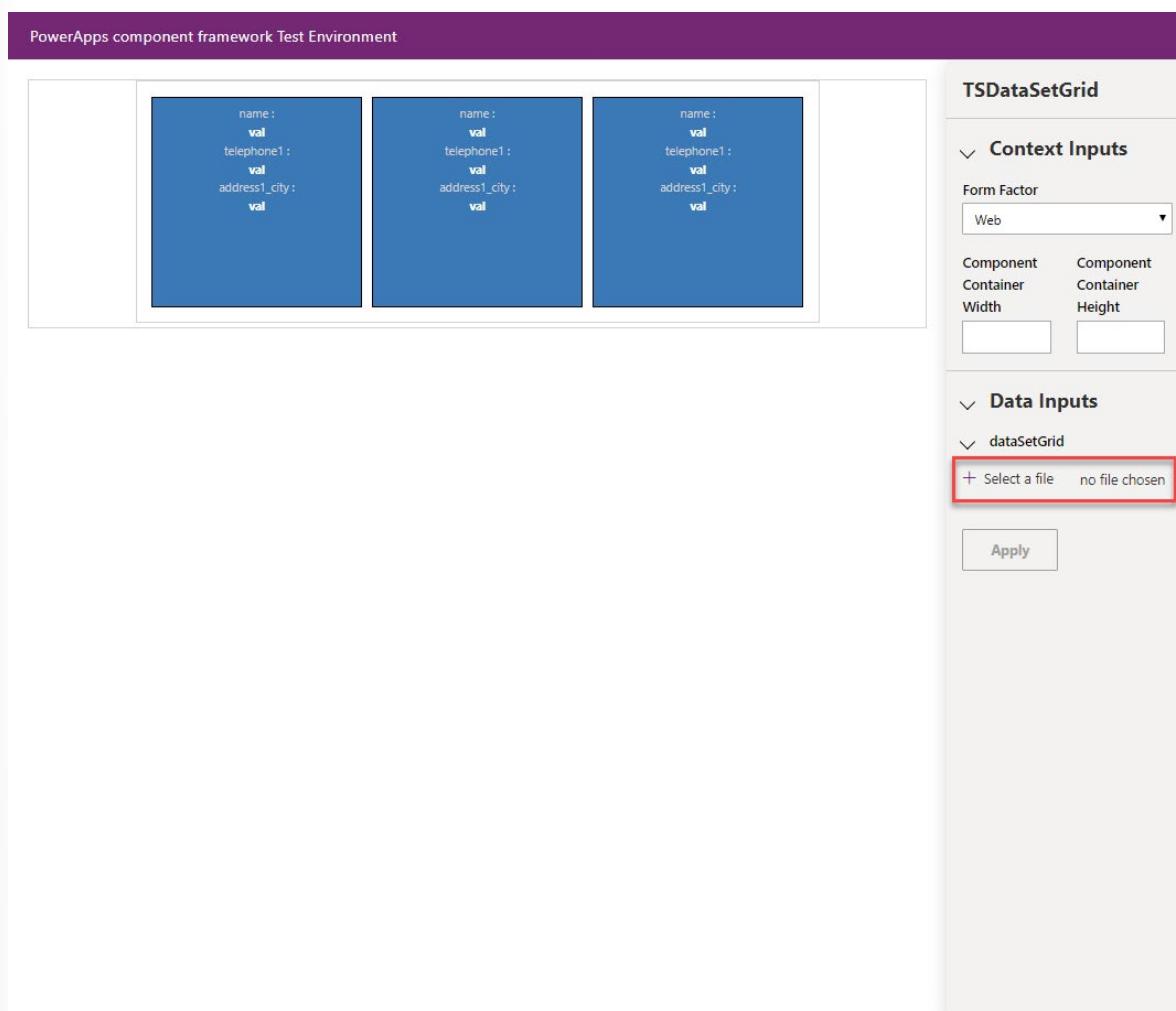
Within the test harness, you can provide your component with various inputs to specify how it is rendered (which is useful for ensuring that the component renders properly under different display contexts).

In the **Context Inputs** drop-down menu, you can select **Form Factor** such as **Web**, **Tablet**, and **Phone**. You can also provide explicit width and height fields to see how the component renders in various dimensions.

## Data Inputs

The **Data Inputs** drop-down menu helps you ensure that the component functions as expected when different data parameters are provided, and also allows you to determine how the component will render based on the type specified in a given field.

If your component is a dataset component, you can load CSV files with mock data. You can export a CSV from a sample target environment against which you are building your component, or you can build a new one.



In the previous example, if you wanted to build a new CSV file, it should look similar to the following sample:

name,telephone1,address1\_city

Joe, 123-555-0100, Seattle

Mary, 123-555-0101, Atlanta

Steve, 123-555-0102, Redmond

After the CSV has successfully loaded, the component will display its records and, as with your property controls, you can assign various types as specified by the component's corresponding type group (which is defined in the component's ControlManifest.Input.xml file).

The screenshot shows the PowerApps component framework Test Environment. On the left, there is a preview area displaying three data cards. Each card contains the following data:

- Card 1 (Joe):** name : Joe, telephone1 : 123-555-0100, address1\_city : Seattle
- Card 2 (Mary):** name : Mary, telephone1 : 123-555-0101, address1\_city : Atlanta
- Card 3 (Steve):** name : Steve, telephone1 : 123-555-0102, address1\_city : Redmond

To the right of the preview is the component configuration interface. It includes the following sections:

- TSDDataSetGrid**
- Context Inputs** (expanded)
  - Form Factor: Web
  - Component Container Width: [Input field]
  - Component Container Height: [Input field]
- Data Inputs** (expanded)
  - dataSetGrid** (expanded)
    - + Select a file: griddata.csv
    - Column: name
    - Type: SingleLine.Text
  - Column: telephone1
  - Type: SingleLine.Text
  - Column: address1\_city
  - Type: SingleLine.Text
- Apply** button

## Debug a code component by using browser debugging

Most modern browsers have a variety of built-in debugging capabilities. Microsoft Edge, Google Chrome, Mozilla Firefox, and Apple Safari each have built-in developer tools that allow for interactive debugging experiences. For more details on each browser, refer to the following links.

Browser	Developer Tools Documentation
Microsoft Edge (Chromium)	<a href="https://docs.microsoft.com/microsoft-edge/devtools-guide-chromium">https://docs.microsoft.com/microsoft-edge/devtools-guide-chromium</a>
Microsoft Edge (EdgeHTML)	<a href="https://docs.microsoft.com/microsoft-edge/devtools-guide">https://docs.microsoft.com/microsoft-edge/devtools-guide</a>
Google Chrome	<a href="https://developers.google.com/web/tools/chrome-devtools">https://developers.google.com/web/tools/chrome-devtools</a>
Mozilla Firefox	<a href="https://developer.mozilla.org/docs/Tools/Debugger">https://developer.mozilla.org/docs/Tools/Debugger</a>

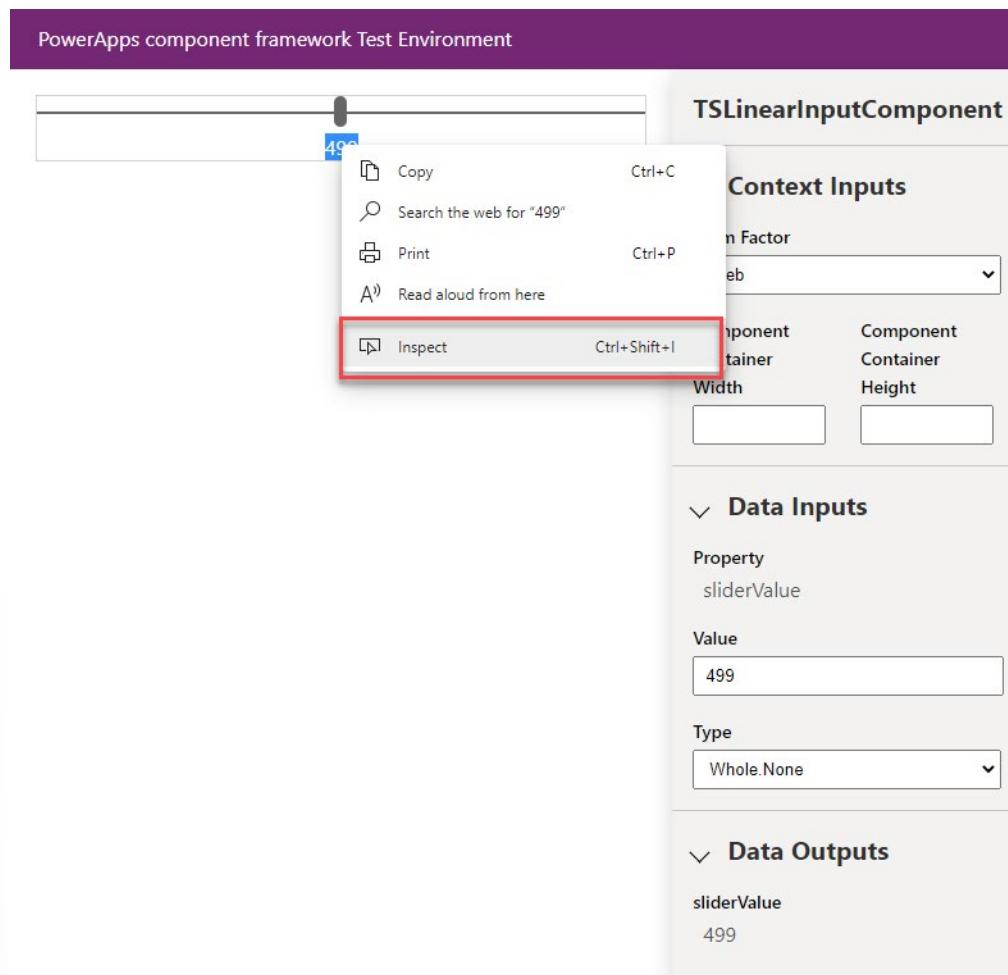
Browser	Developer Tools Documentation
Apple Safari	<a href="https://support.apple.com/guide/safari-developer/welcome/mac">https://support.apple.com/guide/safari-developer/welcome/mac</a>

For this exercise, you will use Microsoft Edge (Chromium). Start Microsoft Edge's DevTools by pressing F12 on your keyboard.

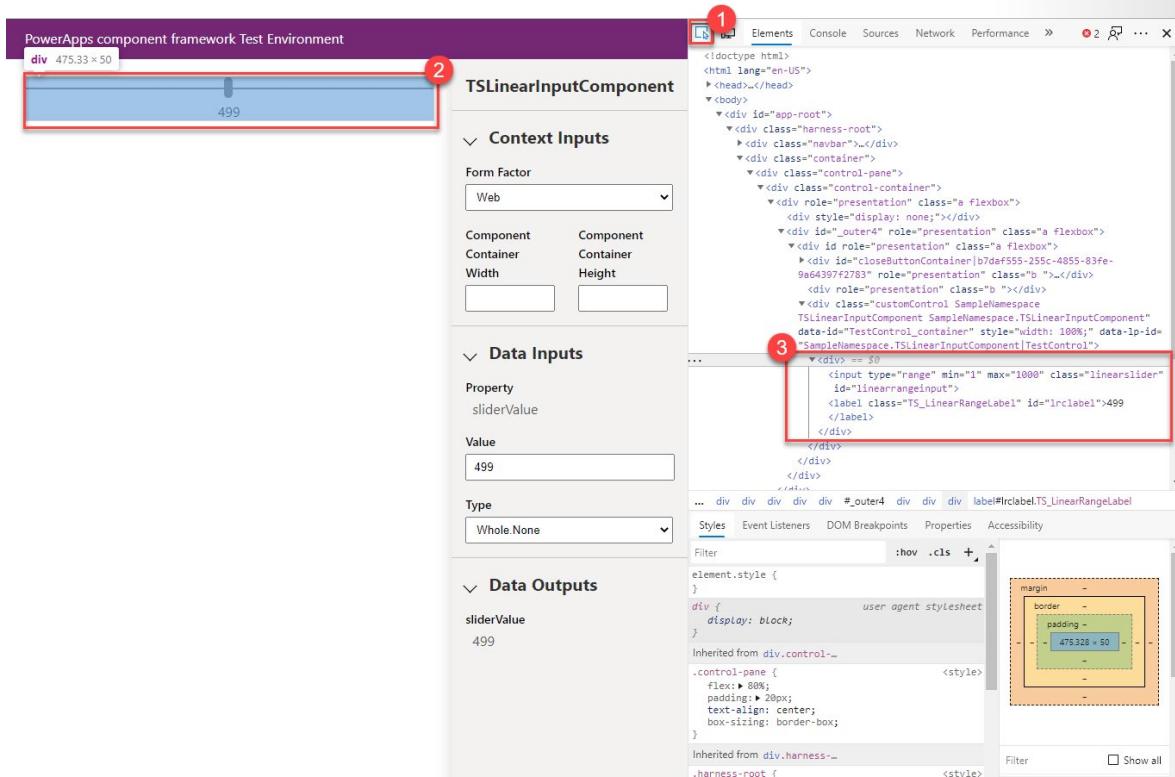
## Inspect your HTML with the Elements panel

In DevTools, the first available tab (**Elements**) shows the Elements panel, which provides you with a way to view the HTML that is rendered within the page. You can go to where your control is being rendered by using the Inspect functionality, which can be accessed in one of two ways:

1. Highlight and right-click an element within your page and then select **Inspect**.



2. Select an element on the Elements panel.



## Inspect console logs and run script in the Console panel

A frequent mechanism for providing debug context within client script is to use the `console.log()` method. If you want to provide logging inside of your component's logic, you can use this method. These logs are displayed in the DevTools console panel whenever they are run, which provides a valuable way to trace logic as it runs within your component. The following image shows an example of a couple of logs that were written by your test harness.

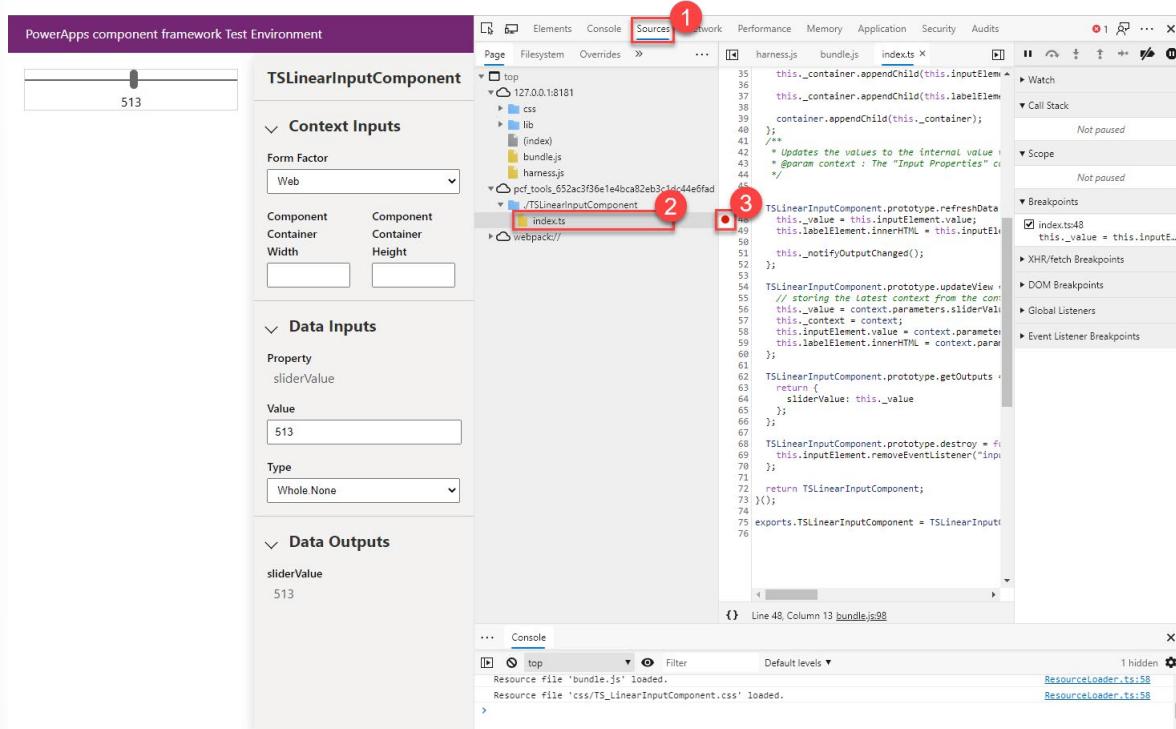
```
Resource file 'bundle.js' loaded.          ResourceLoader.ts:58
Resource file 'css/TS_LinearInputComponent.css' loaded.  ResourceLoader.ts:58
```

You also have the ability to run your own script from within the console. This method can be valuable for testing various method calls and logic from within the context of a live environment.

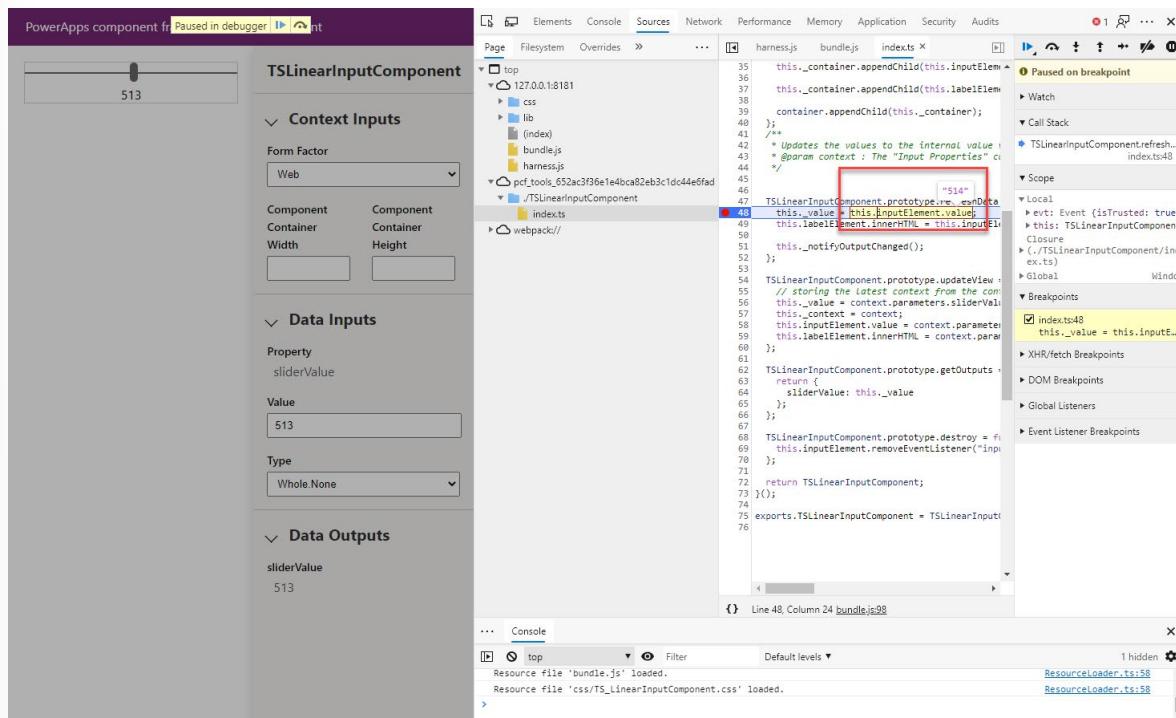
```
Resource file 'bundle.js' loaded.          ResourceLoader.ts:58
Resource file 'css/TS_LinearInputComponent.css' loaded.  ResourceLoader.ts:58
> document.getElementById("linearrangeinput").value
< "499"
```

## Set debugger breakpoints in the Sources panel

One of the most valuable utilities in DevTools is the ability to set debugger breakpoints in your code so that you can inspect variables and the flow of your method implementation. This example sets a breakpoint in your `index.ts` file that you developed as part of your Power Apps component. In the following example, a breakpoint is set to run any time that the `refreshData` method is called.



If you change the value of your slider control, this breakpoint is hit, providing you with the ability to manually step through the logic of the event handler. Additionally, you can inspect values, such as those that have changed, as shown in the following image.



## summary

Writing Power Apps components is a great skillset for developers who want to extend a Power Apps application with custom presentation and logic that is not able to be accomplished through configuration. This module reviewed how to create, package, and debug code components and explained how to install sample components that are found in the community.

## Use advanced features with Power Apps component framework

### Introduction to using React within a Power Apps component

React is a standardized client framework that is built by Facebook for building user interfaces. It provides a declarative way to create interactive UIs and provides a mechanism to encapsulate components to make complex UIs that manage component states and have high interactivity. Because React is written in JavaScript, you can use this framework within a Power Apps component.

If you're new to React, go to <https://reactjs.org/><sup>20</sup>, which provides a wealth of tutorials and resources on how to build React components.

### UI fabric

One of the many great developments from Microsoft has been its implementation of UI Fabric, a collection of UX frameworks that you can use to build fluent experiences that fit seamlessly into a broad range of Microsoft products. Using UI Fabric within your Power Apps component is as simple as referencing its libraries, and it provides a React-specific version that you can use. For more details on UI Fabric, go to

<https://developer.microsoft.com/fabric><sup>21</sup>.

### Implement a sample Facepile component

[!NOTE]

Download files<sup>22</sup> to use with this exercise.

In this example, you'll build a component that uses the Facepile<sup>23</sup>

UI Fabric component. The Facepile shows a list of faces or initials in a horizontal lookup, each circle representing a person.

A practical example of when you might use this lookup is to list contributors to an article or record, such as what you would see in Microsoft Docs, as shown in the following image.

## Common Data Service Developer Guide

03/27/2019 • 2 minutes to read • 

<sup>20</sup> <https://reactjs.org/?azure-portal=true>

<sup>21</sup> <https://developer.microsoft.com/fabric#/?azure-portal=true>

<sup>22</sup> <https://github.com/MicrosoftDocs/mslearn-developer-tools-power-platform/tree/master/power-apps-component-framework>

<sup>23</sup> <https://developer.microsoft.com/fabric-js/components/facepile/facepile/?azure-portal=true>

## Initialize your component's project

To initialize your component's project, follow these steps:

1. Initialize the project by running the following command:

```
pac pcf init --namespace SampleControls --name ReactStandardControl --template field
```

2. Add the following dependencies to your package.json to indicate that you'll be using React and Office UI Fabric libraries.

```
"office-ui-fabric-react": "^6.189.0",
"react": "^16.8.6",
"react-dom": "^16.8.6"
```

3. Add the following dev dependencies to ensure that types for React are installed.

```
"@types/react": "^16.8",
"@types/react-dom": "^16.8"
```

4. Add the following compiler options to your tsconfig.json to indicate we want to use react:

```
"jsx": "react",
"jsxFactory": "React.createElement"
```

5. Run npm install to load dependent libraries into your project.

## Implement your code component's logic

To implement your code component's logic, follow these steps:

1. Open your code component's manifest file (ControlManifest.Input.xml) and replace it with the following logic:

```
<?xml version="1.0" encoding="utf-8" ?>
<manifest>
  <control namespace="SampleControls" constructor="ReactStandardControl" version="0.0.1"
  display-name-key="ReactStandardControl_Display_Key" description-key="ReactStandardControl_
  Desc_Key" control-type="standard">
    <!-- property node identifies a specific, configurable piece of data that the control expects from
    CDS -->
    <property name="numberOfFaces" display-name-key="numberOfFaces" description-key="num-
    berOfFaces" of-type="Whole.None" usage="bound" required="false" />
    <resources>
      <css path="css/ReactStandardControl.css" order="1" />
      <code path="index.ts" order="2"/>
    </resources>
  </control>
</manifest>
```

You'll be adding the supporting files that are found in this manifest later.

2. Open the index.ts file and add the following import statements:

```
import { IInputs, IOutputs } from "./generated/ManifestTypes";
import * as React from "react";
import * as ReactDOM from "react-dom";
import { FacepileBasicExample, IFacepileBasicExampleProps } from "./Facepile";
```

3. Above the constructor method, insert the following private variables:

```
// reference to the notifyOutputChanged method
private notifyOutputChanged: () => void;
// reference to the container div
private theContainer: HTMLDivElement;
// reference to the React props, prepopulated with a bound event handler
private props: IFacepileBasicExampleProps = {
    numberFacesChanged: this.numberFacesChanged.bind(this)
};
```

4. Place the following logic inside of the *init* method:

```
this.notifyOutputChanged = notifyOutputChanged;
this.props.numberOfFaces = context.parameters.numberOfFaces.raw || 3;
this.theContainer = container;
```

5. Add the following logic to the *updateView* method:

```
if (context.updatedProperties.includes("numberOfFaces"))
    this.props.numberOfFaces = context.parameters.numberOfFaces.raw || 3;

// Render the React component into the div container
ReactDOM.render(
    // Create the React component
    React.createElement(
        FacepileBasicExample, // the class type of the React component found in Facepile.tsx
        this.props
    ),
    this.theContainer
);
```

6. Add the following logic to the *getOutputs* method:

```
numberOfFaces: this.props.numberOfFaces
```

7. Add the following logic to the *destroy* method:

```
ReactDOM.unmountComponentAtNode(this.theContainer);
```

8. After you've made the updates, your completed class should look similar to the following example:

```
import { IInputs, IOutputs } from "./generated/ManifestTypes";
import * as React from "react";
import * as ReactDOM from "react-dom";
import { FacepileBasicExample, IFacepileBasicExampleProps } from "./Facepile";

export class ReactStandardControl
    implements ComponentFramework.StandardControl<IInputs, IOutputs> {
```

```
// reference to the notifyOutputChanged method
private notifyOutputChanged: () => void;
// reference to the container div
private theContainer: HTMLDivElement;
// reference to the React props, prepopulated with a bound event handler
private props: IFacepileBasicExampleProps = {
    numberFacesChanged: this.numberFacesChanged.bind(this)
};

/**
 * Empty constructor.
 */
constructor() {}

/**
 * Used to initialize the control instance. Controls can kick off remote server calls and other initialization actions here.
 * Dataset values are not initialized here, use updateView.
 * @param context The entire property bag that is available to control through the Context Object; It contains values as set up by the customizer that are mapped to property names that are defined in the manifest and to utility functions.
 * @param notifyOutputChanged A callback method to alert the framework that the control has new outputs that are ready to be retrieved asynchronously.
 * @param state A piece of data that persists in one session for a single user. Can be set at any point in a control's life cycle by calling 'setControlState' in the Mode interface.
 * @param container If a control is marked control-type='starndard', it will receive an empty div element within which it can render its content.
 */
public init(
    context: ComponentFramework.Context<IInputs>,
    notifyOutputChanged: () => void,
    state: ComponentFramework.Dictionary,
    container: HTMLDivElement
) {
    this.notifyOutputChanged = notifyOutputChanged;
    this.props.numberOfFaces = context.parameters.numberOfFaces.raw || 3;
    this.theContainer = container;
}

/**
 * Called when any value in the property bag has changed. This includes field values, datasets, global values such as container height and width, offline status, control metadata values such as label, visible, and so on.
 * @param context The entire property bag that is available to control through the Context Object; It contains values as set up by the customizer that is mapped to names that are defined in the manifest and to utility functions
 */
public updateView(context: ComponentFramework.Context<IInputs>): void {
    if (context.updatedProperties.includes("numberOfFaces"))
        this.props.numberOfFaces = context.parameters.numberOfFaces.raw || 3;
```

```
// Render the React component into the div container
ReactDOM.render(
  // Create the React component
  React.createElement(
    FacepileBasicExample, // the class type of the React component found in Facepile.tsx
    this.props
  ),
  this.theContainer
);
}

/**
 * Called by the React component when it detects a change in the number of faces shown
 * @param newValue The newly detected number of faces
 */
private numberFacesChanged(newValue: number) {
  // only update if the number of faces has truly changed
  if (this.props.numberOfFaces !== newValue) {
    this.props.numberOfFaces = newValue;
    this.notifyOutputChanged();
  }
}

/**
 * It is called by the framework prior to a control receiving new data.
 * @returns an object based on nomenclature that is defined in the manifest, expecting object[s] for
property marked as "bound" or "output"
*/
public getOutputs(): IOutputs {
  return {
    numberOfFaces: this.props.numberOfFaces
  };
}

/**
 * Called when the control is to be removed from the DOM tree. Controls should use this call for
cleanup,
 * for example, canceling any pending remote calls, removing listeners, and so on.
*/
public destroy(): void {
  ReactDOM.unmountComponentAtNode(this.theContainer);
}
}
```

## Add styling to your code component

To add styling to your code component, follow these steps:

1. Create a new CSS subfolder under the ReactStandardControl folder.
2. Create a new ReactStandardControl.css file inside the CSS subfolder.

3. Add the following style content to the ReactStandardControl.css file:

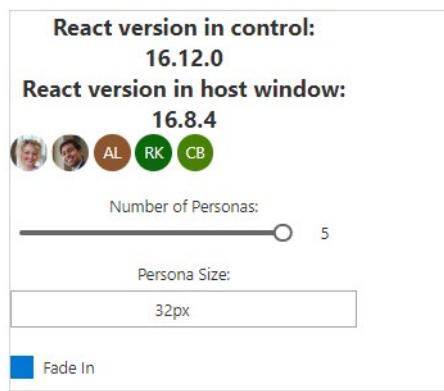
```
.msFacepileExample {  
    max-width: 300px;  
}  
  
.msFacepileExample .control {  
    padding-top: 20px;  
}  
  
.msFacepileExample .ms-Dropdown-container,  
.msFacepileExample .ms-Slider {  
    margin: 10px 0 10px 0;  
}  
  
.msFacepileExample .ms-Dropdown-container .ms-Label {  
    padding-top: 0;  
}  
  
.msFacepileExample .ms-Checkbox {  
    padding-top: 15px;  
}  
  
.exampleCheckbox {  
    margin: 10px 0;  
}  
  
.exampleLabel {  
    margin: 10px 0;  
}
```

4. Save the ReactStandardControl.css file.

## Add React component logic

To add React component logic, follow these steps:

1. Add the following files that are found in the supporting libraries of this exercise to the ReactStandardControl folder:
  - Facepile.tsx
  - FacepileExampledata.ts
  - ReactVersion.Tsx
  - TestImages.ts
2. Build your solution by running npm run build.
3. Upon a successful build, you can test your new Facepile component by running npm start, which should look like the following example:



## Use the formatting API in a Power Apps component

Power Apps component framework exposes a formatting API that can be especially useful when you need to format various values in your application. This unit shows how to use this API by producing an HTML table to illustrate how various methods can be used.

Example Method	Result
formatCurrency()	\$10,250,030.00
formatDecimal()	123,456.28
formatInteger()	12,345
formatLanguage()	English
formatDateYearMonth()	February 2020
getWeekOfYear()	0

## Initialize your component's project

To initialize your component's project, follow these steps:

1. Initialize the project by running the following command:

```
pac pcf init --namespace SampleNamespace --name FormattingAPI --template field
```

2. Run npm install to load dependent libraries into your project.

## Implement your code component's logic

To implement your code component's logic, follow these steps:

1. Open your code component's manifest file (ControlManifest.Input.xml) and replace it with the following logic:

```
<?xml version="1.0" encoding="utf-8" ?>
<manifest>
  <control namespace="SampleNamespace" constructor="FormattingAPI" version="1.1.0" display-name-key="TS_FormattingAPI_Display_Key" description-key="TS_FormattingAPI_Desc_Key" con-
```

```
trol-type="standard">
    <property name="controlValue" display-name-key="controlValue_Display_Key" description-key="-
controlValue_Desc_Key" of-type="SingleLine.Text" usage="bound" required="true" />
    <resources>
        <code path="index.ts" order="1" />
        <css path="css/TS_FormattingAPI.css" order="2" />
    </resources>
</control>
</manifest>
```

You'll be adding the supporting files that are found in this manifest later.

2. Open the index.ts file.
3. Above the constructor method, insert the following private variables:

```
// PCF framework delegate that will be assigned to this object that would be called whenever any
update happens.
private _notifyOutputChanged: () => void;
// Reference to the div element that holds together all the HTML elements that you are creating
as part of this control
private divElement: HTMLDivElement;

// Reference to HTMLTableElement rendered by control
private _tableElement: HTMLTableElement;

// Reference to the control container HTMLDivElement
// This element contains all elements of your custom control example
private _container: HTMLDivElement;

// Reference to ComponentFramework Context object
private _context: ComponentFramework.Context<Inputs>;

// Flag if control view has been rendered
private _controlViewRendered: Boolean;
```

4. Place the following logic inside of the *init* method:

```
this._notifyOutputChanged = notifyOutputChanged;
this._controlViewRendered = false;
this._context = context;

this._container = document.createElement("div");
this._container.classList.add("TSFormatting_Container");
container.appendChild(this._container);
```

5. Add the following logic to the *updateView* method:

```
if (!this._controlViewRendered)
{
    // Render and add HTMLTable to the custom control container element
    let tableElement: HTMLTableElement = this.createHTMLTable();
    this._container.appendChild(tableElement);
```

```
        this._controlViewRendered = true;  
    }  
  
6. Add the following helper methods to generate your HTML table that will display formatted values:
```

```
/**  
 * Helper method to create an HTML Table Row Element  
 * @param key : string value to show in left column cell  
 * @param value : string value to show in right column cell  
 * @param isHeaderRow : true if method should generate a header row  
 */  
private createHTMLTableRowElement(key: string, value: string, isHeaderRow: boolean): HTMLTableRowElement  
{  
    let keyCell: HTMLTableCellElement = this.createHTMLTableCellElement(key, "FormattingControlSampleHtmlTable_HtmlCell_Key", isHeaderRow);  
    let valueCell: HTMLTableCellElement = this.createHTMLTableCellElement(value, "FormattingControlSampleHtmlTable_HtmlCell_Value", isHeaderRow);  
  
    let rowElement: HTMLTableRowElement = document.createElement("tr");  
    rowElement.setAttribute("class", "FormattingControlSampleHtmlTable_HtmlRow");  
    rowElement.appendChild(keyCell);  
    rowElement.appendChild(valueCell);  
  
    return rowElement;  
}  
  
/**  
 * Helper method to create an HTML Table Cell Element  
 * @param cellValue : string value to inject in the cell  
 * @param className : class name for the cell  
 * @param isHeaderRow : true if method should generate a header row cell  
 */  
private createHTMLTableCellElement(cellValue: string, className: string, isHeaderRow: boolean): HTMLTableCellElement  
{  
    let cellElement: HTMLTableCellElement;  
  
    if (isHeaderRow)  
    {  
        cellElement = document.createElement("th");  
        cellElement.setAttribute("class", "FormattingControlSampleHtmlTable_HtmlHeaderCell " +  
        className);  
        let textElement: Text = document.createTextNode(cellValue);  
        cellElement.appendChild(textElement);  
    }  
    else  
    {  
        cellElement = document.createElement("td");  
        cellElement.setAttribute("class", "FormattingControlSampleHtmlTable_HtmlCell " +
```

```
    className);
    let textElement: Text = document.createTextNode(cellValue);
    cellElement.appendChild(textElement);
}
return cellElement;
}
```

7. Add the following method, which contains sample usages of the formatting API, by saving them as table cells.

```
/**
 * Creates an HTML Table that showcases examples of basic methods available to the custom
control
 * The left column of the table shows the method name or property that is being used
 * The right column of the table shows the result of that method name or property
*/
private createHTMLTableElement(): HTMLTableElement
{
    // Create HTML Table Element
    let tableElement: HTMLTableElement = document.createElement("table");
    tableElement.setAttribute("class", "FormattingControlSampleHtmlTable_HtmlTable");

    // Create header row for table
    let key: string = "Example Method";
    let value: string = "Result";
    tableElement.appendChild(this.createHTMLTableRowElement(key, value, true));

    // Example use of formatCurrency() method
    // Change the default currency and the precision or pass in the precision and currency as
additional parameters.
    key = "formatCurrency()";
    value = this._context.formatting.formatCurrency(10250030);
    tableElement.appendChild(this.createHTMLTableRowElement(key, value, false));

    // Example use of formatDecimal() method
    // Change the settings from user settings to see the output change its format accordingly
    key = "formatDecimal()";
    value = this._context.formatting.formatDecimal(123456.2782);
    tableElement.appendChild(this.createHTMLTableRowElement(key, value, false));

    // Example use of formatInteger() method
    // change the settings from user settings to see the output change its format accordingly.
    key = "formatInteger()";
    value = this._context.formatting.formatInteger(12345);
    tableElement.appendChild(this.createHTMLTableRowElement(key, value, false));

    // Example use of formatLanguage() method
    // Install additional languages and pass in the corresponding language code to see its string
value
    key = "formatLanguage()";
    value = this._context.formatting.formatLanguage(1033);
```

```
tableElement.appendChild(this.createHTMLTableRowElement(key, value, false));

// Example of formatDateYearMonth() method
// Pass a JavaScript Data object set to the current time into formatDateYearMonth method to
format the data
// and get the return in Year, Month format
key = "formatDateYearMonth()";
value = this._context.formatting.formatDateYearMonth(new Date());
tableElement.appendChild(this.createHTMLTableRowElement(key, value, false));

// Example of getWeekOfYear() method
// Pass a JavaScript Data object set to the current time into getWeekOfYear method to get the
value for week of the year
key = "getWeekOfYear()";
value = this._context.formatting.getWeekOfYear(new Date().toString());
tableElement.appendChild(this.createHTMLTableRowElement(key, value, false));

return tableElement;
}
```

After you've made the updates, your completed class should look similar to the following example:

This file is part of the Microsoft Power Apps code samples.  
Copyright (C) Microsoft Corporation. All rights reserved.  
This source code is intended only as a supplement to Microsoft Development Tools and/or  
online documentation. See these other materials for detailed information regarding  
Microsoft code samples.

THIS CODE AND INFORMATION ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND,  
EITHER  
EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF  
MERCHANTABILITY AND/OR FITNESS FOR A PARTICULAR PURPOSE.  
\*/

```
import {IInputs, IOutputs} from "./generated/ManifestTypes";

export class FormattingAPI implements ComponentFramework.StandardControl<IInputs, IOutputs>
{
    // PCF framework delegate that will be assigned to this object that would be called whenever any
    update happens.
    private _notifyOutputChanged: () => void;

    // Reference to the div element that holds together all the HTML elements that you are creating
    as part of this control
    private divElement: HTMLDivElement;

    // Reference to HTMLElement rendered by control
    private _tableElement: HTMLElement;

    // Reference to the control container HTMLDivElement
```

```
// This element contains all elements of your custom control example
private _container: HTMLDivElement;

// Reference to ComponentFramework Context object
private _context: ComponentFramework.Context<IInputs>;

// Flag if control view has been rendered
private _controlViewRendered: Boolean;

/**
 * Used to initialize the control instance. Controls can kick off remote server calls and other
initialization actions here.
 * Dataset values are not initialized here, use updateView.
 * @param context The entire property bag is available to control through the Context Object; It
contains values as set up by the customizer and mapped to property names that are defined in the
manifest and to utility functions.
 * @param notifyOutputChanged A callback method to alert the framework that the control has
new outputs ready to be retrieved asynchronously.
 * @param state A piece of data that persists in one session for a single user. Can be set at any
point in a control's life cycle by calling 'setControlState' in the Mode interface.
 * @param container If a control is marked control-type='starndard', it will receive an empty div
element within which it can render its content.
 */
public init(context: ComponentFramework.Context<IInputs>, notifyOutputChanged: () => void,
state: ComponentFramework.Dictionary, container:HTMLDivElement)
{
    this._notifyOutputChanged = notifyOutputChanged;
    this._controlViewRendered = false;
    this._context = context;

    this._container = document.createElement("div");
    this._container.classList.add("TSFormatting_Container");
    container.appendChild(this._container);
}

/**
 * Called when any value in the property bag has changed. This includes field values, datasets,
global values such as container height and width, offline status, control metadata values such as label,
visible, and so on.
 * @param context The entire property bag that is available to control through the Context
Object; It contains values as set up by the customizer that are mapped to names defined in the
manifest and to utility functions
*/
public updateView(context: ComponentFramework.Context<IInputs>): void
{
    if (!this._controlViewRendered)
    {
        // Render and add HTMLTable to the custom control container element
        let tableElement: HTMLTableElement = this.createHTMLTableElement();
        this._container.appendChild(tableElement);
    }
}
```

```
        this._controlViewRendered = true;
    }
}

/***
 * It is called by the framework prior to a control receiving new data.
 * @returns an object based on nomenclature that is defined in the manifest, expecting object[s]
for property marked as "bound" or "output"
*/
public getOutputs(): IOutputs
{
    // no-op: method not used by this example custom control
    return { };
}

/***
 * Called when the control is to be removed from the DOM tree. Controls should use this call for
cleanup,
 * such as canceling any pending remote calls, removing listeners, and so on.
*/
public destroy()
{

}

/***
 * Helper method to create an HTML Table Row Element
 * @param key : string value to show in left column cell
 * @param value : string value to show in right column cell
 * @param isHeaderRow : true if method should generate a header row
*/
private createHTMLTableRowElement(key: string, value: string, isHeaderRow: boolean): HTMLTableElement
{
    let keyCell: HTMLTableCellElement = this.createHTMLTableCellElement(key, "FormattingCon-
trolSampleHtmlTable_HtmlCell_Key", isHeaderRow);
    let valueCell: HTMLTableCellElement = this.createHTMLTableCellElement(value, "Formatting-
ControlSampleHtmlTable_HtmlCell_Value", isHeaderRow);

    let rowElement: HTMLTableElement = document.createElement("tr");
    rowElement.setAttribute("class", "FormattingControlSampleHtmlTable_HtmlRow");
    rowElement.appendChild(keyCell);
    rowElement.appendChild(valueCell);

    return rowElement;
}

/***
 * Helper method to create an HTML Table Cell Element
 * @param cellValue : string value to inject in the cell
 * @param className : class name for the cell
*/
```

```
* @param isHeaderRow : true if method should generate a header row cell
*/
private createHTMLTableCellElement(cellValue: string, className: string, isHeaderRow: boolean): HTMLTableCellElement
{
    let cellElement: HTMLTableCellElement;

    if (isHeaderRow)
    {
        cellElement = document.createElement("th");
        cellElement.setAttribute("class", "FormattingControlSampleHtmlTable_HtmlHeaderCell " + className);
        let textElement: Text = document.createTextNode(cellValue);
        cellElement.appendChild(textElement);
    }
    else
    {
        cellElement = document.createElement("td");
        cellElement.setAttribute("class", "FormattingControlSampleHtmlTable_HtmlCell " + className);
        let textElement: Text = document.createTextNode(cellValue);
        cellElement.appendChild(textElement);
    }
    return cellElement;
}

/**
 * Creates an HTML Table that showcases examples of basic methods that are available to the
custom control
 * The left column of the table shows the method name or property that is being used
 * The right column of the table shows the result of that method name or property
*/
private createHTMLTableElement(): HTMLTableElement
{
    // Create HTML Table Element
    let tableElement: HTMLTableElement = document.createElement("table");
    tableElement.setAttribute("class", "FormattingControlSampleHtmlTable_HtmlTable");

    // Create header row for table
    let key: string = "Example Method";
    let value: string = "Result";
    tableElement.appendChild(this.createHTMLTableRowElement(key, value, true));

    // Example use of formatCurrency() method
    // Change the default currency and the precision or pass in the precision and currency as
additional parameters.
    key = "formatCurrency()";
    value = this._context.formatting.formatCurrency(10250030);
    tableElement.appendChild(this.createHTMLTableRowElement(key, value, false));

    // Example use of formatDecimal() method
}
```

```
// Change the settings from user settings to see the output change its format accordingly
key = "formatDecimal()";
value = this._context.formatting.formatDecimal(123456.2782);
tableElement.appendChild(this.createHTMLTableRowElement(key, value, false));

// Example use of formatInteger() method
// change the settings from user settings to see the output change its format accordingly.
key = "formatInteger()";
value = this._context.formatting.formatInteger(12345);
tableElement.appendChild(this.createHTMLTableRowElement(key, value, false));

// Example use of formatLanguage() method
// Install additional languages and pass in the corresponding language code to see its string
value
key = "formatLanguage()";
value = this._context.formatting.formatLanguage(1033);
tableElement.appendChild(this.createHTMLTableRowElement(key, value, false));

// Example of formatDateYearMonth() method
// Pass a JavaScript Data object set to the current time into formatDateYearMonth method to
format the data
// and get the return in Year, Month format
key = "formatDateYearMonth()";
value = this._context.formatting.formatDateYearMonth(new Date());
tableElement.appendChild(this.createHTMLTableRowElement(key, value, false));

// Example of getWeekOfYear() method
// Pass a JavaScript Data object set to the current time into getWeekOfYear method to get the
value for week of the year
key = "getWeekOfYear()";
value = this._context.formatting.getWeekOfYear(new Date()).toString();
tableElement.appendChild(this.createHTMLTableRowElement(key, value, false));

return tableElement;
}

}
```

## Add styling to your code component

To add styling to your code component, follow these steps:

1. Create a new CSS subfolder under the FormattingAPI folder.
2. Create a new TS\_FormattingAPI.css file inside the CSS subfolder.
3. Add the following style content to the TS\_FormattingAPI.css file:

```
/*
This file is part of the Microsoft Power Apps code samples.
Copyright (C) Microsoft Corporation. All rights reserved.
This source code is intended only as a supplement to Microsoft Development Tools and/or
```

online documentation. See these other materials for detailed information regarding Microsoft code samples.

THIS CODE AND INFORMATION ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER

EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY AND/OR FITNESS FOR A PARTICULAR PURPOSE.

```
/*
.SampleNamespace\FormattingAPI
{
    font-family: 'SegoeUI-Semibold', 'Segoe UI Semibold', 'Segoe UI Regular', 'Segoe UI';
}

.SampleNamespace\FormattingAPI .TSFormatting_Container
{
    overflow-x: auto;
}

.SampleNamespace\FormattingAPI .FormattingControlSampleHtmlTable_HtmlRow
{
    background-color: #FFFFFF;
}

.SampleNamespace\FormattingAPI .FormattingControlSampleHtmlTable_HtmlHeaderCell
{
    text-align: center;
}

.SampleNamespace\FormattingAPI .FormattingControlSampleHtmlTable_HtmlCell,
.SampleNamespace\FormattingAPI .FormattingControlSampleHtmlTable_HtmlHeaderCell
{
    border: 1px solid black;
    padding-left: 3px;
    padding-right: 3px;
}

.SampleNamespace\FormattingAPI .FormattingControlSampleHtmlTable_HtmlInputTextCell
{
    border: 1px solid black;
    padding: 0px;
}

.SampleNamespace\FormattingAPI .FormattingControlSampleHtmlTable_HtmlHeaderCell
{
    font-weight: bold;
    font-size: 16px;
}

.SampleNamespace\FormattingAPI .FormattingControlSampleHtmlTable_HtmlCell_Key
{
    color: #1160B7;
```

```
}

.SampleNamespace\FormattingAPI.FormattingControlSampleHtmlTable_HtmlCell_Value
{
    color: #1160B7;
    text-align: center;
}
```

4. Save the TS\_FormattingAPI.css file.

## Build and run your component

To build and run your component, follow these steps:

1. Build your solution by running npm run build.
2. Upon a successful build, you can test your new formatting API component by running npm start.

## Use the Common Data Service web API in a Power Apps component

A common scenario during component development is the requirement to interface with data in your solution's underlying Common Data Service. The Power Apps component framework exposes a Web API feature to achieve this requirement. This example will illustrate how to perform various CRUD operations by using this feature.

## Initialize your component's project

To initialize your component's project, follow these steps:

1. Initialize the project by running the following command:  
pac pcf init --namespace SampleNamespace --name TSWebAPI --template field
2. Run npm install to load dependent libraries into your project.

## Implement your code component's logic

To implement your code component's logic, follow these steps:

1. Open your code component's manifest file (ControlManifest.Input.xml) and replace it with the following logic:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest>
    <control namespace="SampleNamespace" constructor="TSWebAPI" version="1.0.0" display-name-key="TS_WebAPI_Display_Key" description-key="TS_WebAPI_Desc_Display_Key" control-type="standard">
        <property name="stringProperty" display-name-key="stringProperty_Display_Key" description-key="stringProperty_Desc_Key" of-type="SingleLine.Text" usage="bound" required="true" />
        <resources>
            <code path="index.ts" order="1" />
```

```
<css path="css/TS_WebAPI.css" order="2" />
</resources>
<feature-usage>
  <uses-feature name="WebAPI" required="true" />
</feature-usage>
</control>
</manifest>
```

You'll be adding the supporting files that are found in this manifest later.

2. Open the index.ts file.
3. Above the constructor method, insert the following private variables to support rendering of your component:

```
// Reference to the control container HTMLDivElement
// This element contains all elements of our custom control example
private _container: HTMLDivElement;

// Reference to ComponentFramework Context object
private _context: ComponentFramework.Context<IInputs>;

// Flag if control view has been rendered
private _controlViewRendered: Boolean;

// References to button elements that are rendered by example custom control
private _createEntity1Button: HTMLButtonElement;
private _createEntity2Button: HTMLButtonElement;
private _createEntity3Button: HTMLButtonElement;
private _deleteRecordButton: HTMLButtonElement;
private _fetchXmlRefreshButton: HTMLButtonElement;
private _odataRefreshButton: HTMLButtonElement;

// References to div elements that are rendered by the example custom control
private _odataStatusContainerDiv: HTMLDivElement;
private _resultContainerDiv: HTMLDivElement;
```

4. Add the following private static variables above the constructor to indicate which entity/fields you will be interfacing with in this example. If you want to try against different entities or fields, you can do so by changing their respective values.

```
// Name of entity to use for example Web API calls that are performed by this control
private static _entityName: string = "account";

// Required field on _entityName of type 'single line of text'
// Example Web API calls that are performed by the example custom control will set this field for
new record creation examples
private static _requiredAttributeName: string = "name";

// Value that the _requiredAttributeName field will be set to for new created records
private static _requiredAttributeValue: string = "Web API Custom Control (Sample);
```

```
// Name of currency field on _entityName to populate during record create
// Example Web API calls that are performed by the example custom control will set and read this
// field
private static _currencyAttributeName: string = "revenue";

// Friendly name of currency field (only used for control UI - no functional impact)
private static _currencyAttributeNameFriendlyName: string = "annual revenue";
```

5. Place the following logic inside of the *init* method:

```
this._context = context;
this._controlViewRendered = false;
this._container = document.createElement("div");
this._container.classList.add("TSWebAPI_Container");
container.appendChild(this._container);
```

6. Add the following logic to the *updateView* method:

```
if (!this._controlViewRendered) {
    this._controlViewRendered = true;

    // Render Web API Examples
    this.renderCreateExample();
    this.renderDeleteExample();
    this.renderFetchXmlRetrieveMultipleExample();
    this.renderODataRetrieveMultipleExample();

    // Render result div to display output of Web API calls
    this.renderResultsDiv();
```

7. Add the following helper methods to render the HTML elements in your component:

```
/**
 * Helper method to create HTML button that is used for CreateRecord Web API Example
 * @param buttonLabel : Label for button
 * @param buttonId : ID for button
 * @param buttonValue : Value of button (attribute of button)
 * @param onClickHandler : onClick event handler to invoke for the button
 */
private createHTMLButtonElement(buttonLabel: string, buttonId: string, buttonValue: string | null,
onClickHandler: (event?: any) => void): HTMLButtonElement {
    let button: HTMLButtonElement = document.createElement("button");
    button.innerHTML = buttonLabel;

    if (buttonValue) {
        button.setAttribute("buttonvalue", buttonValue);
    }

    button.id = buttonId;

    button.classList.add("SampleControl_WebAPI_ButtonClass");
```

```
button.addEventListener("click", onClickHandler);
return button;
}

/**
 * Helper method to create HTML Div Element
 * @param elementClassName : Class name of div element
 * @param isHeader : True if 'header' div - adds extra class and post-fix to ID for header elements
 * @param innerText : innerText of Div Element
 */
private createHTMLDivElement(elementClassName: string, isHeader: Boolean, innerText?: string):
HTMLDivElement {
    let div: HTMLDivElement = document.createElement("div");

    if (isHeader) {
        div.classList.add("SampleControl_WebAPI_Header");
        elementClassName += "_header";
    }

    if (innerText) {
        div.innerText = innerText.toUpperCase();
    }

    div.classList.add(elementClassName);
    return div;
}

/**
 * Renders a 'result container' div element to inject the status of the example Web API calls
 */
private renderResultsDiv() {
    // Render header label for result container
    let resultDivHeader: HTMLDivElement = this.createHTMLDivElement("result_container", true,
        "Result of last action");
    this._container.appendChild(resultDivHeader);

    // Div elements to populate with the result text
    this._resultContainerDiv = this.createHTMLDivElement("result_container", false, undefined);
    this._container.appendChild(this._resultContainerDiv);

    // Init the result container with a notification that the control was loaded
    this.updateResultContainerText("Web API sample custom control loaded");
}

/**
 * Helper method to inject HTML into result container div
 * @param statusHTML : HTML to inject into result container
 */
private updateResultContainerText(statusHTML: string): void {
    if (this._resultContainerDiv) {
        this._resultContainerDiv.innerHTML = statusHTML;
```

```
        }

    }

    /**
     * Helper method to inject error string into result container div after failed Web API call
     * @param errorResponse : error object from rejected promise
     */
    private updateResultContainerTextWithErrorResponse(errorResponse: any): void {
        if (this._resultContainerDiv) {
            // Retrieve the error message from the errorResponse and inject into the result div
            let errorHTML: string = "Error with Web API call:";
            errorHTML += "<br />"
            errorHTML += errorResponse.message;
            this._resultContainerDiv.innerHTML = errorHTML;
        }
    }

    /**
     * Helper method to generate Label for Create Buttons
     * @param entityNumber : value to set _currencyAttributeNameFriendlyName field to for this
     * button
     */
    private getCreateRecordButtonLabel(entityNumber: string): string {
        return "Create record with " + TSWebAPI._currencyAttributeNameFriendlyName + " of " +
entityNumber;
    }

    /**
     * Helper method to generate ID for Create button
     * @param entityNumber : value to set _currencyAttributeNameFriendlyName field to for this
     * button
     */
    private getCreateButtonId(entityNumber: string): string {
        return "create_button_" + entityNumber;
    }
}
```

8. Add the following onClick event handlers to trigger the various CRUD operations:

```
    /**
     * Event Handler for onClick of create record button
     * @param event : click event
     */
    private createButtonOnClickHandler(event: Event): void {
        // Retrieve the value to set the currency field to from the button's attribute
        let currencyAttributeValue: Number = parseInt(
            (event.srcElement! as Element)!.attributes.getNamedItem("buttonvalue")!.value
        );

        // Generate unique record name by appending timestamp to _requiredAttributeValue
        let recordName: string = TSWebAPI._requiredAttributeValue + "_" + Date.now();
```

```
// Set the values for the attributes we want to set on the new record
// If you want to set additional attributes on the new record, add to data dictionary as key/value
pair
var data: any = {};
data[TSWebAPI._requiredAttributeName] = recordName;
data[TSWebAPI._currencyAttributeName] = currencyAttributeValue;

// store reference to 'this' so it can be used in the callback method
var thisRef = this;

// Invoke the Web API to create the new record
this._context.webAPI.createRecord(TSWebAPI._entityName, data).then
(
    function (response: ComponentFramework.EntityReference) {
        // Callback method for successful creation of new record

        // Get the ID of the new record created
        let id: string = response.id.guid;

        // Generate HTML to inject into the result div to showcase the fields and values of the
        new record that is created
        let resultHtml: string = "Created new " + TSWebAPI._entityName + " record with below
values:";

        resultHtml += "<br />";
        resultHtml += "<br />";
        resultHtml += "id: " + id;
        resultHtml += "<br />";
        resultHtml += "<br />";
        resultHtml += TSWebAPI._requiredAttributeName + ":" + recordName;
        resultHtml += "<br />";
        resultHtml += "<br />";
        resultHtml += TSWebAPI._currencyAttributeName + ":" + currencyAttributeValue;

        thisRef.updateResultContainerText(resultHtml);
    },
    function (errorResponse: any) {
        // Error handling code here - record failed to be created
        thisRef.updateResultContainerTextWithErrorResponse(errorResponse);
    }
);

/**
 * Event Handler for onClick of delete record button
 * @param event : click event
 */
private deleteButtonOnClickHandler(): void {
    // Invoke a lookup dialog to allow the user to select an existing record of type _entityName to
    delete
    var lookUpOptions: any =
    {
```

```
        entityTypes: [TSWebAPI._entityName]
    };

    // store reference to 'this' so it can be used in the callback method
    var thisRef = this;

    var lookUpPromise: any = this._context.utils.lookupObjects(lookUpOptions);

    lookUpPromise.then
    (
        // Callback method - invoked after user has selected an item from the lookup dialog
        // Data parameter is the item selected in the lookup dialog
        (data: ComponentFramework.EntityReference[]) => {
            if (data && data[0]) {
                // Get the ID and entityType of the record that was selected by the lookup
                let id: string = data[0].id.guid;
                let entityType: string = data[0].etnl;

                // Invoke the deleteRecord method of the WebAPI to delete the selected record
                this._context.webAPI.deleteRecord(entityType, id).then
                (
                    function (response: ComponentFramework.EntityReference) {
                        // Record was deleted successfully
                        let responseld: string = response.id.guid;
                        let responseEntityType: string = response.etnl;

                        // Generate HTML to inject into the result div to showcase the deleted record
                        thisRef.updateResultContainerText("Deleted " + responseEntityType + " record
with ID: " + responseld);
                    },
                    function (errorResponse: any) {
                        // Error handling code here
                        thisRef.updateResultContainerTextWithErrorResponse(errorResponse);
                    }
                );
            },
            (error: any) => {
                // Error handling code here
                thisRef.updateResultContainerTextWithErrorResponse(error);
            }
        );
    }

    /**
     * Event Handler for onClick of calculate average value button
     * @param event : click event
     */
    private calculateAverageButtonOnClickHandler(): void {
        // Build FetchXML to retrieve the average value of _currencyAttributeName field for all _entityName records
```

```
// Add a filter to only aggregate on records that have _currencyAttributeName not set to null
let fetchXML: string = "<fetch distinct='false' mapping='logical' aggregate='true'>";
fetchXML += "<entity name='" + TSWebAPI._entityName + "'>";
fetchXML += "<attribute name='" + TSWebAPI._currencyAttributeName + "' aggregate='avg' alias='average_val' />";
fetchXML += "<filter>";
fetchXML += "<condition attribute='" + TSWebAPI._currencyAttributeName + "' operator='not-null' />";
fetchXML += "</filter>";
fetchXML += "</entity>";
fetchXML += "</fetch>";

// store reference to 'this' so it can be used in the callback method
var thisRef = this;

// Invoke the Web API RetrieveMultipleRecords method to calculate the aggregate value
this._context.webAPI.retrieveMultipleRecords(TSWebAPI._entityName, "?fetchXml=" + fetchXML).
then(
  (
    function (response: ComponentFramework.WebApi.RetrieveMultipleResponse) {
      // Retrieve multiple completed successfully -- retrieve the averageValue
      let averageVal: Number = response.entities[0].average_val;

      // Generate HTML to inject into the result div to showcase the result of the RetrieveMultiple Web API call
      let resultHTML: string = "Average value of " + TSWebAPI._currencyAttributeNameFriendlyName + " attribute for all " + TSWebAPI._entityName + " records: " + averageVal;
      thisRef.updateResultContainerText(resultHTML);
    },
    function (errorResponse: any) {
      // Error handling code here
      thisRef.updateResultContainerTextWithErrorResponse(errorResponse);
    }
  );
}

/***
 * Event Handler for onClick of calculate record count button
 * @param event : click event
 */
private refreshRecordCountButtonOnClickHandler(): void {
  // Generate OData query string to retrieve the _currencyAttributeName field for all _entityName records
  // Add a filter to only retrieve records with _requiredAttributeName field which contains _requiredAttributeValue
  let queryString: string = "?$select=" + TSWebAPI._currencyAttributeName + "&$filter=contains(" +
  + TSWebAPI._requiredAttributeName +
  ",'" + TSWebAPI._requiredAttributeValue + "')";

  // store reference to 'this' so it can be used in the callback method
  var thisRef = this;
```

```
// Invoke the Web API Retrieve Multiple call
this._context.webAPI.retrieveMultipleRecords(TSWebAPI._entityName, queryString).then
(
    function (response: any) {
        // Retrieve Multiple Web API call completed successfully
        let count1: number = 0;
        let count2: number = 0;
        let count3: number = 0;

        // Loop through each returned record
        for (let entity of response.entities) {
            // Retrieve the value of _currencyAttributeName field
            let value: Number = entity[TSWebAPI._currencyAttributeName];

            // Check the value of _currencyAttributeName field and increment the correct counter
            if (value == 100) {
                count1++;
            }
            else if (value == 200) {
                count2++;
            }
            else if (value == 300) {
                count3++;
            }
        }

        // Generate HTML to inject into the fetch xml status div to showcase the results of the
        // OData retrieve example
        let innerHtml: string = "Use above buttons to create or delete a record to see count
update";
        innerHtml += "<br />";
        innerHtml += "<br />";
        innerHtml += "Count of " + TSWebAPI._entityName + " records with " + TSWebAPI._cur-
rencyAttributeName + " of 100: " + count1;
        innerHtml += "<br />";
        innerHtml += "Count of " + TSWebAPI._entityName + " records with " + TSWebAPI._cur-
rencyAttributeName + " of 200: " + count2;
        innerHtml += "<br />";
        innerHtml += "Count of " + TSWebAPI._entityName + " records with " + TSWebAPI._cur-
rencyAttributeName + " of 300: " + count3;

        // Inject the HTML into the fetch xml status div
        if (thisRef._odataStatusContainerDiv) {
            thisRef._odataStatusContainerDiv.innerHTML = innerHtml;
        }

        // Inject a success message into the result div
        thisRef.updateResultContainerText("Record count refreshed");
    },
    function (errorResponse: any) {
```

```
        // Error handling code here
        thisRef.updateResultContainerTextWithErrorResponse(errorResponse);
    }
}
}
```

9. Add the following helper methods to render the results of your CRUD operations within your component:

```
/***
 * Renders example use of CreateRecord Web API
 */
private renderCreateExample() {
    // Create header label for Web API sample
    let headerDiv: HTMLDivElement = this.createHTMLDivElement("create_container", true, "Click to
create "
        + TSWebAPI._entityName + " record");
    this._container.appendChild(headerDiv);

    // Create button 1 to create a record with the revenue field set to 100
    let value1: string = "100";
    this._createEntity1Button = this.createHTMLButtonElement(
        this.getCreateRecordButtonLabel(value1),
        this.getCreateButtonId(value1),
        value1,
        this.createButtonOnClickHandler.bind(this));

    // Create button 2 to create a record with the revenue field set to 200
    let value2: string = "200";
    this._createEntity2Button = this.createHTMLButtonElement(
        this.getCreateRecordButtonLabel(value2),
        this.getCreateButtonId(value2),
        value2,
        this.createButtonOnClickHandler.bind(this));

    // Create button 3 to create a record with the revenue field set to 300
    let value3: string = "300";
    this._createEntity3Button = this.createHTMLButtonElement(
        this.getCreateRecordButtonLabel(value3),
        this.getCreateButtonId(value3),
        value3,
        this.createButtonOnClickHandler.bind(this));

    // Append all button HTML elements to custom control container div
    this._container.appendChild(this._createEntity1Button);
    this._container.appendChild(this._createEntity2Button);
    this._container.appendChild(this._createEntity3Button);
}

/***
 * Renders example use of DeleteRecord Web API

```

```
/*
private renderDeleteExample(): void {
    // Create header label for Web API sample
    let headerDiv: HTMLDivElement = this.createHTMLDivElement("delete_container", true, "Click to
delete " + TSWebAPI._entityName + " record");

    // Render button to invoke DeleteRecord Web API call
    this._deleteRecordButton = this.createHTMLButtonElement(
        "Select record to delete",
        "delete_button",
        null,
        this.deleteButtonOnClickHandler.bind(this));

    // Append elements to custom control container div
    this._container.appendChild(headerDiv);
    this._container.appendChild(this._deleteRecordButton);
}

/**
 * Renders example use of RetrieveMultiple Web API with OData
 */
private renderODataRetrieveMultipleExample(): void {
    let containerClassName: string = "odata_status_container";

    // Create header label for Web API sample
    let statusDivHeader: HTMLDivElement = this.createHTMLDivElement(containerClassName, true,
    "Click to refresh record count");
    this._odataStatusContainerDiv = this.createHTMLDivElement(containerClassName, false, undefined);

    // Create button to invoke OData RetrieveMultiple Example
    this._fetchXmlRefreshButton = this.createHTMLButtonElement(
        "Refresh record count",
        "odata_refresh",
        null,
        this.refreshRecordCountButtonOnClickHandler.bind(this));

    // Append HTML elements to custom control container div
    this._container.appendChild(statusDivHeader);
    this._container.appendChild(this._odataStatusContainerDiv);
    this._container.appendChild(this._fetchXmlRefreshButton);
}

/**
 * Renders example use of RetrieveMultiple Web API with Fetch XML
 */
private renderFetchXmlRetrieveMultipleExample(): void {
    let containerName: string = "fetchxml_status_container";

    // Create header label for Web API sample
    let statusDivHeader: HTMLDivElement = this.createHTMLDivElement(containerName, true,
```

```
"Click to calculate average value of " + TSWebAPI._currencyAttributeNameFriendlyName);
let statusDiv: HTMLDivElement = this.createHTMLDivElement(containerName, false, undefined);

// Create button to invoke Fetch XML RetrieveMultiple Web API example
this._oDataRefreshButton = this.createHTMLButtonElement(
    "Calculate average value of " + TSWebAPI._currencyAttributeNameFriendlyName,
    "odata_refresh",
    null,
    this.calculateAverageButtonOnClickHandler.bind(this));

// Append HTML Elements to custom control container div
this._container.appendChild(statusDivHeader);
this._container.appendChild(statusDiv);
this._container.appendChild(this._oDataRefreshButton);
}
```

After you've made the updates, your completed class should look similar to the following example:

This file is part of the Microsoft Power Apps code samples.

Copyright (C) Microsoft Corporation. All rights reserved.

This source code is intended only as a supplement to Microsoft Development Tools and/or online documentation. See these other materials for detailed information regarding Microsoft code samples.

THIS CODE AND INFORMATION ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND,  
EITHER  
EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF  
MERCHANTABILITY AND/OR FITNESS FOR A PARTICULAR PURPOSE.  
\*/

```
import {IInputs, IOutputs} from "./generated/ManifestTypes";

export class FormattingAPI implements ComponentFramework.StandardControl<IInputs, IOutputs>
{

    // PCF framework delegate that will be assigned to this object that would be called whenever any
    update happens.
    private _notifyOutputChanged: () => void;

    // Reference to the div element that holds together all the HTML elements that we are creating
    as part of this control
    private divElement: HTMLDivElement;

    // Reference to HTMLTableElement that is rendered by the control
    private _tableElement: HTMLTableElement;

    // Reference to the control container HTMLDivElement
    // This element contains all elements of our custom control example
    private _container: HTMLDivElement;

    // Reference to ComponentFramework Context object
```

```
private _context: ComponentFramework.Context<IInputs>;  
  
// Flag if control view has been rendered  
private _controlViewRendered: Boolean;  
  
/**  
 * Used to initialize the control instance. Controls can kick off remote server calls and other  
 initialization actions here.  
 * Dataset values are not initialized here, use updateView.  
 * @param context The entire property bag that is available to control through the Context  
 Object; It contains values as set up by the customizer that are mapped to property names that are  
 defined in the manifest and to utility functions.  
 * @param notifyOutputChanged A callback method to alert the framework that the control has  
 new outputs ready to be retrieved asynchronously.  
 * @param state A piece of data that persists in one session for a single user. Can be set at any  
 point in a control's life cycle by calling 'setControlState' in the Mode interface.  
 * @param container If a control is marked control-type='starndard', it will receive an empty div  
 element within which it can render its content.  
 */  
public init(context: ComponentFramework.Context<IInputs>, notifyOutputChanged: () => void,  
state: ComponentFramework.Dictionary, container: HTMLDivElement)  
{  
    this._notifyOutputChanged = notifyOutputChanged;  
    this._controlViewRendered = false;  
    this._context = context;  
  
    this._container = document.createElement("div");  
    this._container.classList.add("TSFormatting_Container");  
    container.appendChild(this._container);  
}  
  
/**  
 * Called when any value in the property bag has changed. This includes field values, datasets,  
 global values such as container height and width, offline status, control metadata values such as label,  
 visible, and so on.  
 * @param context The entire property bag that is available to control through the Context  
 Object; It contains values as set up by the customizer that are mapped to names that are defined in  
 the manifest and to utility functions.  
 */  
public updateView(context: ComponentFramework.Context<IInputs>): void  
{  
    if (!this._controlViewRendered)  
    {  
        // Render and add HTMLTable to the custom control container element  
        let tableElement: HTMLTableElement = this.createHTMLTableElement();  
        this._container.appendChild(tableElement);  
  
        this._controlViewRendered = true;  
    }  
}
```

```
/*
 * It is called by the framework prior to a control receiving new data.
 * @returns an object based on nomenclature that is defined in manifest, except object[s] for
property that are marked as "bound" or "output"
*/
public getOutputs(): IOOutputs
{
    // no-op: method not used by this example custom control
    return { };
}

/**
 * Called when the control is to be removed from the DOM tree. Controls should use this call for
cleanup,
 * such as canceling any pending remote calls, removing listeners, and so on.
*/
public destroy()
{

}

/**
 * Helper method to create an HTML Table Row Element
 * @param key : string value to show in left column cell
 * @param value : string value to show in right column cell
 * @param isHeaderRow : true if method should generate a header row
*/
private createHTMLTableRowElement(key: string, value: string, isHeaderRow: Boolean): HTMLTableElement
{
    let keyCell: HTMLTableCellElement = this.createHTMLTableCellElement(key, "FormattingCon-
trolSampleHtmlTable_HtmlCell_Key", isHeaderRow);
    let valueCell: HTMLTableCellElement = this.createHTMLTableCellElement(value, "Formatting-
ControlSampleHtmlTable_HtmlCell_Value", isHeaderRow);

    let rowElement: HTMLTableElement = document.createElement("tr");
    rowElement.setAttribute("class", "FormattingControlSampleHtmlTable_HtmlRow");
    rowElement.appendChild(keyCell);
    rowElement.appendChild(valueCell);

    return rowElement;
}

/**
 * Helper method to create an HTML Table Cell Element
 * @param cellValue : string value to inject in the cell
 * @param className : class name for the cell
 * @param isHeaderRow : true if method should generate a header row cell
*/
private createHTMLTableCellElement(cellValue: string, className: string, isHeaderRow: Boolean): HTMLTableElement
```

```
{  
    let cellElement: HTMLTableCellElement;  
  
    if (isHeaderRow)  
    {  
        cellElement = document.createElement("th");  
        cellElement.setAttribute("class", "FormattingControlSampleHtmlTable_HtmlHeaderCell " +  
className);  
        let textElement: Text = document.createTextNode(cellValue);  
        cellElement.appendChild(textElement);  
    }  
    else  
    {  
        cellElement = document.createElement("td");  
        cellElement.setAttribute("class", "FormattingControlSampleHtmlTable_HtmlCell " +  
className);  
        let textElement: Text = document.createTextNode(cellValue);  
        cellElement.appendChild(textElement);  
    }  
    return cellElement;  
}  
  
/**  
 * Creates an HTML Table that showcases examples of basic methods that are available to the  
custom control  
 * The left column of the table shows the method name or property that is being used  
 * The right column of the table shows the result of that method name or property  
*/  
private createHTMLTableElement(): HTMLTableElement  
{  
    // Create HTML Table Element  
    let tableElement: HTMLTableElement = document.createElement("table");  
    tableElement.setAttribute("class", "FormattingControlSampleHtmlTable_HtmlTable");  
  
    // Create header row for table  
    let key: string = "Example Method";  
    let value: string = "Result";  
    tableElement.appendChild(this.createHTMLTableRowElement(key, value, true));  
  
    // Example use of formatCurrency() method  
    // Change the default currency and the precision or pass in the precision and currency as  
additional parameters.  
    key = "formatCurrency()";  
    value = this._context.formatting.formatCurrency(10250030);  
    tableElement.appendChild(this.createHTMLTableRowElement(key, value, false));  
  
    // Example use of formatDecimal() method  
    // Change the settings from user settings to see the output change its format accordingly  
    key = "formatDecimal()";  
    value = this._context.formatting.formatDecimal(123456.2782);  
    tableElement.appendChild(this.createHTMLTableRowElement(key, value, false));  
}
```

```
// Example use of formatInteger() method
// Change the settings from user settings to see the output change its format accordingly.
key = "formatInteger";
value = this._context.formatting.formatInteger(12345);
tableElement.appendChild(this.createHTMLTableRowElement(key, value, false));

// Example use of formatLanguage() method
// Install additional languages and pass in the corresponding language code to see its string
value
key = "formatLanguage";
value = this._context.formatting.formatLanguage(1033);
tableElement.appendChild(this.createHTMLTableRowElement(key, value, false));

// Example of formatDateYearMonth() method
// Pass a JavaScript Data object set to the current time into formatDateYearMonth method to
format the data
// and get the return in Year, Month format
key = "formatDateYearMonth";
value = this._context.formatting.formatDateYearMonth(new Date());
tableElement.appendChild(this.createHTMLTableRowElement(key, value, false));

// Example of getWeekOfYear() method
// Pass a JavaScript Data object set to the current time into getWeekOfYear method to get the
value for week of the year
key = "getWeekOfYear";
value = this._context.formatting.getWeekOfYear(new Date()).toString();
tableElement.appendChild(this.createHTMLTableRowElement(key, value, false));

return tableElement;
}

}
```

10. If you want to view a completed version of this file, you can find one in the solution files of this module.

## Add styling to your code component

To add styling to your code component, follow these steps:

1. Create a new CSS subfolder under the TSWebAPI folder.
2. Create a new TS\_WebAPI.css file inside the CSS subfolder.
3. Add the following style content to the TS\_WebAPI.css file:

```
/*
This file is part of the Microsoft Power Apps code samples.
Copyright (C) Microsoft Corporation. All rights reserved.
This source code is intended only as a supplement to Microsoft Development Tools and/or
online documentation. See these other materials for detailed information regarding
Microsoft code samples.
```

THIS CODE AND INFORMATION ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND,  
EITHER  
EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF  
MERCHANTABILITY AND/OR FITNESS FOR A PARTICULAR PURPOSE.

```
/*
.SampleNamespace\TSWebAPI
{
    font-family: 'SegoeUI-Semibold', 'Segoe UI Semibold', 'Segoe UI Regular', 'Segoe UI';
    color: #1160B7;
}

.SampleNamespace\TSWebAPI .TSWebAPI_Container
{
    overflow-x: auto;
}

.SampleNamespace\TSWebAPI .SampleControl_WebAPI_Header
{
    color: rgb(51, 51, 51);
    font-size: 1rem;
    padding-top: 20px;
}

.SampleNamespace\TSWebAPI .result_container
{
    padding-bottom: 20px;
}

.SampleNamespace\TSWebAPI .SampleControl_WebAPI_ButtonClass
{
    text-decoration: none;
    display: inline-block;
    font-size: 14px;
    cursor: pointer;
    color: #1160B7;
    background-color: #FFFFFF;
    border: 1px solid black;
    padding: 5px;
    text-align: center;
    min-width: 300px;
    margin-top: 10px;
    margin-bottom: 5px;
    display: block;
}
```

4. Save the TS\_WebAPI.css file:

## Build and run your component

To build and run your component, follow these steps:

1. Build your solution by running `npm run build`.
2. Upon successful build, you can test your new formatting API component by running `npm start`. To test the web API functionality, you will need to publish and host the component in a live Power Apps environment. For more details on how to publish Power Apps components, see “Package a code component” in the *Write a custom Power Apps component* module.

## Write a pop-up Power Apps component

Occasionally, it is required that you display a pop-up window to a user of your application. The Power Apps component framework exposes a pop-up API that allows you to achieve this requirement. The following example will show how to build a pop-up window that displays a loader graphic. In long-running operations, this method can help you achieve a satisfying user experience where the underlying UI is blocked from performing operations.

## Initialize your component's project

To initialize your component's project, follow these steps:

1. Initialize the project by running the following command:

```
pac pcf init --namespace SampleNamespace --name PopupComponent --template field
```

2. Run `npm install` to load dependent libraries into your project.

## Implement your code component's logic

To implement your code component's logic, follow these steps:

1. Open your code component's manifest file (`ControlManifest.Input.xml`) and replace it with the following logic:

```
<?xml version="1.0" encoding="utf-8" ?>
<manifest>
  <control namespace="SampleNamespace" constructor="PopupComponent" version="0.0.1"
    display-name-key="PopupComponent_Display_Key" description-key="PopupComponent_Desc_Key"
    control-type="standard">
    <!-- property node identifies a specific, configurable piece of data that the control expects from
    CDS -->
    <property name="sampleProperty" display-name-key="Property_Display_Key" descrip-
      tion-key="Property_Desc_Key" of-type="SingleLine.Text" usage="bound" required="true" />
    <resources>
      <code path="index.ts" order="1"/>
      <css path="css/loader.css" order="1" />
    </resources>
  </control>
</manifest>
```

You'll be adding the supporting files that are found in this manifest later.

2. Open the index.ts file.
3. Above the `constructor` method, insert the following interface method so that you can expose some additional methods that are provided by the pop-up API (`popupStyle` and `shadowStyle`):

```
interface Popup extends ComponentFramework.FactoryApi.Popup.Popup {  
    popupStyle: object;  
    shadowStyle: object;  
}
```

4. Add the following logic to your component's `init` method:

```
this._container = document.createElement('div');  
//===== content of our popup ======  
let popUpContent = document.createElement('div');  
popUpContent.setAttribute("class", "loader");  
  
//===== our Popup object ======  
let popUpOptions: Popup = {  
    closeOnOutsideClick: true,  
    content: popUpContent,  
    name: 'loaderPopup', // unique popup name  
    type: 1, // Root popup  
    popupStyle: {  
        "width": "100%",  
        "height": "100%",  
        "overflow": "hidden",  
        "backgroundColor": "transparent",  
        "display": "flex",  
        "flexDirection": "column",  
        "position": "absolute",  
        "margin-top": 28 + "px"  
    },  
    shadowStyle:{  
        position: "absolute",  
        width: "100%",  
        height: "100%"  
    }  
};  
  
this._popUpService = context.factory.getPopupService();  
  
this._popUpService.createPopup(popUpOptions);  
  
container.appendChild(this._container);  
this._popUpService.openPopup('loaderPopup');
```

## Add styling to your code component

To add styling to your code component, follow these steps:

1. Create a new CSS subfolder under the PopupComponent folder.
2. Create a new loader.css file inside the CSS subfolder.
3. Add the following style content to the loader.css file:

```
.loader {  
    border: 16px solid #f3f3f3; /* Light grey */  
    border-top: 16px solid #3498db; /* Blue */  
    border-radius: 50%;  
    position: fixed;  
    width: 120px;  
    height: 120px;  
    top: 40%;  
    left: 50%;  
    animation: spin 2s linear infinite;  
}  
  
@keyframes spin {  
    0% { transform: rotate(0deg); }  
    100% { transform: rotate(360deg); }  
}
```

4. Save the loader.css file.

## Build and run your component

To build and run your component, follow these steps:

1. Build your solution by running npm run build.
2. Upon a successful build, you can test your new formatting API component by running npm start. This loader component is bound to a text field. To use the component in a model-driven app, it might be beneficial for you to mark this field as hidden if you want to use it in a form. For more details on how to publish Power Apps components, see “Package a code component” in the *Write a custom Power Apps component* module.

## summary

This module reviewed various techniques for doing more advanced operations within a Power Apps component. You learned how to use the React framework in conjunction with Microsoft’s UI Fabric framework. Additionally, you delved into using the formatting API, the Web API, and the pop-up API. These utilities are only a few of the many that are available to you within the Power Apps component framework. You can refer to Microsoft’s **developer documentation**<sup>24</sup> if you want to explore this topic further.

<sup>24</sup> <https://docs.microsoft.com/powerapps/developer/component-framework/overview/?azure-portal=true>



## Module 9 Extend Power Apps Portals

### Introduction to Power Apps portals

#### Introduction

Power Apps portals provide a great way to allow internal and external audiences to view and interact with data from Common Data Service or Dynamics 365. Through the Power Apps interface, you can build an anonymous or authenticated website that provides them with a branded, personalized, self-service experience.

Power Apps portals come with a variety of preconfigured portal solutions that target diverse audiences. Starter portals have many features that add value to the Microsoft Dynamics 365 service apps.

In this module, you will:

- Learn what are Power Apps portals and the business value they provide
- Learn the core components for the content and data access
- Learn how to create portals using a template
- See some of the differences in portal behavior based on security

#### Power Apps portals and what they can do for you

Power Apps portals give internal and external users secure access to your data either anonymously or through commercial authentication providers like LinkedIn, Microsoft, Facebook, and Google, or enterprise providers such as Azure AD B2C and Okta. Portals also allow you to set authentication requirements, customize data for each user, and allow users to submit their information privately with straightforward admin controls.

## Modern audience

Portal capabilities empower online consumers who prefer to find answers on their own through self-service and community options. By using Power Apps portals, you can provide them with a branded, personalized, self-service experience. Portals help you provide an organized, searchable knowledge base to deliver consistent, up-to-date answers and community experience for peer-to-peer support and direct interaction with your subject matter experts. Additionally, portals provide simple navigation with seamless transitions between self and assisted support.

Out-of-the-box mobile optimizations for portal and knowledge articles ensure that customers can get the help that they need, any time and from any device.

- The majority of global consumers now expect brands and organizations to offer a self-service portal.
- Nearly one-third of consumers keep in touch with brands they've done business with to make sure that they are getting the most out of their purchase.
- Half of the consumers are using multiple channels to stay connected, including web, mobile, telephone, social, and self-service channels. Often, consumers use many channels for a single support experience.

Providing a web portal interface takes advantage of these user trends and brings any type of engagement, including partner, group, and employee scenarios, that directly accesses Common Data Service to create a modern connected experience for external users and internal business operations.

## Modern business

Consider a scenario where your business is already using Power Apps. The following are additional capabilities that Power Apps portals can deliver:

- **Provide self-service support** - When your business is growing, rather than having to employ extra staff in call centers, you could use Power Apps portals to add self-service capabilities to your website so that your customers can search knowledge articles, engage with other customers, find answers, and create support cases when needed (that go directly into Dynamics 365 Customer Service), all without a single interaction from your resources.
- **Build a sales pipeline** - When a lead fills out a **Contact Us** form on your company website, this information is recorded in Dynamics 365 Sales where the record can become part of your sales pipeline automatically.
- **Empower employees** - When an employee needs a new computer, they can fill out an online form, where the information will be recorded in Common Data Service so the helpdesk staff can immediately access and process this information.
- **Engage mobile workforce** - Empower agents on any device, wherever they work. Field technicians can process and complete work orders in the field, instantly updating Dynamics 365 Field Service.

## Power Apps portals capabilities

Power Apps portals are built on top of Common Data Service. This architecture comes with a major benefit. All the differentiating features of model-driven Power Apps are the features of Power Apps portals as well, including:

- Centralized management
- Common Data Model
- Roles and permissions
- Forms and views
- Business rules
- Declarative workflows and actions
- Plug-in architecture
- Integration with other services
- Common Data Service extensibility
- Audit

Power Apps portals deliver a complete content management system out of the box, with all content stored in Common Data Service. As a result, content can be edited through the Portals Studio and also directly by using the Portal Management app. Additionally, the robust Common Data Service security model can help secure the content.

## Get started with portals

Businesses that have Dynamics 365 or a custom Common Data Service solution already in place can quickly build portals that are more secure and build their entire website, all without requiring developers.

A Power Apps portal is not automatically provisioned when a new Common Data Service environment is created. You will need to provision a Power Apps portal and determine the name, default URL, language, and template.

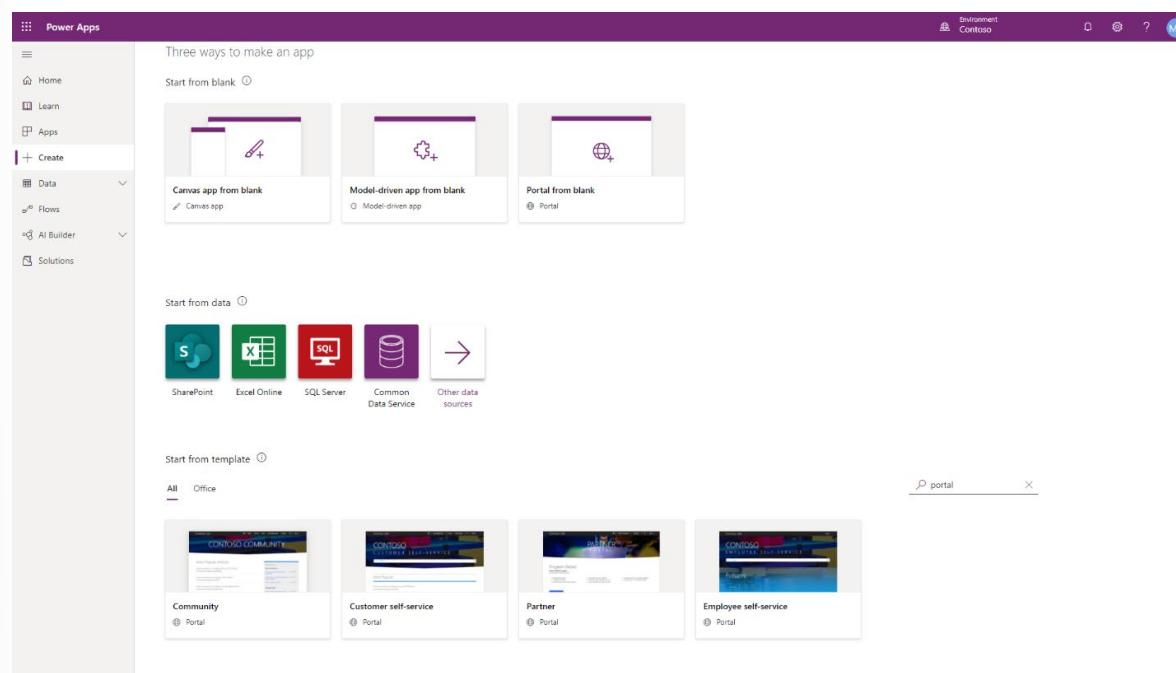
[!IMPORTANT]

To provision a portal, you must be assigned to the System Administrator role of the Common Data Service environment that is selected for the portal.

## Portal templates

When provisioning a Power Apps portal, the most important choices to consider are the audience, workload, and choosing a specific portal template that would best align with the business requirements.

Several portal templates are available that can be provisioned. These templates will accelerate the configuration of portals based on the intended audience and workload.



If you are building a custom business application by using Common Data Service without Dynamics 365 apps enabled, your only choice is the **Portal from blank** option.

If you are using Microsoft Dynamics 365 apps such as Dynamics 365 Sales or Dynamics 365 Service, you have a choice of five additional portal templates:

- Community portal
- Customer self-service portal
- Employee self-service portal
- Partner portal
- Customer portal (Dynamics 365 Supply Chain Management)

[!NOTE]

Specific features and components a specific starter portal can be added to another starter portal, as required.

Make sure that you define the type of audience who will visit the new portal. The audience will determine which options of portal you will be given.

PORTAL TEMPLATES		
Portal template	Audience	Workload
Community	Partner, Customer	Choose this option to provision a portal that is focused on an online community. This portal will contain features such as forums, ideas, blogs, and case management.

<b>PORTAL TEMPLATES</b>		
Customer self-service	Partner, Customer	This option provides the ability for portal users to search knowledge articles, submit cases, and participate in discussion forums to resolve issues.
Employee self-service	Employee	This portal allows employees to access a centralized knowledge article and to also submit cases.
Partner	Partner	Choose this option to build a portal where external partners can manage and collaborate on accounts and opportunities. Add-ons are available for Dynamics 365 Field Service or Dynamics 365 Project Service.
Customer portal	Enterprise B2B	The Dynamics 365 Supply Chain Management Customer portal is a template that provides portal access to Dynamics 365 Supply Chain Management data by using dual-write Common Data Service entities.
Portal from blank	Other	The <b>Portal from blank</b> option is meant for unique line-of-business scenarios where the other templates are not a good fit. The portal can be configured to address a variety of requirements. If <b>Portal from blank</b> is provisioned within a Common Data Service environment with Dynamics 365 apps enabled, specific features from the other portals can be incorporated into the portal later.

## Provision a portal

Only one Power Apps portals can be provisioned for each Common Data Service environment.

The high-level steps to provision a starter portal are:

1. Go to <https://make.powerapps.com><sup>1</sup>.
2. Select a target environment by using the environment selector in the upper-right corner.
3. On the left menu, select + **Create**.

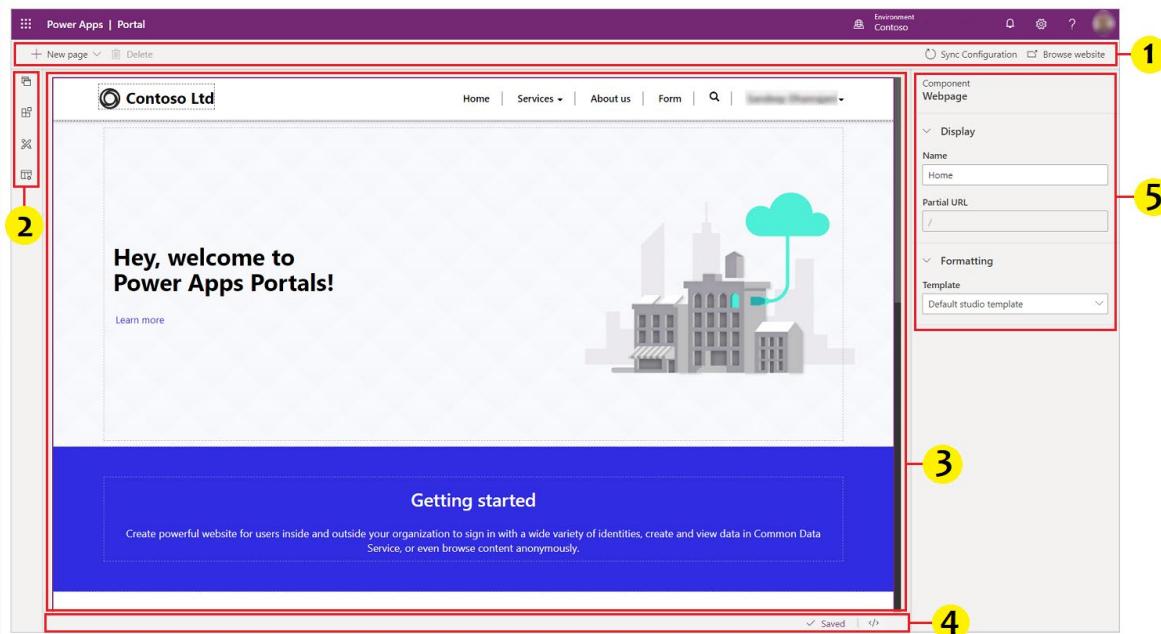
---

<sup>1</sup> <https://make.powerapps.com/>

4. Select **Portal from blank**. If you have Dynamics 365 apps deployed in your Common Data Service environment, additional portal templates, such as Customer self-service, will be available.
5. Provide a name for the portal.
6. Provide a unique address (URL) for the portal.
7. Select the language.
8. Select **Create** to start the portal provisioning process. After portal provisioning has completed, the portal will appear in the list as an app of type Portal.
9. Select the ellipsis (...) next to the portal app name and then select **Browse** to open the portal website.

## Power Apps portals Studio

You can use Power Apps portals Studio to create and customize your website. It contains various options to add and configure webpages, components, forms, and lists. The anatomy of Power Apps portals Studio is as follows:



Power Apps portals Studio components:

1. **Command bar** - Allows you to:
  - Create a webpage.
  - Delete a component.
  - Sync Configuration - synchronizes the latest portal configuration changes in Common Data Service database with your current Studio session. For example, use Sync Configuration to reflect the changes in Studio when using the Portal Management app to change the configuration of pages, forms or any other objects.
  - Browse website - clears the portal cache and opens the current portal page.
2. **Toolbelt** - Allows you to:
  - View and manage webpages

- Add components
  - Edit templates
3. \*\*Canvas\*\* - Contains components that build a webpage.
  4. \*\*Footer\*\* - Displays autosave status and allows you to open-source code editor.
  5. \*\*Properties pane\*\* - Displays properties of webpage and selected components and lets you edit them as required.

## Webpages

Most of a portal's content is represented by webpages. A webpage is a document that is identified by a unique URL in a website. Through parent and child relationships to other webpages, webpages form the hierarchy of a website, that is, its site map.

Webpages can be added and edited by using the Portal Studio, the portal front-side editor, or directly in Common Data Service by using the Portal Management app.

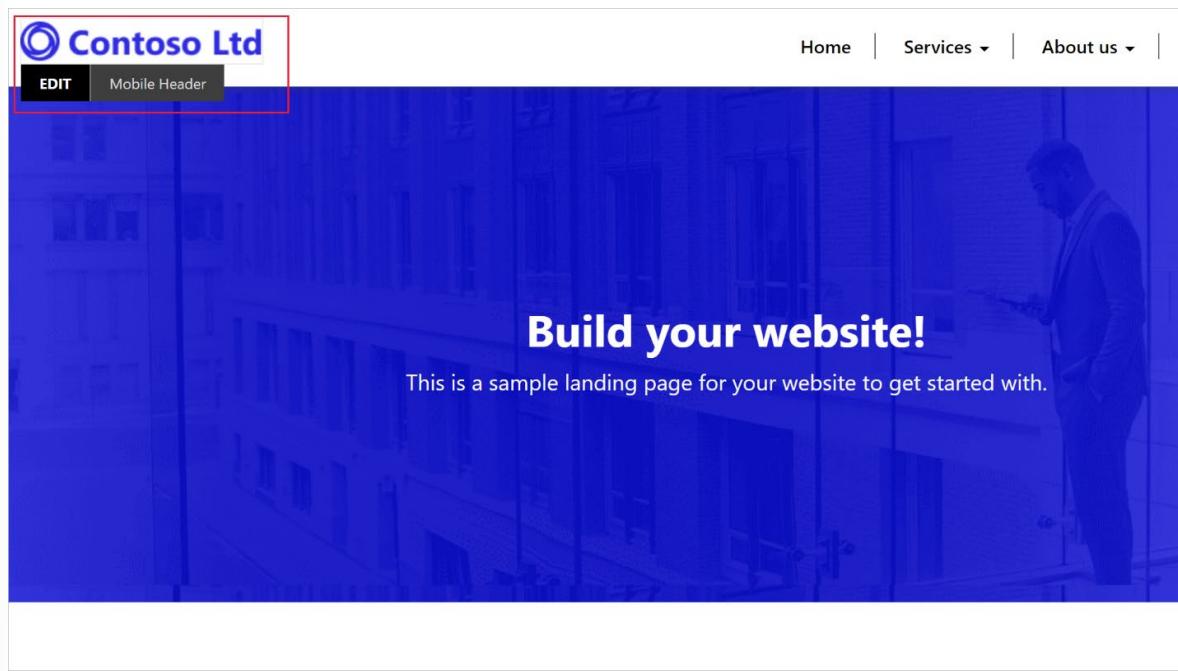
## Page templates

A webpage record does not define how the page looks when it is rendered on the portal. Instead, it is linked to the **Page template** record that defines the layout and the behavior. Think of the webpage as the exact URL and the Page template as the blueprint for displaying the content.

## Content snippets

Content snippets are reusable fragments of editable content that can be placed within a web template. Using snippets allows for targeted editing of parts of a page without affecting the overall content.

Content snippets can include plain text, HTML layout, or template processing instructions, which helps enable dynamic content. In the example below, **Mobile Header** is a content snippet that can be updated with your company's logo to quickly and easily tailor the portal to your needs.



Snippets can be edited by using Portal Studio and Common Data Service records by using the Portal Management app. Here is where you would replace the image source in the **Value** field with your company's logo.

A screenshot of the Power Apps Portal Management app. The left sidebar shows navigation options like Home, Recent, Pinned, Website, Websites, Page Templates, Redirects, Site Markers, Site Settings, Website Bindings, and Settings. Under Content, it lists Entity Forms, Entity Lists, Shortcuts, Web Files, Web Forms, Web Link Sets, Portal Languages, Web Pages, and Web Templates. The main area shows a 'Mobile Header' content snippet. The 'General' tab is selected, displaying fields for Name (Mobile Header), Website (Starter Portal), Display Name (Mobile Header), Type (HTML), and Content Snippet Language (English). Below this, the 'Value (HTML)' section shows the HTML code for the logo: 

```
1 <a href="/"></a>
```

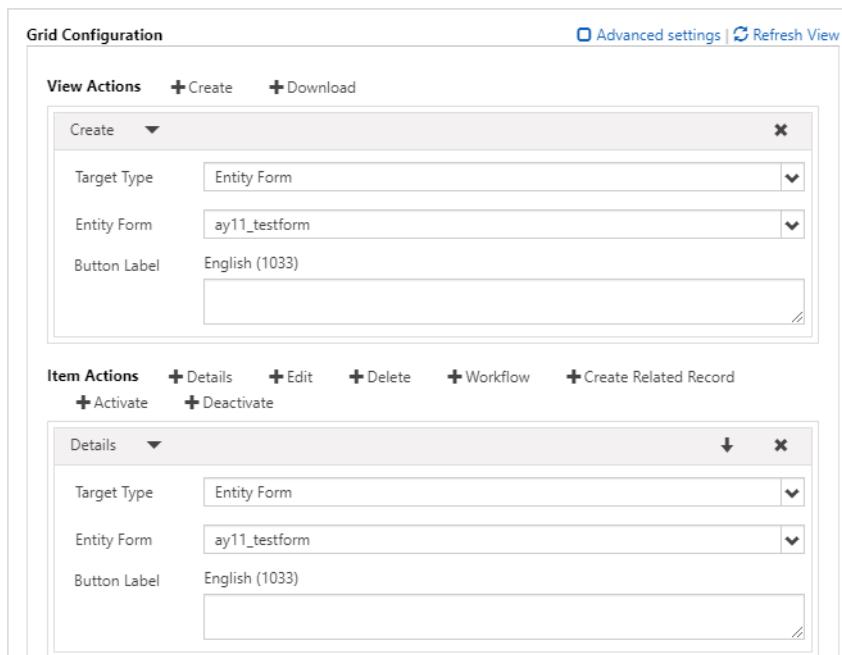
## Entity lists and entity forms

The strength of Power Apps portals is the ability to interact with information and data that is stored in Common Data Service. Entity lists and entity forms are used

in Power Apps portals to define what data should render on the portal from Common Data Service, such as a list of records from an entity or a form to capture and display data for a specific record.

A webpage record can be linked to an **entity list** or an **entity form**. The linked list or form will be used by the template to render the page layout with data from Common Data Service, such as a list of all *Active Contacts* to form the above *Member Directory* list. In the **Properties** pane on the right of the above example, you see that this entity list was created to display the *Active Contacts* view from the *contacts* entity.

Entity lists can include functionality like filtering and sorting and can also have actions associated with them to enable Create/Edit/Read abilities and to trigger workflows. With this, App Makers can determine what will happen when a user opens a record from a list, such as taking them to a form displaying the details of the selected record.



In the above example, the App Maker has dictated that the user will be taken to the **Entity Form** if they want to *create* or view *details* for a specific record from a list.

## Use themes in portals

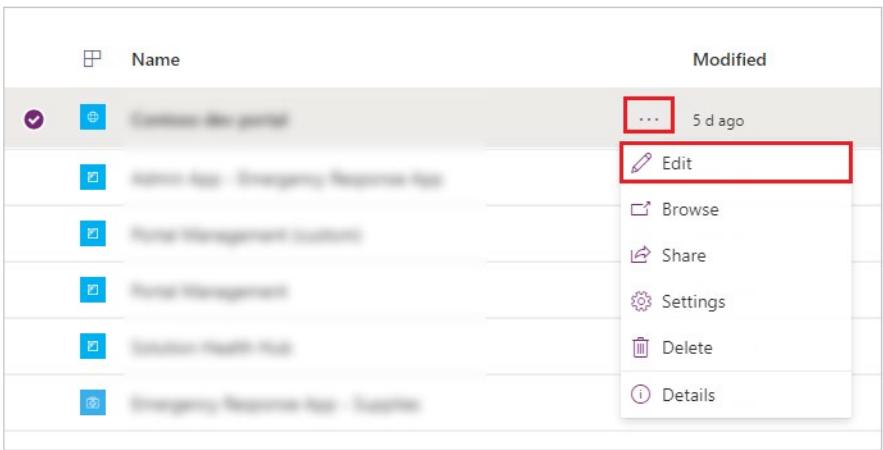
In Power Apps portals, the **Enable basic theme** feature is set to **Off**. When you turn on this feature, you can use default themes called **Presets**. You can also create copies of the preset themes for additional customization.

1. Sign in to Power Apps by navigating to make.powerapps.com
2. Select **Apps** from the left navigation pane, and then select your portal.

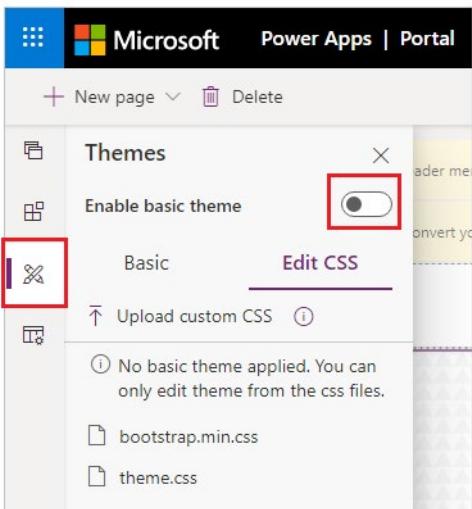
The screenshot shows the Power Apps portal interface. The left sidebar has options like Home, Learn, Apps (which is selected and highlighted with a red box), Create, Data, Flows, AI Builder, and Solutions. The main area has a toolbar with Edit, Browse, Share, Settings, Delete, and Details buttons. Below is a table titled 'Apps Component libraries (preview)'. The table has columns for Name, Modified, Owner, and Type. One row is selected and highlighted with a red box, showing the name 'Customer Portal'.

Name	Modified	Owner	Type
Customer Portal	5 d ago	[redacted]	Portal
Customer Portal - Emergency Response Plan	1 wk ago	[redacted]	Model-driven
Hotel Management System	1 wk ago	[redacted]	Model-driven
Hotel Management	1 mo ago	[redacted]	Model-driven

3. Select **More Commands (...)**, and then select **Edit**.



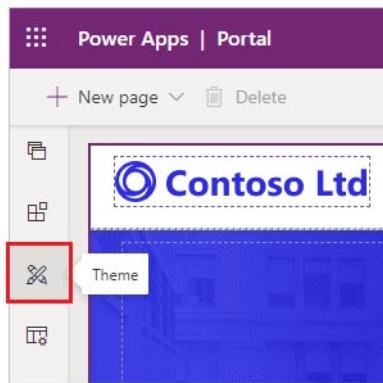
4. Select Themes from the left navigation pane, and then turn on the Enable basic theme toggle.



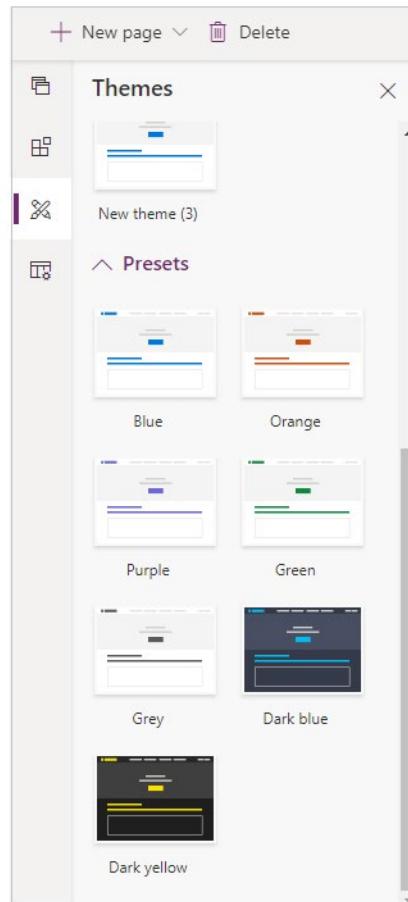
## Change theme for your portal

You can set any existing theme in your portal to the default theme.

1. Sign in to Power Apps by navigating to make.powerapps.com
2. Select **Apps** from the left navigation pane, and then select your portal.
3. Select **More Commands (...)**, and then select **Edit**.
4. Select **Theme** from the components pane.



5. Select any default theme from the available presets (in this example, we selected Green).

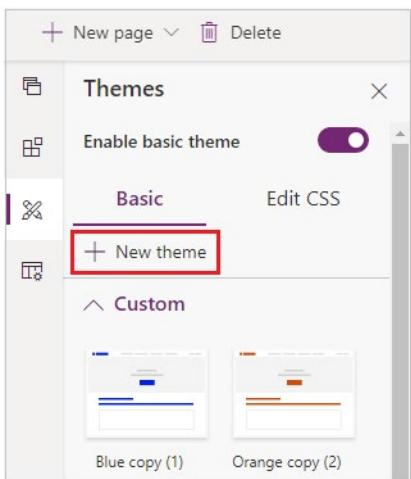


The selected theme is applied to your portal.

## Create a new theme

1. Sign in to Power Apps by navigating to make.powerapps.com
2. Select **Apps** from the left navigation pane, and then select your portal.

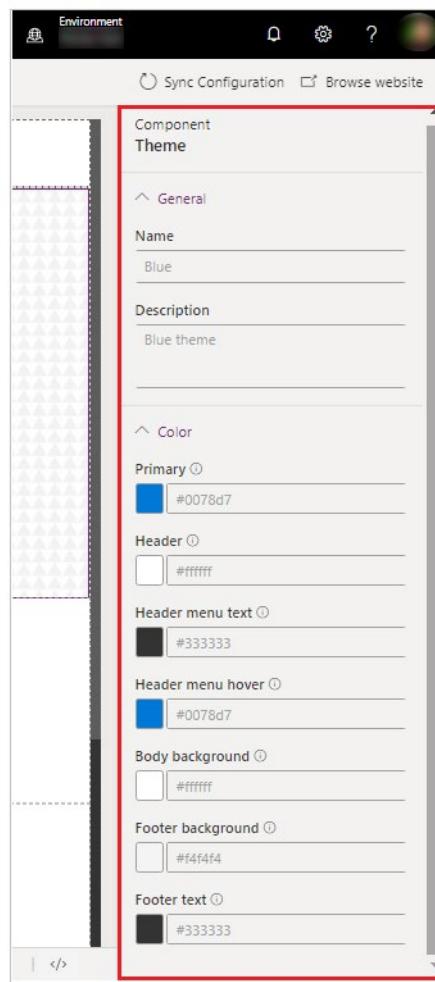
3. Select **More Commands (...)**, and then select **Edit**.
4. Select **Theme** from the components pane.
5. Select **New Theme**.



## Edit theme details

You can update theme name, description, color, and other typography settings in Power Apps Studio.

1. Sign in to Power Apps by navigating to make.powerapps.com
2. Select **Apps** from the left navigation pane, and then select your portal.
3. Select **More Commands (...)**, and then select **Edit**.
4. Select **Theme** from the components pane.
5. Select the theme that's currently applied or select a new theme from the presets. Selecting a theme opens the details pane on the right side of your workspace.

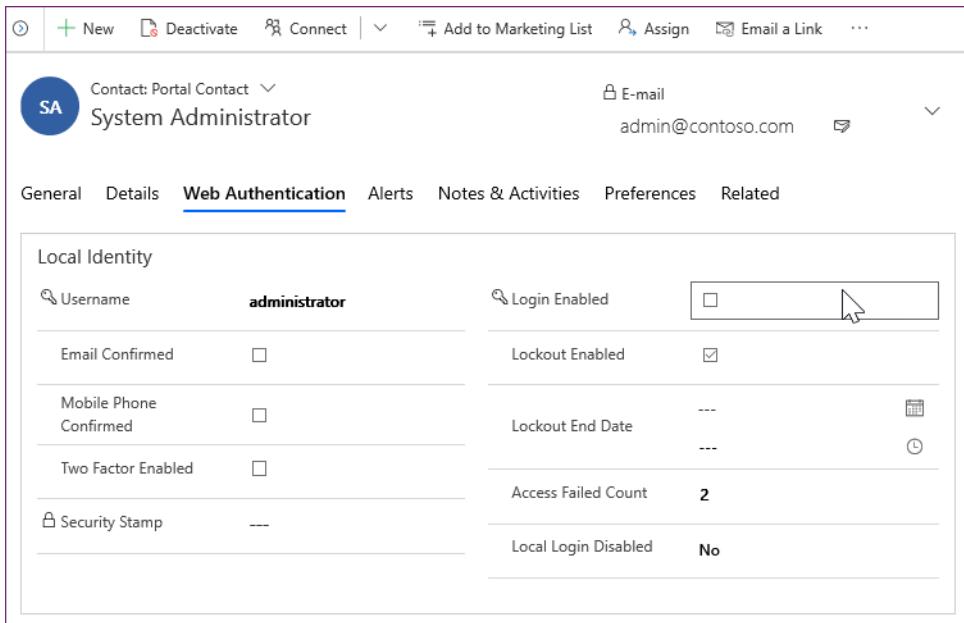


6. Edit theme details such as name, description, and color for different areas.
7. Save and publish the changes.

## Overview of portals security

Users of Power Apps portals are tracked in Common Data Service as contacts.

The Portal Management model-driven app provides access to the contact entity and has forms to manage passwords, view portal-specific contact information, and provide registration and profile management forms for the portal.



All interactions and actions that a portal user takes (for example, leaving a comment on a page) are tied to their contact record in Common Data Service.

## Authentication

Portal users can authenticate by using the following methods:

- **Local authentication** - Basic authentication with usernames and passwords are stored in the Common Data Service contact record internally.
- **External authentication** - Credentials and password management are handled by other identity providers. Supported authentication providers include:
  - OAuth2 (Microsoft, Twitter, Facebook, Google, LinkedIn, Yahoo)
  - Open ID (Azure Active Directory, Azure Active Directory B2C)
  - WS-Federation and SAML 2.0 (used for integration with on-premises Active Directory and other identity services)

Portal administrators can choose to enable or disable any combination of authentication options through portal **Authentication Settings**.

The screenshot shows the Microsoft Power Apps portal interface. On the left, there is a navigation sidebar with options like Home, Learn, Apps, Create, Data, Flows, AI Builder, and Solutions. The main content area is titled 'Identity providers' under 'Contoso Portal > Identity providers'. It lists various provider configurations with columns for 'Provider name', 'Provider type', and 'Status'. Providers listed include Local sign in (Local sign in, Enabled), Azure Active Directory (Azure active directory, Enabled), Azure Active Directory B2C (Azure Active Directory B2C, Click to configure), Facebook (Facebook, Click to configure), LinkedIn (LinkedIn, Click to configure), Google (Google, Click to configure), Twitter (Twitter, Click to configure), and Microsoft (Microsoft, Click to configure). A note at the bottom says, 'You can configure additional providers using Portal Management app. Learn more'.

[!IMPORTANT]

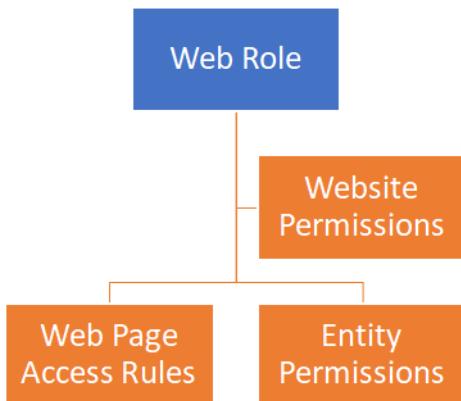
Azure Active Directory B2C is the recommended identity provider for authentication. If another provider support is required, then it can be configured in Azure Active Directory B2C.

## Authorization

After the user is authenticated and associated with a contact, Power Apps portals use numerous entities to define authorization, that is, what a user is allowed to do. Selecting **Share** from the portal app options will provide information on how to share the portal app with internal and external users.

The screenshot shows the Power Apps portal interface. On the left, there's a navigation bar with options like Home, Learn, Apps, Create, Data, Flows, AI Builder, and Solutions. The main area is titled 'Apps' and shows a list of four apps: Contoso Portal, Class Room App, Solution Health Hub, and Portal Management. Each app entry includes a small icon, the name, modified date, owner, and type (Portal, Model-driven, or Model-driven). To the right of the app list is a 'Share this portal' dialog. It has two main sections: 'Share with internal Users' and 'Share with External Users'. The 'Share with internal Users' section contains steps to create a security role (by going to Security Roles) and assign users (by going to the Users page under Security). The 'Share with External Users' section contains steps to add users to portal web roles (by going to Web Roles) and invite users (by going to Contacts).

**Web roles** allow an administrator to control user access to portal content and Common Data Service records.



A web role can be associated with the following records:

- **Website permissions** - Define what (if any) front-side editing permissions that a web role should have.
- **Webpage access rules** - Define what pages are visible to a web role and what actions can be taken.
- **Entity permissions** - Define what access a web role has to individual Common Data Service entities.

A portal contact might be assigned to one or more web roles at a time. Access rules and permissions of individual roles are combined to determine the resulting permissions set.

One of the web roles in the portal can be marked as **Anonymous** and all of the others are **Authenticated**. These roles allow you to apply permissions and

access rules to all portal users based on whether they are signed in.

If a user is not signed in, they will view the portal with the Anonymous web role permissions, which should be the most restrictive permissions.

## Summary

Power Apps portals extend Common Data Service access to external audiences such as customers, employees, or partners. This access allows businesses to extend and scale their operations as they reduce call center costs, manual processing, and resolution times while simultaneously improving user satisfaction, transparency, and scale of operations.

This module covered the following concepts:

- How Power Apps portals can add value to customer Power Platform and Dynamics 365 solutions.
- Core components for configuring a portal: webpages, templates, entity forms, content snippets, and entity lists.
- Using and creating themes in Power Apps portals
- Fundamentals of portals security and which methods are used to control access to the content and data.

# Access Common Data Service in Power Apps portals

## Introduction

A typical use case of Microsoft Power Apps portals is to extend functionality from Dynamics 365 applications to internal and external audiences. The portal templates that are based on Dynamics 365 apps offer built-in functionality to these audiences by adding features like case life cycle, knowledge article access, partner opportunity management, and so on.

Additionally, Power Apps portals can extend applications that are built on Common Data Service and then display the data and business logic and make them available on your portal to an external audience, based on user permissions.

Consider the following scenarios:

- You have extended case management features of Dynamics 365 Customer Service. Your app now handles product warranties, return management authorization (RMA), refunds, and product replacements. You can extend the Customer self-service portal to bring this functionality to your customers, allowing them to register the warranty, make a claim, fill in an RMA form directly on the site, and so on.
- You have built an application in Common Data Service to track charitable donations, from both individual and corporate donors, and fundraising campaigns. You can build a portal to allow individuals to donate online and for employers to view and match their employees' donations.
- You use Common Data Service to track the progress of your certification programs from the application process, to the evaluation of various skill assessments, to awarding a certification level to a candidate. A Power Apps portal can be configured to allow online applications, provide access to update evaluations, and allow candidates to view their progress.

[!NOTE]

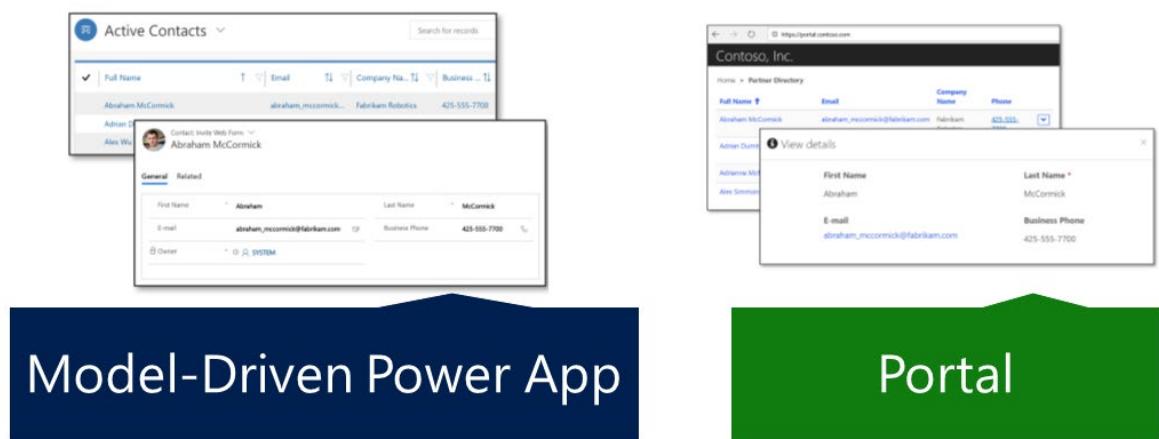
A Power Apps portal can only access Common Data Service data in the same instance where the portal is provisioned. Connecting to multiple instances and accessing data across the instances or across the tenants is not supported.

Model-driven Power Apps are low-code/no-code methods that you can use to build unique line-of-business applications (see **What are model-driven apps in Power Apps?**<sup>2</sup>). Part of the app creation process is to define the model and UI elements such as views and forms (see **Understand model-driven app components**<sup>3</sup>). Power Apps portals extends these UI elements to the web by using entity lists, entity forms, and web forms:

- **Entity lists** - Define how the list of Common Data Service records is displayed on the portal pages. They are defined by one or more model-driven app entity views and include functionality like filtering and sorting.
- **Entity forms** - Add the ability for the portal pages to interact with the records in a specific entity by using a model-driven app form definition as a layout template.
- **Web forms** - Render one or more model-driven app forms on a portal website with support for single or multi-step navigation and conditional branching logic.

<sup>2</sup> <https://docs.microsoft.com/powerapps/maker/model-driven-apps/model-driven-app-overview/?azure-portal=true>

<sup>3</sup> <https://docs.microsoft.com/powerapps/maker/model-driven-apps/model-driven-app-components/?azure-portal=true>



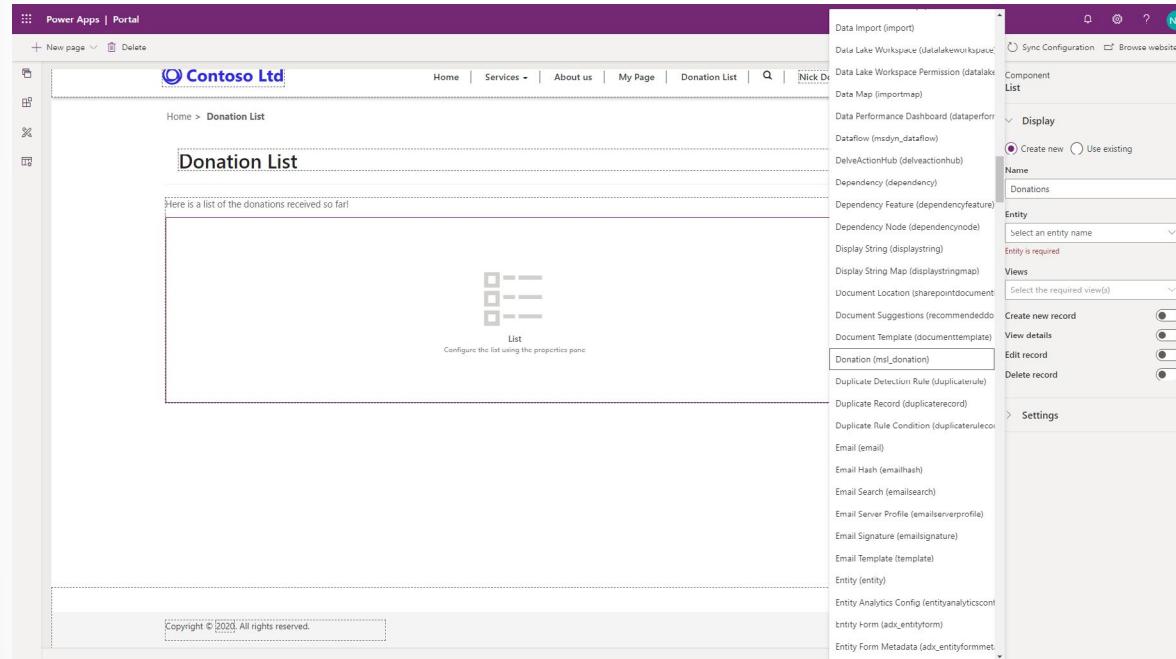
The image shows two side-by-side screenshots. On the left, titled 'Model-Driven Power App' in a blue box, is a screenshot of a Power Apps portal page titled 'Active Contacts'. It displays a list of contacts with columns for Full Name, Email, Company Name, and Business Phone. A specific contact, 'Abraham McCormick', is selected, showing a detailed view with fields for First Name, Last Name, E-mail, Business Phone, and Owner. On the right, titled 'Portal' in a green box, is a screenshot of a standard web-based portal page titled 'Contoso, Inc.' showing a similar contact list and detail view for the same contact.

## Use entity lists to display multiple records

The **Entity Lists** component allows a maker to display a list of Common Data Service records on a portal webpage by using configuration only. Entity lists are defined by using model-driven app views and can be further configured to filter data based on entity permissions. Entity lists can have additional features enabled such as running workflows and navigating to show detail records.

### Create an entity list

An entity list can be added as a component to a webpage in Power Apps portals Studio. After placing the list component on a page, the maker will need to set the properties of the entity list. Then, the maker will choose the entity and one or more model-driven views to be used to render the entity list on the page.



The image shows the Power Apps Portal Studio interface. On the left, there's a preview of a portal page titled 'Contoso Ltd' with a 'Donation List' section. The preview shows a placeholder for the entity list. On the right, the 'Component List' pane is open, specifically the 'Display' section under 'Entity Lists'. It shows configuration options like 'Create new' (radio button selected), 'Name' (set to 'Donations'), 'Entity' (dropdown set to 'Select an entity name'), and 'Views' (dropdown set to 'Select the required view(s)'). Below these, there are toggle switches for 'Create new record', 'View details', 'Edit record', and 'Delete record'. At the bottom of the configuration pane, there's a 'Settings' section. The bottom of the studio interface shows a copyright notice: 'Copyright © 2020. All rights reserved.'

## List rendering

Adding a list component in portals Studio will add the corresponding Liquid tag to the webpage content.

When the webpage is requested, the list rendering process is as follows:

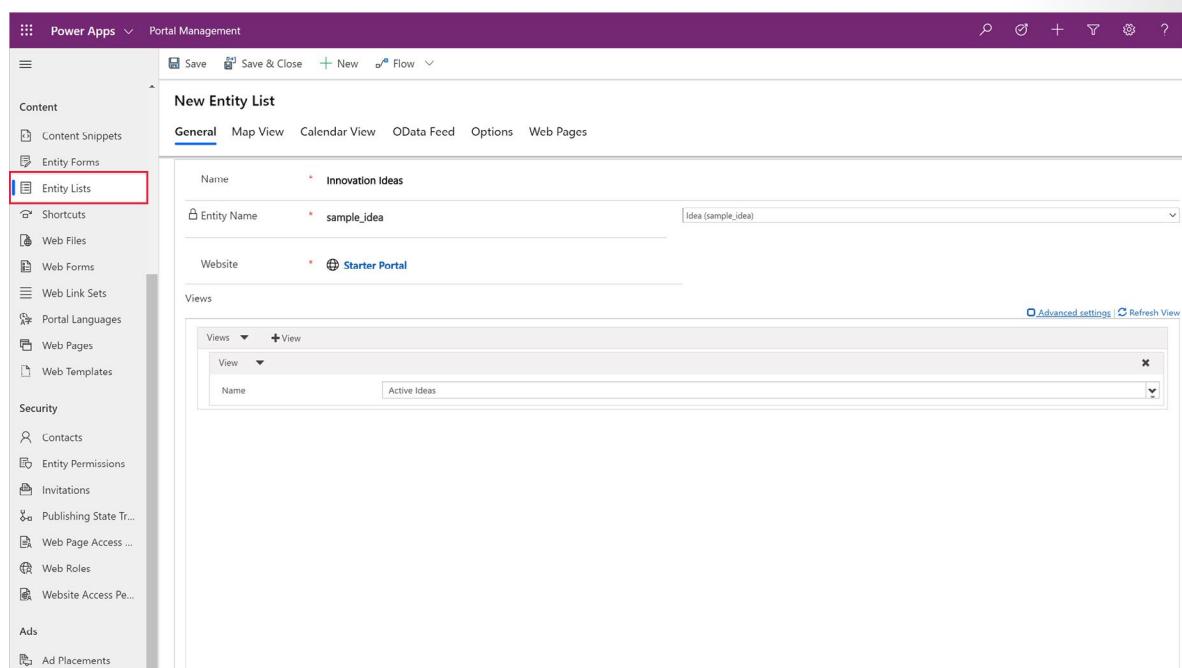
1. The webpage is retrieved.
2. The webpage will render the entity list based on the Liquid tag that was added when the list component was added to a webpage.

```
{% include 'entity_list' key: '<<Entity List Name>>' %}
```

## Configure the entity list

Power Apps portals Studio provides a basic interface for creating and configuring an entity list. Makers can customize all entity list features and properties by using the Portal Management app. To access entity lists in the Portal Management app:

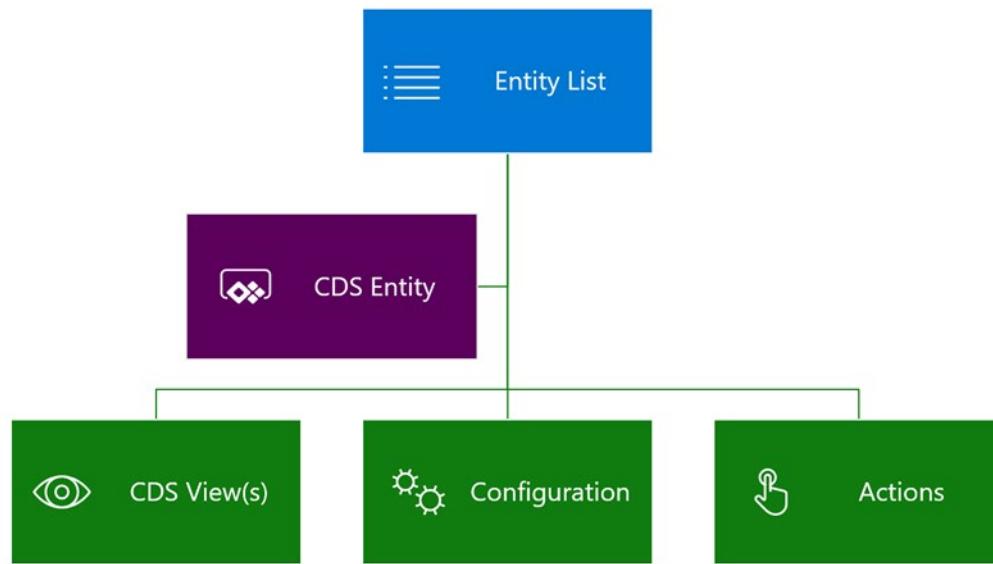
1. Go to the **Power Apps maker portal**<sup>4</sup>.
2. Select the target environment by using the environment selector in the upper-right corner.
3. From the **Apps** list, locate and open the Portal Management app (type will be Model-driven).
4. Select **Entity Lists** in the left navigation.
5. Open the list that you previously created in portals Studio.



An entity list can be as simple or as complicated as your business requirements specify. The only required properties for the entity list, other than the name and the website, are the target **Entity Name** and one or more **Views**.

<sup>4</sup> <https://make.powerapps.com/?azure-portal=true>

Entity lists are highly configurable and have many settings that define the list behavior. Lists can also include actions for the user to interact with the items on the list.



The following sections explain some of the most common features and settings.

[!NOTE]

Most of the options that add interactive elements, such as buttons, support customization of the elements in multiple languages. For example, if multiple views are enabled, the name for each of the views in the view selector can be customized for each of the enabled portal languages.

## Views

Selected view(s) define the Common Data Service entity fields, list layout, and the default sort order.

**Multiple views** - If more than one view has been specified, a drop-down list will be rendered to allow the user to switch between the views.

**Sorting and pagination** - Sorting is enabled on any of the displayed columns and the page size is configurable.

The screenshot shows a list of contacts from the 'Entitlement Contacts' view. The columns are 'Email' and 'Company Name'. Red annotations highlight several features:

- View selector:** A dropdown menu labeled 'Entitlement Contacts' with a red arrow pointing to it.
- Sorting:** An arrow pointing to the 'Email' column header, which has a downward arrow indicating sorting.
- Hyperlink to view:** A red box around the email address 'winifred\_pollard@fabrikam.com' for Winifred Pollard.
- Pagination:** An arrow pointing to the page navigation bar at the bottom, which shows pages 1 through 44.

	Email	Company Name	
Yvonne McKay	ymckay@proseware.com	Proseware, Inc.	<input type="button" value="▼"/>
Wilson Pais	wparis@cohowinery.com	Coho Winery	<input type="button" value="▼"/>
Winifred Pollard	winifred_pollard@fabrikam.com		<input type="button" value="▼"/>
Wilson Chew	WilsonChew@fabrikam.com		<input type="button" value="▼"/>
William Flash	williamf@alpineskihouse.com	Alpine Ski House	<input type="button" value="▼"/>

[!NOTE]

Entity lists include the **Web Page for Details View** and **Web Page for Create** general properties. These properties are for backward compatibility only. The functionality is included as part of the **View Actions** and **Item Actions** grid configuration.

## Configuration

The following sections describe the different types of configuration that are available in entity lists.

### Filter and search

Entity lists provide you with several options to filter and search list data:

**Search** - When quick search is enabled, the portal will render a text search box. It is similar to the quick search feature in model-driven apps. Quick search runs across the view columns and allows you to locate the information within the larger lists by using plain text input. Portal users can use the asterisk wildcard character to search on partial text.

**Portal filters** - The list data can be filtered by the current portal user, the current portal user's parent account, and the current portal website. This feature enables some common scenarios without needing additional configuration:

- List of product reviews that are left by the current user
- List of campus buildings for the current user's department (account)
- List of all draft pages for the current website, only when multiple portals are provisioned

If the current portal user and current portal user's parent account filters are enabled, the portal will render a drop-down list to allow the user to view their own data (My) or their parent account's data (account name will be displayed).

**Metadata filters** - Records in the list can be filtered on any of the list columns, including ranges, lookups, option sets, and custom FetchXML expressions. Portal users have access to an interactive filtering panel when the entity list is rendered.

The screenshot shows a search interface with several filter options. On the left, there are dropdowns for 'Currency' (Euro) and 'Email' (Yes or No), with an 'Apply' button below. A red box highlights the 'Metadata filters' section, which contains dropdowns for 'Southridge Video' (selected), 'My Southridge Video' (Eric Gilmore), and 'Portal filters' (Sato, John Abercrombie). To the right is a search bar with a magnifying glass icon and a 'Search' button. Below the search bar is a table with columns 'Company Name', 'Phone', and 'Currency'. The table lists three entries for 'Southridge Video' with details like phone numbers (408-875-4575, 111-587-4587, 408-875-4582) and currency (Euro).

## Display options

Views can be rendered as traditional grid lists, a calendar, or a map. Delivering list content as an OData feed is also supported.

The screenshot displays two views side-by-side. On the left is a map of Europe with various locations marked by purple circles, each containing a number from 1 to 16. A search bar at the top left allows searching for locations near Amsterdam within 2500 km. On the right is a calendar for March 2019. Each day of the month has a grid of events. For example, March 1st has three events: 'D365 Saturday Amsterdam | 5/18/2019 3:30 PM' (link), 'User Group Summit Europe | 3/27/2019 8:00 AM' (link), and 'eXtreme365 Amsterdam | 3/24/2019 8:00 AM' (link). The calendar also includes a 'Find an event' search bar and navigation buttons for Year, Month, Week, and Day.

Alternative views might require some additional configuration such as start and end date fields for a calendar or latitude and longitude fields for a map.

### [!IMPORTANT]

Map and calendar views require page templates that can render the view. When a starter portal is provisioned, **Rewrite** page templates like **Page** or **Full Page** support this functionality. Pages that use web templates will render the default entity list view.

## Actions

Entity lists can have actions associated with them to enable per list commands, like Create and Download (as a Microsoft Excel spreadsheet), or per record commands like View or Edit, and to trigger workflows.

Full Name	Email	Company	Phone	Currency
Yvonne McKay	ymckay@proseware.com	Proseware, Inc.	407-967-2242	Euro
Winifred Pollard	winifred_pollard@fabrikam.com			
Wilson Pais	wparis@cohownery.com	Coho Winery	456-698-4582	
Wilson Chew	wilson_chew@fabrikam.com			
William Flash	williamf@alpineskihouse.com		54-4589	

For more information, see [About entity lists<sup>5</sup>](#).

Watch the following video to learn how to configure and display the view on a portal page and add interactive filtering capabilities on an existing Common Data Service entity.



<https://www.microsoft.com/videoplayer/embed/RE4ArhG>

Now that you've observed how the entity lists are built and rendered, you can learn how portals can display and interact with individual records.

## Use entity forms to interact with Common Data Service

Entity Forms add the ability for portal pages to interact with records in a specific entity by using a model-driven form definition as a layout template. Similar to entity lists, entity forms are data-driven configurations that allow users to add a form to collect or display data in the portal without the need for a developer to create a custom form. Entity forms are defined by using model-driven forms and can be placed into webpages in the portal.

Entity forms can display most field types and subgrids but, currently, they can't display Power Apps Component Framework controls.

<sup>5</sup> <https://docs.microsoft.com/powerapps/maker/portals/configure/entity-lists/?azure-portal=true>

This website uses sample data for illustration purposes. You can build pages using multiple templates available.

**Idea Detail**

General

Originating challenge  
Smarter manufacturing

Name \*  
Solar Powered Refrigerator

Investment Required  
10000

Description

Doner ham pastrami, chuck buffalo salami andouille jerky meatloaf biltong tail kevin bresaola. Ham hock meatloaf andouille rump pastrami swine sirloin pork chop strip steak drumstick. T-bone pancetta shankle rump landjaeger sirloin. Pancetta tri-tip turducken filet mignon porchetta [landjaeger](#) biltong tenderloin.

TNCYM  
Generate a new image  
Play the audio code

Submit

Copyright © 2020. All rights reserved.

## Common uses

The following table explains the common scenarios where entity forms can be used.

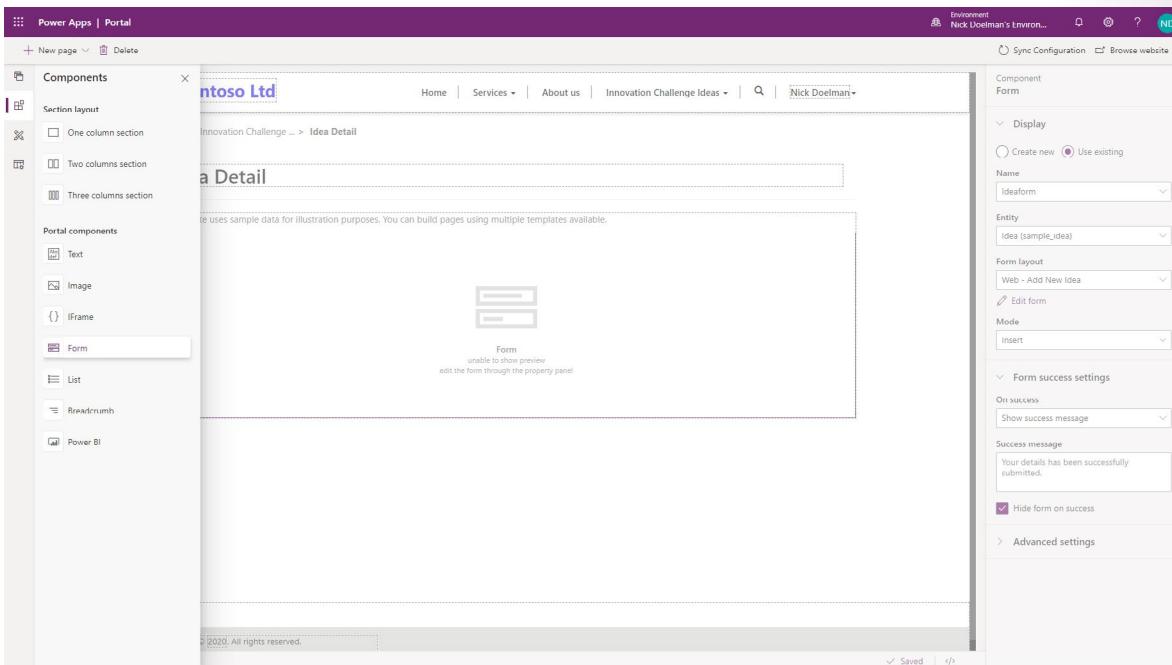
Scenario	Description
<b>Layout</b>	Entity forms can be configured and used in read-only mode as a layout mechanism. Think of entity forms as informational pages about employees, products, or any other Common Data Service entity. For example, you might have a custom entity in your Common Data Service instance that describes open positions that are available within your organization. Instead of crafting a special template to render the page, you can create a separate model-driven form for the entity, create a new webpage by using portals Studio, and then add an <b>Entity Form</b> component to the page. Any published changes in a form layout in a model-driven app will automatically apply to the webpage.
<b>Information capture</b>	Forms can be used on the portal for data capture from anonymous or authenticated users. For example, a simple lead entity form might be rendered on a <b>Contact Us</b> page to record anonymous requests as leads in Dynamics 365 Sales. For authenticated users, a portal might use a survey page to collect product feedback from customers into a custom Product Feedback entity.

Scenario	Description
<b>Record management</b>	Typically used in authenticated scenarios, entity forms allow various combinations of Create, Retrieve, Update, and Delete (CRUD) operations on an entity to be defined within a set of related webpages. For example, customers can retrieve and read their own cases and create new ones, partners can edit their company profiles, and employees can view the list of assets that are allocated to them by the company.
<b>Web apps</b>	Entity forms, when used in conjunction with entity lists and subgrids, and with the functionality extended by JavaScript, allow developers to build complete web applications.

## Create an entity form

When creating a new entity form, your first step is to decide the **Entity** and **Form Name** that you will be rendering.

While reuse of forms that are part of a model-driven app is possible, the common practice is to design portal-specific model-driven forms (that might or might not be included in the model-driven apps).



- Portal forms are more concise with less information presented, especially when external audiences are involved.
- Portal processes are separate from the internal use of a staff-facing, model-driven app (including Dynamics 365 apps). Dedicated portal forms are easier to maintain because any changes in the model-driven apps need to be manually applied to the portal forms, giving you an opportunity to review the requirements and assess usability aspects.

- Certain limitations exist on the form and fields that are rendering, for example, PCF controls are not rendered. For more information, see [About entity forms<sup>6</sup>](#).
- Client-side business rules and JavaScript, which are essential parts of a model-driven form, will not run on the portal. It's easy to overlook and might result in unintended consequences.
- Special considerations need to be given when you are rendering related records, notes, and a timeline because not all functional aspects are supported (or required) in the portals.

## Mode

The form mode can be **Read Only**, **Insert**, or **Edit**. This mode defines if the form is used to generate a layout, capture the data, or provide full editing capabilities for Common Data Service records.

[!NOTE]

A form that is in **Edit** mode will be rendered as **Read Only** if the user does not have write privileges for the record.

If the mode is **Insert**, no additional information is required. For **Read Only** and **Edit** modes, the form will need to "know" the entity record to display and update. The **Record Source Type** setting defines how this information is passed to the form:

- **Query String** - This setting is default when you are creating an entity form in portals Studio. When the page that contains the form is displayed, the record identifier is expected to be part of the query string, for example `https://contoso.powerappspartals.com/contacts/edit/?id=<contact guid>`. Usually, this setting is done automatically when the form is linked to an entity list. This setting is by far the most common.
- **Current Portal User** - This option is configured within the Portal Management app. When this option is selected, the form will load the information from the current portal user record without using additional information from the page URL. Commonly, this option is used to render a user profile form. The **Entity Name** field in this case must be set to **Contact** because portal users are represented by the contact entity.
- **Record Associated to Current Portal User** - This option is configured within the Portal Management app. Selecting this option allows you to edit associated records, such as the current user's parent account details. **Relationship Name** must be specified to identify the record to edit. The entity type that is selected must match the selection in the **Entity Name** field. This option is useful in partner scenarios where the partner organization would have multiple portal users. Some of these users could be authorized to edit the parent account record.

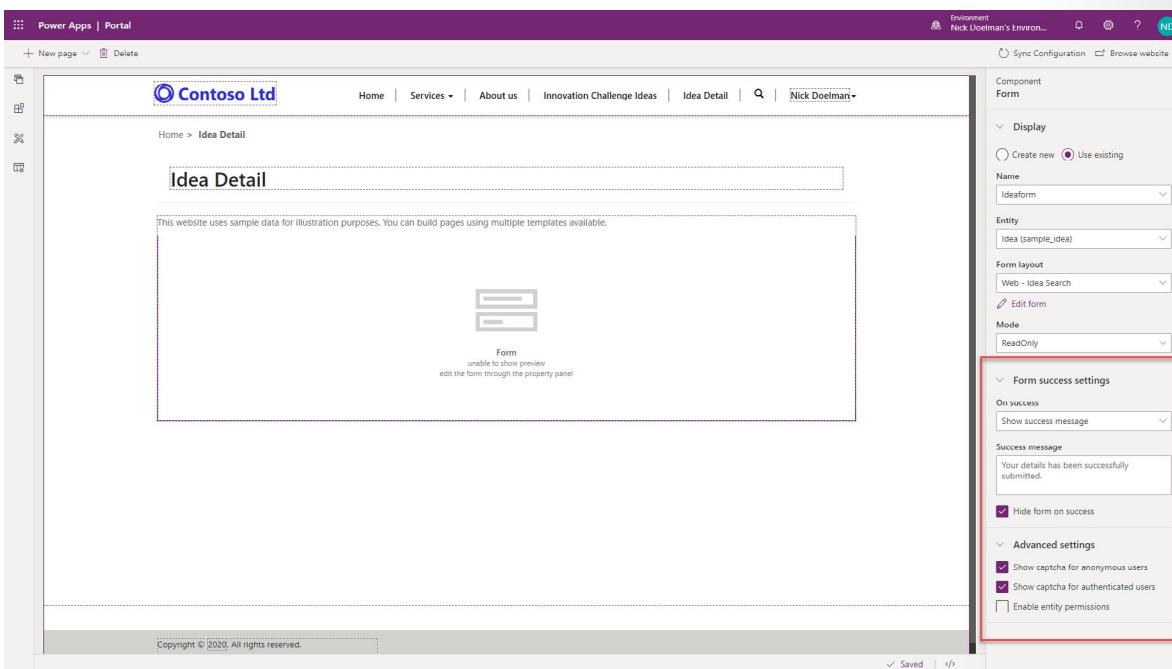
**Entity**, **Form**, and **Mode** are the details that are required to render the form.

## Configure the entity form

Additional configuration options to control form appearance and behavior are available within portals Studio.

---

<sup>6</sup> <https://docs.microsoft.com/powerapps/maker/portals/configure/entity-forms/?azure-portal=true>



## Form success settings

The form success settings determine the actions that are taken when a form is successfully submitted to a portal.

- **Show success message** - Shows a message when a form is submitted.
- **Redirect to webpage** - Automatically navigates to a webpage on success of the form submission.
- **Redirect to URL** - Redirects to a URL (portal or external).

## Advanced settings

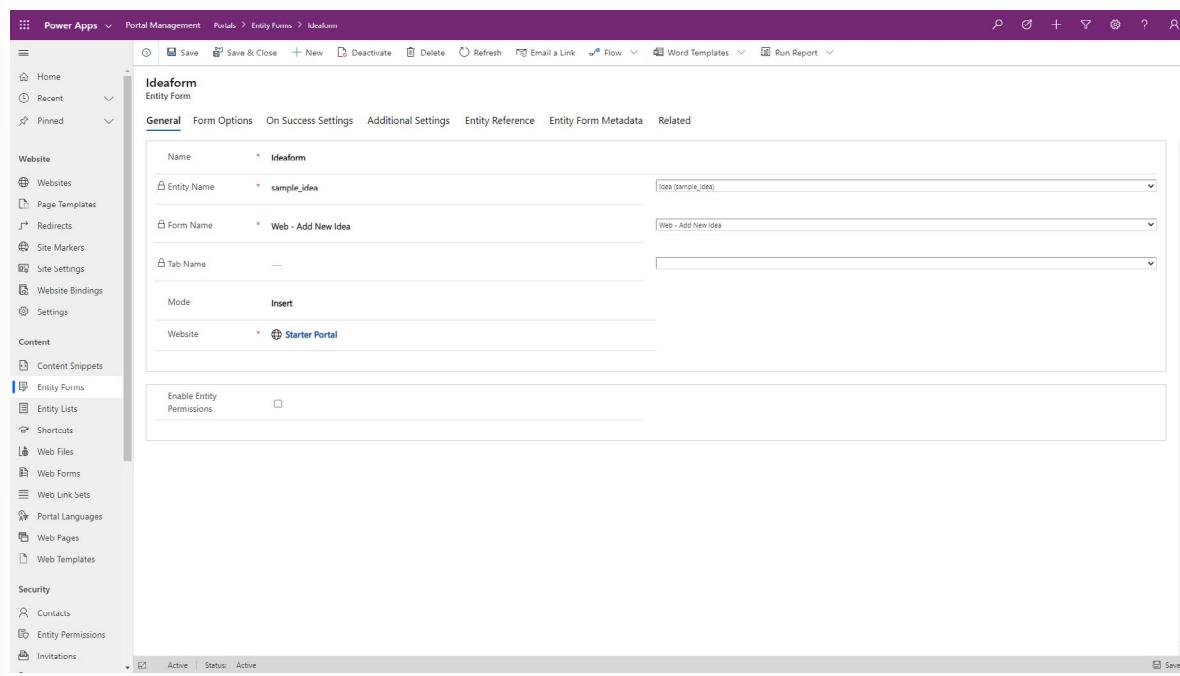
The **Advanced settings** feature determines if a captcha is displayed for anonymous or authenticated users. Also, the setting helps determine if entity permissions are enabled for the particular entity.

## Additional entity form settings

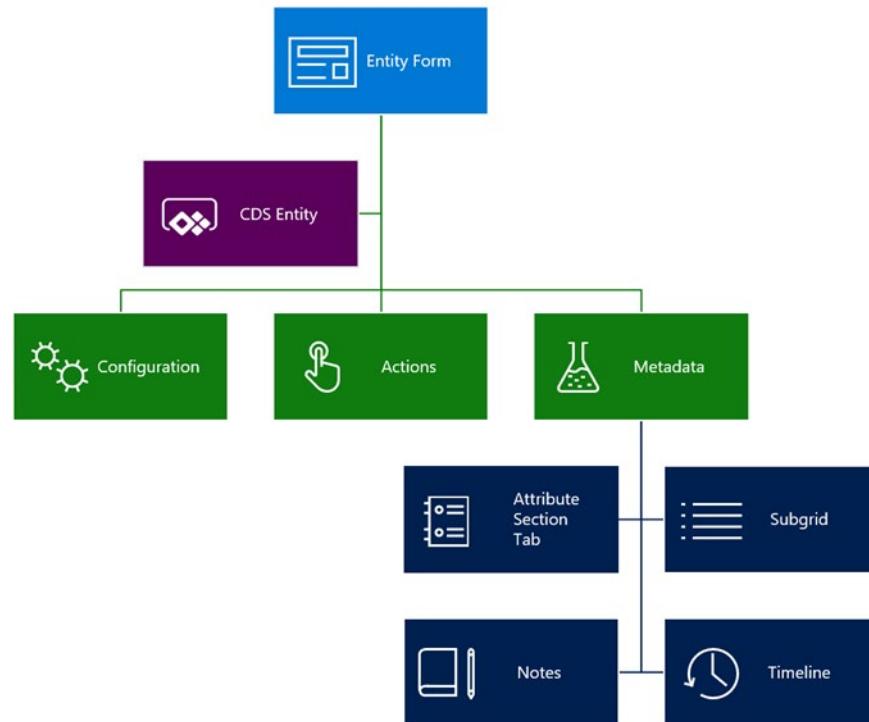
Additional configuration settings are available for entity forms in the Portal Management app. To access an entity form in the Portal Management app:

1. Go to the **Power Apps maker portal**<sup>7</sup>.
2. Select the target environment by using the environment selector in the upper-right corner.
3. From the **Apps** list, locate and open the Portal Management app (type will be Model-driven).
4. Select **Entity Forms** in the left navigation.
5. Open the form that you previously created in portals Studio.

<sup>7</sup> <https://make.powerapps.com/?azure-portal=true>



Forms can include **Actions** for the user to interact with the record. Detailed configurations for each form element are available by using additional **Entity Form Metadata** records.



]

## Configuration

The following sections describe the different ways that you can configure entity forms.

## Form options

Most form options support customization of the Cascading Style Sheets (CSS) elements to change visual appearance. Text elements such as labels, messages, and tooltips can be specified in multiple languages. For example, the default message after the form submission is "Saved," but it can be customized for each of the enabled portal languages.

Another form option includes **Control validation behavior**, where you can determine whether to mark all fields as required regardless of the form setting, for example.

## Additional settings

Additional settings define other aspects of form behavior, such as styling, translation of the UI elements, and so on.

[!TIP]

Some of the settings and configuration options are hidden. Select the **Advanced settings** check box to display all available options.

**Associate current portal user on insert** - This option can be used to keep track of which portal contacts created or updated the record. This setting creates a portal equivalent of the **Created By** and **Modified By** fields in Common Data Service. You can also set parental relationships where applicable. For example, if a new account record is created, you might want to set the current contact as a primary contact automatically.

**Add attach file** - A number of options are available to add a file upload control to your form. Configuration is flexible and supports multiple files, storage options, mime type, and size restrictions (for example, you can restrict uploads to images up to 2 MB in size).

**Geolocation** - An entity form can be configured to display a map control to display an existing location as a pin on a map or to provide a user with the ability to specify a location. For more information, see [Add Geolocation<sup>8</sup>](#).

The form's map control requires additional configuration to tell it what the various location fields are, to assign values to them, or retrieve values from them. For more information, see [Geolocation configuration for entity forms<sup>9</sup>](#).

<sup>8</sup> <https://docs.microsoft.com/powerapps/maker/portals/configure/add-geolocation/?azure-portal=true>

<sup>9</sup> <https://docs.microsoft.com/en-us/powerapps/maker/portals/configure/entity-forms#geolocation-configuration-for-entity-forms>

### Location

Address 1

3000 E 1st Ave, Denver, CO 80206

Address 1: Street 1  
3000 E 1st Ave

Address 1: City  
Denver

Address 1: County

Address 1: State/Province  
CO

Address 1: Country/Region  
United States

Address 1: ZIP/Postal Code  
80206

Address 1: Latitude  
39.71794

Address 1: Longitude  
-104.95213

## Entity reference

**Entity reference** provides a way to associate the current record that is being created or updated with another target record. This feature is useful if you have multiple steps with multiple entity types and want to relate the resulting records or if the page is passed in a query string of a record ID that you want to associate.

For example, you might have an event page that displays information about an upcoming webinar. You want to include a registration button that redirects visitors to the registration page where the registration form is displayed. You can pass the event identifier in a query string and, when the registration form is submitted, you'll be able to automatically link the registration information to the event.

## Actions

Because an entity form deals with an individual entity record, numerous actions can be run against this record, such as Update, Delete, Deactivate, and so on. These actions, that at runtime are displayed as command buttons, can be configured by selecting **Additional Settings > Action Button Configuration**.

All commands include options to rename the buttons and change their placement on the form.

## Entity form metadata

**Entity form metadata** records allow you to control the appearance and behavior of individual form elements, including:

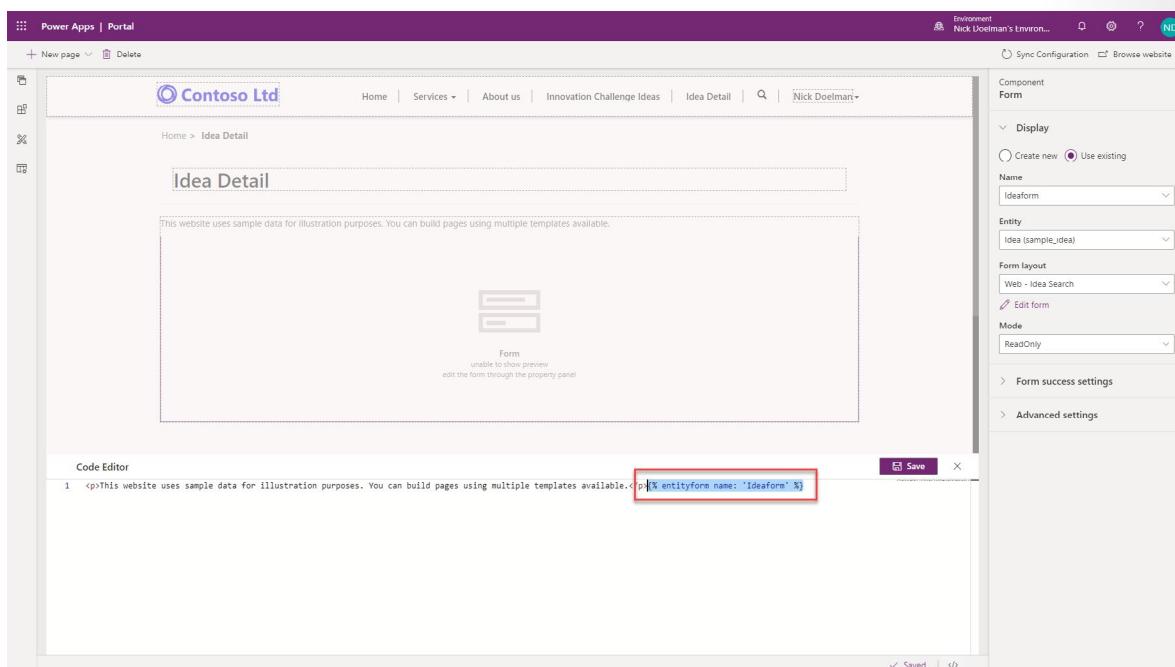
- Appearance of the fields, sections, and tabs. An individual field's default values, validation behavior, and other aspects can be defined.
- Subgrids configuration, which allows you to define actions for related records, similar to entity list actions.
- Behavior of notes and timeline sections, and if new records can be added. This option is commonly used to allow portal users to enter comments about the record, for example, a case in process.

## Add an entity form to your portal

An entity form defines the required behavior but does not contain information about how and where on the site that the form should be rendered. Two methods to render an entity form in a portal are:

- The entity form can be explicitly specified as a target for an entity list action like Create, Update, or Edit. In this case, the form will be rendered in a modal pop-up window. Certain limitations apply when the form is rendered in a pop-up window, for example, the ability to create related records from subgrids is not available.
- Similar to an entity list, an entity form component can be added to a webpage from portals Studio. This action will place a Liquid tag on the webpage copy to render the entity form.

```
{% entityform name: '<<entity form name>>' %}
```



Because forms can submit information back to the server for processing, you will have less control over the rendering of entity forms as compared to the entity lists.

The following video shows how to extend entity list functionality with entity forms and display a Common Data Service record in a pop-up window or on a separate webpage.

(How to extend entity list functionality)[<https://www.microsoft.com/videoplayer/embed/RE4AJtN>]

## Exercise-Use an entity list and entity form

The purpose of this exercise is to provide you with hands-on experience in adding a list and a form to a Power Apps portal.

### Learning objectives

At the end of these exercises, you will be able to:

- Add a list component to a portal page.

- Add a form component to a portal page.
- Configure the list component to drill down and view details of the record.

## Prerequisites

The following prerequisites are necessary for completing these exercises:

- Access to the Power Apps maker portal.
- Ideally, you will have the default sample Common Data Service apps and data, but you can use other Common Data Service entities.
- You will need to have provisioned a Power Apps portal to complete the exercise. If you do not have a provisioned portal, go to **Create Portal<sup>10</sup>** to create one.

[!TIP]

The exercises work best when you have sample data to work with. When you provision a Common Data Service environment, you have the opportunity to add sample apps and data. Review the **Create Environment<sup>11</sup>** steps to provision a Common Data Service environment with sample apps and data.

## Scenario

Your organization has provisioned a Power Apps portal and wants to display a list of ideas that the research team is considering on a public webpage. The requirement is also to allow visitors to drill down and view additional details.

## High-level steps

To finish the exercise, complete the following tasks:

- Create a webpage and add a list component that is linked to a Common Data Service entity.
- Create a child webpage with a form component to display details for a Common Data Service entity.
- Modify the list component to allow visitors to drill down and view the detail record on the form page.

## Create a webpage for the entity list

In this exercise, you will create a webpage to contain the **Entity Lists** component.

1. Go to the **maker portal<sup>12</sup>** and sign in.
2. Locate your portal app, select the ellipsis (...), and then select **Edit** to open portals Studio.
3. From the command bar, select + **New page**, and from **Fixed Layouts**, select **Page with title**.
4. In the Properties pane, fill in the following information:
  - **Name** - Ideas
  - **Partial URL** - ideasview
5. Click in the canvas area to save the webpage.

---

<sup>10</sup> <https://docs.microsoft.com/powerapps/maker/portals/create-portal/?azure-portal=true>

<sup>11</sup> <https://docs.microsoft.com/en-us/power-platform/admin/create-environment#azure-portal=true#create-an-environment-with-a-database>

<sup>12</sup> <https://make.powerapps.com/?azure-portal=true>

## Add and configure a list component

To add and configure a list component, follow these steps:

1. On the canvas, select the **page copy** component, and then from the tool belt, select **Components**.
2. In the **Portal components** section, select the **List** component.
3. In the Properties pane, enter the following values for the **List** component:
  - **Name** - Ideas List
  - **Entity** - Idea (sample idea) or choose another entity from your own app
  - **Views** - Active Ideas
4. Select **Browse** from the command bar and ensure that you can see a list of idea data records from Common Data Service.

## Create a webpage for the entity form

Your next task is to create a webpage to contain the **Entity Forms** component to view record details. This webpage will be a child page of the **Ideas list** page.

1. From the tool belt, select **Pages** and locate the **Ideas** page that you created previously.
2. Select the ellipsis (...) and then select **Add a child page**.
3. In the Properties pane, fill in the following information:
  - **Name** - Idea Detail
  - **Partial URL** - ideasdetail
4. Click in the canvas area to save the webpage.

## Add and configure a form component

To add and configure a form component, follow these steps:

1. On the canvas, select the **page copy** component, and then from the tool belt, select **Components**.
2. In the **Portal components** section, select the **Form** component.
3. In the Properties pane, enter the following values for the **Form component**:
  - **Name** - Ideas Detail
  - **Entity** - Idea (sample idea) or choose another entity from your own app
  - **Form Layout** - Information (you can select **Edit** to modify your form or create your own form specifically for the portal)
  - **Mode** - ReadOnly
4. Click in the canvas area to save the webpage.

## Modify the list page to navigate to the form component

Your last task is to modify the list page to navigate to the form component:

1. From the tool belt, select **Pages** and locate the **Ideas** page that contains the list component.
2. Select the list component on the canvas.

3. In the Properties pane, activate the **View Details** option.
4. Fill in the following information:
  - **Target Type** - Form
  - **Form** - Ideas Detail
5. Click in the canvas area to save the webpage.
6. Select **Browse** from the command bar and ensure that you can see a list of idea data records from Common Data Service and then select an idea to view details.

The screenshot shows a web browser displaying a Power Apps portal for 'Contoso Ltd'. The page title is 'Innovation Challenge Ideas'. The URL in the address bar is 'https://contosoltd.sharepoint.com/:w/sites/ContosoLtd/InnovationChallengeIdeas'. The page header includes links for 'Home', 'Services', 'About us', 'Innovation Challenge Ideas', a search bar, and a 'Sign in' button. Below the header, a breadcrumb trail shows 'Home > Innovation Challenge Ideas'. The main content area displays a table of 'Innovation Challenge Ideas' with the following data:

Name	Originating challenge	Number of Votes	Idea Score	Created On
Connected quality control	Connected Operations	10	8	5/30/2020 11:55 PM
Fleet automation	Connected Operations	8	9	5/30/2020 11:55 PM
Cloud computing	Servitization	7	9	5/30/2020 11:55 PM
Tiny Homes	3D Printing	7	6	5/30/2020 11:55 PM
Integrated service management	Connected Operations	6		5/30/2020 11:55 PM
Rapid prototyping	Smarter manufacturing	6		5/30/2020 11:55 PM
Solar panels	Enterprise sustainability	5	6	5/30/2020 11:55 PM
CO2-absorbing artificial trees	Enterprise sustainability	3	8	5/30/2020 11:55 PM
Business intelligence	Big data	2		5/30/2020 11:55 PM
Data analytics	Big data	2		5/30/2020 11:55 PM

Pagination controls at the bottom show pages 1 and 2. A copyright notice at the bottom of the page reads 'Copyright © 2020. All rights reserved.'

Adding components to a webpage will allow you to quickly view Common Data Service records.

## Summary

The ability to display and interact with Common Data Service data is a core benefit of implementing a Power Apps portal. Entity lists and entity forms are simple to configure to enable access to Common Data Service data in portals and turn content portals into functional web apps. They are configurable and extendable to satisfy many business requirements.

By now, you should be able to:

- Describe the portal components that are available to display and interact with Common Data Service data on a portal.
- Identify various features of portal components.
- Configure the **Entity Forms** and **Web Form Metadata** features to access individual records.
- Display a list of data and an associated drill-down list for details.

## Next steps

You have learned how to enable access to Common Data Service data for portal audiences. The next step would be to learn how to use Liquid code in the portal to satisfy complex requirements.

# Extend Power Apps portals

## Introduction

A variety of configuration-only tools and features are available for building a Microsoft Power Apps portals application. Examples of these tools and features include entity lists, entity forms, web forms, and the ability to integrate other technologies such as SharePoint and Microsoft Power BI. You might encounter features that are not easily configured by using the available low-code or no-code features.

Liquid template language in portal pages and templates begins to extend your web application and allows you to manipulate and display content in a variety of ways.

Power Apps portals features can also be further extended by using standard web technologies such as HTML, JavaScript, and Cascading Style Sheets (CSS).

You might also encounter situations where you need to update or create data in Common Data Service without submitting an entity form or a web form. A common method is to build a standalone web application with an API that can be called from the Power Apps portal and that will act as a liaison to update Common Data Service.

After you have configured and customized your portal, another concern that you might have is determining how to ensure that your work is saved in a source control system and how to ensure that your portal can be deployed to a test or production portal.

## Server-side extensibility

Power Apps portals do not support client-side business rules, JavaScript web resource, or Power Apps component framework controls that are commonplace with the model-driven forms. As a result, portal deployments will occasionally become blocked because expectations of the same or similar form behavior can't be met. Instead, entity lists, entity forms, and web forms include a custom JavaScript option that allows developers to add scripts that implement equivalent functionality.

However, the portals are based on model-driven apps that are underpinned by Common Data Service. In fact, portals deliver functionality that is already available in a model-driven app but only to the portal users, meaning that:

- Entity-scope business rules still apply because they are implemented on the server.
- Workflows and Microsoft Power Automate flows are still triggered, regardless of whether a triggering action was run inside a model-driven app or a portal. The workflows can also be called explicitly by using form configuration on the portal.
- When a portal user interacts with Common Data Service records, the server-side code runs as usual. Developers can pass relevant portal context to plug-ins when the records are updated by portal users who are employing entity forms. That approach adds some server-side code extensibility to the portals and enables various integration scenarios.

## Development life cycle

Application lifecycle management (ALM) is important as the applications that your organization builds become more complex and as more of your company depends on their stability.

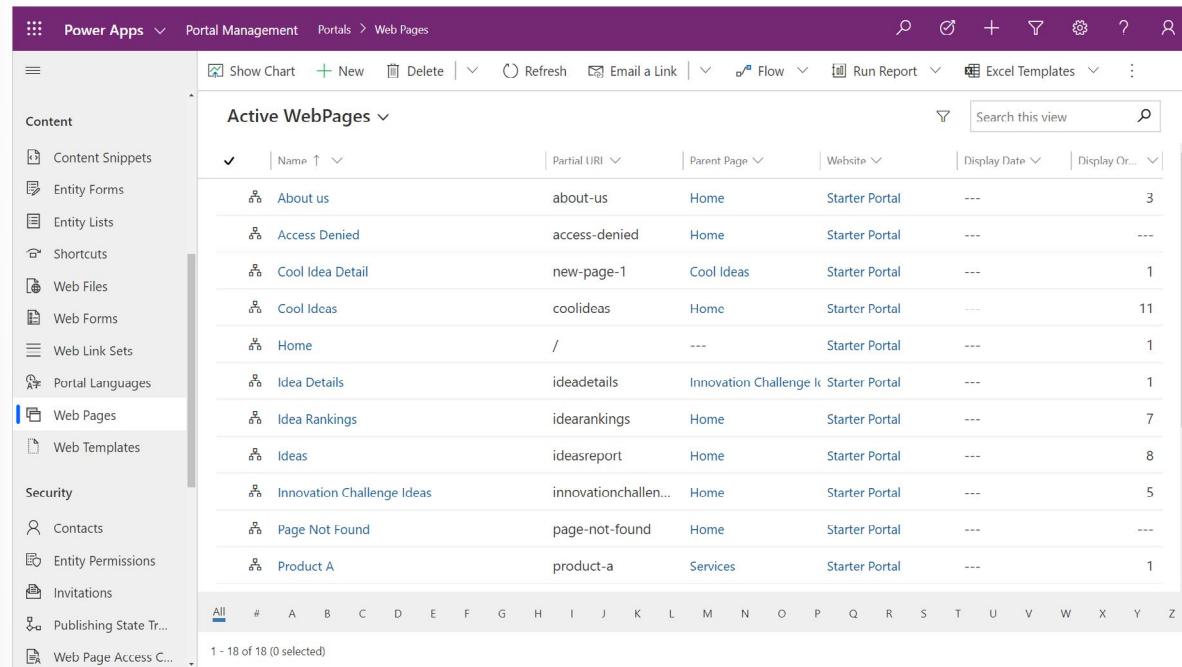
If you create a model-driven Power App, a Power Automate flow, or any other object to build a business solution on a Common Data Service environment, you would typically contain these assets in a *solution* and export the single solution file from one environment to another. The solution file can also be *un-*

packed into discreet file components and uploaded to a source control system (such as Microsoft Azure DevOps) that will maintain a repository of the solution and track the various changes. Many tools are provided by Microsoft and the community to automate the ALM process.

For more information, see **Application lifecycle management (ALM) with Microsoft Power Platform<sup>13</sup>**.

## Portal metadata

Power Apps portals configuration is stored directly within Common Data Service as records known as *portal metadata*. When you provision a Power Apps portal, the portal template that you choose will populate a number of portal-specific Common Data Service entities with data that describe the website structure, webpages, content, entity list configuration, entity form configuration, and so on. As you configure the portal by using Power Apps portals Studio or other tools, you are changing or adding to the portal metadata.



The screenshot shows the Power Apps Portal Management interface. The left sidebar lists categories: Content (Content Snippets, Entity Forms, Entity Lists, Shortcuts, Web Files, Web Forms, Web Link Sets, Portal Languages, Web Pages, Web Templates), Security (Contacts, Entity Permissions, Invitations), and Publishing State Tr... (Web Page Access C...). The main area is titled "Active WebPages" and displays a list of webpage records. The columns are: Name, Partial URI, Parent Page, Website, Display Date, and Display Order. The list includes entries like "About us", "Access Denied", "Cool Idea Detail", "Cool Ideas", "Home", "Idea Details", "Idea Rankings", "Ideas", "Innovation Challenge Ideas", "Page Not Found", and "Product A". At the bottom, there are navigation links for All, #, A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z, and a note indicating 1 - 18 of 18 (0 selected).

Portal metadata can't be added to a Common Data Service solution file. The challenge is how to move the portal metadata records from one environment to another. While model-driven Power Apps have built-in features to export and import data, these features work for a single entity and must be sequenced based on data relationships.

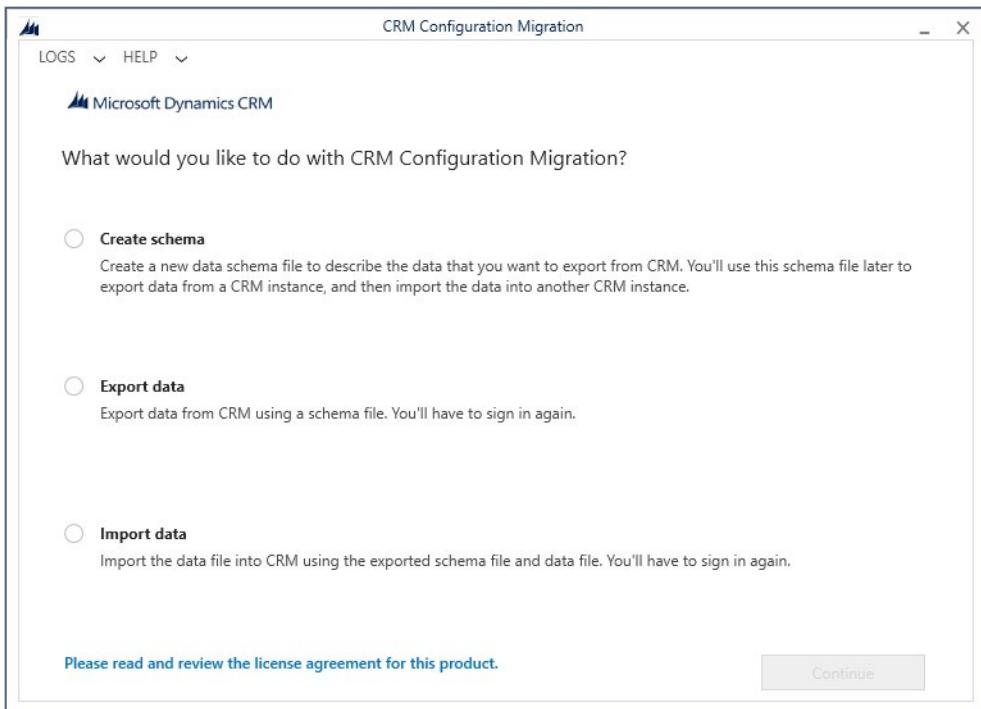
Portal metadata is stored in over 50 interconnected, unique entities. In addition, functionality is built into the portal solutions that will be triggered as records are created. For example, as a webpage record is added to the portal, a corresponding webpage record will be created for each provisioned language. Using an import tool might inadvertently duplicate webpage records.

You can move portal metadata from one environment to another by using tools that are available from the community and Microsoft.

<sup>13</sup> <https://docs.microsoft.com/power-platform/alm/?azure-portal=true>

## Configuration Migration tool

The **Configuration Migration** tool is an application that can be run as a Windows desktop application or invoked as a PowerShell cmdlet. The tool can export and import a set of Common Data Service records and maintain the entity relationships.



[!NOTE]

The **Configuration Migration** tool is designed for smaller datasets (specifically configuration data, such as portal metadata) and is not suited for high-volume data migrations or integrations.

The **Configuration Migration** tool will use a schema file to determine which Common Data Service entities to export to a data file. Microsoft has a predefined schema file for each of the portal templates to identify the specific portal metadata Common Data Service entities.

After you have set up your Power Apps portal configuration, you can run the **Configuration Migration** tool to export your portal metadata to an XML file by using the schema that corresponds to your portal template.

Ideally, the portal metadata file should be uploaded to a source control repository, where the changes can be tracked and used to deploy to other environments.

You will also need to deploy any configurations that are made to Common Data Service, such as new entities and model-driven forms and views that are used by portal entity lists and entity forms. You would move this configuration by using *solutions*. For more information, see **Solutions Overview**<sup>14</sup>.

On your destination environment, you can then run the **Configuration Migration** tool to import the portal metadata file to update the portal configuration.

[!CAUTION]

Using the **Configuration Migration** tool will move over all and overwrite existing portal metadata. Situations might occur where content is specific to the environment (for example, hyperlinks to specific

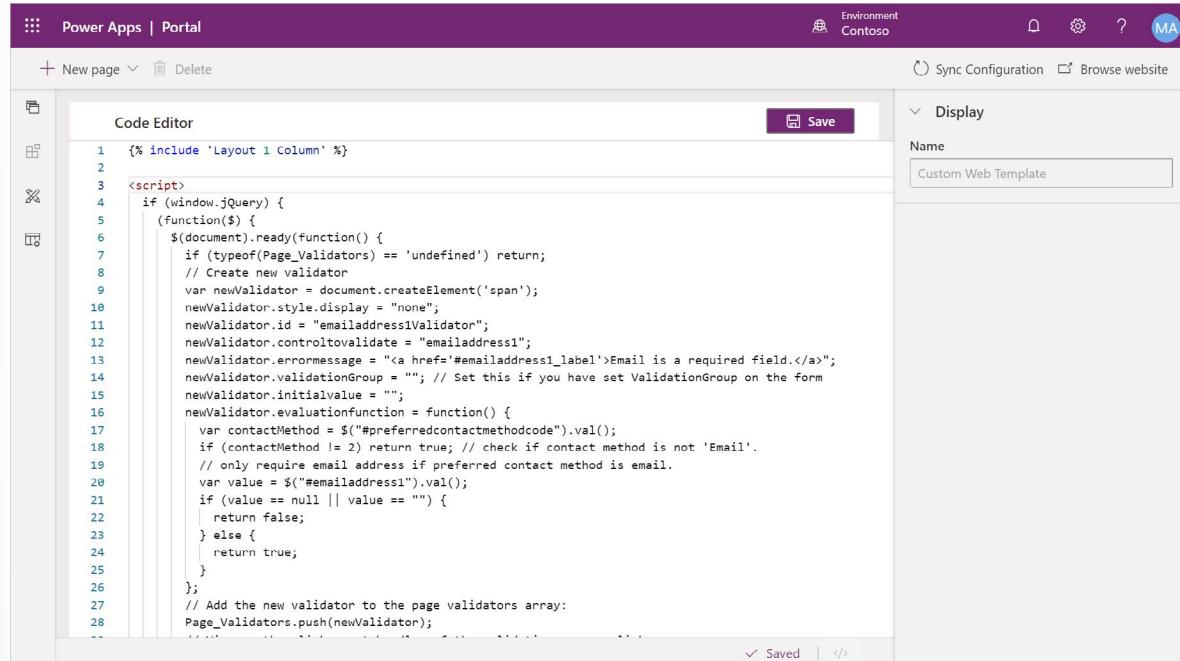
<sup>14</sup> <https://docs.microsoft.com/powerapps/maker/common-data-service/solutions-overview/?azure-portal=true>

URLs). After the data has been migrated, you might need to update some content directly in the destination portal.

For more information, see [Migrate portal configuration<sup>15</sup>](#).

## Extend with scripts

Power Apps portals does not support client-side business rules or custom JavaScript processes that are available on model-driven forms. However, custom JavaScript can be added directly to a portal page in the Power Apps portals Studio source code editor and to web templates.



```
1  <% include 'Layout 1 Column' %>
2
3  <script>
4    if (window.jQuery) {
5      (function($) {
6        $(document).ready(function() {
7          if (typeof(Page_Validators) == 'undefined') return;
8          // Create new validator
9          var newValidator = document.createElement('span');
10         newValidator.style.display = "none";
11         newValidator.id = "emailaddress1Validator";
12         newValidator.controltovalidate = "emailaddress1";
13         newValidator.errormessage = "<a href='#emailaddress1_label'>Email is a required field.</a>";
14         newValidator.validationGroup = ""; // Set this if you have set ValidationGroup on the form
15         newValidator.initialvalue = "";
16         newValidator.evaluationfunction = function() {
17           var contactMethod = $("#preferredcontactmethodcode").val();
18           if (contactMethod != 2) return true; // check if contact method is not 'Email'.
19           // only require email address if preferred contact method is email.
20           var value = $("#emailaddress1").val();
21           if (value == null || value == "") {
22             return false;
23           } else {
24             return true;
25           }
26         };
27         // Add the new validator to the page validators array:
28         Page_Validators.push(newValidator);
29       })(jQuery);
30     }
31   
```

In addition, the **Entity Lists**, **Entity Forms**, and **Web Forms** features include a custom JavaScript field that allows developers to add scripts that implement the required functionality.

The functionality can range from hiding controls, to replacing input controls with user-friendly equivalents, to invoking external web services and implementing sophisticated integration scenarios.

Out of the box, Power Apps portals includes the **jQuery<sup>16</sup>** library that makes manipulation of page content and appearance a reasonably simple task.

## Controls and fields

A default **Contact Us** form that creates a lead in Common Data Service includes a mandatory **Topic** field. This field could be confusing to site visitors. You can't use CSS to hide the control because it leaves the red asterisk that indicates that the field is mandatory. JavaScript helps make your job of hiding the field relatively simple. To set this field to a predefined value and then hide it from view, go to portals Studio and add the following script to the source code of the page that contains the **Contact Us** form:

```
<script>
$(document).ready
(
```

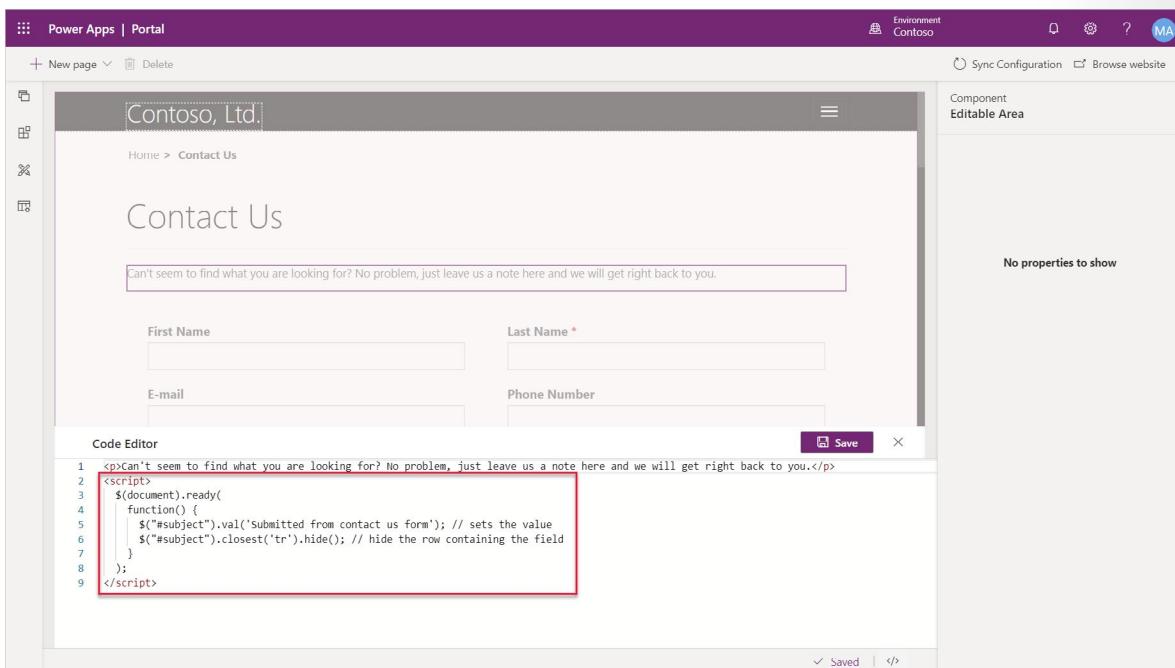
<sup>15</sup> <https://docs.microsoft.com/powerapps/maker/portals/admin/migrate-portal-configuration/?azure-portal=true>

<sup>16</sup> <https://jquery.com/?azure-portal=true>

```

function()
{
    $("#subject").val('Submitted from contact us form'); // sets the value
    $("#subject").closest('tr').hide(); // hide the row containing the field
}
);
</script>

```



JavaScript is a simple way to add supplementary behavior or functionality to your Power Apps portal. It can be added directly to a page or added to the **Entity Form** configuration (through the Portal Management app) or as part of a web template, depending on your scoping needs. Script can be used on the form to:

- Implement business rules.
- Add dependencies between elements like dependent option sets.
- Perform additional form validation.

These options provide reasonable alternatives to client-side business rules and scripting in model-driven forms.



<https://www.microsoft.com/videoplayer/embed/RE4AmL6>

For more information, see **Add custom JavaScript<sup>17</sup>**.

<sup>17</sup> <https://docs.microsoft.com/powerapps/maker/portals/configure/add-custom-javascript?azure-portal=true>

## Advanced CSS

Cascading Style Sheets (CSS) is a language that determines the style of a webpage by describing how its HTML elements are to be displayed, including text, fonts, colors, backgrounds borders, and margins.

Making changes to the style of your portal pages can be as simple as applying CSS statements directly to a page in the **Custom CSS** field. For example, if you need to increase the height of the navigation bar at the top of a home page to fit a logo, you can edit the page properties and add a custom CSS statement.

```
.navbar-static-top.navbar { min-height: 100px; }
```

[!NOTE]

CSS statements that are added directly to a webpage will apply to that webpage only.

This approach works for small adjustments on a single page, but a better approach is to record customizations in one or more CSS files and then apply them to the entire portal or parts of it.

## Portal themes

Portals use the **Bootstrap front-end framework**<sup>18</sup> to control the design and layout of the website. Bootstrap is a package of HTML and CSS design templates for typography, forms, buttons, navigation, and other elements, including optional JavaScript extensions. One appealing feature of Bootstrap is that it offers responsive layout out of the box; it automatically adjusts your website so that it has a pleasing appearance on all devices from small phones to large desktops.

A theme determines the appearance of all webpages in your portal to ensure visual consistency. It controls the navigational structure, the banner, colors and fonts, and other visual elements of a webpage.

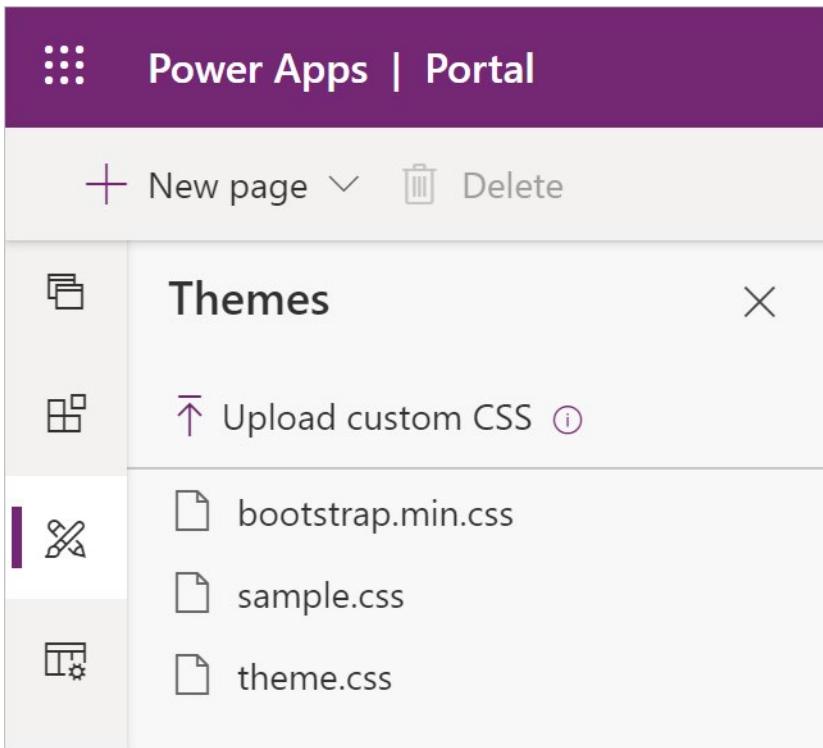
The web templates that are included in a starter portal are implemented by using standard Bootstrap components with minimal additional custom styles. As a result, you can take advantage of the customization options that are provided by Bootstrap to tailor the theme in a way that's applied consistently to the entire portal.

## Upload custom CSS

Any starter portal has two files already included as child web files of the home page: `bootstrap.min.css` and `theme.css`. These files define the default styles and theme for your entire portal. You can upload and edit additional stylesheets in portals Studio by using the **Themes** section on the tool belt.

---

<sup>18</sup> <http://getbootstrap.com/?azure-portal=true>



Make sure that you consider how you want to approach style modifications for your portal, such as:

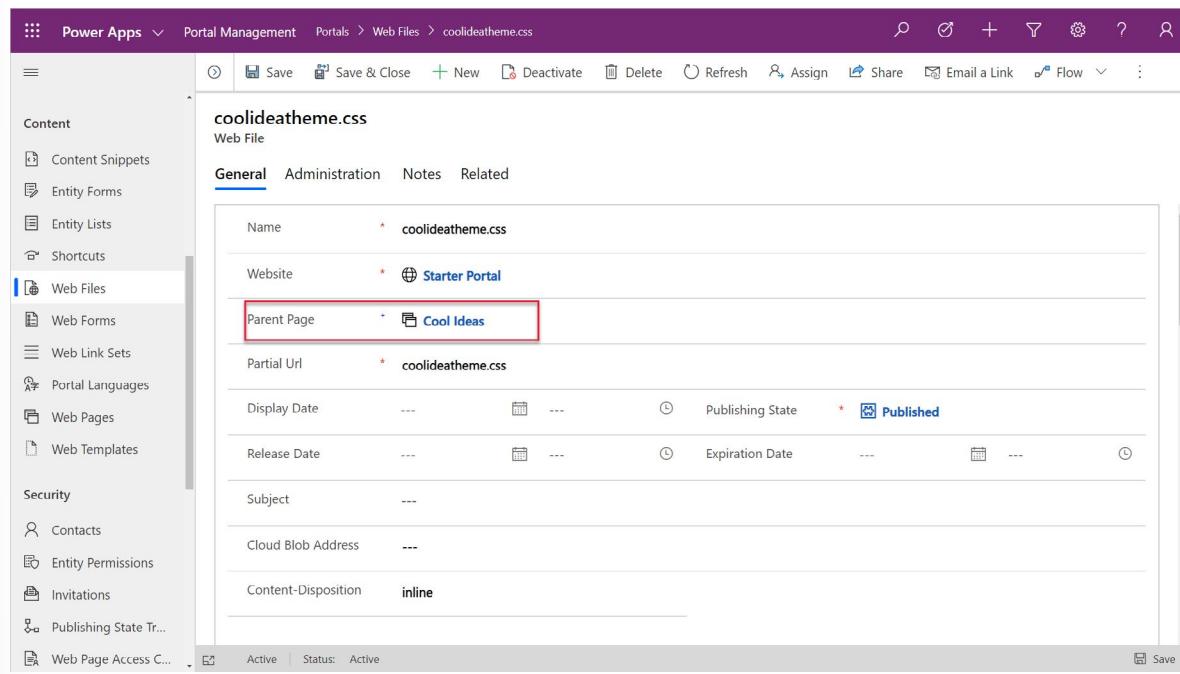
- Creating complete styling for the entire site and then replacing the content of the .css file. This method works well if you have access to skilled designers who can ensure that all relevant elements are defined. This approach creates centrally controlled styling and ensures consistency throughout the portal.
- Redefining only the elements that require modifications, such as colors and font size. Create and upload the CSS file that contains only these incremental adjustments. This method works well if your target design is close to the starter portal design and only minor styling modifications are required. This approach allows incremental modifications that can be undone.

[!WARNING]

If you decide to overwrite `bootstrap.min.css` or `theme.css` files, make sure that you download a backup copy of these files prior to replacing them. If your replacement CSS is invalid or incomplete, you will not be able to undo the replacement. Consequently, you will have to restore the content of these files, potentially using a Power Apps portals app if the portal is rendered non-functional.

## Apply customizations to specific portal areas

You can add customizations to specific pages or sections of your portal by adding a **Web File** record by using the Portal Management app with an attachment to a CSS file. You will need to specify the parent page in the **Web File** record so that the theme will be applied to the parent page and all descendants of that page. This approach makes it possible for you to build fully customized sections of your site.

**[!IMPORTANT]**

The partial URL must end in **.css** for the portal to recognize it and apply it to the webpage and its child pages.

## Customize Bootstrap

The standard way to create a custom version of Bootstrap is **through the official Bootstrap site<sup>19</sup>**. However, due to the popularity of Bootstrap, many other sites have also been created for this purpose. These sites might provide a user-friendly interface for Bootstrap customization or predesigned versions of Bootstrap for you to download. **The official Bootstrap customizer<sup>20</sup>** site has more information about Bootstrap customization.

**[!TIP]**

When you are customizing Bootstrap, only select elements that require modification. For example, if you want to replace the standard fonts with your corporate standard fonts, select the **Typography** component of Bootstrap. This approach will help reduce the chances of accidentally rewriting other CSS elements.

After you have customized Bootstrap, it will generate one or more files that you would want to upload as web files. Unless your intent is to completely *replace* the original styles, make sure that you avoid using `bootstrap.css`, `bootstrap.min.css`, `theme.css`, or `theme.min.css` files in your partial URLs because of how the portal handles multiple CSS files.

## Background images

When you are starting portal customizations, one of the most common requests that you might have is to replace the background images. These images are applied by using CSS, but they can be replaced without changing CSS. Look for `.jpg` web files under the portal's home page, for example `homehero`.

<sup>19</sup> <http://getbootstrap.com/customize/#less-variables/?azure-portal=true>

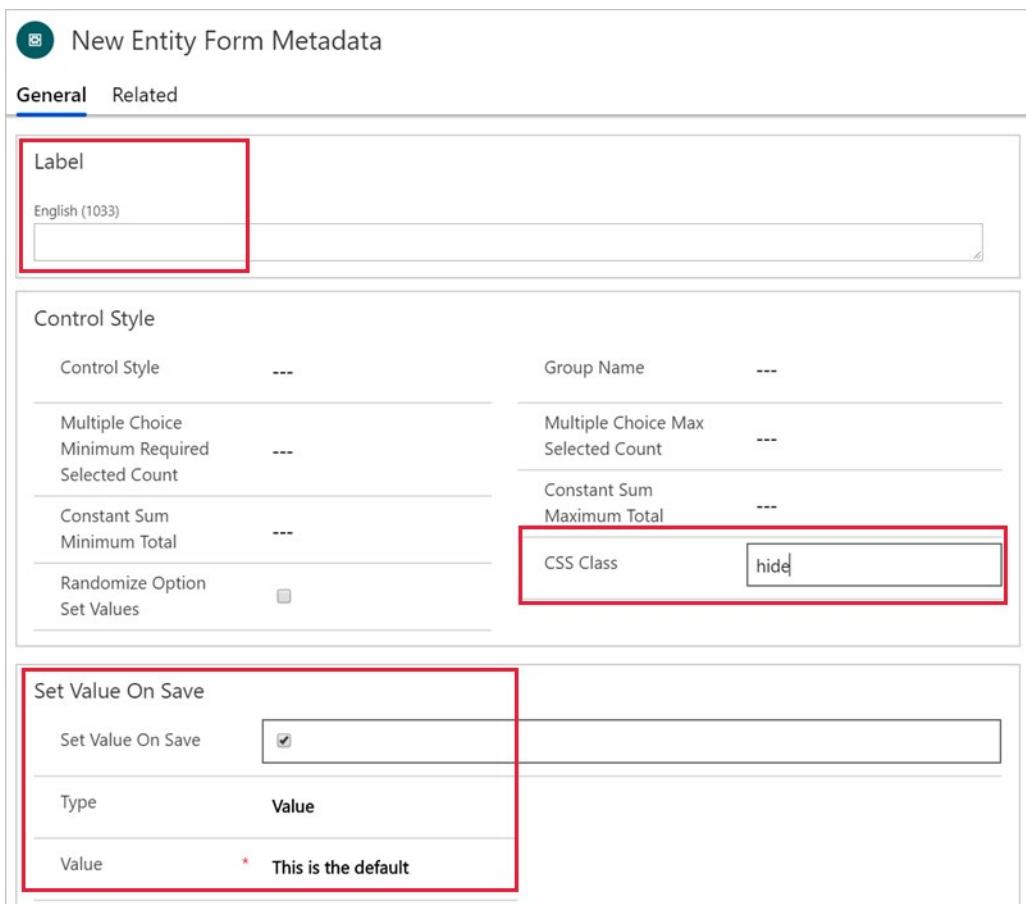
<sup>20</sup> <http://getbootstrap.com/customize/?azure-portal=true>

.jpg. Your only requirement is to replace the attachments of these web files with your own images. Make sure that the size of the new images is compatible to maintain consistent layout.

## CSS for simple adjustments

CSS is often overlooked when simple adjustments are required, such as hiding the element or adjusting margins to fit a wide label. Bootstrap framework is the foundation of the styling in Power Apps portals, and it defines many styles that control appearance and multi-column responsive layout. For example, if you want to hide a particular field and submit a default value instead of collecting the user input, you can use **Entity Form Metadata** for the field and then define the following parameters:

- Set **Label** to a single space.
- Set **CSS Class** to **hide**.
- Set the default value as required.



Instead of resorting to JavaScript, you can accomplish this task by using configuration only. Using CSS, where possible, has certain advantages:

- CSS applies on load; it's faster because you don't have to wait for the document to finish loading.
- CSS is always on and continues to apply to the relevant elements, even if JavaScript changes the document structure and adds or removes the elements.
- For certain tasks that are related to an element's visibility and responsive layout, CSS can be simpler to use than JavaScript, meaning less maintenance and less room for error.

# Advanced client-side development

Using JavaScript and CSS to manipulate client-side visibility and functionality of portal pages can help you achieve considerable success in satisfying some key business requirements. To satisfy more complex scenarios, a developer can be creative and use other strategies to extend Power Apps portals.

## Partner libraries

JavaScript can use other JavaScript libraries that deliver functionality such as UX enhancements (masked controls, for example), real-time communications (SignalR), sophisticated UI building (Angular, Vue, React), and other various business services like address validations, map API, routing services, logistics, and so on.

Power Apps portals create a clean, responsive layout with predictable element names, which helps make manipulating the data and UI easier.

For an example of a sophisticated implementation that can be hosted in Power Apps portals and that uses Angular framework for communications, go to the **Set up the event website<sup>21</sup>** documentation for Dynamics 365 Marketing.

## API calls

Combined with custom web templates that use Liquid code to deliver *data* instead of HTML, JavaScript can call *back* into Power Apps portals to retrieve the data and then use it to create alternative UX. For example, instead of using the **Entity List** calendar view, you can retrieve list data in JSON format instead and then use other calendaring libraries, like FullCalendar, to create sophisticated calendar visualization. Alternatively, you can retrieve opportunity data and use another charting JavaScript library for sophisticated interactive charting.

[!IMPORTANT]

When you use web templates, it is only possible to implement REST service for GET operations. No other operation, such as PUT, is supported by Liquid code.

## Companion app

Situations might occur where you want to communicate securely with external services while maintaining the security context, such as when you are processing online payments. Power Apps portals enables this scenario by providing support for **OAuth 2.0 implicit grant flow within your portal<sup>22</sup>**.

This feature allows a customer to make client-side calls to external APIs and secure them by using OAuth implicit grant flow. This method helps ensure that the identity information of a signed-in user is passed in a secured manner to the external calls.

In this scenario, you build a custom web application and Power Apps portals would communicate to this application by using JavaScript to call the API.

CSS and JavaScript enable a number of integration and extensibility scenarios that range from simple UI adjustments, to validation and data input, to sophisticated client-side applications that interact with other services.

---

<sup>21</sup> <https://docs.microsoft.com/dynamics365/marketing/set-up-event-portal/?azure-portal=true>

<sup>22</sup> <https://docs.microsoft.com/powerapps/maker/portals/oauth-implicit-grant-flow/?azure-portal=true>

# Exercise-Add advanced client-side functionality

The purpose of this hands-on lab is to demonstrate how to add JavaScript code to a page to render data from Common Data Service as a chart by using another charting library. The data has been retrieved from Common Data Service by using a web template that acts as a REST API endpoint.

The exercises work best when you have sample data to work with. Depending on the environment that you are working with, you might want to install some sample data to assist with the exercises. Power Platform does provide the ability to add sample data as needed. If the environment that you are working in doesn't have sample data installed, follow the steps in the **Add or remove sample data**<sup>23</sup> documentation to install the sample data into your environment.

## Learning objectives

At the end of these exercises, you will be able to:

- Build a webpage that acts as a REST endpoint that returns data from Common Data Service.
- Add inline code to a content webpage to retrieve the data by using the endpoint.
- Use an external JavaScript library to consume the retrieved data.

**Estimated time to complete this exercise:** 15 to 20 minutes

## Prerequisites

For this exercise, make sure that the following parameters are set up in your environment:

- A Power Apps portal that is provisioned. If you do not have a Power Apps portal available, follow the **Create Portal**<sup>24</sup> instructions to create one.
- Access to the Power Apps maker portal.

## High-level steps

To finish the exercise, complete the following tasks:

1. Create a web template with Liquid code to retrieve data about accounts in Common Data Service and then return the data in JSON format.
2. Add **Page Template** and **Web Page** records that use the web template that you created.
3. Open a content page and add JavaScript code that retrieves the data.
4. Add a charting library to the page and JavaScript code by using the library to build a graph with the retrieved data.

## Create a web template

To create a web template, follow these steps:

1. Open **Dynamics 365 Home**<sup>25</sup>.
2. Select the Portals Management app.

<sup>23</sup> <https://docs.microsoft.com/power-platform/admin/add-remove-sample-data/?azure-portal=true>

<sup>24</sup> <https://docs.microsoft.com/powerapps/maker/portals/create-portal>

<sup>25</sup> <https://home.dynamics.com/?azure-portal=true>

3. Select **Web Templates**.

4. Select **+ New**.

5. Enter the following values:

- **Name** - getAccounts
- **Website** - Select your current website
- **Source** - Enter the following content
- **MIME Type** - application/json

```
{% fetchxml accounts %}  
<fetch>  
  <entity name="account">  
    <attribute name="name" />  
    <attribute name="numberofemployees" />  
    <attribute name="revenue" />  
  </entity>  
</fetch>  
{% endfetchxml %}  
[  
{% for account in accounts.results.entities -%}  
{  
  "x": {{ account.numberofemployees }},  
  "y": {{ account.revenue }},  
  "z": {{ account.revenue | divided_by: account.numberofemployees }},  
  "name": "{{ account.name }}"  
}{% unless forloop.last %},{% endunless %}  
{% endfor -%}  
]
```

6. Press **Save & Close**.

This Liquid code retrieves the list of accounts and then generates a data structure in JSON format. The data structure is already prepared for plotting by assigning appropriate labels to data points:

- **name** - Company name
- **x** - Number of employees
- **y** - Company revenue in thousands
- **z** - Revenue for each employee (calculated)

## Create a page template and a webpage

To create a page template and a webpage, follow these steps:

1. Select **Page Templates**.

2. Select **+ New**.

3. Enter the following values:

- **Name** - getAccounts

- **Website** - Select your current website
  - **Type** - Select **Web Template**
  - **Web Template** - Select **getAccounts**
  - **Use Website Header and Footer** - Clear the check box
4. Select **Save & Close**.
  5. Select **Web Pages**.
  6. Select **+ New**.
  7. Enter the following values:
    - **Name** - getAccounts
    - **Website** - Select your current website
    - **Parent Page** - Select **Home**
    - **Partial URL** - getAccounts
    - **Page Template** - getAccounts
    - **Publishing State** - Published
  8. Select **Save & Close**.

[!!IMPORTANT]

If you have not previously configured entity permissions for the account entity, your API page will return an empty array. Complete the next task to set up the permissions if you have not done so previously.

## Add entity permissions

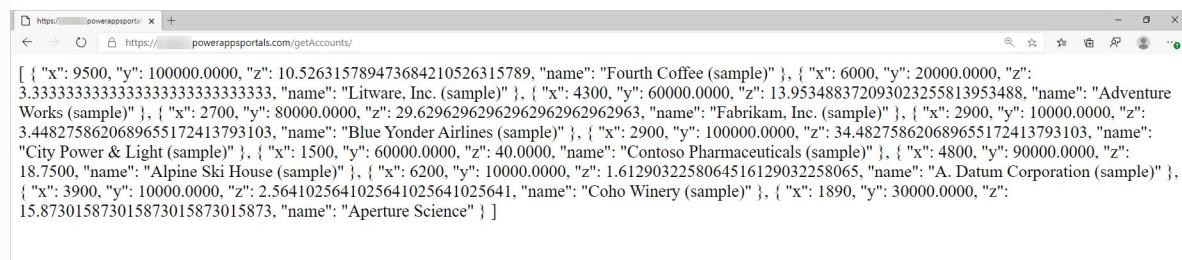
To add entity permissions, follow these steps:

1. Switch to the Portal Management app.
2. Select **Entity Permissions**.
3. Select **+ New**.
4. Enter the following values:
  - **Name** - Account Directory
  - **Entity Name** - Select account entity
  - **Website** - Select your current website
  - **Scope** - Select **Global**
  - **Privileges** - Select **Read**
5. Select **Save**.
6. Scroll to the **Web Roles** subgrid.
7. Select **Add Existing Web Role**.
8. Locate and select **Anonymous users** and **Authenticated users**.
9. Select **Add**.

## Test the REST webpage

Go to <https://yourportal.powerappspartials.com/getAccounts>.

Your output should look like the following example:



The screenshot shows a browser window with the URL <https://powerappspartials.com/getAccounts/>. The page displays a JSON array of objects representing accounts. Each object has properties: x, y, z, name, and a detailed description. The names listed include "Fourth Coffee (sample)", "Adventure Works (sample)", "Litware, Inc. (sample)", "Fabrikam, Inc. (sample)", "Blue Yonder Airlines (sample)", "City Power & Light (sample)", "Alpine Ski House (sample)", "A. Datum Corporation (sample)", "Coho Winery (sample)", and "Aperture Science". The JSON array is as follows:

```
[{"x": 9500, "y": 10000.0000, "z": 10.526315789473684210526315789, "name": "Fourth Coffee (sample)"}, {"x": 6000, "y": 20000.0000, "z": 3.33333333333333333333333333333333, "name": "Litware, Inc. (sample)"}, {"x": 4300, "y": 60000.0000, "z": 13.953488372093023255813953488, "name": "Adventure Works (sample)"}, {"x": 2700, "y": 80000.0000, "z": 29.62962962962962962962962962962963, "name": "Fabrikam, Inc. (sample)"}, {"x": 2900, "y": 100000.0000, "z": 3.4482758620689655172413793103, "name": "Blue Yonder Airlines (sample)"}, {"x": 2900, "y": 100000.0000, "z": 34.482758620689655172413793103, "name": "City Power & Light (sample)"}, {"x": 1500, "y": 60000.0000, "z": 40.0000, "name": "Contoso Pharmaceuticals (sample)"}, {"x": 4800, "y": 90000.0000, "z": 18.7500, "name": "Alpine Ski House (sample)"}, {"x": 6200, "y": 10000.0000, "z": 1.6129032258064516129032258065, "name": "A. Datum Corporation (sample)"}, {"x": 3900, "y": 10000.0000, "z": 2.5641025641025641025641025641, "name": "Coho Winery (sample)"}, {"x": 1890, "y": 30000.0000, "z": 15.873015873015873015873015873, "name": "Aperture Science"}]
```

## Add code to retrieve the data

To add code to retrieve the data, follow these steps:

1. Open Power Apps portals Studio in a new browser tab and then follow these steps:
  1. Go to **Power Apps maker portal**<sup>26</sup>.
  2. Select the target environment by using the environment selector in the upper-right corner.
  3. From the **Apps** list, select the application of type **Portal**.
  4. Select the **Edit** menu.
2. Select the **Pages** icon on the tool belt on the left side.
3. Select an existing page from the hierarchy, for example **Product B** located under the **Services** page.

[!NOTE]

The names and hierarchy of pages on your portal might differ.

4. Select the **Page Copy** area on the page.
5. Select **Components** on the tool belt.
6. Select **One-column section**.
7. Select **Added section** and then select the **source code editor** icon.
8. Insert the following code as the content of the innermost **div** element:

```
<script>
function makeChart(rData) {
    console.log(rData);
}

$(document).ready(function() {
    $.get('/getAccounts', makeChart, 'json');
});
</script>
```

9. Select **Save**.

<sup>26</sup> <https://make.powerapps.com/?azure-portal=true>

10. Select **Browse website**.

11. When the page is displayed, press the **F12** key to display browser developer tools.

12. Verify that the console output contains the same data as previously retrieved by the REST API page.

```
▼ (11) [{}], [{}], [{}], [{}], [{}], [{}], [{}], [{}], [{}], [{}], [{}]
  ► 0: {x: 9500, y: 100000, z: 10.526315789473685, name: "Fourth Coffee (sample)"}
  ► 1: {x: 6000, y: 20000, z: 3.3333333333333335, name: "Litware, Inc. (sample)"}
  ► 2: {x: 4300, y: 60000, z: 13.953488372093023, name: "Adventure Works (sample)"}
  ► 3: {x: 2700, y: 80000, z: 29.62962962962963, name: "Fabrikam, Inc. (sample)"}
  ► 4: {x: 2900, y: 10000, z: 3.4482758620689653, name: "Blue Yonder Airlines (sample)"}
  ► 5: {x: 2900, y: 100000, z: 34.48275862068966, name: "City Power & Light (sample)"}
  ► 6: {x: 1500, y: 60000, z: 40, name: "Contoso Pharmaceuticals (sample)"}
  ► 7: {x: 4800, y: 90000, z: 18.75, name: "Alpine Ski House (sample)"}
  ► 8: {x: 6200, y: 10000, z: 1.6129032258064515, name: "A. Datum Corporation (sample)"}
  ► 9: {x: 3900, y: 10000, z: 2.5641025641025643, name: "Coho Winery (sample)"}
  ► 10: {x: 1890, y: 30000, z: 15.873015873015873, name: "Aperture Science"}
    length: 11
  __proto__: Array(0)
```

[!IMPORTANT]

If you have not previously configured entity permissions for the account entity, your API call will return an empty array. Make sure that you have completed the **Add entity permissions** task.

## Add external library functionality

This exercise uses Highcharts.js library (free for personal or non-profit use) to create a bubble chart based on the data.

1. Switch to portals Studio.
2. Locate and open the content page that you previously modified.
3. Select the section that you previously modified.
4. Insert the following code either above or below the previous code:

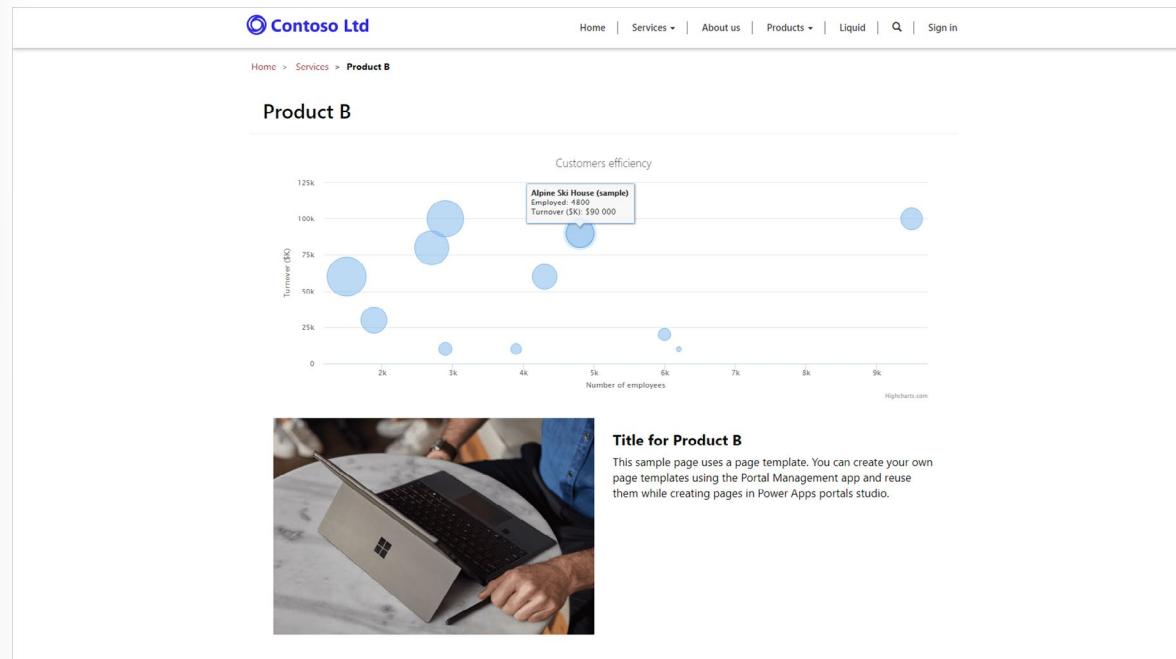
```
<script src="https://code.highcharts.com/highcharts.js"></script>
<script src="https://code.highcharts.com/highcharts-more.js"></script>
<figure>
  <div class="mychart"></div>
</figure>
```

5. Modify the **makeChart** function as follows:

```
function makeChart(rData) {
  console.log(rData);
  Highcharts.chart($('.mychart')[0], {
    title: {
      text: "Customers efficiency"
    },
    legend: {
      enabled: false
    },
    xAxis: {
      title: {
```

```
        text: "Number of employees"
    }
},
yAxis: {
    title: {
        text: "Turnover ($K)"
    }
},
tooltip: {
    pointFormat: '<strong>{point.name}</strong><br/>Employed: {point.x}<br>Turnover ($K): ${point.y}',
    headerFormat: ""
},
series: [
{
    type: 'bubble',
    data: rData
}]
});
}
}
```

6. Select **Save**.
7. Select **Browse website**.
8. The output should now include the bubble chart. Hover over the bubbles to verify the data.



## Summary

This module discussed extending portal functionality by using software development, and it examined application lifecycle management (ALM) techniques in portal deployments.

By now, you should be able to:

- Explain where software development can resolve particular portal feature requirements.
- Employ portal ALM for Power Apps portals.
- Apply JavaScript code to portal assets.
- Use CSS to address specific portal development requirements.

## References

**Power Apps portals technical documentation**<sup>27</sup>

**Power Apps portals Community**<sup>28</sup>

**XrmToolBox Portal Records Mover**<sup>29</sup> - Configuration Migration tool alternative

**XrmToolBox Portal Code Editor**<sup>30</sup> - Edit portal components from the desktop without the Power Apps Management app

---

<sup>27</sup> <https://docs.microsoft.com/powerapps/maker/portals/overview/?azure-portal=true>

<sup>28</sup> <https://powerusers.microsoft.com/t5/Power-Apps-Portals/bd-p/PowerAppsPortals/?azure-portal=true>

<sup>29</sup> <https://www.xrmtoolbox.com/plugins/MscrmTools.PortalRecordsMover/?azure-portal=true>

<sup>30</sup> <https://www.xrmtoolbox.com/plugins/MscrmTools.PortalCodeEditor/?azure-portal=true>

# Build custom Power Apps portals web templates

## Introduction

Liquid is an **open-source template language**<sup>31</sup> that can be used to add dynamic content to pages. Liquid code can be used anywhere in the portals where HTML or text content can be entered, including content in webpages, content snippets, and web templates.

Liquid has various language constructs that can help you build content, apply transformations, and control the implementation flow. Ability to access Common Data Service data by using FetchXML query language or directly retrieving entities by identifiers makes Liquid the primary choice for building data-driven webpages. It also includes a number of special tags (or processing instructions) that specifically target the building of reusable web templates. The following sections explain how templates are used in Microsoft Power Apps portals.

## Templates

In Power Apps portals, the webpage does not define how the page looks when it's rendered on the portal. Instead, it's linked to the **Page Template** record that defines the layout and behavior.

The two types of templates that you can use in Power Apps portals are:

- **Rewrite** - These templates use server-side processing to implement specialized behavior that is required by some of the components and special pages such as an error page, site map, search, and others.
- **Web Template** - The linked layout template defines how the content of the page is rendered for output.

**Rewrite** templates offer limited customizations. Liquid template language can be used as part of the content or inside content snippets, but the page layout and behavior are predefined.

Contrarily, the **Web Template** option is entirely template-based and uses Liquid to define how the content is rendered. Templates are flexible. A template can include other templates for parts of the content. A template can also be based on another template, extending the base functionality. The real power of web templates comes from the ability to contain Liquid code that adds processing capabilities to the static content, including access to Common Data Service data.

[!TIP]

Web templates can also contain HTML, Cascading Style Sheets (CSS), and JavaScript.

You can use web templates to define an entire webpage, a part of a page, or common elements such as the site header and footer. This approach creates a consistent appearance and behavior throughout the portal and helps make it easier to modify the appearance and data that is rendered. The following example shows what a typical simple template might look like:

```
<div class="container">
  <div class="page-heading">
    { % include 'Breadcrumbs' % }
    { % include 'Page Header' % }
  </div>
  <div class="row">
```

---

<sup>31</sup> <http://dotliquidmarkup.org/?azure-portal=true>

```

<div class="col-md-12">
{%
  block main %
  {%
    include 'Page Copy' %

    {%
      if page.adx_entitylist %
        {%
          include 'entity_list' key:page.adx_entitylist.id %
        }
      endif %

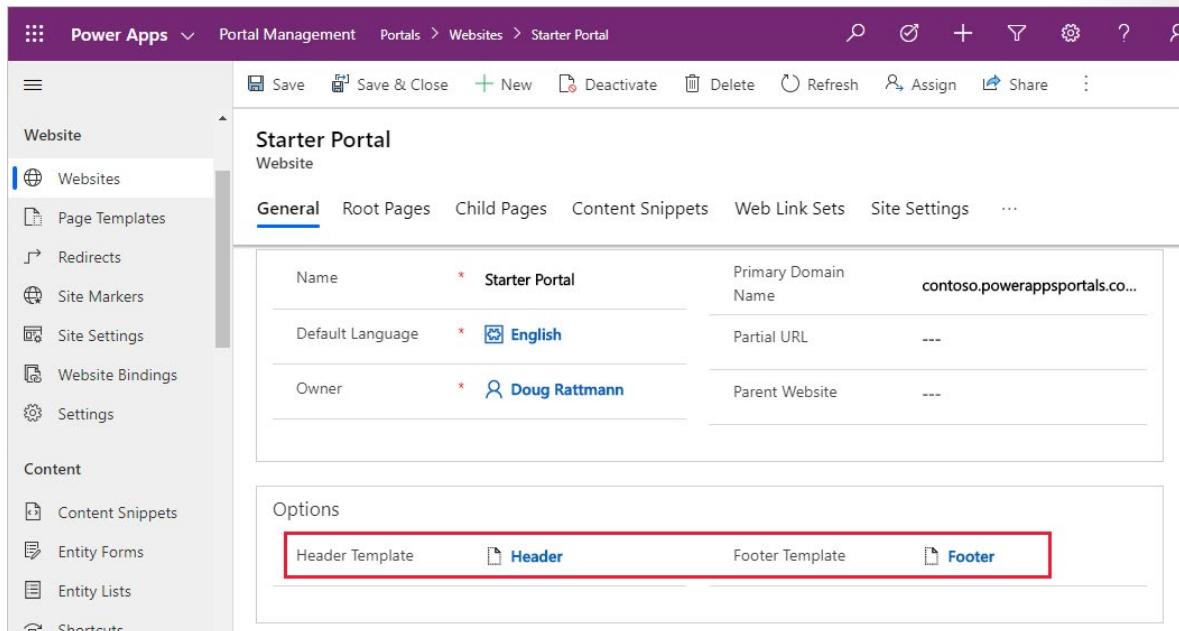
      {%
        if page.adx_entityform %
          {%
            entityform id: page.adx_entityform.id %
          }
        endif %

        {%
          endblock %
        }
      
```

This simple template includes other templates to render common bits and pieces, it defines content block that can be rendered differently by a derived template, and it renders Common Data Service data by using related **Entity List** and **Entity Form** records.

## Web templates as website headers and footers

Web templates can be used to override the global header and footer that is used by a Power Apps portal. To accomplish this task, set the **Header Template** or **Footer Template** field of your website to the web template of your choice.



If you override the **Header Template** field, your selected template assumes responsibility for rendering the primary navigation, sign-in/sign-out links, search interface, and so on, for your site interface elements that are normally handled by the default header template.

## [!TIP]

If you don't specify a header or a footer template in the **Website** record, the default content is rendered. To remove, specify a blank template.

## Create web templates

**Web Template** is a simple entity that contains the following attributes:

- **Name** - When a template is included in other content, or extended by other templates, it is referred to by this name.
- **Source** - The source content of the template. It can be static text, an HTML fragment, or most often, a layout by using Liquid.
- **MIME type** - Defines what MIME type that the server will send to the client when the template is rendered. If value is not provided, the value is assumed to be `text/html`, which is a standard type for HTML pages. It's possible to create a web template that will render specialized content. For example, you can create a web template that will return some Common Data Service data in `json` format. In this case, the MIME type would be set to `application/json`.

## Web template management

Liquid implementations within Power Apps portals contain a number of **Template tags**<sup>32</sup> that help manage templates and promote reusability.

### FetchXML tag

The `fetchxml` tag allows users to query data from Common Data Service and renders the results in a page.

```
{% fetchxml varResults %}  
  <!-- Fetchxml query -->  
  <fetch>  
    <entity name="account">  
      <attribute name="name" />  
    </entity>  
  </fetch>  
{% endfetchxml %}
```

The `varResults` variable will contain the results of the query.

```
{% for account in varResults.results.entities %}  
  {{ account.name }}<br />  
{% endfor %}
```

*Important* Entity permissions are always applied to the `fetchxml` tag.

---

<sup>32</sup> <https://docs.microsoft.com/powerapps/maker/portals/liquid/template-tags/?azure-portal=true>

## Comment tag

With the `comment` tag, any content within the block will not be rendered, and any Liquid code within will not be run. This tag is useful for including extended comments in complex templates and to temporarily comment out blocks of code when you are troubleshooting.

```
This is a {% comment %}very useful {% endcomment %}comment.
```

## Raw tag

The `raw` tag allows the output of Liquid code on a page without having it parsed and implemented. This tag is useful for generating content that uses conflicting client-side syntax, such as Handlebars.

```
Hello, {% raw %}{{ user.fullname }}.{% endraw %} Nice to meet you.
```

## Include tag

The `include` tag includes the contents of one template in another, by name. This tag allows for the reuse of common template fragments in multiple places, for example, rendering of social links. The included template will have access to any variables that are defined in the parent template, and it is possible to pass parameters.

```
{% include 'Breadcrumbs', separator: '>' %}
```

This logic will include the output that is generated by the **Breadcrumbs** template that will have a `separator` variable set to the > symbol.

## Block tag

By using the `block` tag, you can define a block within a template, which defines a region that can be optionally overwritten by the templates that extend the current template.

## Extend tag

When used in conjunction with the `block` tag, the `extend` tag provides template inheritance. This tag allows multiple templates to use a shared layout while overriding specific areas of the parent layout. When `extends` is used, it must be the first content in the template and can only be followed by one or more `block` tags.

## Base template

The base template logic is as follows:

```
<div>
Hello
{% block content %}default content{% endblock %}
</div>
```

## Child template

The child template logic is as follows:

```
{% extends 'Base Template' %}  
{% block content %}Power Apps portals{% endblock%}
```

The child template will generate the following output:

```
<div>  
  Hello  
  Power Apps portals  
</div>
```

The typical use of the `extend` and `block` tags is to separate layout and content templates. Parent or base templates usually define broad page layout, for example, if it's a single column or a two-column page. Child templates that extend the base can only define the content of the blocks as specified by the parent.

For a comprehensive Liquid code sample that demonstrates template inheritance with `block/extend` and `include` tags, see [Create a custom page template<sup>33</sup>](#).

## Web templates best practices

The following steps will help you improve template structure and make the web templates more manageable:

- Choose descriptive names for your templates because they will be referenced or included as part of other templates.
- Break down the page layout and separate layout from content. These layouts will be candidates for extendable templates.
- Look out for repeatable and reusable fragments; these fragments can be defined as sub templates and will use the `include` tag.
- Use the `include` tag to help reuse layout and commonly used JavaScript fragments. Make your JavaScript templates "smart" by using parameters that are available in the `include` tag. These fragments will be inserted inline.
- Move larger, reusable blocks of JavaScript to separate files and insert them as a reference instead. The client's browser will load them separately and they can be cached. In addition, externally loaded scripts can be minified, which will result in better performance.
- Find parts of the page that need to be translated for multilingual implementation. Define these pages as either page content or content snippets.
- Identify editable fragments. Determine which parts of the webpage where you want users to be responsible for content management and maintenance.
- Be generous with the layout when you are writing a template, but avoid excessive blank lines by using hyphens in your tags, for example:

```
{% if username %}  
  Hello, {{ user.firstname }}
```

---

<sup>33</sup> <https://docs.microsoft.com/powerapps/maker/portals/liquid/create-custom-template>

```
{% endif %}
```

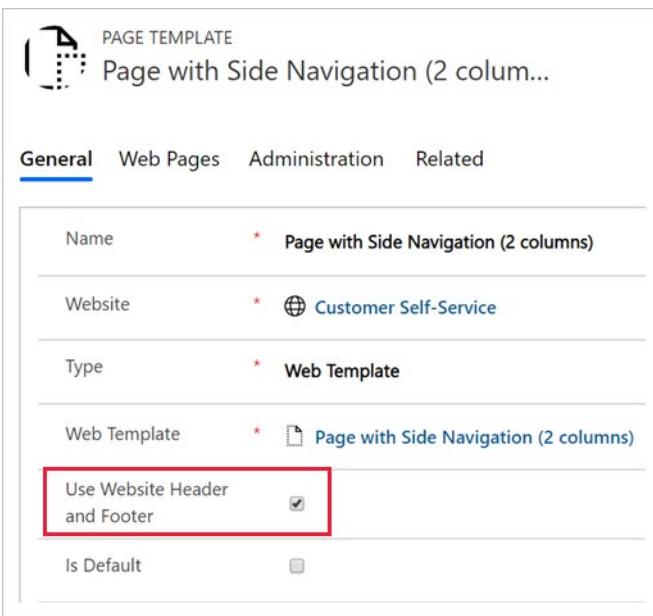
- Use the `comments` tag to describe complex parts of the template.
- Study templates that are already deployed with the starter portal and look for Liquid techniques in [Create advanced templates for portals<sup>34</sup>](#).

Numerous ready-to-use Liquid templates are installed when you provision a starter portal: **Built-in web templates<sup>35</sup>**. Their names are fixed, and the templates are not available for editing.

For more information, see [Work with Liquid templates<sup>36</sup>](#).

## Web templates as API

The **Page Template** entity has a setting that specifies whether the page should use the common website header and footer templates when the web template is used.



When the website header and footer are not used, the template assumes responsibility for generating the entire page output. In the case when you're rendering HTML, this output includes everything from the doctype to the root `<html>` tags, and everything in between. This approach could be useful in a couple different scenarios:

- Special purpose pages need to appear different from the rest of the portal, for example, marketing campaign landing pages.
- The web template generates non-HTML content, returning data in XML, json, or other formats.

For example, you can create a web template that returns a list of accounts, or any other data that the current user has access to, in json format.

```
{% entityview logical_name:'account', name:'Active Accounts' %}  
[  
  {% for acc in entityview.records -%}
```

<sup>34</sup> <https://docs.microsoft.com/powerapps/maker/portals/liquid/create-custom-template/?azure-portal=true>

<sup>35</sup> <https://docs.microsoft.com/powerapps/maker/portals/liquid/store-content-web-templates#built-in-web-templates/?azure-portal=true>

<sup>36</sup> <https://docs.microsoft.com/powerapps/maker/portals/liquid/liquid-overview/?azure-portal=true>

```
{  
    "name": "{{ acc.name }}",  
    "phone": "{{ acc.telephone1 }}"  
}{% unless forloop.last %},{% endunless %}  
{% endfor -%}  
]  
{% endentityview %}
```

**[!NOTE]**

In this example, instead of the `entityview` tag, you can use a FetchXML query inside the `fetchxml` tag. Using inline FetchXML adds some flexibility to the query. The query can be built dynamically by using template parameters or even a `request` object that contains query string parameters of a current HTTP page request.

This template would be used without a header and footer, with the MIME type set to `application/json`. The output would be similar to the following example:

```
[  
{  
    "name": "A Datum Corporation",  
    "phone": "425-555-0182"  
,  
{  
    "name": "A Datum Fabrication",  
    "phone": "303-555-0134"  
,  
{  
    "name": "A Datum Integration",  
    "phone": "512-555-0163"  
,  
{  
    "name": "A. Datum",  
    "phone": "+86-23-4444-0100"  
,  
{  
    "name": "Adventure Works",  
    "phone": "+27-264-1234567"  
}  
]
```

The page that uses this web template wouldn't be used by people who are in a browser; instead, it will be called from JavaScript code, effectively defining an API endpoint for your solution. JavaScript on another page would be able to load and render this data as required.

Authorization will be in place, and accessing Common Data Service data by using this "headless" template is no different than if the output was rendered as HTML. The calling page would typically require user authentication prior to calling the endpoint.

## Exercise-Create an extendable web template

The purpose of this hands-on lab is to introduce the concept of building and extending Liquid templates.

The exercises work best when you have sample data to work with. Depending on the environment that you are working with, you might want to install some sample data to assist with the exercises. Common Data Service does provide the ability to add sample data as needed. If the environment that you are working in does not have sample data installed, follow the steps in the [Add or remove sample data<sup>37</sup>](#) documentation to install the sample data into your environment.

## Learning objectives

At the end of these exercises, you will be able to:

- Extend Liquid templates by using `extends` and `block` tags.
- Reuse Liquid templates by using the `include` tag.
- Apply entity permissions to the results of the new template.

## Prerequisites

For this exercise, you will need to have the following parameters set up in your environment:

- A Power Apps portal that is provisioned. If you do not have a Power Apps portal available, follow the [Create Portal<sup>38</sup>](#) instructions to create one.
- Access to the Power Apps maker portal.

## High-level steps

To finish the exercise, you need to complete the following high-level tasks:

- Create a partial template by accessing Common Data Service data to use as a layout block.
- Create a new template that extends a two-column layout web template.
- Overwrite the side panel to include the partial template.
- Change the template of an existing webpage.
- Configure entity permissions to display data to anonymous users.

## Detailed steps

To complete the exercise, you will build a new page template that includes a side panel that lists all accounts in Common Data Service.

### Create a partial template

Your first task is to create a partial template that will not be used to render a page but will instead be inserted into another template.

1. Open <https://home.dynamics.com>.
2. Select the Portals Management app.
3. Select **Web Templates**.
4. Select **New**.

<sup>37</sup> <https://docs.microsoft.com/power-platform/admin/add-remove-sample-data>

<sup>38</sup> <https://docs.microsoft.com/powerapps/maker/portals/create-portal>

5. Enter the following values:

- **Name** - Directory
- **Website** - Select your current website
- **Source** - Enter the following content:

```
{% fetchxml accounts %}  
<fetch>  
  <entity name="account">  
    <attribute name="name" />  
  </entity>  
</fetch>  
{% endfetchxml %}  
  
{% if accounts.global_permission_granted %}  
  <ul>  
    {% for account in accounts.results.entities %}  
      <li>{{ account.name }}</li>  
    {%- endfor -%}  
  </ul>  
{% else %}  
  <div class="alert alert-warning">You do not have permissions to access the directory.</div>  
{% endif %}
```

6. Select **Save & Close**.

## Extend an existing template

Next, you will create a new template that extends an existing Liquid template and then insert the template that you previously created.

1. Select **Web Templates**.

2. Select **New**.

3. Enter the following values:

- **Name** - Directory Template
- **Website** - Select your current website
- **Source** - Enter the following content:

```
{% extends "Layout 2 Column Wide Left" %}  
  
{% block aside %}  
  <h2>Directory</h2>  
  {% include 'Directory' %}  
{% endblock %}
```

4. Select **Save & Close**.

## Create a page template and associate with that page

In this exercise, you will create a page template that will use your new web template and will include the Directory output.

1. Select **Page Templates**.
2. Select **New**.
3. Enter the following values:
  - **Name** - Directory Page Template
  - **Website** - Select the current website
  - **Type** - Select **Web Template**
  - **Web Template** - Select **Directory Template**
  - **Entity Name** - Select **Web Page**
4. Select **Save & Close**.

## Test

Your next step is to test that your new template works:

1. Open Power Apps portals Studio in a new browser tab. Then, follow these steps:
  1. Go to Power Apps maker portal at <https://make.powerapps.com>.
  2. Select the target environment by using the environment selector in the upper-right corner.
  3. From the **Apps** list, select the application of type **Portal**.
  4. Select the **Edit** menu.
2. On the toolbelt, select the **Pages** icon.
3. Select an existing page, for example **Product A** under **Services**. Note: names and hierarchy of pages on your portal might differ.
4. Locate the **Template** property in the **Component** panel on the right side.
5. Select **Directory Page Template** as the new template.  
The list of accounts should be displayed because portals Studio runs under the maker account and uses Common Data Service security instead of entity permissions to filter the data.
6. Select **Browse website**.  
The message "You do not have permissions to access the directory" should be displayed.

## Add entity permissions

Follow these steps to add entity permissions:

1. Return to the Portal Management app.
2. Select **Entity Permissions**.
3. Select **+ New**.

4. Enter the following values:
  - **Name** - Account Directory
  - **Entity Name** - Select the account entity
  - **Website** - Select your current website
  - **Scope** - Select **Global**
  - **Privileges** - Select **Read**
5. Select **Save**.
6. Scroll to the **Web Roles** subgrid.
7. Select **Add Existing Web Role**.
8. Locate and select **Anonymous users** and **Authenticated users**.
9. Select **Add**.

## Test

Your final task is to test your new template:

1. Switch to portals Studio.
2. Select **Browse website**. Note: This command rebuilds the site cache. A simple browser page refresh will not be sufficient to update the data.

The page should now be displayed and include the list of accounts.

The screenshot shows a web browser displaying a Power Apps portal for 'Contoso Ltd'. The top navigation bar includes links for Home, Services, About us, Products, Liquid, and Sign in. Below the navigation, a breadcrumb trail shows 'Home > Products > Product A'. The main content area features a product card for 'Product A' with a thumbnail image of a camera. To the right of the product card is a 'Directory' sidebar containing a list of sample organizations: Fourth Coffee (sample), Litware, Inc. (sample), Adventure Works (sample), Fabrikon, Inc. (sample), Blue Yonder Airlines (sample), City Power & Light (sample), Contoso Pharmaceuticals (sample), Alpine Ski House (sample), A. Datum Corporation (sample), Coho Winery (sample), and Aperture Science.

## Summary

This module explained custom web templates and how you can build and configure templates to extend and enhance a Power Apps portal.

By now, you should be able to:

- Describe the relationship between web templates, page templates, and webpages.
- Explain how custom web templates can be built by using HTML, CSS, Liquid, and JavaScript.
- Use Liquid to build and structure web templates.

- 
- Embed custom CSS and JavaScript into web templates.
  - Reference other web templates in a web template.

## Next steps

Your next steps would be to learn why and when to use various development techniques and then learn how to apply these techniques, tools, and methods to address specific business requirements.



---

## Module 10 Integrate with Power Platform and Common Data Service

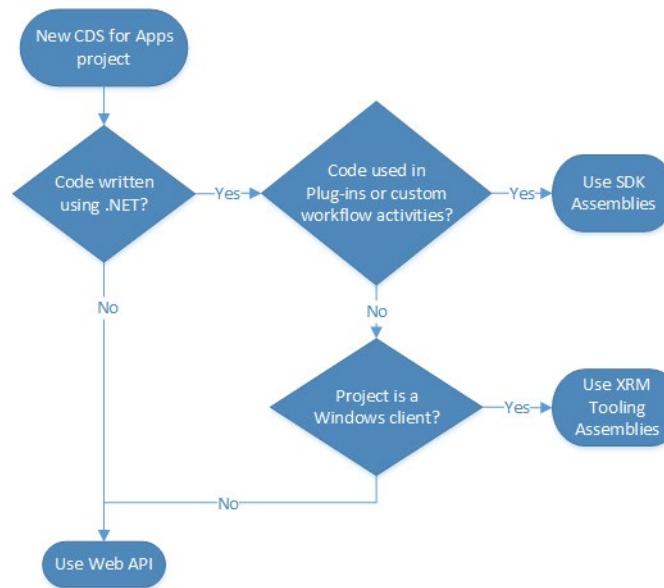
### Work with Common Data Service Web API

#### Web API vs. the Organization Service

Two main ways for interacting with data and metadata in Common Data Service are the Web API and the Organization Service. You can use the Organization Service when writing plug-ins or workflow extensions by using the .NET Framework SDK assemblies that are provided by Microsoft.

At the time of this writing, all data operations go through the underlying Organization Service. As such, the Web API allows you to perform the same operations as the Organization Service but in a RESTful style (more information on REST will be provided later in this module).

The following graphic is a decision tree that can help you determine when to use the Web API versus the Organization Service (through SDK Assemblies) and also when to use XRM Tooling Assemblies (available from Microsoft) for Windows applications.



This module focuses on the Web API; for more information on the Organization Service, see [Use the Common Data Service Organization Service<sup>1</sup>](#) documentation found in the Common Data Service Developer Guide.

If you'd like to learn more about the XRM Tooling Assemblies that are available for Windows client applications, see [Build Windows client applications using the XRM tools<sup>2</sup>](#) documentation, which is also found in the Common Data Service Developer Guide. XRM is considered the predecessor to Common Data Service, so you'll frequently see this name embedded into documentation and tooling that is provided by Microsoft.

## OData 4.0

Common Data Service Web API is implemented as an OData (OASIS Open Data Protocol) service. By using an industry standard such as this, you're allotted all the benefits of open standard development, such as cross-platform compatibility and simple implementation against a common, predefined protocol.

Common Data Service doesn't provide specific assembly tooling for the Web API; instead, we recommend that you use community libraries that conform to the OData protocol. If you want more information about which libraries exist, go to <https://www.odata.org/libraries/><sup>3</sup>.

OData provides the ability to interact with Common Data Service data by means of CRUD operations that are available through standard GET, POST, PATCH, and DELETE HTTP methods (more information about these methods will be provided later in this module). You can also perform almost any other operation that is exposed through Common Data Service [Event Framework<sup>4</sup>](#) by invoking an OData *function* or *action* that

<sup>1</sup> <https://docs.microsoft.com/powerapps/developer/common-data-service/org-service/overview/?azure-portal=true>

<sup>2</sup> <https://docs.microsoft.com/powerapps/developer/common-data-service/xrm-tooling/build-windows-client-applications-xrm-tools/?azure-portal=true>

<sup>3</sup> <https://www.odata.org/libraries/?azure-portal=true>

<sup>4</sup> <https://docs.microsoft.com/powerapps/developer/common-data-service/event-framework/?azure-portal=true>

has the same name as its corresponding Organization Service message. CRUD-related messages are the only ones that are not available because those messages are achieved by means of the previously-described standard HTTP methods.

## REST

Fundamentally, OData protocol is a set of RESTful (REpresentational State Transfer) APIs, a time-tested industry standard for providing interoperability between systems. REST consists of six guiding principles that restrict the way that a service can process and respond to requests to ensure that non-functional properties of that service are met (such as performance, scalability, simplicity, reliability, and so on).

For the purposes of this lesson, you should keep in mind that OData conforms to these constraints and that the Common Data Web API in turn also conforms to the constraints because it implements the OData 4.0 protocol.

## FetchXML

FetchXML is a robust query language that allows for complex querying of Common Data Service data. Microsoft provides a way for users to run FetchXML queries within a Web API query. This approach can be valuable if you encounter a scenario where FetchXML is a better fit for your particular query versus using OData query syntax. More information about FetchXML is provided in a later unit in this module.

## Authenticate against Common Data Service by using OAuth

Microsoft's Common Data Service uses OAuth 2.0 as its authorization standard. OAuth provides an industry-standardized mechanism for authenticating client applications against a resource.

## Authentication vs. authorization

**Authentication** is the process or action of verifying the identity of a user or process. Microsoft's solution to this verification process is Azure Active Directory (Azure AD). Azure AD supports a number of options to verify the identity of a user or process. Abstracting your identity provider allows for a good separation of concerns because managing usernames and passwords can be a difficult (and risky) process.

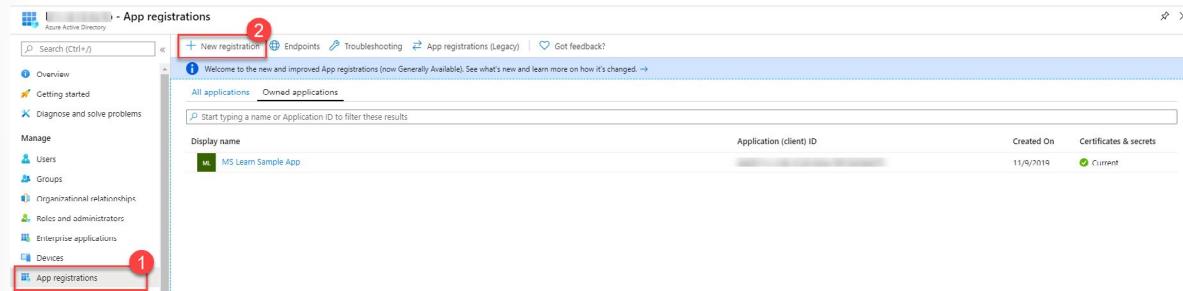
**Authorization** is the process or action of verifying whether an *authenticated* user is *authorized* to access the resources that are being provided. Presently, authorization to Common Data Service is at the tenant level, and permissions are delegated to the application based on the current signed-in user. Therefore, you won't use OAuth to govern app-level security, which is instead handled by means of the Power Apps admin center.

If you want to learn more about the concepts of authentication and authorization, go to the Azure Active Directory section of Microsoft Docs: [Authentication basics<sup>5</sup>](https://docs.microsoft.com/azure/active-directory/develop/authentication-scenarios/?azure-portal=true).

## Register Common Data Service apps with Azure AD

To successfully connect to your Common Data Service, you must first register an app with Azure Active Directory, which can be completed in the [Azure portal<sup>6</sup>](https://portal.azure.com/?azure-portal=true). Depending on the type of app that you want to make, a few different settings are available for you to configure (web apps versus native apps that are installed natively on a device). For more information on settings that are required for each of these types, see [Types of app registration<sup>7</sup>](https://docs.microsoft.com/powerapps/developer/common-data-service/authenticate-oauth?azure-portal=true).

To register an app with Azure AD, you can go to the **App Registrations** section of the Azure Active Directory menu blade and then select **New Registration**.



The screenshot shows the 'App registrations' section of the Azure Active Directory portal. The left sidebar includes links for Overview, Getting started, Diagnose and solve problems, Manage (Users, Groups, Organizational relationships, Roles and administrators, Enterprise applications, Devices), and App registrations (which is highlighted with a red box and labeled '1'). At the top center, there's a search bar, a '+ New registration' button (highlighted with a red box and labeled '2'), and other navigation links like Endpoints, Troubleshooting, App registrations (Legacy), and Get feedback?

Specify the name of your app and what type of account access you need. If you're registering a web app, specify a Redirect URI by going to the **Authentication** section, setting the type to **Web**, and then entering a redirect URI.

### Register an application

#### \* Name

The user-facing display name for this application (this can be changed later).

MS Learn Sample App

1

#### Supported account types

Who can use this application or access this API?

- Accounts in this organizational directory only (Paritta Group only - Single tenant)  
 Accounts in any organizational directory (Any Azure AD directory - Multitenant)  
 Accounts in any organizational directory (Any Azure AD directory - Multitenant) and personal Microsoft accounts (e.g. Skype, Xbox)

2

Help me choose...

#### Redirect URI (optional)

We'll return the authentication response to this URI after successfully authenticating the user. Providing this now is optional and it can be changed later, but a value is required for most authentication scenarios.

Web https://callbackurl

3

<sup>5</sup> <https://docs.microsoft.com/azure/active-directory/develop/authentication-scenarios/?azure-portal=true>

<sup>6</sup> <https://portal.azure.com/?azure-portal=true>

<sup>7</sup> <https://docs.microsoft.com/powerapps/developer/common-data-service/authenticate-oauth?azure-portal=true>

The following list summarizes when to use the different account types:

- **Accounts in this organizational directory only (Single tenant)**

All user and guest accounts in your directory can use your application or API.

*Use this option if your target audience is internal to your organization.*

- **Accounts in any organizational directory (Any Azure AD directory - Multitenant)**

All users with a work or school account from Microsoft can use your application or API, including schools and businesses that use Microsoft 365.

*Use this option if your target audience is business or educational customers and to enable multitenancy.*

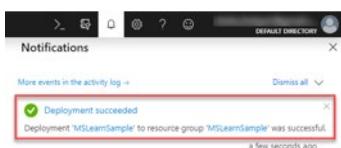
- **Accounts in any organizational directory (Any Azure AD directory - Multitenant) and personal Microsoft accounts (for example, Skype and Xbox)**

All users with a work, school, or personal Microsoft account can use your application or API. It includes schools and businesses that use Microsoft 365 and personal accounts that are used to sign in to services like Xbox and Skype.

For this example, `https://callbackurl` was entered for the **Redirect URI** because you will be accessing with Postman in a later lesson, which is a public application. Note this URL because you will need it to connect to Common Data Service successfully.

Depending on the complexity of your application setup, you might want to configure other authentication settings. Refer to the Azure Active Directory documentation for steps on how to complete this task.

You will also need to set a few flags to **true** in the app's manifest to allow access from Postman and other external applications. This access is only being granted for demonstration purposes only. We recommend that you read the **Azure Active Directory app manifest**<sup>8</sup> documentation to fully understand the implications of these flags.



Next, go to the **API permissions** tab and ensure that your application is granted user impersonation access to your Common Data Service environment.

<sup>8</sup> <https://docs.microsoft.com/azure/active-directory/develop/reference-app-manifest/?azure-portal=true>

The first screenshot shows the 'MSLearnSample - Queues' blade in the Azure portal. A red box labeled '1' highlights the 'Queues' link under the 'Entities' section. A red box labeled '2' highlights the '+ Queue' button at the top. A red box labeled '3' highlights the 'Name' field where 'MSLearnSampleQueue' is entered. The second screenshot shows the 'MSLearnSample - Shared access policies' blade. A red box labeled '1' highlights the 'Shared access policies' link under the 'Settings' section. A red box labeled '2' highlights the 'Manage, Send, Listen' claim for the 'RootManageSharedAccessKey' policy.

You should now have an app endpoint that can successfully connect to your Common Data Service environment.

## Interact with Common Data Service Web API by using Postman

Postman is a valuable tool to use for composing ad hoc Web API queries and verifying the behavior of Web API Operations without having to write a program to do so. If you're new to Postman, see the **First 5 things to try if you're new to Postman**

[Postman<sup>9</sup>](#)

article for an introduction to the topic.

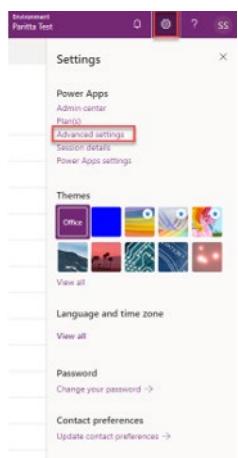
## Register an app in Azure Active Directory

To connect Postman to Common Data Service, you must first ensure that you have an application registered in your Azure AD environment.

<sup>9</sup> <https://blog.getpostman.com/2018/04/11/first-5-things-to-try-if-youre-new-to-postman/?azure-portal=true>

## Get the current version of your Web API endpoint

You can find the version number of your Web API endpoint by going to your environment's **Advanced settings** in the maker portal.



From **Advanced settings**, go to **Settings > Customizations** and then go to the **Developer Resources** page. Your version number will be indicated under the Instance Web API region, as shown in the following image.

**Developer Resources**

**Getting Started**

- Developer Center
- Developer Forums
- SDK NuGet Packages
- SDK Download
- Sample Code
- Developer Overview

**Connect your apps to this instance of Dynamics 365**

**Instance Web API**  
HTTP REST API providing access to this instance of Dynamics 365. For more information see [Microsoft Dynamics 365 Web API](#).

Service Root URL  [Download OData Metadata](#)

**Instance Reference Information**  
Use this information to uniquely identify this instance of Dynamics 365. You can use this to retrieve the current URL for this instance. For more information see [Azure extensions for Microsoft Dynamics 365](#).

ID	<input style="width: 150px; height: 20px;" type="text"/>
Unique Name	<input style="width: 150px; height: 20px;" type="text"/>

**Connect your apps to the Dynamics 365 Discovery Service**

**Discovery Web API**  
HTTP REST API providing connection information for the set of Dynamics 365 instances to which the caller has access. For more information see [Discover the URL for your organization with Discovery Web API](#).

Endpoint Address  [Download OData Metadata](#)

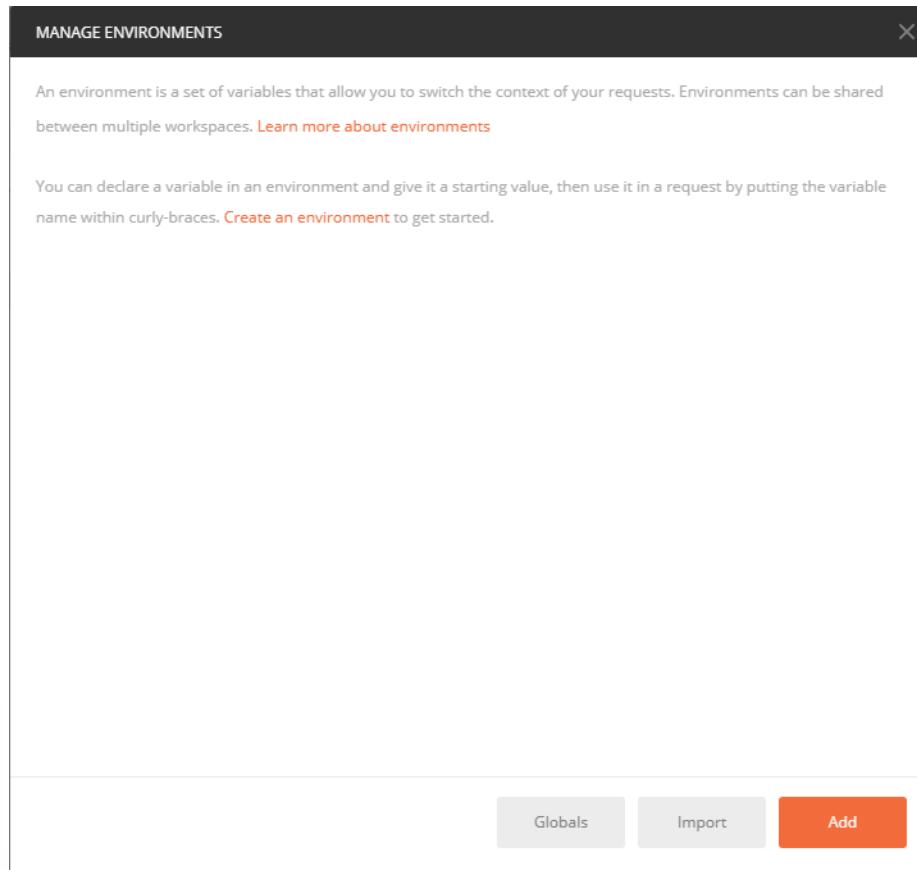
## Set up Postman

To start the setup process, install **Postman**<sup>10</sup>.

After Postman has been installed, you will use a feature that Postman has to manage environment variables so that you can cleanly manage entities such as your endpoint URLs and client IDs.

To create a Postman environment that you can use to connect with your Common Data Service instance, follow these steps:

1. Launch the Postman desktop application.
2. Select the **Environment Options** gear icon in the top-right corner.
3. In the **Manage Environments** dialog box, select the **Add** button to add a new environment.



4. In the dialog box that opens, enter a name for the environment and then add the following key/value pairs into the editing space.

Variable	Initial value
url	<code>https://[ORGANIZATION].crm.dynamics.com</code>
clientid	<code>&lt;APPID&gt;</code>
version	<code>9.1</code>
webapiurl	<code>{url}/api/data/v{version}/</code>

<sup>10</sup> <https://www.getpostman.com/?azure-portal=true>

The screenshot shows the 'Manage environments' dialog box in Postman. At the top, there's a table with two columns: 'Variable' and 'Initial value'. Below it is a list of environment variables with their current values. A note at the bottom explains how variables are reused across different places.

Variable	Initial value
callback	<a href="https://callbackurl">https://callbackurl</a>
authurl	<a href="https://login.microsoftonline.com/common/oauth2/authorize?resource={{url}}">https://login.microsoftonline.com/common/oauth2/authorize?resource={{url}}</a>

MANAGE ENVIRONMENTS		X
Environment Name		
MS Learn Sample App		
VARIABLE	INITIAL VALUE ⓘ	CURRENT VALUE ⓘ
<input checked="" type="checkbox"/> url	<a href="https://[REDACTED].crm.dynamics.com">https://[REDACTED].crm.dynamics.com</a>	<a href="https://[REDACTED].crm.dynamics.com">https://[REDACTED].crm.dynamics.com</a>
<input checked="" type="checkbox"/> clientid	[REDACTED] ...	[REDACTED]
<input checked="" type="checkbox"/> version	9.1	9.1
<input checked="" type="checkbox"/> webapiurl	<a href="{{url}}/api/data/v{{version}}/">{{url}}/api/data/v{{version}}/</a>	<a href="{{url}}/api/data/v{{version}}/">{{url}}/api/data/v{{version}}/</a>
<input checked="" type="checkbox"/> callback	<a href="https://callbackurl">https://callbackurl</a>	<a href="https://callbackurl">https://callbackurl</a>
<input checked="" type="checkbox"/> authurl	<a href="https://login.microsoftonline.com/common/oauth2/authorize?resource={{url}}">https://login.microsoftonline.com/common/oauth2/authorize?resource={{url}}</a>	<a href="https://login.microsoftonline.com/common/oauth2/authorize?resource={{url}}">https://login.microsoftonline.com/common/oauth2/authorize?resource={{url}}</a>
Add a new variable		

ⓘ Use variables to reuse values in different places. The current value is used while sending a request and is never synced to Postman's servers. The initial value is auto-updated to reflect the current value. [Change this behaviour from Settings](#). [Learn more about variable values](#) X

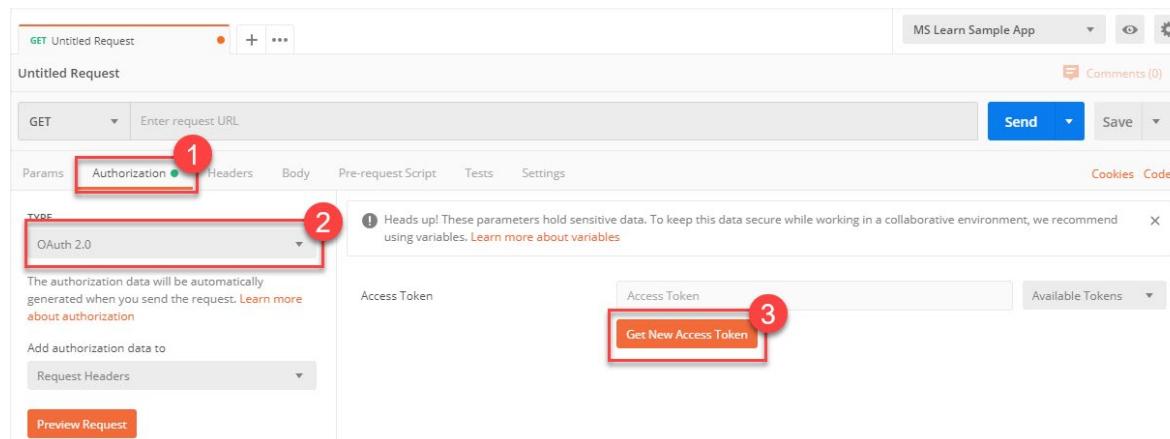
Cancel Update

5. Replace the instance URL placeholder value with the URL of your Common Data Service instance and then select **Add** to save the environment.
6. Close the **Manage environments** dialog box.

## Generate an access token to use with your environment

To connect by using OAuth 2.0, you must have an access token. To get a new access token, follow these steps:

1. Make sure that the new environment you created is selected.
2. Select the **Authorization** tab.
3. Set the **Type** to **OAuth 2.0**.
4. Verify that you have selected the environment that you created.
5. Select **Get New Access Token**.



6. Set the following values in the dialog box. Select **Implicit** from the **Grant Type** drop-down menu. You can set the **Token Name** to whatever you like and then leave other keys set to default values.

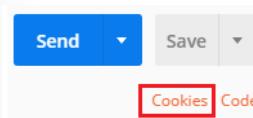
A screenshot of the 'GET NEW ACCESS TOKEN' dialog box. It contains the following fields:

- Token Name: MSLearnToken01
- Grant Type: Implicit
- Callback URL: {{callback}}
- Auth URL: {{authurl}}
- Client ID: {{clientid}}
- Scope: e.g. read:org
- State: State
- Client Authentication: Send as Basic Auth header

A large red box highlights the 'Request Token' button at the bottom center of the dialog.

[!NOTE]

If you are configuring environments in Postman for multiple Common Data Service instances by using different user credentials, you might need to delete the cookies that are cached by Postman. Select the **Cookies** link, which can be found under the **Send** button, and remove the saved cookies from the **Manage Cookies** dialog box.



Some of these cookies are persistent. You can delete some cookies in groups, but you might have to delete others individually. You might need to complete this process twice to ensure that no cookies remain.

7. Select **Request Token**. When you make this selection, an Azure Active Directory sign-in page appears. Enter your username and password.
8. After the token is generated, scroll to the bottom and select **Use Token**. This selection closes the **Manage Access Tokens** dialog box.
9. After you have added a token, you can select which token to apply to requests. On the **Available Tokens** drop-down list, select the token that you have just created. The Authorization header will be added to the Web API request.

## Test your connection

Create a new Web API request to test the connection with your Common Data Service instance. Use the **WhoAmI function<sup>11</sup>** by following these steps:

1. Select GET as the HTTP method and add {{webapiurl}}WhoAmI in the editing space.



2. Select **Send** to send this request.

If your request is successful, you will see the data from the **WhoAmIResponse ComplexType<sup>12</sup>** that is returned by the **WhoAmI Function<sup>13</sup>**.

A screenshot of the Postman response view. At the top, tabs show 'Body', 'Cookies [2]', 'Headers [21]', and 'Test Results'. Status information at the top right says 'Status: 200 OK' and 'Time: 520ms'. The main area displays a JSON response with line numbers 1 through 6. The JSON content includes '@odata.context', 'BusinessUnitId', 'UserId', and 'OrganizationId' fields.

## Use OData to query data

OData is a time-tested language for querying RESTful APIs. This lesson explores a few ways of performing CRUD operations against Common Data Service data.

<sup>11</sup> <https://docs.microsoft.com/dynamics365/customer-engagement/web-api/whoami/?azure-portal=true>

<sup>12</sup> <https://docs.microsoft.com/dynamics365/customer-engagement/web-api/whoamiresponse/?azure-portal=true>

<sup>13</sup> <https://docs.microsoft.com/dynamics365/customer-engagement/web-api/whoami/?azure-portal=true>

## Create records

To create records, use the HTTP POST method.

```
POST [Organization URI]/api/data/v9.0/accounts HTTP/1.1
Content-Type: application/json; charset=utf-8
OData-MaxVersion: 4.0
OData-Version: 4.0
Accept: application/json

{
    "name": "Sample Account",
    "creditonhold": false,
    "address1_latitude": 47.639583,
    "description": "This is the description of the sample account",
    "revenue": 5000000,
    "accountcategorycode": 1
}
```

## Retrieve records

To retrieve records, use the HTTP GET method.

The following sample retrieves an account with the ID of 00000000-0000-0000-000000000001:

```
GET [Organization URI]/api/data/v9.0/accounts(00000000-0000-0000-000000000001)
```

Other methods are available for querying data by using standard OData query syntax. For more information on this process, see **Retrieve an entity record using the Web API<sup>14</sup>**.

## Update records

Depending on what you are trying to accomplish, you can choose between one of two methods for updating records:

- If you are performing an update of multiple attribute values, use the HTTP PATCH method. PATCH methods provide upsert capability if you provide an ID value as part of your request, a beneficial feature when you are synchronizing data between systems.
- If you are updating a single attribute value, use the HTTP PUT method. This method can't be used with navigation properties like lookups because that requires a reference to be removed as well.

The following example updates an account entity:

```
PATCH [Organization URI]/api/data/v9.0/accounts(00000000-0000-0000-000000000001) HTTP/1.1
Content-Type: application/json
```

---

<sup>14</sup> <https://docs.microsoft.com/powerapps/developer/common-data-service/webapi/retrieve-entity-using-web-api/?azure-portal=true>

```
OData-MaxVersion: 4.0
OData-Version: 4.0

{
    "name": "Updated Sample Account",
    "creditonhold": true,
    "address1_latitude": 47.639583,
    "description": "This is the updated description of the sample account",
    "revenue": 6000000,
    "accountcategorycode": 2
}
```

If you want to retrieve data from the entity that you are updating, you can use the `return=representation` request header. If you want to control which properties are returned, you can add a `$select` query to your PATCH URL. In the following example, the header has been added and the `$select` has been amended to only include the name, creditonhold, and address1 attributes.

#### **Request**

```
PATCH [Organization URI]/api/data/v9.0/accounts(00000000-0000-0000-0000-000000000001)?$select=name,creditonhold,address1_latitude,description,revenue,accountcategorycode,createdon HTTP/1.1
OData-MaxVersion: 4.0
OData-Version: 4.0
Accept: application/json
Content-Type: application/json; charset=utf-8
Prefer: return=representation

{"name": "Updated Sample Account"}
```

#### **Response**

```
HTTP/1.1 200 OK
Content-Type: application/json; odata.metadata=minimal
Preference-Applied: return=representation
OData-Version: 4.0

{
    "@odata.context": "[Organization URI]/api/data/v9.0/$metadata#accounts/$entity",
    "@odata.etag": "W/\"536537\"",
    "accountid": "00000000-0000-0000-000000000001",
    "accountcategorycode": 1,
    "description": "This is the description of the sample account",
    "address1_latitude": 47.63958,
    "creditonhold": false,
    "name": "Updated Sample Account",
    "createdon": "2016-09-28T23:14:00Z",
    "revenue": 5000000.0000,
    "_transactioncurrencyid_value": "048dddaa-6f7f-e611-80d3-00155db5e0b6"
```

```
}
```

The following is a sample PUT request where an account's name for a given record is updated:

```
PUT [Organization URI]/api/data/v9.0/accounts(00000000-0000-0000-000000000001)/name HTTP/1.1
Content-Type: application/json
OData-MaxVersion: 4.0
OData-Version: 4.0

{"value": "Updated Sample Account Name"}
```

## Delete records

To delete records, use the HTTP DELETE method. The operation is straightforward, where you provide the URI of the entity record that you want to delete, as shown in the following request:

```
DELETE [Organization URI]/api/data/v9.0/accounts(00000000-0000-0000-000000000001) HTTP/1.1
Content-Type: application/json
OData-MaxVersion: 4.0
OData-Version: 4.0
```

## Additional operations

Common Data Service provides a number of other predefined operations that you can trigger through Web API requests. For a full inventory of the available capabilities, see **Perform operations using the Web API<sup>15</sup>**.

## Use FetchXML to query data

FetchXML is a robust query language that was developed by Microsoft to enable complex operations to be performed against Common Data Service data. It provides considerably more filtering options than the OData connectors, and it also allows you to see the labels for reference data such as Option Set labels or lookup fields.

You can run FetchXML queries by using the following methods:

- Common Data Service Web API
- Organization Service

You can also apply a custom FetchXML filter to a lookup field within a model-driven app. This approach allows for more complex lookup scenarios that aren't easily accomplished out of the box. For more information on how to complete this process, see the **addCustomFilter Client API Reference<sup>16</sup>**.

---

<sup>15</sup> <https://docs.microsoft.com/powerapps/developer/common-data-service/webapi/perform-operations-web-api/?azure-portal=true>

<sup>16</sup> <https://docs.microsoft.com/powerapps/developer/model-driven-apps/clientapi/reference/controls/addcustomfilter/?azure-portal=true>

## Example FetchXML query

The following sample query retrieves the accountid and name fields from the account entity:

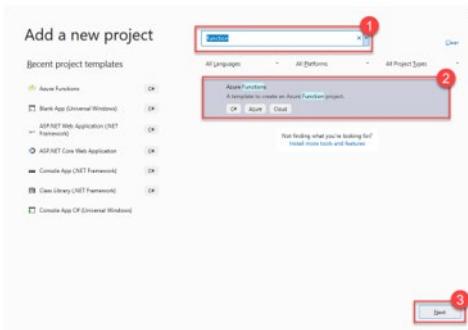
```
<fetch mapping='logical'>
    <entity name='account'>
        <attribute name='accountid'/>
        <attribute name='name'/>
    </entity>
</fetch>
```

If you need to filter this data, for example by state, you could do something like the following sample query:

```
<fetch mapping='logical'>
    <entity name='account'>
        <attribute name='accountid'/>
        <attribute name='name'/>
        <filter type='and'>
            <condition attribute='address1_stateorprovince' operator='eq'
value='WA' />
        </filter>
    </entity>
</fetch>
```

## Tools to help build your queries

If you're querying data against a model-driven app that is built on Common Data Service, the simplest method for building FetchXML queries is to use Advanced Find, which contains a **Download FetchXML** function. You are limited to the UI on what filters and related data that you can pull, but it's frequently an excellent first start so that you don't have to compose the XML anew.



Another notable tool, called **FetchXML Builder**, can be found at XrmToolBox. With this tool, you can construct FetchXML in ways that Advanced Find cannot (aggregates, outer joins, "has no" queries, and attributes from multi-level linked

entities). It also provides a wealth of other features, which you can find at their website: [FetchXML Builder<sup>17</sup>](#).

## Additional resources

Go to Microsoft Docs to discover in-depth details on the FetchXML Query Language. The [Use FetchXML to query data<sup>18</sup>](#) section covers several advanced topics such as how to handle paging, querying hierarchical data, and much more.

# Call Power Automate actions from WebAPI

Power Automate contains a feature that allows power users to create their own custom sets of business logic called *actions*. From a developer perspective, Microsoft Docs defines an action as a “reusable operation, which may have observable side effects.” These side effects include creating or updating records, or anything you can accomplish by using a Classic Common Data Service workflow.

An additional feature of actions is their ability to be invoked through the Web API. As a result, you can encapsulate your low-code automations in such a way that you can use them in nearly any scenario by interacting with them through their associated Web API request and response.

While this lesson doesn't cover how to create actions in detail, it's assumed that you already have a basic understanding of how to build them. If you need more information on how to use actions, you can review the Power Automate documentation on them. See the [Classic Common Data Service Actions<sup>19</sup>](#) section for detailed information on how to build Common Data Service actions.

## Run actions with the Web API

When a new action is created in your Common Data Service solution, the framework also creates a corresponding Web API request message. You can run these requests by using a POST operation similar to the following example:

```
POST [Organization URI]/api/data/v9.0/WinOpportunity HTTP/1.1
Accept: application/json
Content-Type: application/json; charset=utf-8
OData-MaxVersion: 4.0
OData-Version: 4.0

{
  "Status": 3,
  "OpportunityClose": {
    "subject": "Won Opportunity",
    "opportunityid@odata.bind": "[Organization URI]/api/data/v9.0/opportunities(b3828ac8-917a-e511-80d2-00155d2a68d2)"
  }
}
```

---

<sup>17</sup> <https://fetchxmlbuilder.com/?azure-portal=true>

<sup>18</sup> <https://docs.microsoft.com/powerapps/developer/common-data-service/use-fetchxml-construct-query/?azure-portal=true>

<sup>19</sup> <https://docs.microsoft.com/power-automate/workflow-processes/?azure-portal=true>

```
}
```

The above example is an HTTP request call to the WinOpportunity action for an opportunity with an opportunityid value of b3828ac8-917a-e511-80d2-00155d2a68d2.

## Call pre-built Common Data Service actions

Common Data Service comes with a set of existing actions that you can use to perform common operations. Some of these actions might only apply to model-driven or Dynamics 365 apps. For example, the CalculatePrice action calculates the price in an opportunity/quote/order/invoice, so it would only apply to a Dynamics 365 Sales app where that functionality exists.

For more information about which pre-built actions are available for use through the Web API, see the [Web API Action Reference<sup>20</sup>](#).

## Unbound vs. bound actions

An action can be built as either unbound (meaning, it doesn't have a Common Data Service entity correlated to it) or bound (meaning, the logic is tied to a specific entity record). SQL developers could

consider these actions as similar to Stored Procedures (unbound actions) versus Triggers (bound actions). However, unlike SQL Triggers, you can also run bound actions on demand by providing a corresponding entity record ID as its parameter.

Unbound actions are beneficial for generic logic that might need to run outside the context of a specific entity record, such as the WinOpportunity action that was referenced earlier. To run a bound action against a specific record, you must provide the ID of that record in the URI of your request:

```
POST [Organization URI]/api/data/v9.0/contacts(94d8c461-a27a-e511-80d2-00155d2a68d2)/Microsoft.Dynamics.CRM.new_AddNoteToContact HTTP/1.1
Accept: application/json
Content-Type: application/json; charset=utf-8
OData-MaxVersion: 4.0
OData-Version: 4.0

{
  "NoteTitle": "New Note Title",
  "NoteText": "This is the text of the note"
}
```

The preceding example calls a custom action that was previously built in the solution called new\_AddNoteToContact for a contact with the ID of 94d8c461-a27a-e511-80d2-00155d2a68d2 and then passes the parameters

<sup>20</sup> <https://docs.microsoft.com/dynamics365/customer-engagement/web-api/actions?view=dynamics-ce-odata-9/?azure-portal=true>

NoteTitle and NoteText to the action. The expected behavior for this procedure would be that a new note would be added to the corresponding contact with the title and text provided.

## More details

For more details on how to call actions in code, see the [Use Web API actions<sup>21</sup>](#) article in Microsoft Docs.

# Use the Web API to impersonate another user

Occasionally, you might need to run logic for another user. Within the context of Common Data Service, the logic that you're running will apply all appropriate role and object-based security based on the user whom you're impersonating. This method can be especially effective when you are integrating external systems with a Common Data Service solution where the integration account is a system account versus the user who actually invoked the request.

## Implement a Web API request with user impersonation

When calling any Web API method, you can provide a CallerObjectId in your message header to indicate that you want the message to be run for that particular user. The value of this parameter is their Azure Active Directory (Azure AD) object ID. The Azure Graph API provides a method to query Azure AD user data. For more information, see the [Azure AD Graph API Reference<sup>22</sup>](#).

## Determine the user who performed an operation

If you need to see the ID of the user who actually performed an operation for another user, you can query the record's createdonbehalfby value, which will contain this detail. For example, if you want to see if an account record was created through user impersonation rather than by the actual user, you could query that account record to compare their createdby and createdonbehalfby values by using the following query:

```
GET [Organization URI]/api/data/v9.0/accounts(00000000-0000-0000-000000000000)?$select=name&$expand=createdby($select=fullname),createdonbehalfby($select=fullname),owninguser($select=fullname) HTTP/1.1
Accept: application/json
OData-MaxVersion: 4.0
OData-Version: 4.0
```

If the record was created by an impersonating account, you can expect a response similar to the following example:

```
HTTP/1.1 200 OK
Content-Type: application/json; odata.metadata=minimal
ETag: W/"506868"
```

<sup>21</sup> <https://docs.microsoft.com/powerapps/developer/common-data-service/webapi/use-web-api-actions/?azure-portal=true>

<sup>22</sup> <https://docs.microsoft.com/previous-versions/azure/ad/graph/api/users-operations/?azure-portal=true>

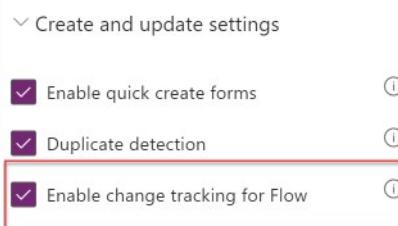
```
{  
    "@odata.context": "[Organization URI]/api/data/v9.0/$metadata#ac-  
counts(name,createdby(fullname,azureactivedirectoryobjectid),createdonbe-  
halfby(fullname,azureactivedirectoryobjectid),owninguser(fullname,azureac-  
tivedirectoryobjectid))/$entity",  
    "@odata.etag": "W/\"2751197\"",  
    "name": "Sample Account created using impersonation",  
    "accountid": "00000000-0000-0000-000000000003",  
    "createdby": {  
        "@odata.etag": "W/\"2632435\"",  
        "fullname": "Impersonated User",  
        "azureactivedirectoryobjectid": "e39c5d16-675b-48d1-8e67-667427e9c084",  
        "systemuserid": "75df116d-d9da-e711-a94b-000d3a34ed47",  
        "ownerid": "75df116d-d9da-e711-a94b-000d3a34ed47"  
    },  
    "createdonbehalfby": {  
        "@odata.etag": "W/\"2632445\"",  
        "fullname": "Actual User",  
        "azureactivedirectoryobjectid": "3d8bed3e-79a3-47c8-80cf-269869b2e9f0",  
        "systemuserid": "278742b0-1e61-4fb5-84ef-c7de308c19e2",  
        "ownerid": "278742b0-1e61-4fb5-84ef-c7de308c19e2"  
    },  
    "owninguser": {  
        "@odata.etag": "W/\"2632435\"",  
        "fullname": "Impersonated User",  
        "azureactivedirectoryobjectid": "e39c5d16-675b-48d1-8e67-667427e9c084",  
        "systemuserid": "75df116d-d9da-e711-a94b-000d3a34ed47",  
        "ownerid": "75df116d-d9da-e711-a94b-000d3a34ed47"  
    }  
}
```

## Track entity data changes with change tracking and the Web API

Commonly, organizations need to synchronize data across multiple systems. Common Data Service provides a feature to efficiently manage this scenario, called *change tracking*. With change tracking enabled, you can build applications to query for these changes.

### Enable change tracking for an entity

You can enable change tracking on an entity by setting the **Enable change tracking for Flow** flag on the entity settings in the PowerApps maker portal, found under the **More settings > Create and update settings** section. As its name indicates, this feature can be used frequently by Power Automate to allow you to trigger automations based on changes that are made to a given entity record.



## Delta links

According to the OData 4.0 documentation, delta links are “opaque, service-generated links that the client uses to retrieve subsequent changes to a result.” Microsoft has built this standard into change tracking in Common Data Service to provide a common, standard method for interfacing with change data as it occurs over time. For more information on delta links, see **OData Version 4.0 - Delta Links<sup>23</sup>**.

After issuing a change tracking request, your response will contain a delta link, which you'll then pass to your next request to get changes that have occurred since the initial request. It will be up to your app to maintain the delta link, similarly to how you might have done tracking with date/time stamps in the past.

## Retrieve changes in entities by using the Web API

After you've enabled change tracking on an entity, you can retrieve its changes through the Web API by adding `odata.track-changes` as a preference header in your web request. The following example requests changes from the account entity for a specific set of fields (name, account number, telephone1, and fax).

```
GET [Organization URI]/org1/api/data/v9.1/accounts?$select=name,accountnumber,telephone1,fax HTTP/1.1
Prefer: odata.track-changes
Cache-Control: no-cache
OData-Version: 4.0
Content-Type: application/json
```

On the initial request, you'll receive the entire set of data that applies to the query because a delta has not been applied. However, the response will show an `@odata.deltaLink` value. This link contains a `deltatoken` parameter, which is intended to be passed into your subsequent request when you want to see what changes have been applied since this query was run. Think of this process as similar to how you would handle paging in OData.

The following example is a response to the preceding account request:

```
{
    "@odata.context": "[Organization URI]/data/v9.0/$metadata#ac-
```

<sup>23</sup> [https://docs.oasis-open.org/odata/odata/v4.0/cs01/part1-protocol/odata-v4.0-cs01-part1-protocol.html#\\_Toc365046305](https://docs.oasis-open.org/odata/odata/v4.0/cs01/part1-protocol/odata-v4.0-cs01-part1-protocol.html#_Toc365046305)

```
counts(name,telephone1,fax) /$delta",
    "@odata.deltaLink": "[Organization URI]/api/data/v9.1/ac-
counts?$select=name,telephone1,fax&$deltatoken=919058%2108%2f22%2f2017%20
08%3a21%3a20",
"value":
[
{
    "@odata.etag": "W/\"915244\",
    "name": "Monte Orton",
    "telephone1": "555000",
    "fax": "10101",
    "accountid": "60c4e274-0d87-e711-80e5-00155db19e6d"
},
{
    "@odata.context": "[Organization URI]/api/data/v9.0/$metadata-
#accounts/$deletedEntity",
    "id": "2e451703-c686-e711-80e5-00155db19e6d",
    "reason": "deleted"
}
]
```

In this scenario, consider that some time has passed, and you now want to see if any other changes have been applied. You can use the @odata.deltaLink parameter to run your query (or extract its deltatoken if you are managing connection strings differently), and see what changes have happened since your last request.

The following example shows how you would retrieve changes since that last request by running the returned @odata.deltaLink:

```
GET [Organization URI]/api/data/v9.1/accounts?$select=name,accountnum-
ber,telephone1,fax&$deltatoken=919042%2108%2f22%2f2017%2008%3a10%3a44
```

In this scenario, imagine that a new account was created and another one was deleted. Your response would then look similar to the following example:

```
{
    "@odata.context": "[Organization URI]/api/data/v9.1/$metadata#ac-
counts(name,telephone1,fax) /$delta",
    "@odata.deltaLink": "[Organization URI]/api/data/v9.1/ac-
counts?$select=name,telephone1,fax&$deltatoken=919058%2108%2f22%2f2017%20
08%3a21%3a20",
"value":
[
{
    "@odata.etag": "W/\"915244\",
    "name": "Monte Orton",
    "telephone1": "555000",
    "fax": "10101",
    "accountid": "60c4e274-0d87-e711-80e5-00155db19e6d"
```

```
    },
    {
        "@odata.context": "[Organization URI]/api/data/v9.1/$metadata-
#accounts/$deletedEntity",
        "id": "2e451703-c686-e711-80e5-00155db19e6d",
        "reason": "deleted"
    }
]
}
```

As shown in the example, another delta link was provided, which you can then use the next time you need to see what changes have occurred.

## Retrieve the number of changes in a delta

You can also retrieve the number of changes made in a given delta by adding the \$count parameter to your request.

```
GET [Organization URI]/api/data/v9.1/accounts/$count?$deltato-
ken=919042%2108%2f22%2f2017%2008%3a10%3a44
```

## Summary

Common Data Service provides a robust API to enable interaction with its underlying data and actions. This module reviewed a number of ways that you can query and manipulate Common Data Service data and explained how to authenticate to a Web API endpoint by using OAuth. Additionally, this module covered a number of other topics such as user impersonation and how to track data changes with Common Data Service's change tracking feature.

# Integrate Common Data Service Azure solutions

## Common Data Service Azure Solutions overview

Because Common Data Service is built as part of the robust Microsoft Power Platform, it provides numerous capabilities to facilitate integration with Microsoft Azure solutions. This lesson covers some of the standard integration practices regarding Azure solutions and how they pertain to Common Data Service.

### Comparing Azure's integration solutions

Azure comes with a variety of other solutions, which can also be integrated with Common Data Service. The following sections provide a summary of each solution. However, this module will limit further education in future lessons to the solutions that have pre-configured utilities within Common Data Service that help streamline your integrations, such as Microsoft Azure Service Bus, Microsoft Azure Event Hubs, and Microsoft Azure Logic Apps.

### Logic Apps

Azure Logic Apps provides a robust visual interface in which you can orchestrate complex integrations across your various environments. It has many of the same capabilities as Microsoft Power Automate workflows, so if you are familiar with preparing workflows in that context, you should be able to recognize their similarities. Both provide the ability to use pre-built and custom-built connectors, which you can use to connect to whichever system is needed. Most importantly, Logic Apps comes with a Common Data Service connector that will allow you to trigger actions based on various Common Data Service events (such as a record being created or updated).

### Azure Service Bus

Azure Service Bus is Microsoft's cloud messaging as a service (MaaS) platform. Messages are sent to and received from queues for point-to-point communications. Service Bus also provides a publish-subscribe mechanism that uses its Topics feature, which will not be covered in this module.

### Azure API Management

Because API needs are handled by Common Data Service Web API, and because Common Data Service provides a role-based security model that can handle most integration scenarios, you might not need to consider Microsoft Azure API Management often. However, if you ever need to write a custom and/or proprietary API to sit on top of the Common Data Service Web API, you could use Azure's API Management services.

## Event Grid

Microsoft Azure Event Grid is an event-driven, publish-subscribe framework that allows you to handle various events. While Common Data Service does not provide out-of-the-box capabilities to integrate with an Event Grid instance as it does with Service Bus or Event Hubs, it is a viable item to consider when you are in need of event-driven integrations.

## Event Hubs

Azure Event Hubs is Microsoft's version of Apache Kafka and provides a real-time data ingestion service that supports millions of events per second. This service is good for large streams of data that need to be ingested in real time (which might occur when you are trying to capture items like application telemetry within a business application) but are not common in most business application integration scenarios. Technically, Event Hubs is not seen as an integration solution in Azure (it's seen as an analytics solution given that its predominant applications are with "big data"). Common Data Service provides a utility to publish events to an Event Hub. In the context of Common Data Service, while provided as an endpoint, Azure Service Bus traditionally becomes the preferred method for integration of actual business data because it comes with far less overhead.

## Choosing the right Azure integration solution

If you're struggling to figure out which Azure integration solution best suits your needs, consider the information in the following table.

IF YOU WANT TO...	USE THIS
Create workflows and orchestrate business processes to connect hundreds of services in the cloud and on-premises.	<b>Logic Apps</b> ( <a href="https://azure.microsoft.com/services/logic-apps/?azure-portal=true">https://azure.microsoft.com/services/logic-apps/?azure-portal=true</a> )
Connect on-premises and cloud-based applications and services to implement messaging workflows	<b>Service Bus</b> ( <a href="https://azure.microsoft.com/services/service-bus/?azure-portal=true">https://azure.microsoft.com/services/service-bus/?azure-portal=true</a> )
Publish your APIs for internal and external developers to use when connecting to backend systems that are hosted anywhere.	<b>API Management</b> ( <a href="https://azure.microsoft.com/services/api-management/?azure-portal=true">https://azure.microsoft.com/services/api-management/?azure-portal=true</a> )
Connect supported Azure and third-party services by using a fully managed event-routing service with a publish-subscribe model that simplifies event-based app development.	<b>Event Grid</b> ( <a href="https://azure.microsoft.com/services/event-grid/?azure-portal=true">https://azure.microsoft.com/services/event-grid/?azure-portal=true</a> )
Continuously ingest data in real time from up to hundreds of thousands of sources and stream a million events per second.	<b>Event Hubs</b> ( <a href="https://azure.microsoft.com/services/event-hubs/?azure-portal=true">https://azure.microsoft.com/services/event-hubs/?azure-portal=true</a> )

For in-depth guidance on Azure's broader Integration Services framework, refer to the Azure Integration Services whitepaper, which is found [here<sup>24</sup>](#).

Another article to reference is one found on the Azure website: [Choose between Azure messaging services - Event Grid, Event Hubs, and Service Bus<sup>25</sup>](#).

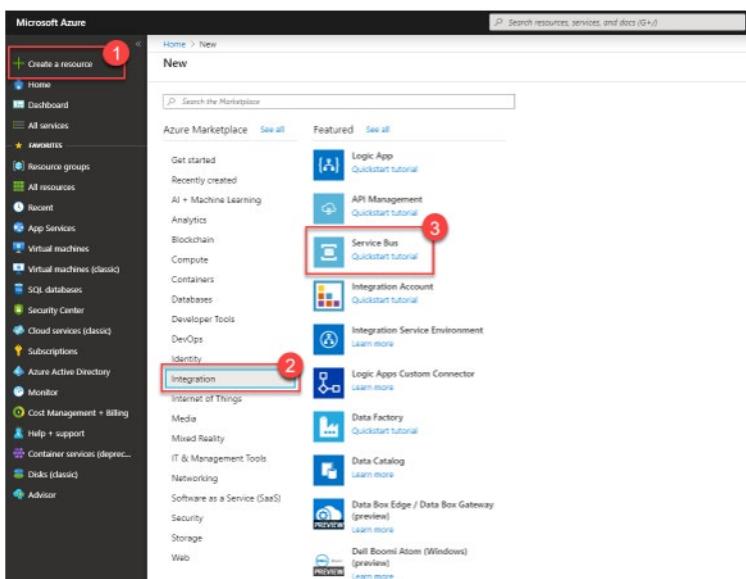
## Expose Common Data Service data to Azure Service Bus

Common Data Service provides a variety of pre-built mechanisms for externalizing its data for integration purposes. This lesson covers how to accomplish exposing Common Data Service data to Azure Service Bus by using Common Data Service's Service Endpoint Registration feature, which is found in its Plug-in Registration Tool.

### Set up your Azure Service Bus environment

Create your Azure Service Bus namespace and message queue with the following steps.

1. Sign in to the [Azure portal<sup>26</sup>](#).
2. In the left navigation pane of the portal, select **+ Create a resource**, select **Integration**, and then select **Service Bus**.



3. Enter the appropriate details for your namespace and then select **Create**.

<sup>24</sup> <https://azure.microsoft.com/product-categories/integration/?azure-portal=true>

<sup>25</sup> <https://docs.microsoft.com/azure/event-grid/compare-messaging-services/?azure-portal=true>

<sup>26</sup> <https://portal.azure.com/?azure-portal=true>

Home > New > Create namespace

Create namespace

Name \*

MSLearnSample

.servicebus.windows.net

Pricing tier (View full pricing details) \*

Basic

Subscription \*

Sandbox

Resource group \*

(New) MLearnSample

Create new

Location \*

South Central US

It might take a few minutes for your resource to provision. When it's finished, you should see something similar to the following image in the **Notifications** area of your Azure portal:

MS Learn Sample App - Manifest

Search (Ctrl+ /)

Save Discard Upload Download

The editor below allows you to update this application by directly modifying its JSON representation.

```
1 {
2     "id": "ada72c66-ad42-4ba9-92fd-3a6307d8e283",
3     "acceptMappedClaims": null,
4     "accessTokenAcceptedVersion": null,
5     "addIns": [],
6     "allowPublicClient": true, 2
7     "appId": "44af011c-c146-4128-b9ce-f00162044d75",
8     "appRoles": [],
9     "oauth2AllowUrlPathMatching": false,
10    "createdDateTime": "2019-11-09T22:32:23Z",
11    "groupMembershipClaims": null,
12    "identifierUris": [],
13    "informationalUrls": {
14        "termsOfService": null,
15        "support": null,
16        "privacy": null,
17        "marketing": null
18    },
19    "keyCredentials": [],
20    "knownClientApplications": [],
21    "logoUrl": null,
22    "logoutUrl": null,
23    "name": "MS Learn Sample App",
24    "oauth2AllowIdTokenImplicitFlow": true, 3
25    "oauth2AllowImplicitFlow": true, 4
26    "oauth2Permissions": [],
27    "oauth2RequirePostResponse": false,
28    "optionalClaims": null,
29    "orgRestrictions": [],
30    "parentalControlSettings": {
31        "countriesBlockedForMinors": [],
32        "legalAgeGroupRule": "Allow"
33    }
}
```

4. After your resource has been created, go to your newly created namespace to create a new queue.

**Request API permissions**

Select an API

Microsoft APIs   APIs my organization uses   My APIs

Commonly used Microsoft APIs

**Microsoft Graph**  
Take advantage of the tremendous amount of data in Office 365, Enterprise Mobility + Security, and Windows 10. Access Azure AD, Excel, Intune, Outlook/Exchange, OneDrive, OneNote, SharePoint, Planner, and more through a single endpoint.

**Azure Data Catalog**  
Programmatic access to Data Catalog resources to register, annotate and search data assets

**Azure DevOps**  
Integrate with Azure DevOps and Azure DevOps server

**Azure Rights Management Services**  
Allow validated users to read and write protected content

**Azure Service Management**  
Programmatic access to much of the functionality available through the Azure portal

**Data Export Service for Microsoft Dynamics 365**  
Export data from Microsoft Dynamics CRM organization to an external destination

**Dynamics 365 Business Central**  
Programmatic access to data and functionality in Dynamics 365 Business Central

**Dynamics CRM**  
Access the capabilities of CRM business software and ERP systems

**Flow Service**  
Embed flow templates and manage flows

**Intune**  
Programmatic access to Intune data

5. Note a few items that are found in your namespace so that Common Data Service has the correct credentials to connect to your new queue. For this procedure, use the existing Shared access policy that was auto created as part of your namespace. If you want further access limitations, you can create a Shared access policy for your individual queue as well.

**Request API permissions**

< All APIs

**Dynamics CRM**  
<https://admin.services.crm.dynamics.com/> Docs

What type of permissions does your application require?

**Delegated permissions**  
Your application needs to access the API as the signed-in user.

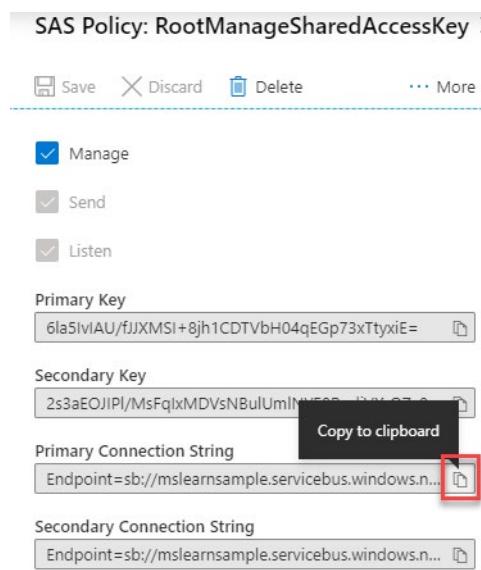
**Application permissions**  
Your application runs as a background service or daemon without a signed-in user.

Select permissions

Type to search

Permission	Admin Consent Required
<input checked="" type="checkbox"/> user_impersonation Access Common Data Service as organization users	

6. From within your Shared access policy, copy your **Primary Connection String** and store it for future use because you'll need this string as part of your Service Bus Endpoint configuration in Common Data Service:



## Register Service Bus Endpoint in Common Data Service

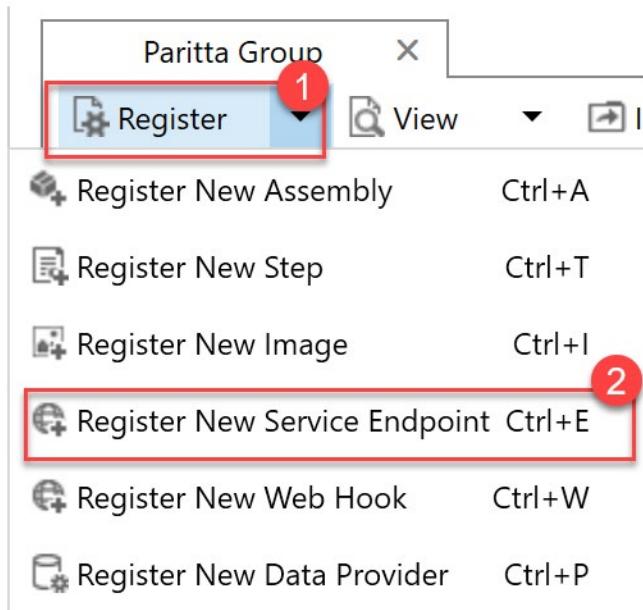
Now that you have set up a message queue in Azure, you can provide Common Data Service with the required configuration information to access it.

[!NOTE]

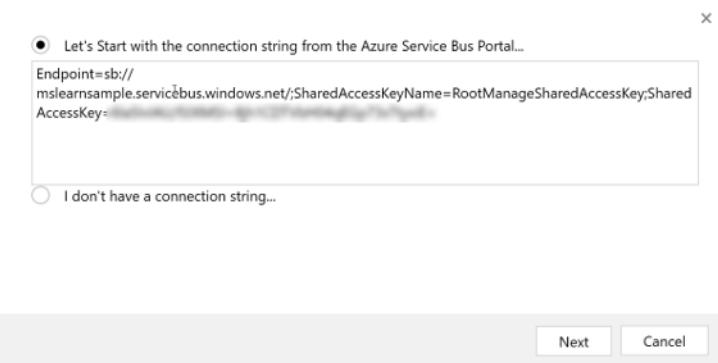
You will be using Common Data Service's Plug-in Registration Tool to configure the publishing of your Common Data Service data to your Service Bus. This tool is provided as part of Microsoft's Common Data Service developer tooling, which is found at NuGet. For more information on how to install the Plug-in Registration Tool through NuGet, see [Download tools from NuGet<sup>27</sup>](#).

1. Open the Plug-in Registration Tool and connect to your Common Data Service environment.
2. When connected to the environment, register your Service Bus Endpoint by selecting **Register** and then selecting **Register New Service Endpoint**.

<sup>27</sup> <https://docs.microsoft.com/powerapps/developer/common-data-service/download-tools-nuget>

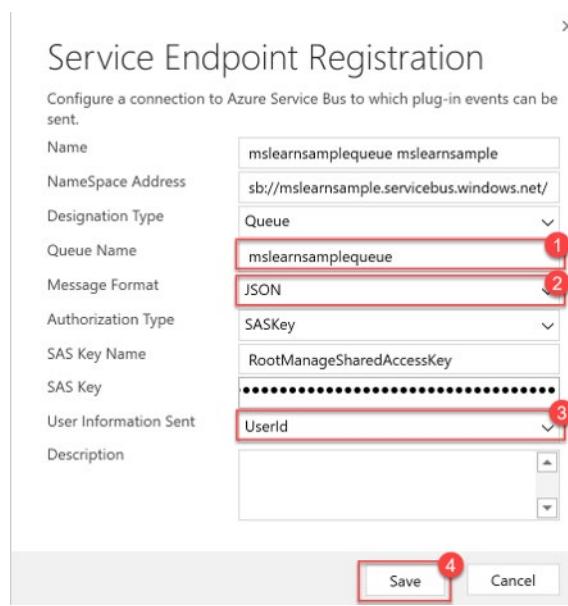


3. Copy and paste the **Primary Connection String** value that you referenced earlier when setting up your Service Bus instance, and then select **Next**.



4. All the fields from your connection string should prepopulate on the form. For this example, you'll be writing a one-way queue publisher, so you can leave the **Designation Type** as **Queue**. Common Data Service supports many other designation types to support various messaging protocols.

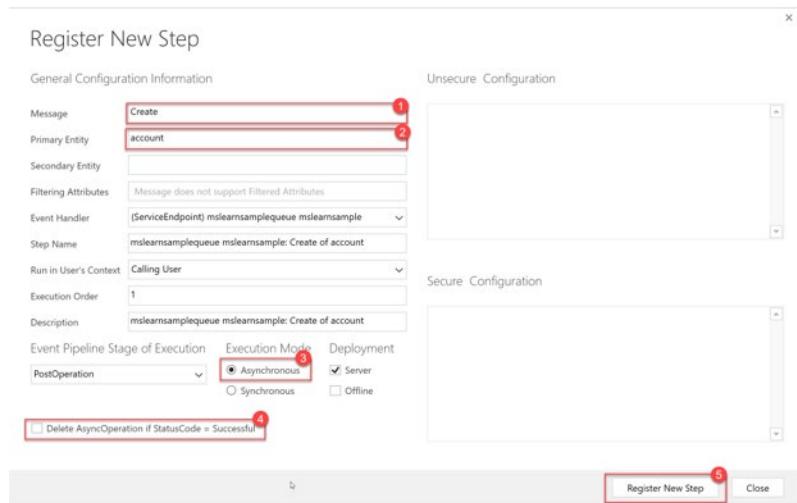
Enter your queue name into the **Queue Name** field and specify **Message Format** as **JSON**. Common Data Service supports the .NETBinary, JSON, and XML message formats. You are using JSON for this message because it's become an industry standard messaging format because of its portability and lightweight nature. Lastly, to have your user information sent to your queue, you can select **User Id** in the **User Information Sent** drop-down list.



## Register a Service Bus integration step

In this scenario, you'll register an integration step that publishes a message to a Service Bus queue every time an account is created. By registering a step, you can define the entity and message combination, and you can define what conditions will cause the message that is being processed by Common Data Service to be sent on the Service Bus to the Azure queue.

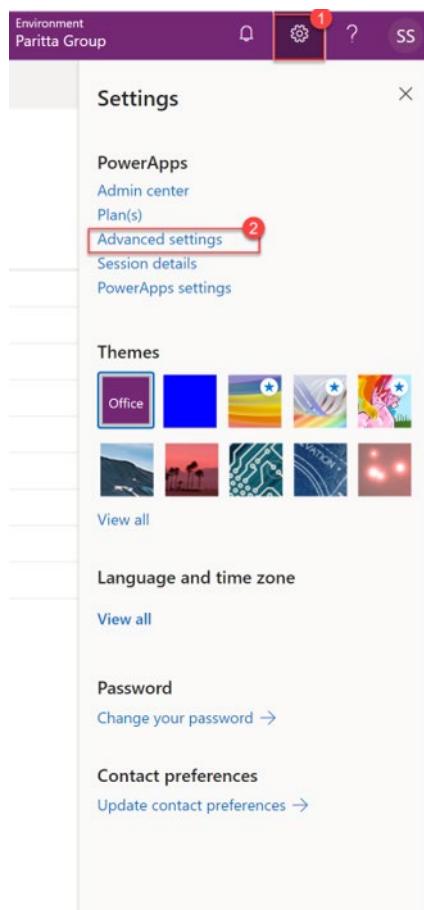
1. Register a new step for your Service Endpoint by right-clicking and selecting **Register New Step**.
2. Enter the following details to register a new integration step that will be started on creation of an account record. Make sure that you clear the **Delete AsyncOperation if StatusCode = Successful** flag. Clearing this flag is only for testing purposes so you can verify that the created System Job records show that the Service Bus integration step has successfully started on creation of an account record. In a real-world production scenario, we recommend that you leave this value selected.



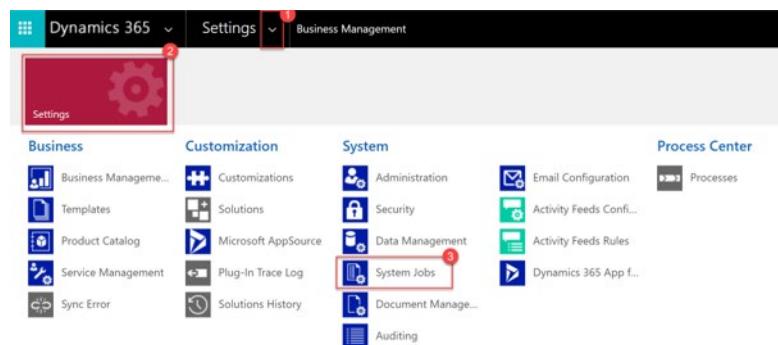
## Test your Service Bus integration

Test your Service Bus integration with the following steps:

1. To test your Service Bus integration, go to your Common Data Service environment and create an account.
2. To see if the integration ran, go to **Settings > Advanced Settings** and select the **System Jobs** view.



3. Go to the **System Jobs** view by going to **Settings > System Jobs**.



4. Verify that your integration step ran successfully by viewing it in the **System Jobs** view. If it ran successfully, the **Status Reason** should be **Succeeded**. You'll also use this view to troubleshoot integration runs in the event that an error occurs. In the case of failure, open the System Job record to view the error information.

System Jobs				
Enti...	All	View:	All System Jobs	
<input type="checkbox"/>	System Job T...	System Job Name	Regarding	Status Reason
	System Event	mslearnsamplequeue mslearnsample: Create of ...	My Test Account	Succeeded
	Calculate Roll...	Calculate rollup fields for the account entity		Succeeded SYSTEM

5. Because the integration step ran successfully, you can now verify that the account creation message has arrived in your Service Bus queue by going to the queue in the Azure portal.

A screenshot of the Azure Service Bus Queue overview page for 'mslearnsamplequeue'. The left sidebar shows 'Overview', 'Access control (IAM)', 'Diagnose and solve problems', 'Settings', 'Metrics (preview)', 'Properties', 'Locks', 'Export template', 'Support + troubleshooting', and 'New support request'. The main area shows the queue URL: 'https://MSLearnSample.servicebus.windows.net/mslearnsamplequeue'. It displays metrics: Active message count (1 MESSAGES), Scheduled message count (0 MESSAGES), Dead-letter message count (0 MESSAGES), Transfer message count (0 MESSAGES), and Transfer dead-letter message count (0 MESSAGES). A large circular progress bar indicates '100% FREE SINCE'.

## Types of supported Service Bus contracts

Common Data Service supports a variety of methods to consume Azure Service Bus queue messages: queue, one-way, two ways, or REST. If using two-way and REST, you are able to return a string of information back to Common Data Service.

## Queue

Queue listeners are not required for this type of queue. You can consume queued messages in a "destructive" or "non-destructive" read. A

destructive read will read the message from the queue and remove it, whereas a non-destructive read will not remove the message from the queue. This method is useful for “send-and-forget” scenarios where it is not critical that the message is received at a given point of time.

## Topic

Topic listeners are similar to queue listeners, except that one or more listeners can subscribe to receive messages for a given topic. This type is useful if you require multiple consumers for a given message.

## One-way

One-way contracts require that an active event listener be available to consume a message that is posted to the Service Bus queue. If no active listener is available, the post will fail. If the post fails, Common Data Service will retry posting the message in exponentially larger and larger time spans until the asynchronous system job is eventually canceled. In this case, the System Job status of this event will be set to **Failed**.

## Two-way

Two-way contracts are similar to one-way contracts, except that they also provide the ability to return a string value from the listener. If you've registered a custom Azure-aware plug-in to post your message, you can then consume this returned data within your plug-in. A common application of this scenario might be if you want to retrieve the ID of a record that was created in an external system as part of your listener's process to maintain it in your Common Data Service environment.

## REST

REST contracts are similar to two-way contracts except that it is exposed on a REST endpoint.

## Write a queue listener

In the previous exercise, you registered a Service Endpoint that publishes messages to a Service Bus Endpoint whenever account data is updated in your Common Data Service environment. This exercise now describes how to consume those messages.

1. Create a C# Console Application in Visual Studio that targets .NET 4.6.2 or higher.
2. Add the helper library, which is found in the resources section of this exercise.
3. Add the following NuGet packages:
  - WindowsAzure.ServiceBus
  - Microsoft.CrmSdk.CoreAssemblies
4. In the application's Main method, paste the following code. Replace the Endpoint URL with your Azure Service Bus Namespace's Endpoint URL and the queue name if it differs:

```
string connectionString =@[ENDPOINT URL];
string queueName = "mslearnsamplequeue";
QueueClient queueClient = QueueClient.CreateFromConnectionString(connectionString, queueName,
ReceiveMode.PeekLock);
```

5. To consume your message, use the “OnMessage” method, which provides the ability to process a Service Bus queue message in an event-driven message pump. For this exercise, you'll use Serializer methods that are provided by the Microsoft.Xrm.Sdk package so you can serialize the incoming message into the Common Data Service RemoteExecutionContext data type. Depending on the type of message that was registered, you need to serialize based on the corresponding message format (.NETBinary, JSON, or XML).

```
queueClient.OnMessage(message =>
{
    //get RemoteExecutionContext based on Message Format
    RemoteExecutionContext context = null;

    if (message.ContentType == "application/msbin1") //NETBinary Message Format
    {
        context = message.GetBody<RemoteExecutionContext>();
    }
    else if (message.ContentType == "application/json") //JSON Message Format
    {
        context = message.GetBody<RemoteExecutionContext>(
            new DataContractJsonSerializer(typeof(RemoteExecutionContext)));
    }
    else if (message.ContentType == "application/xml") //XML Message Format
    {
        context = message.GetBody<RemoteExecutionContext>(
            new DataContractSerializer(typeof(RemoteExecutionContext)));
    }
    try
    {
        //serialize the entire context leveraging the SerializeContext extension method
        Console.WriteLine(context.SerializeContext());
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.ToString());
    }
});
```

6. Lastly we are going to add a Console.ReadLine() to our main method to allow for multiple messages to be processed. Note this is not a scalable method for handling event-processing however is sufficient enough for our exercise's purposes. You'd want to have a more scalable solution that you host in an Azure Durable function or other service of your preference.

```
Console.ReadLine();
```

7. Hit F5 to run your application. If there are already messages in your queue from your previous exercise, they should get processed and their message contents should be displayed on the console screen. If not, you can invoke an update by making an update to an Account in your Common Data Service environment.

## Publish Common Data Service events with webhooks

Another method for publishing events from Common Data Service to an external service is to register webhooks. A webhook is an HTTP-based mechanism for publishing events to any Web API-based service of your choosing. This method allows you to write your own custom code that is hosted on external services as a point-to-point.

### Webhooks vs. Azure Service Bus

When considering integration mechanisms, you have a few available options. It's important that you consider various elements when choosing a given method.

Consider using Azure Service Bus when:

- High scale asynchronous processing/queueing is a requirement.
- Multiple subscribers might be needed to consume a given Common Data Service event.
- You want to govern your integration architecture in a centralized location.

Consider using webhooks when:

- Synchronous processing against an external system is required as part of your process (Common Data Service only supports asynchronous processing against Service Bus Endpoints).
- The external operation that you are performing needs to occur immediately.
- You want the entire transaction to fail unless the webhook payload is successfully processed by the external service.
- A third-party Web API endpoint already exists that you want to use for integration purposes.
- SAS authentication isn't preferred and/or feasible (webhooks support authentication through authentication headers and query string parameter keys).

### Webhook authentication options

The following table describes the three authentication options that you can use to consume a webhook message from a given endpoint.

Type	Description
HttpHeader	Includes one or more key value pairs in the header of the HTTP request. Example: Key1: Value1, Key2: Value2

Type	Description
WebhookKey	Includes a query string by using code as the key and a value that is required by the endpoint. When registering the webhook by using the Plug-in Registration Tool, only enter the value. Example: ?code=00000000-0000-0000-0000-000000000001
HttpQueryString	Includes one or more key value pairs as query string parameters. Example: ?Key1=Value1&Key2=Value2

## Webhook HTTP headers

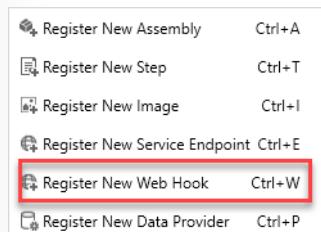
The following table shows the HTTP headers that are passed to your service as part of a webhook call. You can use these headers as part of your processing method if you are writing a new webhook processor.

Key	Value Description
x-request-id	A unique identifier for the request
x-ms-dynamics-organization	The name of the tenant who sent the request
x-ms-dynamics-entity-name	The logical name of the entity that passed in the execution context data
x-ms-dynamics-request-name	The name of the event that the webhook step was registered for
x-ms-correlation-request-id	Unique identifier for tracking any type of extension. This property is used by the platform for infinite loop prevention. In most cases, this property can be ignored. This value can be used when you are working with technical support because it can be used to query telemetry to understand what occurred during the entire operation.
x-ms-dynamics-msg-size-exceeded	Sent only when the HTTP payload size exceeds the 256KB

## Register a Webhook endpoint

Webhook endpoint registration is performed similarly to Service Endpoint registration, by using the Plug-in Registration Tool.

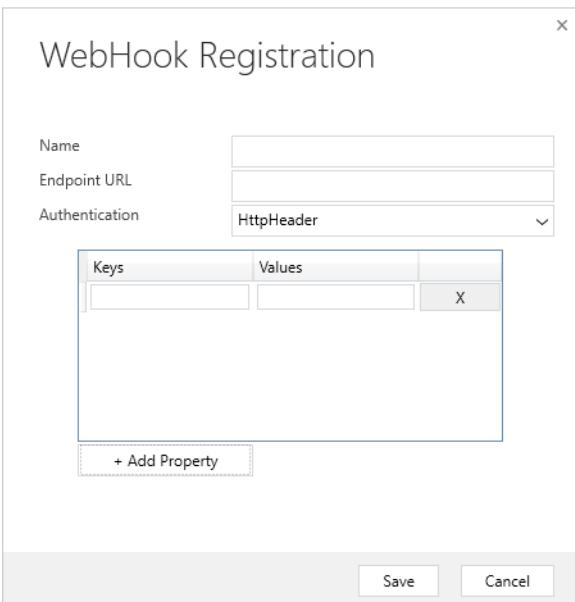
Within the Plug-in Registration Tool, you can register a new webhook by selecting **Register New Web Hook** under the **Register** menu option.



The following **WebHook Registration** dialog box will appear, where you can configure the URL of your endpoint, along with any authentication options.

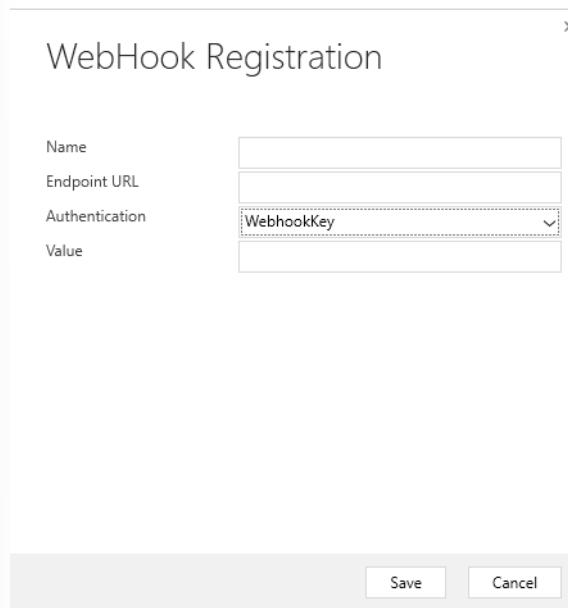
## Register with HTTPHeader Authentication

If **HttpHeader** authentication is selected, the screen will prompt you to add **Keys** and **Values** that will be passed as part of your HTTP request. Commonly, the keys and values might include an OAuth bearer token or other various authentication formats.



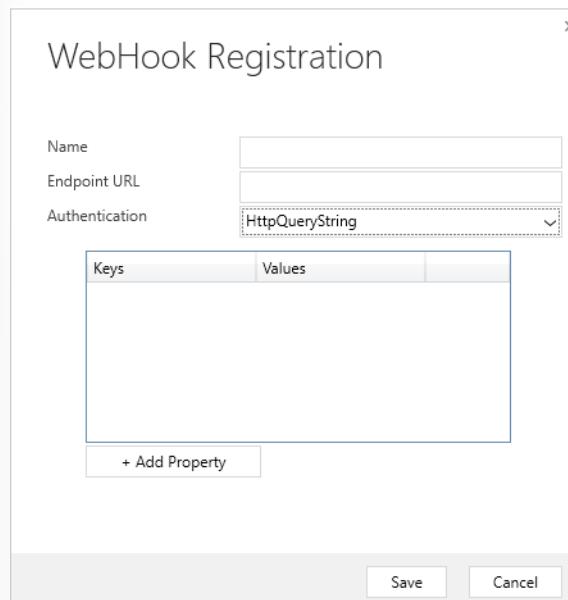
## Register with WebhookKey Authentication

If **WebhookKey** is specified as the **Authentication** method, a query string is passed to the URL with the given key in the format ?code=[web hook key]. This method is handy when you are calling Azure Functions because it uses this code parameter by default to perform its authentication.



## Register with HTTPQueryString Authentication

You can pass Query String parameters by specifying **HttpQueryString** as the **Authentication** option. As with the **HTTPHeader** option, it presents the option to pass a set of key/value pairs to your Web API. You could also pass additional parameters, and even manually pass the "code" parameter that is expected through Azure Functions in this manner.



## Processes Common Data Service events

The previous exercise reviewed how to register webhooks that expose Common Data Service data to an external Web API. In this

exercise, you will build an example Web API by using Azure Functions to illustrate how to consume a published webhook event.

## Azure Functions vs. plug-ins

Microsoft Azure Functions provides a great mechanism for doing small units of work, similar to what you would use plug-ins for in Common Data Service. In many scenarios it might make sense to offload this logic into a separate component, such as an Azure Function, to reduce load on the Common Data Service's application host. You have the availability to run functions in a synchronous capacity because Common Data Service webhooks provide the Remote Execution Context of the given request.

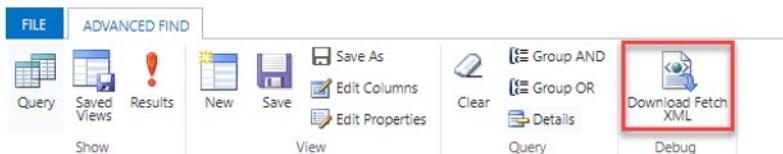
However, Azure Functions does not explicitly run within the Common Data Service's implementation pipeline, so if you need to update data in the most high-performing manner, such as autoformatting a string value before it posts to Common Data Service, we still recommend that you use a plug-in to perform this type of operation.

## Write an Azure Function that processes Common Data Service events

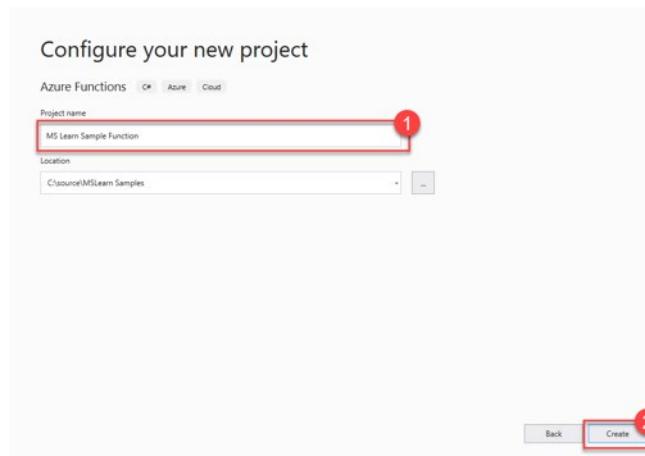
To start writing an Azure Function that processes Common Data Service events, you will use Visual Studio 2019's Azure development template to create and publish your Function. Visual Studio provides a number of tools that are available to help make Azure development simple. Therefore, you are required to have Azure Development Tools installed in your Visual Studio 2019 instance. If you don't have the feature installed, you can add it through the Visual Studio Installer.

## Create your Azure Function project

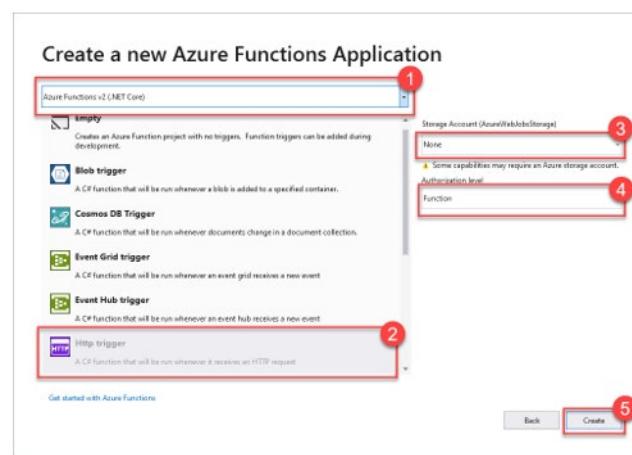
1. Create a new Azure Function project by using the Azure Functions template. You can find this template by creating a new project and then entering "function" in the search bar.



2. Give your Function project a descriptive name and then select **Create**.



3. Specify the Azure Functions v2 (.NET Core) type and ensure that **Http trigger** template is selected. You can set **Storage Account** to **None** because you won't be using storage for this exercise; however, you might need it in other scenarios. Set the **Authorization level** to **Function** and then select **Create**.



Your sample project should be created now, with the following template code found in the Function's .cs file:

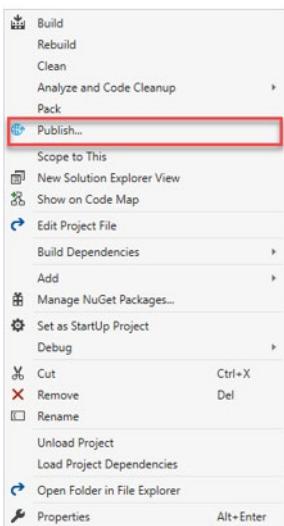
```
public static class Function1
{
    [FunctionName("Function1")]
    public static async Task<IActionResult> Run(
        [HttpTrigger(AuthorizationLevel.Function, "get", "post", Route
= null)] HttpRequest req,
        ILogger log)
    {
        log.LogInformation("C# HTTP trigger function processed a re-
quest.");
        string name = req.Query["name"];
        string requestBody = await new StreamReader(req.Body).ReadToEn-
```

```
dAsync() ;  
    dynamic data = JsonConvert.DeserializeObject(requestBody);  
    name = name ?? data?.name;  
  
    return name != null  
        ? (ActionResult)new OkObjectResult($"Hello, {name}")  
        : new BadRequestObjectResult("Please pass a name on the  
query string or in the request body");  
    }  
}
```

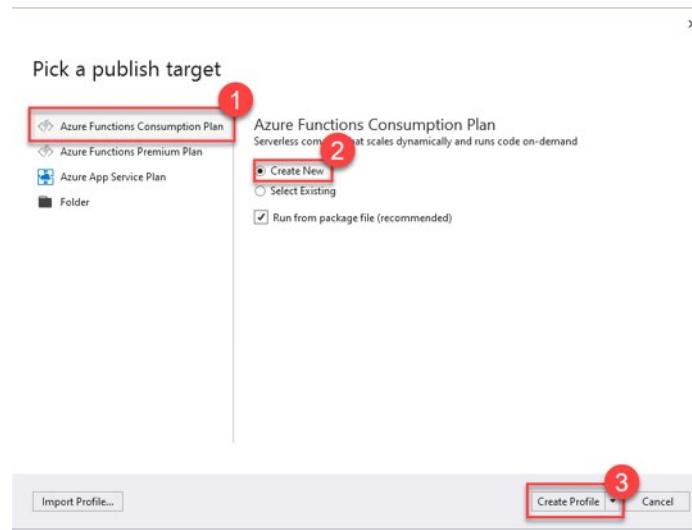
You will be replacing this code later, but first, you will publish your Function to ensure that everything works correctly.

## Publish your Azure Function to Azure

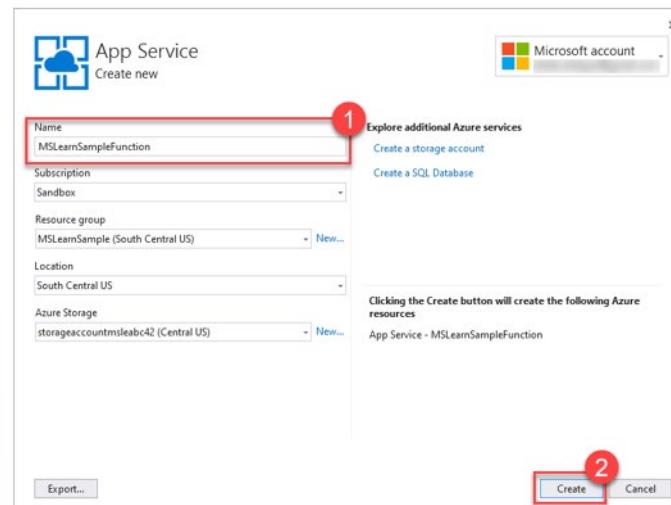
1. Right-click your project and select **Publish...** from the context menu to test the publishing of your Function to Azure App Service.



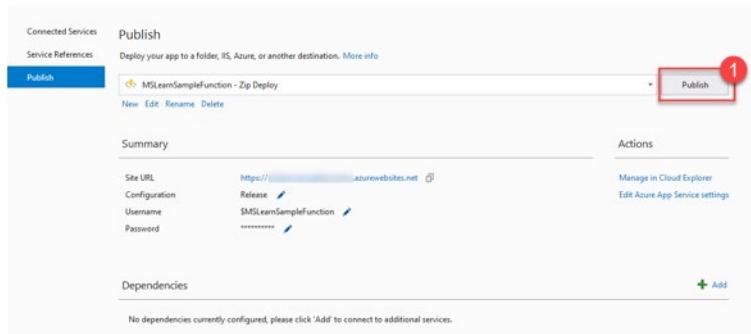
2. Create a new publish target if you haven't already. If you'd like to select an existing App Service plan, you can skip this step and select the **Select Existing** option instead.



3. If you are creating a new publish target, name your new App Service, verify that its **Subscription**, **Resource group**, **Location**, and **Azure Storage** fields are correct, and then select **Create**.



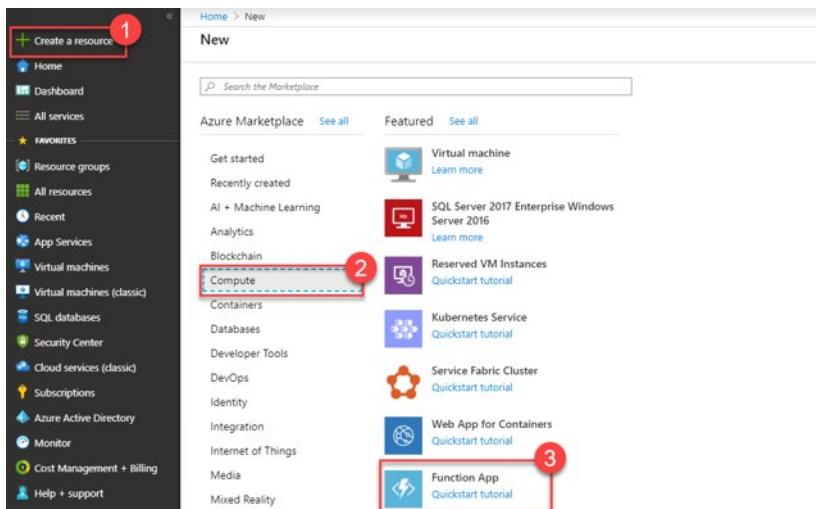
4. After your Publish profile is created, select **Publish** to deploy your Function to Azure. The Function is published by default in release mode. If you'd like to debug this function (more on this later), you'll want to publish the Function in Debug mode.



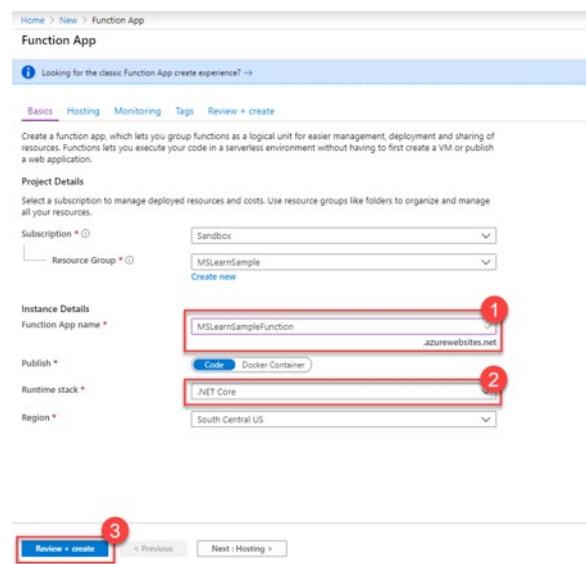
## Another method to create Azure Functions

If you want to manually create your Azure Function without the help of Visual Studio 2019, you can do so from the Azure portal:

1. Sign in to your Azure environment and create a Function App by selecting **Create a resource**.



2. To create an Azure Function App, specify its name and runtime stack, and then verify that the **Subscription**, **Resource group**, and **Region** fields are correct.



## [!NOTE]

This lesson doesn't cover the details of building a new Azure Function assembly.

## Update your Function's logic to interact with Common Data Service data

1. If needed, change your Function's `FunctionName` and corresponding class name to something more meaningful (that is, `MSLearnFunction`).
2. Copy and paste the `RemoteExecutionContextHelper.cs` file that is found in the Resources section of this exercise into your project. The file contains the following code, which will print Remote Execution Context details of the Common Data Service message:

```
public static string SerializeContext(dynamic context)
```

3. Add the following `using` statements to your Function:

```
using Newtonsoft.Json.Linq;
```

```
using RemoteExecutionContextHelper;
```

4. Copy and paste the following code into your Function's `Run` method:

```
log.LogInformation("C# HTTP trigger function processed a request.");  
  
string queryParams = "";  
foreach (var q in req.Query)  
{  
    queryParams += $"Key: {q.Key} Value: {q.Value}\n";  
}
```

```

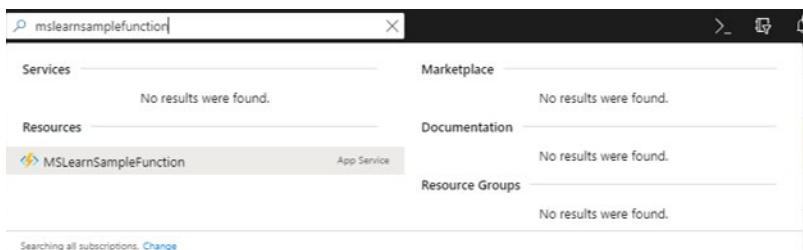
        string requestBody = await new StreamReader(req.Body).ReadToEndAsync();
        dynamic data = JsonConvert.DeserializeObject(requestBody);
        string requestHeader = "";
        foreach (var h in req.Headers)
        {
            requestHeader += $"Key: {h.Key} Value: {h.Value}\n";
        }
        log.LogInformation("Query Parameters:\n" + queryParams);
        log.LogInformation("Request Header: \n" + requestHeader);
        log.LogInformation("Request Body:\n" + requestBody);
        string requestBodyFormatted = JValue.Parse(requestBody).ToString(Formatting.Indented);
        log.LogInformation("Request Body Formatted:\n" + requestBodyFormatted);

        try
        {
            string context = RemoteExecutionContextSerializer.SerializeContext(data);
            log.LogInformation("Context: " + context);

            return (ActionResult)new OkObjectResult(data.InitiatingUserId);
        }
        catch (Exception ex)
        {
            return new BadRequestObjectResult(ex.ToString());
        }
    }
}

```

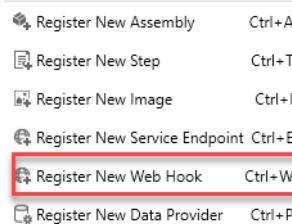
5. Build your Function and publish it to Azure by right-clicking the project and then selecting **Publish....**
6. Verify that your Function has been published by going to the Azure portal. You can either manually select it from within the resource group that you specified when you created the Function, or you can search for it by name in the Azure portal, as shown in the following image.



## Register a Common Data Service webhook that calls your Azure Function

In this exercise, you'll use the Plug-in Registration Tool to register a webhook that calls your new Azure Function.

1. Open the Plug-in Registration Tool and sign in to your Common Data Service environment.
2. Register a new webhook by selecting **Register New Web Hook** under the **Register** menu option.



3. Get your Function's URL from the Azure portal by selecting **Get function URL**.

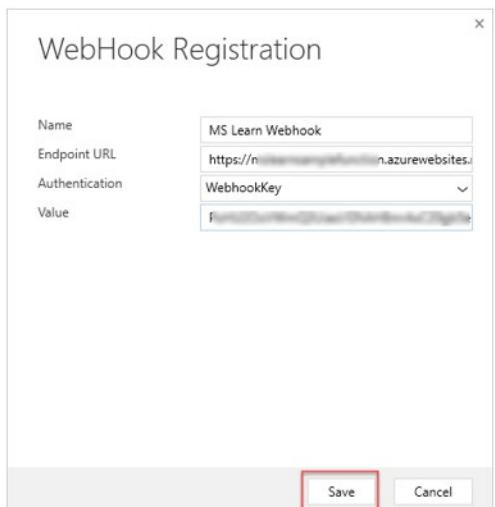
A screenshot of the Azure Functions blade for the 'MSLearnSampleFunction - MsLearnFunction' app. The left sidebar shows the 'Functions (Read Only)' section with 'f MsLearnFunction' selected (marked with a red circle). The right pane shows the 'function.json' file content and a 'Get function URL' button (marked with a red box).

```
1 {
2   "generatedBy": "Microsoft.NET.Sdk.Functions-1.0.29",
3   "configurationSource": "attributes",
4   "bindings": [
5     {
6       "type": "httpTrigger",
7       "methods": [
8         "get",
9         "post"
10      ],
11      "route": "MsLearnFunction"
12    }
13  ]
14}
```

4. Paste the copied value into a text editor, which should look like the following image.

`https://[AppServiceUrl].azurewebsites.net/api/MsLearnFunction?code=[WebhookKey]`

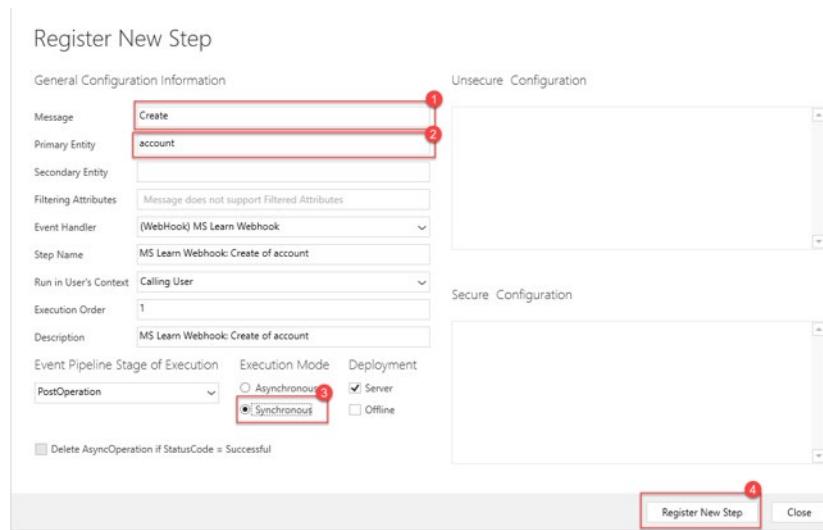
5. Cut and paste the code query string value from the copied URL and place into the **Value** section of the WebHook Registration string (make sure to remove the **code=** portion).



6. Register a new step that will post a message on creation of a new account. Register a new step by right-clicking your new webhook assembly and then selecting **Register New Step**.



7. Configure the step as illustrated in the following figure. Because you will be building this webhook to run synchronously, ensure that the flag is set when you are registering the new step.



## Test your webhook integration

1. To test your Service Bus integration, go to your Common Data Service environment and create an account.
2. Go to your Function in the Azure portal and view the log details of your function's invocation.

3. Browse through each log entry to see what data was passed from your Common Data Service webhook to your Azure Function.

Invocation Details X

Run in Application Insights

DATE (UTC)	MESSAGE	LOG LEVEL
2019-12-02 01:09:24.571	Executing 'MsLearnSampleFunction' (Reason='This functi...')	Information
2019-12-02 01:09:27.606	C# HTTP trigger function processed a request.	Information
2019-12-02 01:10:00.252	Query Parameters: Key: code Value: 4diFftirrJlabvfnCV1r...	Information
2019-12-02 01:10:00.402	Request Header: Key: Connection Value: Keep-Alive Key: ...	Information
2019-12-02 01:10:00.415	Request Body: {"BusinessUnitId": "10201b1a-6d3c-e711-8...	Information
2019-12-02 01:10:00.473	Request Body Formatted: { "BusinessUnitId": "10201b1a-6...	Information
2019-12-02 01:10:00.721	Context: ----- Userid: c8a260d3-e023-e911-a972-000...	Information
2019-12-02 01:10:00.751	Executed 'MsLearnSampleFunction' (Succeeded, Id=0b2ef...	Information

Executing 'MsLearnSampleFunction' (Reason='This function was programmatically called via the host APIs.',  
Id=0b2effbe-8180-4fd8-add2-9ec20930ee78)

## Summary

Common Data Service has a wealth of pre-built methods for interacting with Azure solutions. Whether you are intending to integrate with external systems or distribute workloads away from your Common Data Service environment, the Microsoft Power Platform provides a wealth of utilities to make a seamless and relatively easy experience. In this module, we reviewed how to integrate with various Azure solutions such as Azure Service Bus and Azure Functions, along with providing an overview of Azure Solutions that are commonly used to integrate with Common Data Service.