

# Table of Contents

## Overview

[What are logic apps?](#)

[B2B with Enterprise Integration Pack](#)

[Connectors](#)

## Get started

[Create your first logic app](#)

[Templates for logic apps](#)

[Create logic apps from templates](#)

## How To

### Build

[Build and deploy logic apps with Visual Studio](#)

[Add conditions and run workflows](#)

[Add switch statements](#)

[Add custom code with Azure Functions](#)

[Loops, scopes, and data debatching](#)

[Author logic app definitions](#)

[Call, trigger, or nest logic apps](#)

[Create custom APIs for logic apps](#)

[Call custom APIs for logic apps](#)

[Handle errors and exceptions](#)

[Handle content types](#)

[Secure your logic apps](#)

[Troubleshooting](#)

[Limits and config](#)

[B2B with Enterprise Integration Pack](#)

[Integration accounts](#)

[Partners](#)

[Agreements](#)

[B2B processing](#)

- [XML processing](#)
- [Flat file processing](#)
- [Validate XML](#)
- [Add schemas for XML validation](#)
- [Transform XML](#)
- [Add maps for XML transform](#)
- [Add certificates for B2B security](#)
- [Store metadata for artifacts](#)
- [AS2 enterprise integration](#)
- [AS2 encoding](#)
- [AS2 decoding](#)
- [EDIFACT enterprise integration](#)
- [EDIFACT encoding](#)
- [EDIFACT decoding](#)
- [X12 enterprise integration](#)
- [X12 encoding](#)
- [X12 decoding](#)
- [Disaster recovery](#)
- [Access on-premises data](#)
- [Connect to on-premises data](#)
- [Install data gateway](#)
- [Automate & deploy](#)
- [Create automated deployment templates](#)
- [Publish from Visual Studio](#)
- [Manage & monitor](#)
- [Manage logic apps with Visual Studio](#)
- [Monitor logic apps](#)
- [Manage integration accounts](#)
- [Monitor B2B messages](#)
- [Examples, scenarios, and walkthroughs](#)
- [Overview](#)
- [Create a serverless social dashboard](#)

[Call logic apps with Azure Functions](#)

[Add error and exception handling](#)

[B2B processing](#)

[Pricing & billing](#)

[Pricing](#)

[Usage metering](#)

[Reference](#)

[Workflow definition language](#)

[Workflow actions and triggers](#)

[REST API](#)

[PowerShell](#)

[Schema History](#)

[GA](#)

[Preview](#)

[Resources](#)

[Service updates](#)

[MSDN Forum](#)

[Stack Overflow](#)

# What are Logic Apps?

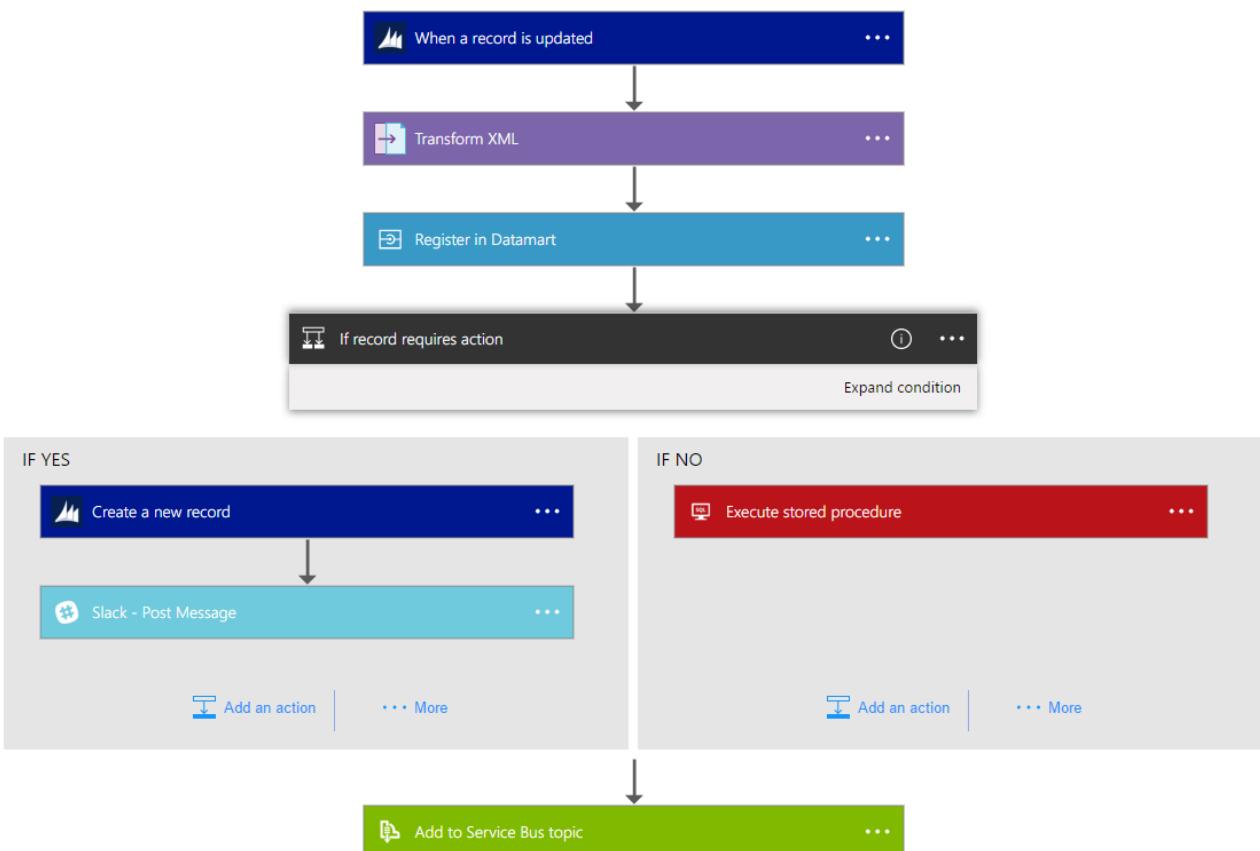
1/25/2017 • 4 min to read • [Edit Online](#)

Logic Apps provide a way to simplify and implement scalable integrations and workflows in the cloud. It provides a visual designer to model and automate your process as a series of steps known as a workflow. There are [many connectors](#) across the cloud and on-premises to quickly integrate across services and protocols. A logic app begins with a trigger (like 'When an account is added to Dynamics CRM') and after firing can begin many combinations actions, conversions, and condition logic.

The advantages of using Logic Apps include the following:

- Saving time by designing complex processes using easy to understand design tools
- Implementing patterns and workflows seamlessly, that would otherwise be difficult to implement in code
- Getting started quickly from templates
- Customizing your logic app with your own custom APIs, code, and actions
- Connect and synchronise disparate systems across on-premises and the cloud
- Build off of BizTalk server, API Management, Azure Functions, and Azure Service Bus with first-class integration support

Logic Apps is a fully managed iPaaS (Integration Platform as a Service) allowing developers not to have to worry about building hosting, scalability, availability and management. Logic Apps will scale up automatically to meet demand.



As mentioned, with Logic Apps, you can automate business processes. Here are a couple examples:

- Move files uploaded to an FTP server into Azure Storage
- Process and route orders across on-premises and cloud systems

- Monitor all tweets about a certain topic, analyze the sentiment, and create alerts and tasks for items needing followup.

Scenarios such as these can be configured all from the visual designer and without writing a single line of code. Get started [building your logic app now](#). Once written - a logic app can be [quickly deployed and reconfigured](#) across multiple environments and regions.

## Why Logic Apps?

Logic Apps brings speed and scalability into the enterprise integration space. The ease of use of the designer, variety of available triggers and actions, and powerful management tools make centralizing your APIs simpler than ever. As businesses move towards digitalization, Logic Apps allows you to connect legacy and cutting-edge systems together.

Additionally, with our [Enterprise Integration Account](#) you can scale to mature integration scenarios with the power of a [XML messaging, trading partner management](#), and more.

- **Easy to use design tools** - Logic Apps can be designed end-to-end in the browser or with Visual Studio tools. Start with a trigger - from a simple schedule to when a GitHub issue is created. Then orchestrate any number of actions using the rich gallery of connectors.
- **Connect APIs easily** - Even composition tasks that are easy to describe are difficult to implement in code. Logic Apps makes it easy to connect disparate systems. Want to connect your cloud marketing solution to your on-premises billing system? Want to centralize messaging across APIs and systems with an Enterprise Service Bus? Logic apps are the fastest, most reliable way to deliver solutions to these problems.
- **Get started quickly from templates** - To help you get started we've provided a [gallery of templates](#) that allow you to rapidly create some common solutions. From advanced B2B solutions to simple SaaS connectivity, and even a few that are just 'for fun' - the gallery is the fastest way to get started with the power of Logic Apps.
- **Extensibility baked-in** - Don't see the connector you need? Logic Apps is designed to work with your own APIs and code; you can easily create your own API app to use as a custom connector, or call into an [Azure Function](#) to execute snippets of code on-demand.
- **Real integration horsepower** - Start easy and grow as you need. Logic Apps can easily leverage the power of BizTalk, Microsoft's industry leading integration solution to enable integration professionals to build the solutions they need. Find out more about the [Enterprise Integration Pack](#).

## Logic App Concepts

The following are some of the key pieces that comprise the Logic Apps experience.

- **Workflow** - Logic Apps provides a graphical way to model your business processes as a series of steps or a workflow.
- **Managed Connectors** - Your logic apps need access to data and services. Managed connectors are created specifically to aid you when you are connecting to and working with your data. See the list of connectors available now in [managed connectors](#).
- **Triggers** - Some Managed Connectors can also act as a trigger. A trigger starts a new instance of a workflow based on a specific event, like the arrival of an e-mail or a change in your Azure Storage account.
- **Actions** - Each step after the trigger in a workflow is called an action. Each action typically maps to an operation on your managed connector or custom API apps.
- **Enterprise Integration Pack** - For more advanced integration scenarios, Logic Apps includes capabilities from BizTalk. BizTalk is Microsoft's industry leading integration platform. The Enterprise Integration Pack connectors allow you to easily include validation, transformation, and more in to your Logic App workflows.

## Getting Started

- To get started with Logic Apps, follow the [create a Logic App](#) tutorial.
- [View common examples and scenarios](#)
- [You can automate business processes with Logic Apps](#)
- [Learn How to Integrate your systems with Logic Apps](#)

# Overview: B2B scenarios and communication with the Enterprise Integration Pack

2/15/2017 • 2 min to read • [Edit Online](#)

For business-to-business (B2B) workflows and seamless communication with Azure Logic Apps, you can enable enterprise integration scenarios with Microsoft's cloud-based solution, the Enterprise Integration Pack.

Organizations can exchange messages electronically, even if they use different protocols and formats. The pack transforms different formats into a format that organizations' systems can interpret and process. Organizations can exchange messages through industry-standard protocols, including [AS2](#), [X12](#), and [EDIFACT](#). You can also secure messages with both encryption and digital signatures.

If you are familiar with BizTalk Server or Microsoft Azure BizTalk Services, the Enterprise Integration features are easy to use because most concepts are similar. One major difference is that Enterprise Integration uses integration accounts to simplify the storage and management of artifacts used in B2B communications.

Architecturally, the Enterprise Integration Pack is based on "integration accounts". These accounts are cloud-based containers that store all your artifacts, like schemas, partners, certificates, maps, and agreements. You can use these artifacts to design, deploy, and maintain your B2B apps and also to build B2B workflows for logic apps. But before you can use these artifacts, you must first link your integration account to your logic app. After that, your logic app can access your integration account's artifacts.

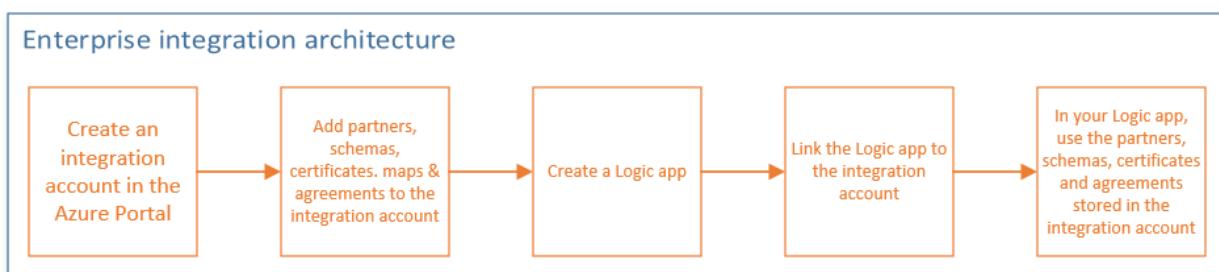
## Why should you use enterprise integration?

- With enterprise integration, you can store all your artifacts in one place -- your integration account.
- You can build B2B workflows and integrate with third-party software-as-service (SaaS) apps, on-premises apps, and custom apps by using the Azure Logic Apps engine and all its connectors.
- You can create custom code for your logic apps with Azure functions.

## How to get started with enterprise integration?

You can build and manage B2B apps with the Enterprise Integration Pack through the Logic App Designer in the [Azure portal](#). You can also manage your logic apps with [PowerShell](#).

Here are the high-level steps you must take before you can create apps in the Azure portal:



## What are some common scenarios?

Enterprise Integration supports these industry standards:

- EDI - Electronic Data Interchange
- EAI - Enterprise Application Integration

## Here's what you need to get started

- An Azure subscription with an integration account
- Visual Studio 2015 to create maps and schemas
- Microsoft Azure Logic Apps Enterprise Integration Tools for Visual Studio 2015 2.0

## Try it now

[Deploy a fully operational sample AS2 send & receive logic app](#) that uses the B2B features for Azure Logic Apps.

## Learn more

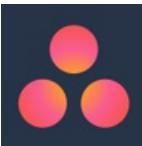
- [Agreements](#)
- [Business to Business \(B2B\) scenarios](#)
- [Certificates](#)
- [Flat file encoding/decoding](#)
- [Integration accounts](#)
- [Maps](#)
- [Partners](#)
- [Schemas](#)
- [XML message validation](#)
- [XML transform](#)
- [Enterprise Integration Connectors](#)
- [Integration Account Metadata](#)
- [Monitor B2B messages](#)
- [Tracking B2B messages in OMS portal](#)

# Connectors list

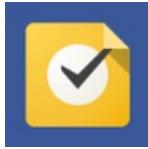
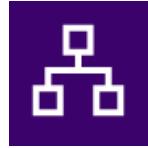
3/30/2017 • 2 min to read • [Edit Online](#)

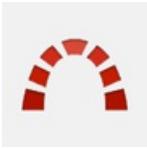
Select a connector to learn how to build workflows quickly.

## Standard connectors

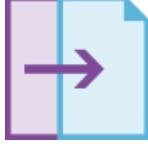
CONNECTORS			
 API/Web App	 Appfigures	 Asana	 Azure DocumentDB
 Azure ML	 Azure Functions	 Azure Blob Storage	 Basecamp 3
 Bitly	 BizTalk Server	 Blogger	 Box
 Campfire	 Cognitive Services Text Analytics	 Common Data Service	 DB2
 Delay	 Dropbox	 Dynamics 365	 Dynamics 365 for Financials
 Dynamics 365 for Operations	 Easy Redmine	 Facebook	 File System

## CONNECTORS

 FTP	 GitHub	 Google Calendar	 Google Drive
 Google Sheets	 Google Tasks	 HipChat	 HTTP
 HTTP Swagger	 HTTP Request	 HTTP Response	 Informix
 Insightly	 Instagram	 Instapaper	 JIRA
 MailChimp	 Mandrill	 Microsoft Translator	 Nested Logic App
 Office 365 Outlook	 Office 365 Users	 Office 365 Video	 OneDrive
 OneDrive for Business	 Oracle DB	 Outlook.com	 PagerDuty
 Pinterest	 Project Online	 Query	 Recurrence

CONNECTORS			
			
Redmine	RSS	Salesforce	SendGrid
			
Service Bus	SFTP	SharePoint	Slack
			
Smartsheet	SMTP	SparkPost	SQL Server
			
Todoist	Trello	Twilio	Twitter
			
Vimeo	Visual Studio Team Services	Webhook	WordPress
			
Wunderlist	Yammer	YouTube	

### Integration account connectors

INTEGRATION ACCOUNT CONNECTORS			
			
XML validation	XML transform	Flat file encoding	Flat file decoding

## INTEGRATION ACCOUNT CONNECTORS

			
<b>AS2 decoding</b>	<b>AS2 encoding</b>	<b>X12 decoding</b>	<b>X12 encoding</b>

### NOTE

If you want to get started with Azure Logic Apps before signing up for an Azure account, go to [Try Logic Apps](#). You can immediately create a short-lived starter logic app in App Service. No credit cards required, no commitments.

## Enterprise connectors

Use the enterprise connectors to create logic apps for B2B scenarios that include EAI and EDI.

	
---	---

### Connectors can be triggers

Several connectors provide triggers that can notify your app when specific events occur. For example, the FTP connector has the OnUpdatedFile trigger. You can build either a logic app, PowerApp, or Flow that listens to this trigger and performs an action whenever the trigger fires.

There are two types of triggers:

- Poll Triggers: These triggers poll your service at a specified frequency to check for new data. When new data is available, a new instance of your app runs with the data as input. To prevent the same data from being consumed multiple times, the trigger may clean up data that has been read and passed to your app.
- Push Triggers: These triggers listen for data on an endpoint or for an event to occur, then, triggers a new instance of your app. The Twitter connector is one such example.

### Connectors can be actions

You can also use connectors as actions in your apps. Actions are useful for looking up data, which can then be used in running your app. For example, you might need to look up customer data from a SQL database when processing an order. Or, you might need to write, update, or delete data in a destination table. You can perform these tasks using the actions provided by the connectors. Actions map to operations that are defined in the Swagger metadata.

## Next Steps

- [Create your first logic app](#)
- [Create custom APIs for logic apps](#)
- [Monitor your logic apps](#)

# Create your first logic app workflow to automate processes between cloud apps and cloud services

4/3/2017 • 5 min to read • [Edit Online](#)

Without writing any code, you can automate business processes more easily and quickly when you create and run workflows with [Azure Logic Apps](#). This first example shows how to create a basic logic app workflow that checks an RSS feed for new content on a website. When new items appear in the website's feed, the logic app sends email from an Outlook or Gmail account.

To create and run a logic app, you need these items:

- An Azure subscription. If you don't have a subscription, you can [start with a free Azure account](#). Otherwise, you can [sign up for a Pay-As-You-Go subscription](#).

Your Azure subscription is used for billing logic app usage. Learn how [usage metering](#) and [pricing](#) work for Azure Logic Apps.

Also, this example requires these items:

- An Outlook.com, Office 365 Outlook, or Gmail account

## TIP

If you have a personal [Microsoft account](#), you have an Outlook.com account. Otherwise, if you have an Azure work or school account, you have an [Office 365 Outlook](#) account.

- A link to a website's RSS feed. This example uses the RSS feed for the [MSDN Channel 9 website](#):

`https://s.ch9.ms/Feeds/RSS`

## Add a trigger that starts your workflow

A [trigger](#) is an event that starts your logic app workflow and is the first item that your logic app needs.

1. Sign in to the [Azure portal](#).
2. From the left menu, choose **New > Enterprise Integration > Logic App** as shown here:

Microsoft Azure New > Enterprise Integration

The screenshot shows the Microsoft Azure portal interface. On the left, the sidebar has a 'New' button highlighted with a red box and a red number '1'. Below it are various service icons: Dashboard, All resources, Resource groups, App Services, SQL databases, NoSQL (DocumentDB), Virtual machines, and Load balancers. The 'Enterprise Integration' icon is highlighted with a red box and a red number '2'. In the center, a 'New' window is open with a search bar 'Search the marketplace'. Below it is a 'MARKETPLACE' section with categories: Compute, Networking, Storage, Web + Mobile, Databases, Intelligence + analytics, Internet of Things, and Enterprise Integration, which is also highlighted with a red box and a red number '2'. On the right, the 'Enterprise Integration' blade is open, titled 'Enterprise Integration'. It shows a 'FEATURED APPS' section with three items: 'Logic App' (highlighted with a red box and a red number '3'), 'On-premises data gateway (preview)', and 'Integration Account'. Each item has a brief description and a 'PREVIEW' badge.

**TIP**

You can also choose **New**, then in the search box, type `logic app`, and press Enter. Then choose **Logic App > Create**.

3. Name your logic app and select your Azure subscription. Now create or select an Azure resource group, which helps you organize and manage related Azure resources. Finally, select the datacenter location for hosting your logic app. When you're ready, choose **Pin to dashboard** and then **Create**.

Create logic app

Logic App

① \* Name: MyFirstLogicApp

② \* Subscription: Visual Studio Enterprise

③ \* Resource group: Create new: MyLogicAppResourceGroup

④ Location: West US

**Information:** You can add triggers and actions to your Logic App after creation.

⑤  Pin to dashboard

⑥ **Create** Automation options

This is a screenshot of the 'Create logic app' dialog box. At the top, it says 'Create logic app' and 'Logic App'. There are four numbered steps: 1. 'Name' field containing 'MyFirstLogicApp'. 2. 'Subscription' dropdown set to 'Visual Studio Enterprise'. 3. 'Resource group' section with 'Create new' radio button selected and 'MyLogicAppResourceGroup' entered. 4. 'Location' dropdown set to 'West US'. Below these is an information box with an info icon and text: 'You can add triggers and actions to... your Logic App after creation.' At the bottom are two buttons: a blue 'Create' button and a 'Automation options' link.

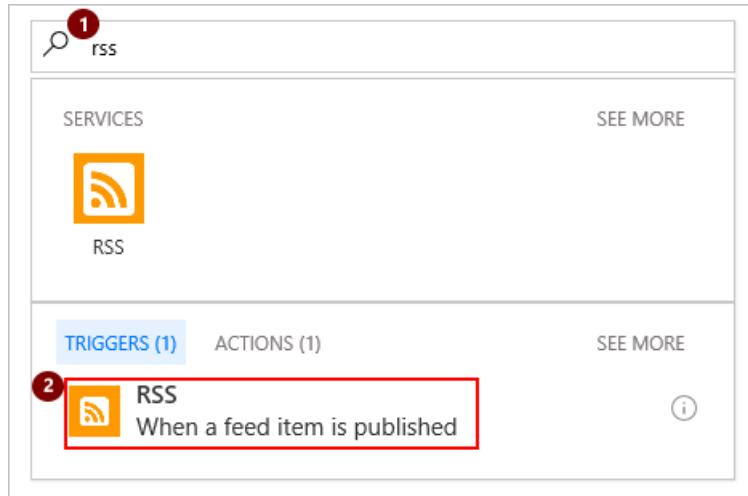
#### NOTE

When you select **Pin to dashboard**, your logic app appears on the Azure dashboard after deployment, and opens automatically. If your logic app doesn't appear on the dashboard, on the **All resources** tile, choose **See More**, and select your logic app. Or on the left menu, choose **More services**. Under **Enterprise Integration**, choose **Logic Apps**, and select your logic app.

4. When you open your logic app for the first time, the Logic App Designer shows templates that you can use to get started. For now, choose **Blank Logic App** so you can build your logic app from scratch.

The Logic App Designer opens and shows available services and *triggers* that you can use in your logic app.

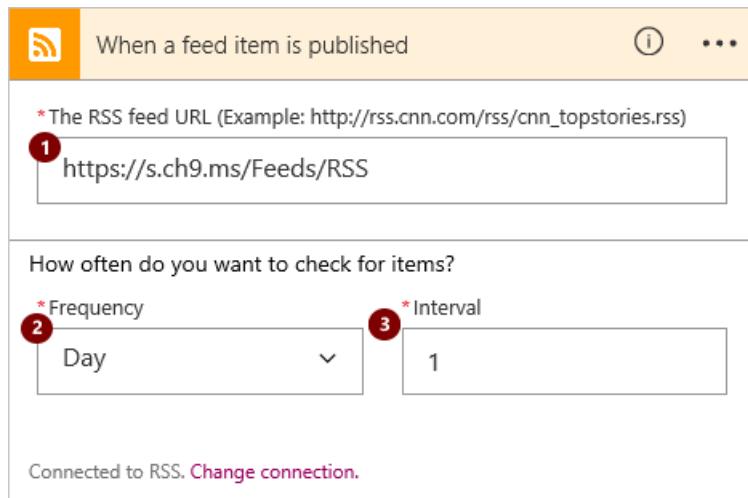
5. In the search box, type **RSS**, and select this trigger: **RSS - When a feed item is published**



6. Enter the link for the website's RSS feed that you want to track.

You can also change **Frequency** and **Interval**. These settings determine how often your logic app checks for new items and returns all items found during that time span.

For this example, let's check every day for new items posted to the MSDN Channel 9 website.



7. Save your work for now. (On the designer command bar, choose **Save**.)

The screenshot shows the Logic App designer interface. At the top, there are buttons for Save (highlighted with a red box), Discard, Run, Designer, Code view, and Templates. Below the header, the trigger configuration is shown: "When a feed item is published". A note says "\* The RSS feed URL (Example: http://rss.cnn.com/rss/cnn\_topstories.rss)". The URL input field contains "https://s.ch9.ms/Feeds/RSS". Under "How often do you want to check for items?", there are dropdowns for "Frequency" (set to "Day") and "Interval" (set to "1"). A note at the bottom says "Connected to RSS. Change connection."

When you save, your logic app goes live, but currently, your logic app only checks for new items in the specified RSS feed. To make this example more useful, we add an action that your logic app performs after your trigger fires.

## Add an action that responds to your trigger

An *action* is a task performed by your logic app workflow. After you add a trigger to your logic app, you can add an action to perform operations with data generated by that trigger. For our example, we now add an action that sends email when new items appear in the website's RSS feed.

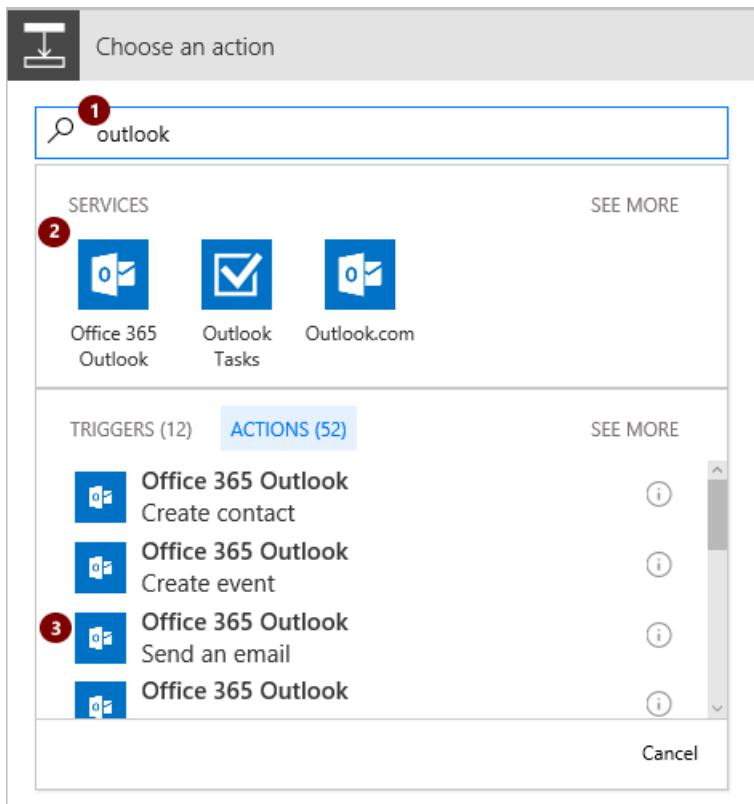
1. In the designer, under your trigger, choose **New step > Add an action** as shown here:

The screenshot shows the Logic App designer interface. A note at the top says "Connected to RSS. Change connection.". Below it, there is a callout box with a red border and a red number "1" containing the text "+ New step". At the bottom, there is a horizontal menu with three items: "Add an action" (highlighted with a red box and a red number "2"), "Add a condition", and "More".

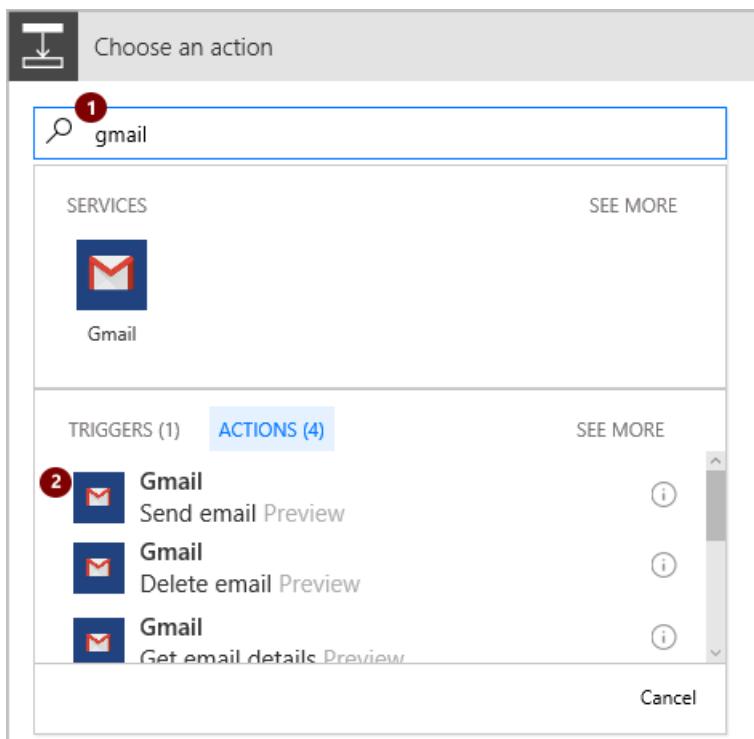
The designer shows [available connectors](#) so that you can select an action to perform when your trigger fires.

2. Based on your email account, follow the steps for Outlook or Gmail.

- To send email from your Outlook account, in the search box, enter `outlook`. Under **Services**, choose **Outlook.com** for personal Microsoft accounts, or choose **Office 365 Outlook** for Azure work or school accounts. Under **Actions**, select **Send an email**.



- To send email from your Gmail account, in the search box, enter `gmail`. Under **Actions**, select **Send email**.



- When you're prompted for credentials, sign in with the username and password for your email account.
- Provide the details for this action, like the destination email address, and choose the parameters for the data to include in the email, for example:

\* To  
username@contoso.com

\* Subject  
New Channel 9 content from Primary feed link

\* Body  
 Feed published on Feed title

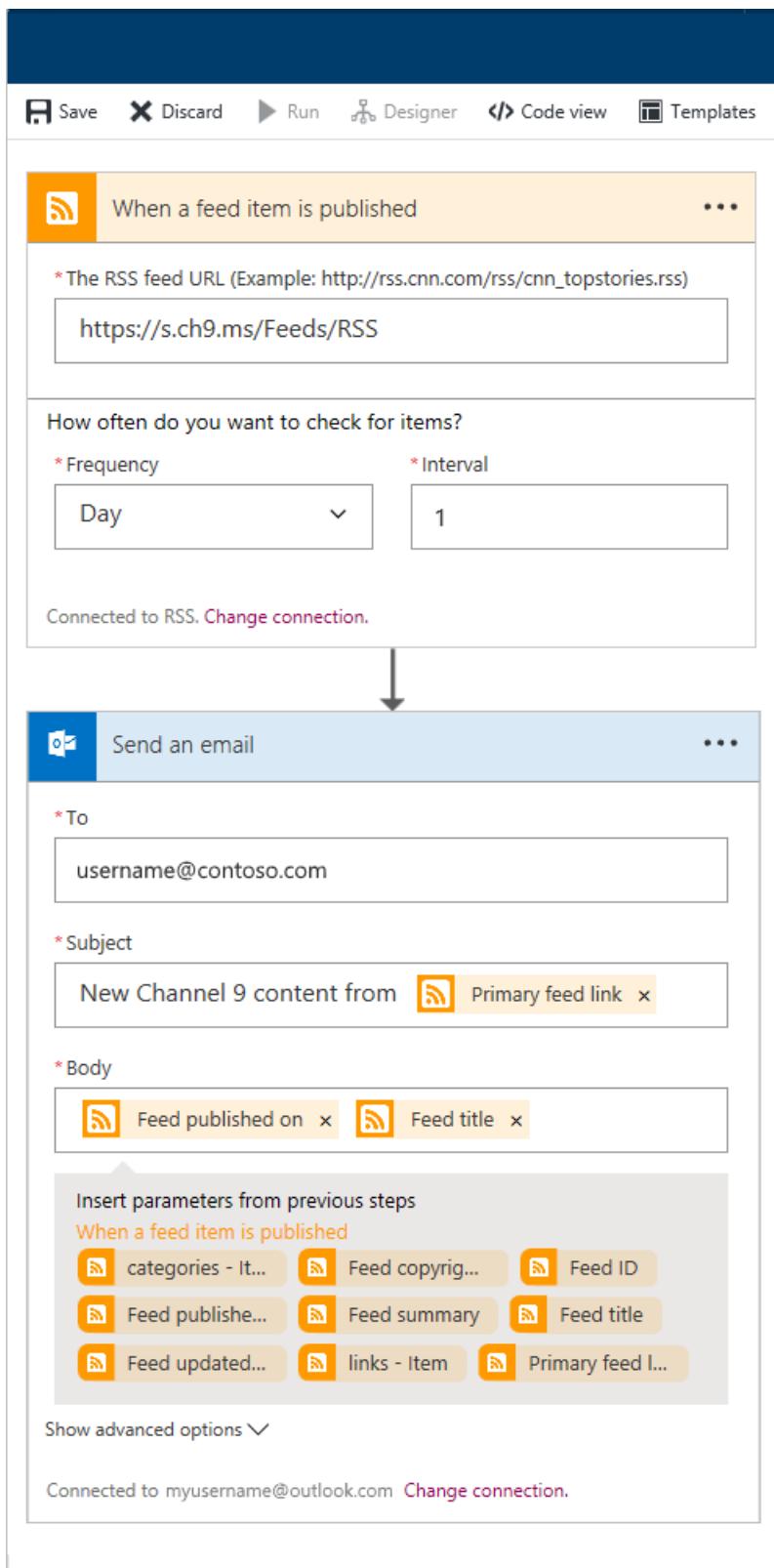
Insert parameters from previous steps  
When a feed item is published

categories - It... Feed copyrig... Feed ID  
 Feed publishe... Feed summary Feed title  
 Feed updated... links - Item Primary feed I...

Show advanced options

Connected to myusername@outlook.com [Change connection.](#)

So if you chose Outlook, your logic app might look like this example:



5. Save your changes. (On the designer command bar, choose **Save**.)
6. You can now manually run your logic app for testing. On the designer command bar, choose **Run**. Otherwise, you can let your logic app check the specified RSS feed based on the schedule that you set up.

If your logic app finds new items, the logic app sends email that includes your selected data. If no new items are found, your logic app skips the action that sends email.
7. To monitor and check your logic app's run and trigger history, on your logic app menu, choose **Overview**.

The screenshot shows the Azure Logic Apps Overview page for a logic app named 'MyLogicAppResourceGroup'. It includes sections for 'Essentials' (Resource group, Location, Subscription), 'Runs history' (listing two successful runs), and 'Trigger History' (listing three triggers: 'When\_a\_feed\_item\_is\_published' with three fired events). The 'Runs history' table has columns: STATUS, START TIME, IDENTIFIER, DURATION. The 'Trigger History' table has columns: STATUS, START TIME, FIRED.

STATUS	START TIME	IDENTIFIER	DURATION
Succeeded	3/22/2017, 9:33 AM	08587114064...	2.12 Seconds
Succeeded	3/21/2017, 2:40 PM	08587114744...	626 Milliseconds

STATUS	START TIME	FIRED
Skipped	3/22/2017, 9:35 AM	
Succeeded	3/22/2017, 9:33 AM	Fired
Succeeded	3/22/2017, 9:33 AM	

#### TIP

If you don't find the data that you expect, on the command bar, try choosing **Refresh**.

To learn more about your logic app's status or run and trigger history, or to diagnose your logic app, see [Troubleshoot your logic app](#).

#### NOTE

Your logic app continues running until you turn off your app. To turn off your app for now, on your logic app menu, choose **Overview**. On the command bar, choose **Disable**.

Congratulations, you just set up and run your first basic logic app. You also learned how easily you can create workflows that automate processes, and integrate cloud apps and cloud services - all without code.

## Manage your logic app

To manage your app, you can perform tasks like check the status, edit, view history, turn off, or delete your logic app.

1. Sign in to the [Azure portal](#).
2. On the left menu, choose **More services**. Under **Enterprise Integration**, choose **Logic Apps**. Select your logic app.

In the logic app menu, you can find these logic app management tasks:

TASK	STEPS
View your app's status, execution history, and general information	Choose <b>Overview</b> .
Edit your app	Choose <b>Logic App Designer</b> .

TASK	STEPS
View your app's workflow JSON definition	Choose <b>Logic App Code View</b> .
View operations performed on your logic app	Choose <b>Activity log</b> .
View past versions for your logic app	Choose <b>Versions</b> .
Turn off your app temporarily	Choose <b>Overview</b> , then on the command bar, choose <b>Disable</b> .
Delete your app	Choose <b>Overview</b> , then on the command bar, choose <b>Delete</b> . Enter your logic app's name, and choose <b>Delete</b> .

## Get help

To ask questions, answer questions, and learn what other Azure Logic Apps users are doing, visit the [Azure Logic Apps forum](#).

To help improve Azure Logic Apps and connectors, vote on or submit ideas at the [Azure Logic Apps user feedback site](#).

## Next steps

- [Add conditions and run workflows](#)
- [Logic app templates](#)
- [Create logic apps from Azure Resource Manager templates](#)

# Configure a workflow using a pre-built template or pattern to get started quickly

2/28/2017 • 3 min to read • [Edit Online](#)

## What are logic app templates

A logic app template is a pre-built logic app that you can use to quickly get started creating your own workflow.

These templates are a good way to discover various patterns that can be built using logic apps. You can either use these templates as-is or modify them to fit your scenario.

## Overview of available templates

There are many available templates currently published in the logic app platform. Some example categories, as well as the type of connectors used in them, are listed below.

### **Enterprise cloud templates**

Templates that integrate Dynamics CRM, Salesforce, Box, Azure Blob, and other connectors for your enterprise cloud needs. Some examples of what can be done with these templates include organizing your leads and backing up your corporate file data.

### **Enterprise integration pack templates**

Configurations of VETER (validate, extract, transform, enrich, route) pipelines, receiving an X12 EDI document over AS2 and transforming it to XML, as well as X12 and AS2 message handling.

### **Protocol pattern templates**

These templates consist of logic apps that contain protocol patterns such as request-response over HTTP as well as integrations across FTP and SFTP. Use these as they exist, or as a basis for creating more complex protocol patterns.

### **Personal productivity templates**

Patterns to help improve personal productivity include templates that set daily reminders, turn important work items into to-do lists, and automate lengthy tasks down to a single user approval step.

### **Consumer cloud templates**

Simple templates that integrate with social media services such as Twitter, Slack, and email, ultimately capable of strengthening social media marketing initiatives. These also include templates such as cloudy copying, which can help increase productivity by saving time spent on traditionally repetitive tasks.

## How to create a logic app using a template

To get started using a logic app template, go into the logic app designer. If you're entering the designer by opening an existing logic app, the logic app automatically loads in your designer view. However, if you're creating a new logic app, you see the screen below.

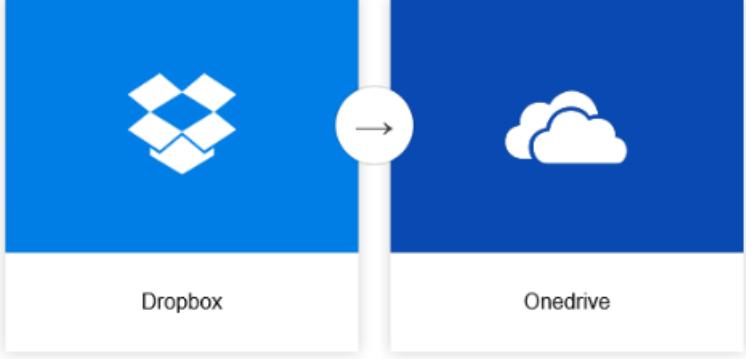
## Templates

Choose a template below to create your Logic App.

 Blank LogicApp	  When a new file is created in Dropbox, copy it to OneDrive	  If you approve a new file in SharePoint, move it to a different folder	  Send me an email when a new item is added to a SharePoint Online list
 Send me an email when a new file is added in SharePoint Online	 Save my email attachments to a SharePoint document library	 Post to Slack if a new tweet matches with some hashtag	 Share my Tweets on Facebook

From this screen, you can either choose to start with a blank logic app or a pre-built template. If you select one of the templates, you are provided with additional information. In this example, we use the *When a new file is created in Dropbox, copy it to OneDrive* template.

When a new file is created in Dropbox, copy it to OneDrive



Dropbox → Onedrive

Make sure your files end up in both Dropbox and OneDrive. This template will copy all new files that are created in Dropbox to a specific folder in OneDrive.

[Use this template](#)

If you choose to use the template, just select the *use this template* button. You'll be asked to sign in to your accounts based on which connectors the template utilizes. Or, if you've previously established a connection with these connectors, you can select continue as seen below.

To use this template:

 Dropbox [Switch account](#)

 OneDrive [Switch account](#)

[Continue](#)

After establishing the connection and selecting *continue*, the logic app opens in designer view.

When a file is created

**FOLDER\***

Select a folder

**FREQUENCY\***

Minute

**INTERVAL\***

3

Show advanced options

Connected to [redacted] Change connection.

↓

**Create file**

**FOLDER PATH\***

/backup from Dropbox

**FILE NAME\***

File name

**FILE CONTENT\***

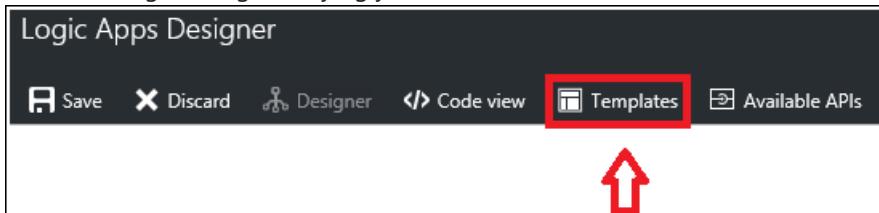
File content

Connected to [redacted] Change connection.

+ New step

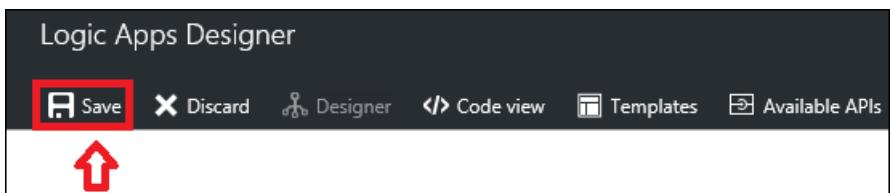
In the example above, as is the case with many templates, some of the mandatory property fields may be filled out within the connectors; however, some might still require a value before being able to properly deploy the logic app. If you try to deploy without entering some of the missing fields, you'll be notified with an error message.

If you wish to return to the template viewer, select the *Templates* button in the top navigation bar. By switching back to the template viewer, you lose any unsaved progress. Prior to switching back into template viewer, you'll see a warning message notifying you of this.



## How to deploy a logic app created from a template

Once you have loaded your template and made any desired changes, select the save button in the upper left corner. This saves and publishes your logic app.



If you would like more information on how to add more steps into an existing logic app template, or make edits in general, read more at [Create a logic app](#).

# Create a Logic App using a template

1/31/2017 • 2 min to read • [Edit Online](#)

Templates provide a quick way to use different connectors within a logic app. Logic apps includes Azure Resource Manager templates for you to create a logic app that can be used to define business workflows. You can define which resources are deployed, and how to define parameters when you deploy your logic app. You can use this template for your own business scenarios, or customize it to meet your requirements.

For more details on the Logic app properties, see [Logic App Workflow Management API](#).

For examples of the definition itself, see [Author Logic App definitions](#).

For more information about creating templates, see [Authoring Azure Resource Manager Templates](#).

For the complete template, see [Logic App template](#).

## What you deploy

With this template, you deploy a logic app.

To run the deployment automatically, select the following button:



## Parameters

With Azure Resource Manager, you define parameters for values you want to specify when the template is deployed. The template includes a section called Parameters that contains all of the parameter values. You should define a parameter for those values that will vary based on the project you are deploying or based on the environment you are deploying to. Do not define parameters for values that will always stay the same. Each parameter value is used in the template to define the resources that are deployed.

When defining parameters, use the **allowedValues** field to specify which values a user can provide during deployment. Use the **defaultValue** field to assign a value to the parameter, if no value is provided during deployment.

We will describe each parameter in the template.

### **logicAppName**

The name of the logic app to create.

```
"logicAppName": {  
    "type": "string"  
}
```

### **testUri**

```
"testUri": {  
    "type": "string",  
    "defaultValue": "http://azure.microsoft.com/en-us/status/feed/"  
}
```

# Resources to deploy

## Logic app

Creates the logic app.

The template uses a parameter value for the logic app name. It sets the location of the logic app to the same location as the resource group.

This particular definition runs once an hour, and pings the location specified in the **testUri** parameter.

```
{
  "type": "Microsoft.Logic/workflows",
  "apiVersion": "2016-06-01",
  "name": "[parameters('logicAppName')]",
  "location": "[resourceGroup().location]",
  "tags": {
    "displayName": "LogicApp"
  },
  "properties": {
    "definition": {
      "$schema": "http://schema.management.azure.com/providers/Microsoft.Logic/schemas/2016-06-01/workflowdefinition.json#",
      "contentVersion": "1.0.0.0",
      "parameters": {
        "testURI": {
          "type": "string",
          "defaultValue": "[parameters('testUri')]"
        }
      },
      "triggers": {
        "recurrence": {
          "type": "recurrence",
          "recurrence": {
            "frequency": "Hour",
            "interval": 1
          }
        }
      },
      "actions": {
        "http": {
          "type": "Http",
          "inputs": {
            "method": "GET",
            "uri": "@parameters('testUri')"
          },
          "runAfter": {}
        }
      },
      "outputs": {}
    },
    "parameters": {}
  }
}
```

# Commands to run deployment

To deploy the resources to Azure, you must be logged in to your Azure account and you must use the Azure Resource Manager module. To learn about using Azure Resource Manager with either Azure PowerShell or Azure CLI, see:

- [Using Azure PowerShell with Azure Resource Manager](#)
- [Using the Azure CLI for Mac, Linux, and Windows with Azure Resource Management](#).

The following examples assume you already have a resource group in your account with the specified name.

## PowerShell

```
New-AzureRmResourceGroupDeployment -TemplateUri https://raw.githubusercontent.com/Azure/azure-quickstart-templates/master/101-logic-app-create/azuredeploy.json -ResourceGroupName ExampleDeployGroup
```

## Azure CLI

```
az group deployment create --template-uri https://raw.githubusercontent.com/Azure/azure-quickstart-templates/master/101-logic-app-create/azuredeploy.json -g ExampleDeployGroup
```

# Design, build, and deploy Azure Logic Apps in Visual Studio

3/13/2017 • 5 min to read • [Edit Online](#)

Although the [Azure portal](#) offers a great way for you to create and manage Azure Logic Apps, you might want to use Visual Studio for designing, building, and deploying your logic apps. Visual Studio provides rich tools like the Logic App Designer for you to create logic apps, configure deployment and automation templates, and deploy to any environment.

To get started with Azure Logic Apps, learn [how to create your first logic app in the Azure portal](#).

## Installation steps

To install and configure Visual Studio tools for Azure Logic Apps, follow these steps.

### Prerequisites

- [Visual Studio 2015](#)
- [Latest Azure SDK](#) (2.9.1 or greater)
- [Azure PowerShell](#)
- Access to the web when using the embedded designer

### Install Visual Studio tools for Azure Logic Apps

After you install the prerequisites:

1. Open Visual Studio. On the **Tools** menu, select **Extensions and Updates**.
2. Expand the **Online** category so you can search online.
3. Browse or search for **Logic Apps** until you find **Azure Logic Apps Tools for Visual Studio**.
4. To download and install the extension, click **Download**.
5. Restart Visual Studio after installation.

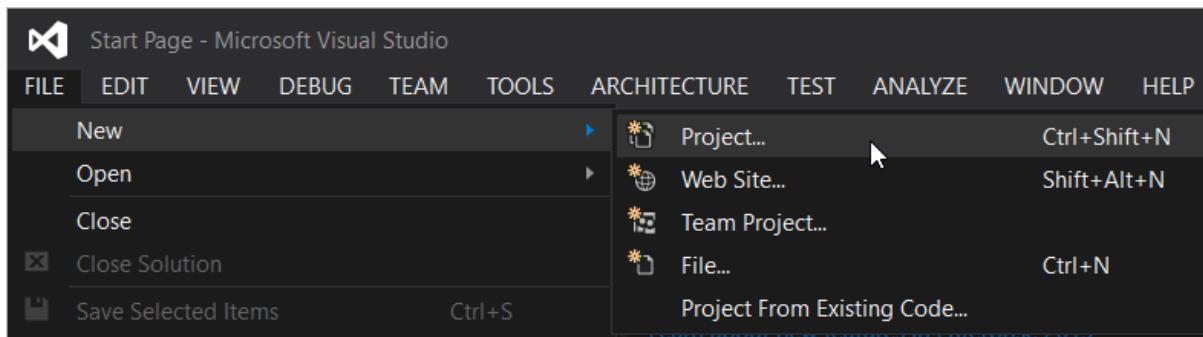
#### NOTE

You can also download Azure Logic Apps Tools for Visual Studio directly from the [Visual Studio Marketplace](#).

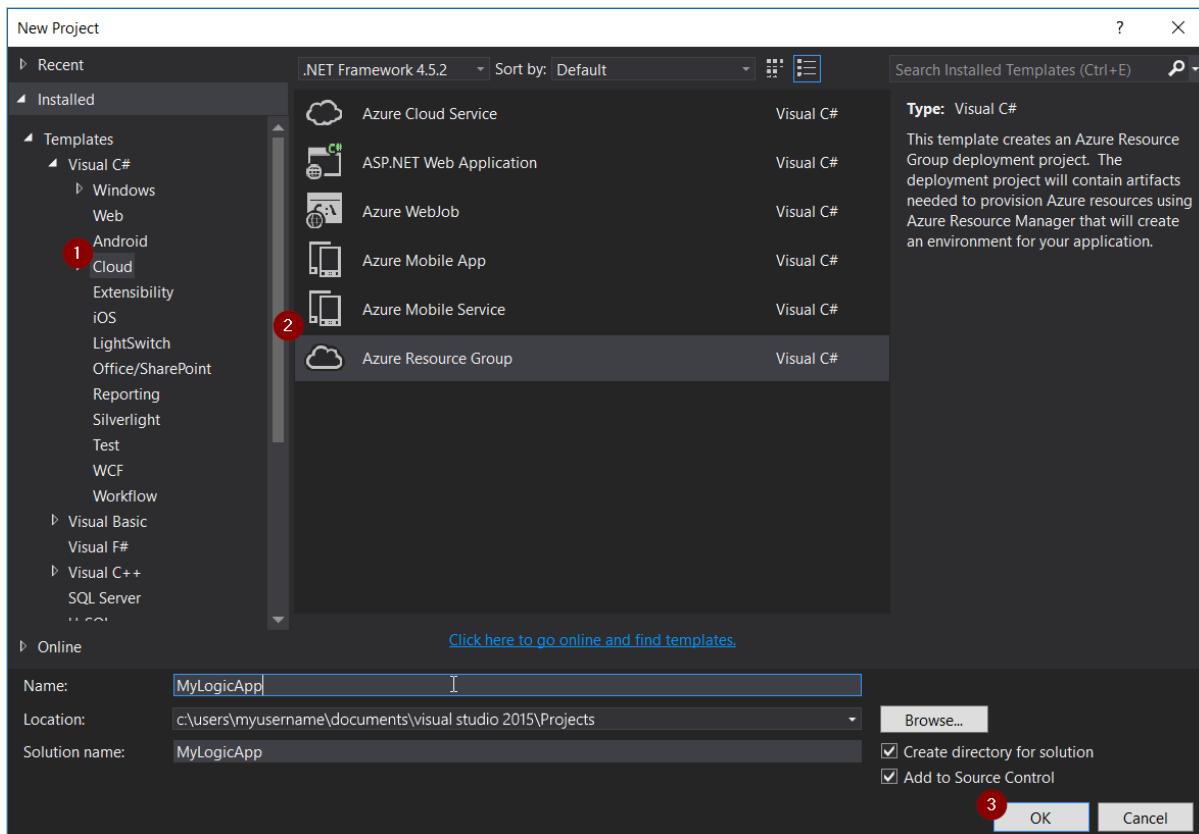
After you finish installation, you can use the Azure Resource Group project with Logic App Designer.

## Create your project

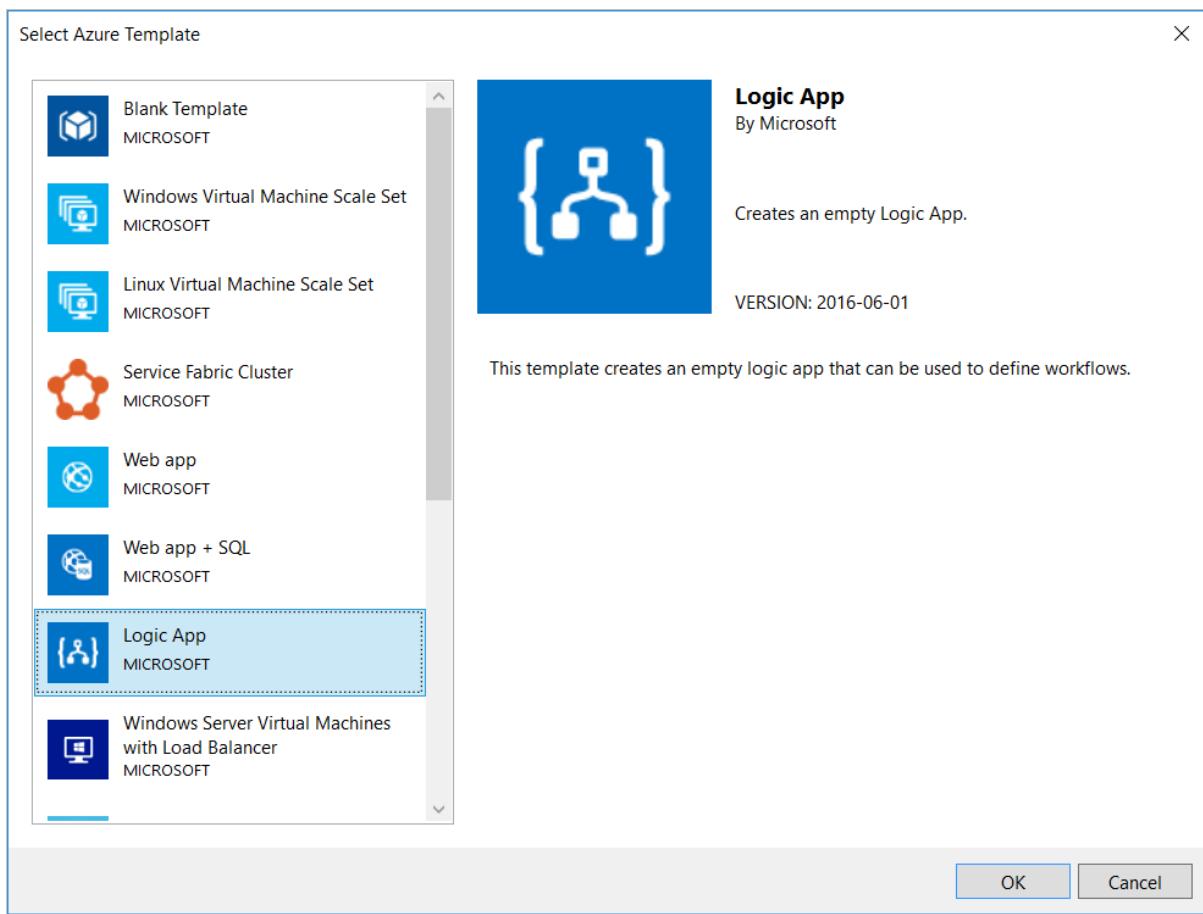
1. On the **File** menu, go to **New**, and select **Project**. Or to add your project to an existing solution, go to **Add**, and select **New Project**.



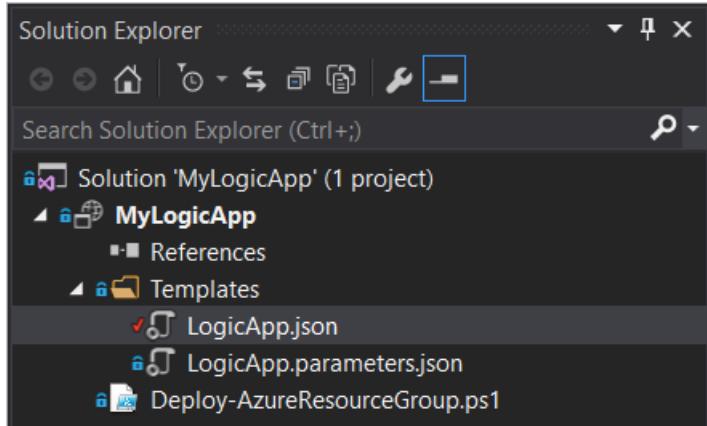
2. In the **New Project** window, find **Cloud**, and select **Azure Resource Group**. Name your project, and click **OK**.



3. Select the **Logic App** template, which creates a blank logic app deployment template for you to use. After you select your template, click **OK**.



You've now added your logic app project to your solution. In the Solution Explorer, your deployment file should appear.



## Create your logic app with Logic App Designer

When you have an Azure Resource Group project that contains a logic app, you can open the Logic App Designer in Visual Studio to create your workflow.

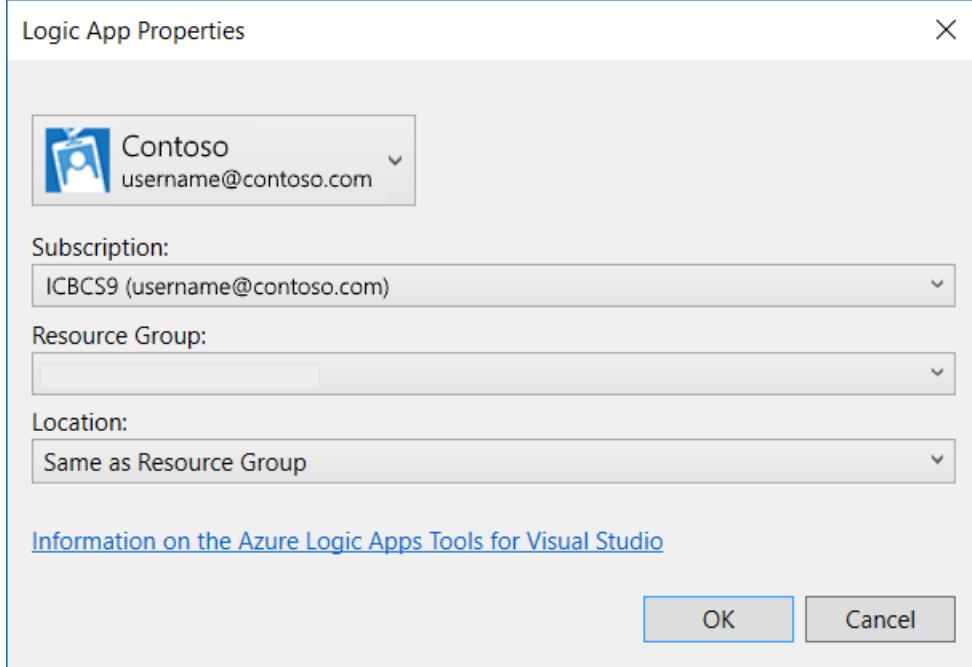
### NOTE

The designer requires an internet connection to query connectors for available properties and data. For example, if you use the Dynamics CRM Online connector, the designer queries your CRM instance to show available custom and default properties.

1. Right-click your `<template>.json` file, and select **Open with Logic App Designer**. (`ctrl+L`)
2. Choose your Azure subscription, resource group, and location for your deployment template.

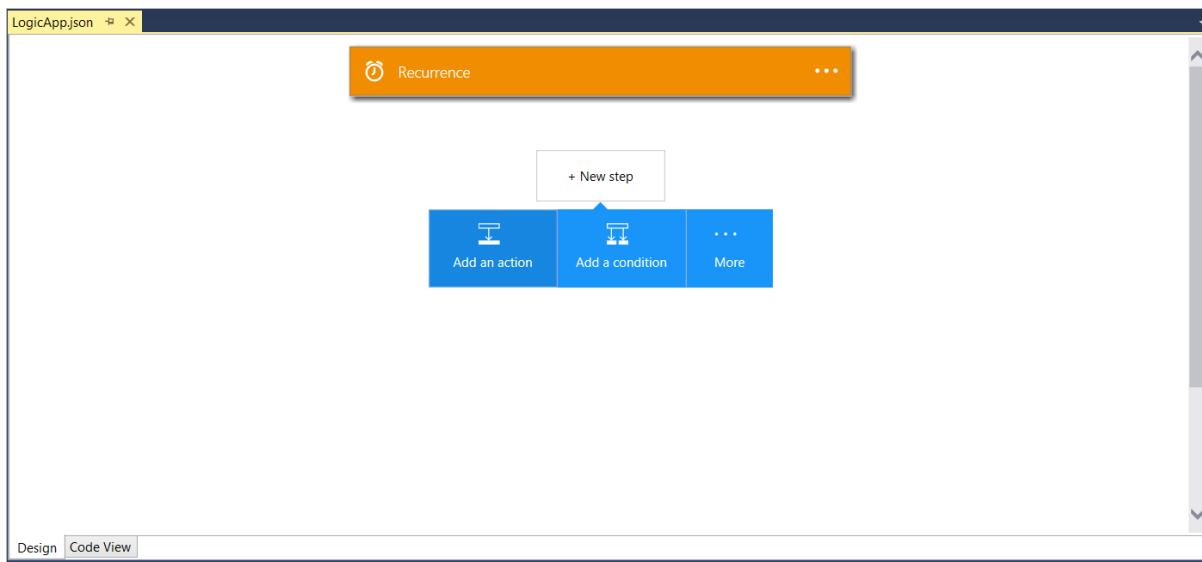
#### NOTE

Designing a logic app creates API Connection resources that query for properties during design. Visual Studio uses your selected resource group to create those connections during design time. To view or change any API Connections, go to the Azure portal, and browse for **API Connections**.



The designer uses the definition in the `<template>.json` file for rendering.

3. Create and design your logic app. Your deployment template is updated with your changes.



Visual Studio adds `Microsoft.Web/connections` resources to your resource file for any connections your logic app needs to function. These connection properties can be set when you deploy, and managed after you deploy in **API Connections** in the Azure portal.

#### Switch to JSON code view

To show the JSON representation for your logic app, select the **Code View** tab at the bottom of the designer.

To switch back to the full resource JSON, right-click the `<template>.json` file, and select **Open**.

#### Add references for dependent resources to Visual Studio deployment templates

When you want your logic app to reference dependent resources, you can use [Azure Resource Manager template](#)

[functions](#), like parameters, in your logic app deployment template. For example, you might want your logic app to reference an Azure Function or integration account that you want to deploy alongside your logic app. Follow these guidelines about how to use parameters in your deployment template so that the Logic App Designer renders correctly.

You can use logic app parameters in these kinds of triggers and actions:

- Child workflow
- Function app
- APIM call
- API connection runtime URL

And you can use these template functions: list below, includes parameters, variables, resourceId, concat, and so on. For example, here's how you can replace the Azure Function resource ID:

```
"parameters":{  
    "functionName": {  
        "type":"string",  
        "minLength":1,  
        "defaultValue": "<FunctionName>"  
    }  
,
```

And where you'd use parameters:

```
"MyFunction": {  
    "type": "Function",  
    "inputs": {  
        "body":{},  
        "function":{  
            "id": "[resourceId('Microsoft.Web/sites/functions', 'functionApp', parameters('functionName'))]"  
        }  
    },  
    "runAfter":{}  
}
```

#### NOTE

For the Logic App Designer to work when you use parameters, you must provide default values, for example:

```
"parameters": {  
    "IntegrationAccount": {  
        "type":"string",  
        "minLength":1,  
  
        "defaultValue": "/subscriptions/<subscriptionID>/resourceGroups/<resourceGroupName>/providers/Microsoft.Logic/integrationAccounts/<integrationAccountName>"  
    }  
,
```

#### Save your logic app

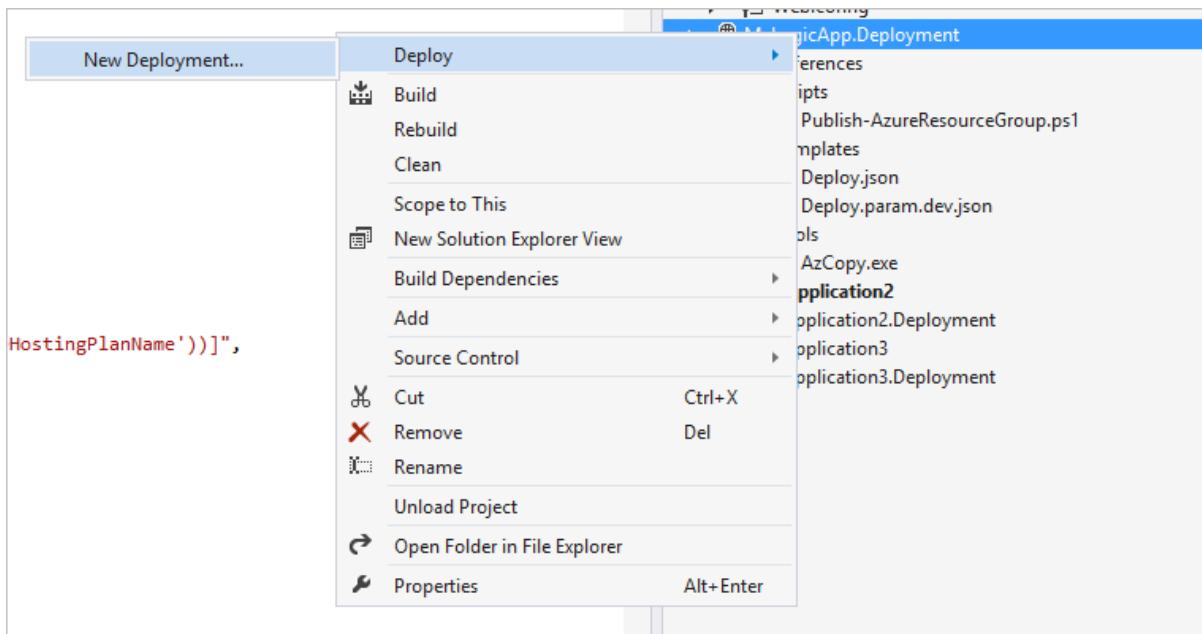
To save your logic app at anytime, go to **File > Save**. (`Ctrl+S`)

If your logic app has any errors when you save your app, they appear in the Visual Studio **Outputs** window.

## Deploy your logic app from Visual Studio

After configuring your app, you can deploy directly from Visual Studio in just a couple steps.

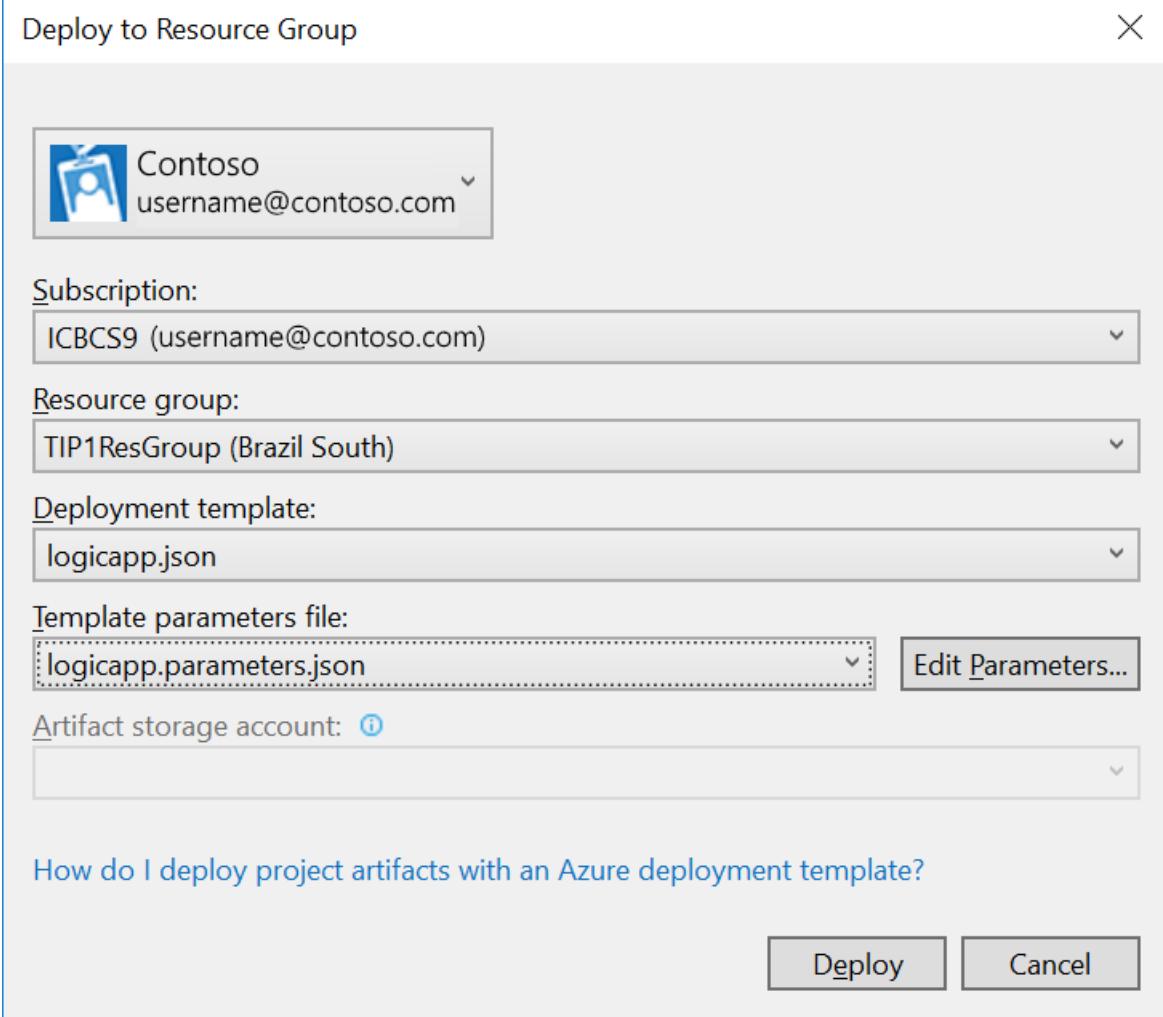
1. In Solution Explorer, right-click your project, and go to **Deploy > New Deployment...**



2. When you're prompted, sign in to your Azure subscription.
3. Now you must select the details for the resource group where you want to deploy your logic app. When you're done, select **Deploy**.

**NOTE**

Make sure that you select the correct template and parameters file for the resource group. For example, if you want to deploy to a production environment, choose the production parameters file.



The deployment status appears in the **Output** window. You might have to select **Azure Provisioning** in the **Show output from** list.

```
Output
Show output from: Azure Provisioning
05:43:57 - 
05:43:57 - Transfer summary:
05:43:57 - -----
05:43:57 - Total files transferred: 3
05:43:57 - Transfer successfully: 3
05:43:57 - Transfer failed: 0
05:43:57 - Elapsed time: 00:00:00:02
05:43:57 - Done building target "UploadDrop" in project "MyLogicApp.Deployment.deployproj".
05:43:57 - Done building project "MyLogicApp.Deployment.deployproj".
05:43:57 - Build succeeded.
05:43:57 - The following parameter values will be used for this deployment:
05:43:57 -   dropLocation: https://deploymentlogs.blob.core.windows.net/webapplication2
05:43:57 -   dropLocationSasToken: <securestring>
05:43:57 -   webSitePackage: MyLogicApp/package.zip
05:43:57 -   webSiteName: blogordns
05:43:57 -   webSiteHostingPlanName: noentuhoneuth
05:43:57 -   webSiteLocation: westus
05:43:57 -   webSiteHostingPlanSKU: Free
05:43:57 -   webSiteHostingPlanWorkerSize: 0
05:43:57 - Starting deployment. This may take a while...
05:43:57 - Uploading template to Azure storage account 'deploymentlogs'.
05:43:58 - Creating resource group 'apiapps-c70beaef-6ec0-49a4-8215-7d55a08b7414-2015-04-11T10' in location 'westus'.
05:44:12 - Creating deployment 'VS_deploy' in resource group 'apiapps-c70beaef-6ec0-49a4-8215-7d55a08b7414-2015-04-11T10'.
```

In the future, you can edit your logic app in source control, and use Visual Studio to deploy new versions.

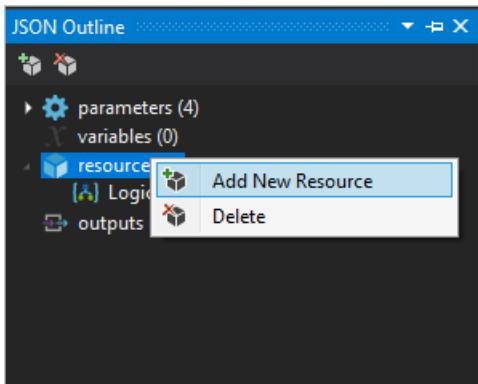
#### NOTE

If you change the definition in the Azure portal directly, those changes are overwritten when you deploy from Visual Studio next time.

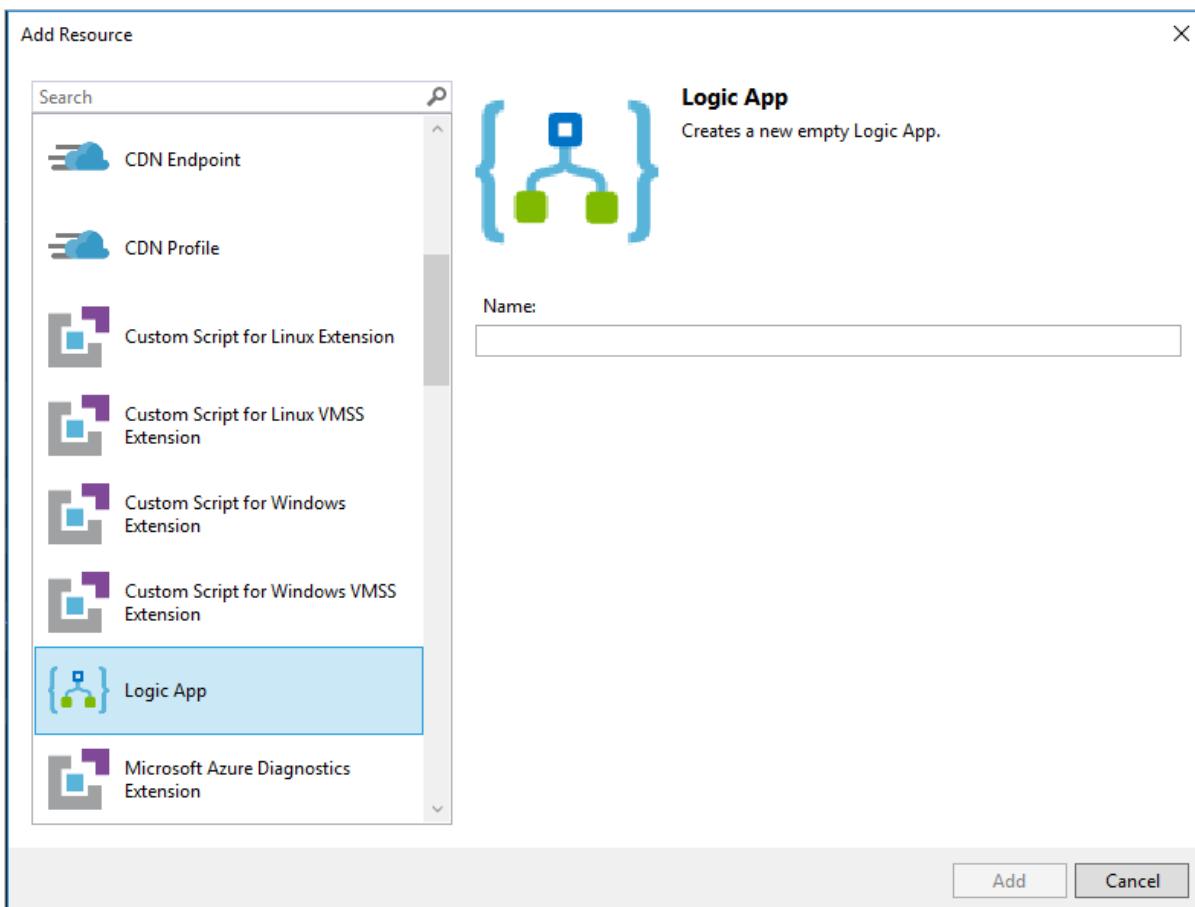
# Add your logic app to an existing Resource Group project

If you have an existing Resource Group project, you can add your logic app to that project in the JSON Outline window. You can also add another logic app alongside the app you previously created.

1. Open the `<template>.json` file.
2. To open the JSON Outline window, go to **View > Other Windows > JSON Outline**.
3. To add a resource to the template file, click **Add Resource** at the top of the JSON Outline window. Or in the JSON Outline window, right-click **resources**, and select **Add New Resource**.



4. In the **Add Resource** dialog box, find and select **Logic App**. Name your logic app, and choose **Add**.



## Next Steps

- [Manage logic apps with Visual Studio Cloud Explorer](#)
- [View common examples and scenarios](#)
- [Learn how to automate business processes with Azure Logic Apps](#)
- [Learn how to integrate your systems with Azure Logic Apps](#)

# Use Logic Apps features

2/15/2017 • 4 min to read • [Edit Online](#)

In the [previous topic](#), you created your first logic app. Now you'll build a fuller process with Azure Logic Apps. This topic introduces the following new Azure Logic Apps concepts:

- Conditional logic, which executes an action only when a certain condition is met.
- Code view to edit an existing logic app.
- Options for starting a workflow.

Before you complete this topic, you should complete the steps in [Create a new logic app](#). In the [Azure portal](#), browse to your logic app and click **Triggers and Actions** in the summary to edit the logic app definition.

## Reference material

You may find the following documents useful:

- [Management and runtime REST APIs](#) - including how to invoke Logic apps directly
- [Language reference](#) - a comprehensive list of all supported functions/expressions
- [Trigger and action types](#) - the different types of actions and the inputs they take
- [Overview of App Service](#) - description of what components to choose when to build a solution

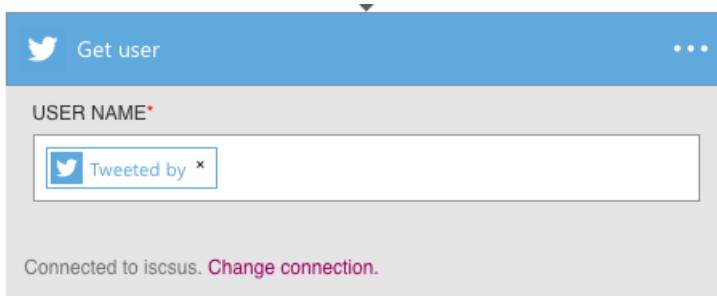
## Add conditional logic to your logic app

Although your logic app's original flow works, we could improve some areas.

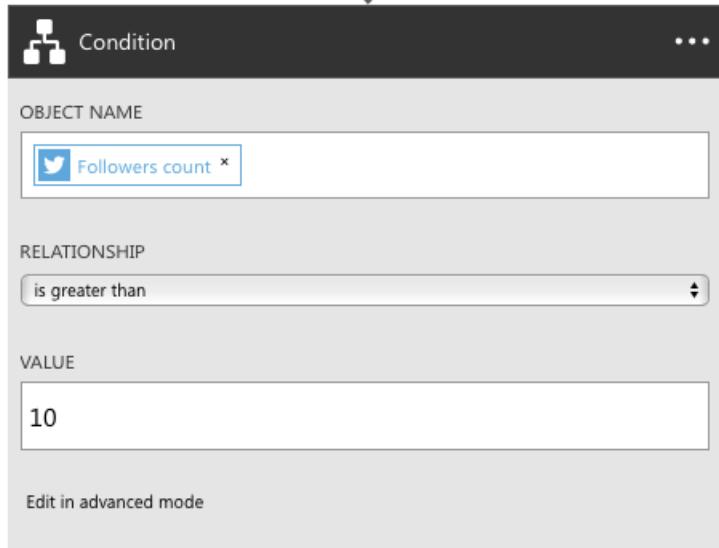
### Conditional

Your first logic app might result in you getting too many emails. The following steps add conditional logic so that you receive email only when the tweet comes from someone with a specific number of followers.

1. In the Logic App Designer, choose **New Step (+) > Add an action**.
2. Find and add the **Get User** action for Twitter.
3. To get the information about the Twitter user, find and add the **Tweeted by** field from the trigger.



4. Choose **New Step (+) > Add a condition**.
5. To filter on the number of followers that users have, under **Object name**, choose **Add dynamic content**.
6. In the search box, find and add the **Followers count** field.
7. Under **Relationship**, select **is greater than**.
8. In the **Value** box, enter the number of followers you want users to have.



9. Finally, drag the **Send email** box into the **If Yes** box.

Now you get emails only when the follower count meets your condition.

## Repeat actions over a list with forEach

The `forEach` loop specifies an array to repeat an action over. If it is not an array, the flow fails. For example, if you have `action1` that outputs an array of messages, and you want to send each message, you can include this `forEach` statement in the properties of your action: `forEach : "@action('action1').outputs.messages"`

## Edit the code definition for a logic app

Although you have the Logic App Designer, you can directly edit the code that defines a logic app.

1. On the command bar, choose **Code view**.

A full editor opens and shows the definition you edited.

```

17     }
18   },
19   "triggers": {
20     "recurrence": {
21       "recurrence": {
22         "frequency": "Minute",
23         "interval": 1
24       },
25       "type": "Recurrence",
26       "name": "recurrence"
27     }
28   },
29   "actions": {
30     "twitterconnector": {
31       "type": "Apiapp",
32       "version": "2015-01-14",
33       "inputs": {
34         "host": {
35           "id": "/subscriptions/423db32d-4f58-4220-961c-b59f14c962f1/resourcegroups/bpmdemo002/providers/Microsoft.AppService/apiapps/twitterconnector002",
36           "gateway": "https://bpmdemo002proxysite.azurewebsites.net"
37         },
38         "operation": "SearchTweets",
39         "parameters": {
40           "query": "@concat('#',parameters('queryString'))",
41           "count": 5
42         }
43       },
44       "conditions": [],
45       "name": "twitterconnector"
46     },
47     "twitterconnector0": {
48       "repeat": "@actions('twitterconnector').outputs.body",
49       "type": "Apiapp",
50       "version": "2015-01-14",
51       "inputs": {
52         "host": {
53           "id": "/subscriptions/423db32d-4f58-4220-961c-b59f14c962f1/resourcegroups/bpmdemo002/providers/Microsoft.AppService/apiapps/twitterconnector002",
54           "gateway": "https://bpmdemo002proxysite.azurewebsites.net"
55         },
56         "operation": "SearchUser",
57         "parameters": {
58           "user_id": "",
59           "screen_name": "@repeatItem().Tweeted_By"
60         },
61         "authentication": {
62           "type": "Raw",
63           "scheme": "Zumo",
64           "parameter": "@parameters('/subscriptions/423db32d-4f58-4220-961c-b59f14c962f1/resourcegroups/bpmdemo002/providers/Microsoft.AppService/apiapps/twitterconnector002')"
65         }
66       },
67       "conditions": [

```

In the text editor, you can copy and paste any number of actions within the same logic app or between logic apps. You can also easily add or remove entire sections from the definition, and you can also share definitions with others.

## 2. To save your edits, choose **Save**.

### Parameters

Some Logic Apps capabilities are available only in code view, for example, parameters. Parameters make it easy to reuse values throughout your logic app. For example, if you have an email address that you want use in several actions, you should define that email address as a parameter.

Parameters are good for pulling out values that you are likely to change a lot. They are especially useful when you need to override parameters in different environments. To learn how to override parameters based on environment, see the [REST API documentation](#).

This example shows how to update your existing logic app so that you can use parameters for the query term.

### 1. In code view, find the `parameters : {}` object, and add a topic object:

```

"topic" : {
  "type" : "string",
  "defaultValue" : "MicrosoftAzure"
}

```

### 2. Go to the `twitterconnector` action, find the query value, and replace that value with

```
#@{parameters('topic')}
```

To join two or more strings, you can also use the `concat` function. For example,

```
@concat('#',parameters('topic'))
```

### 3. When you're done, choose **Save**.

Now every hour, you get new tweets that have more than five retweets delivered to a folder called **tweets** in your Dropbox.

To learn more about Logic App definitions, see [author Logic App definitions](#).

## Start logic app workflows

You have different options for starting the workflow defined in your logic app. You can always start a workflow on-demand in the [Azure portal](#).

### Recurrence triggers

A recurrence trigger runs at an interval that you specify. When the trigger has conditional logic, the trigger determines whether the workflow needs to run. A trigger indicates the workflow should run by returning a `200` status code. When the workflow doesn't need to run, the trigger returns a `202` status code.

### Callback using REST APIs

To start a workflow, services can call a logic app endpoint. To start this kind of logic app on-demand, choose **Run now** on the command bar. See [Start workflows by calling logic app endpoints as triggers](#).

# Use switch statement in Logic Apps

2/14/2017 • 3 min to read • [Edit Online](#)

When authoring a workflow, you often need to take different actions based on the value of an object or expression. For example, you may want your Logic App to behave differently based on the status code of an HTTP request, or the selected option of an approval email.

These scenarios can be achieved by using a switch statement: Logic App evaluates a token or expression, and chooses the case with the same value to execute actions within. Only one case should match the switch statement.

## TIP

Like all programming language, switch statement only supports equality operators. Use a condition statement if you need other relational operators (for example, greater than).

To ensure deterministic execution behavior, cases must contain a unique and static value instead of dynamic tokens or expression.

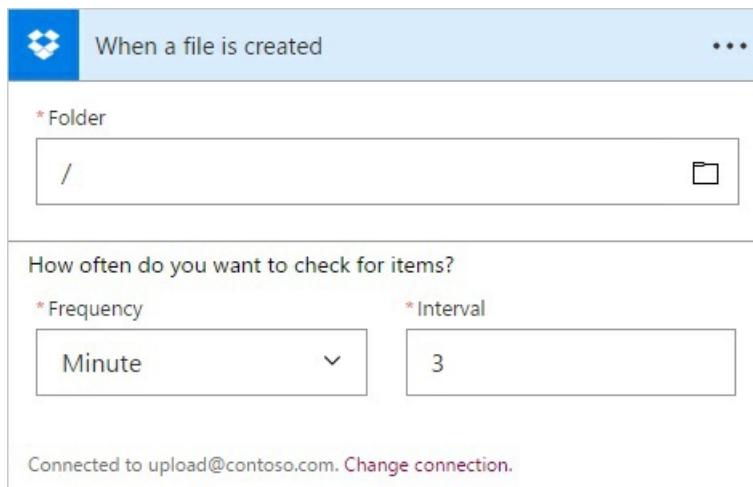
## Prerequisites

- Active Azure subscription.
  - If you don't have an active Azure subscription, [create a free account](#), or try [Logic Apps for free](#).
- [Basic knowledge of Logic Apps](#).

## Working with switch statement in designer

To demonstrate the usage of switch statement, let's create a Logic App that monitors files uploaded to Dropbox. The Logic App will send out an approval email to determine if it should be transferred to SharePoint. We will use switch statement to take different actions depending on the value approver selected.

1. Start by create a Logic App, and select **Dropbox - When a file is created** trigger.

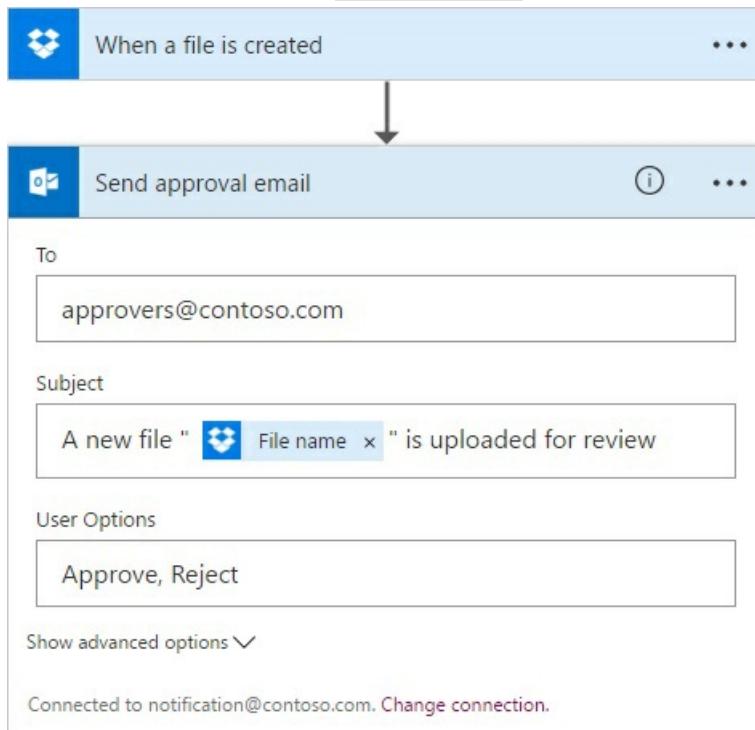


2. Follow up the trigger with an **Outlook.com - Send approval email** action.

## TIP

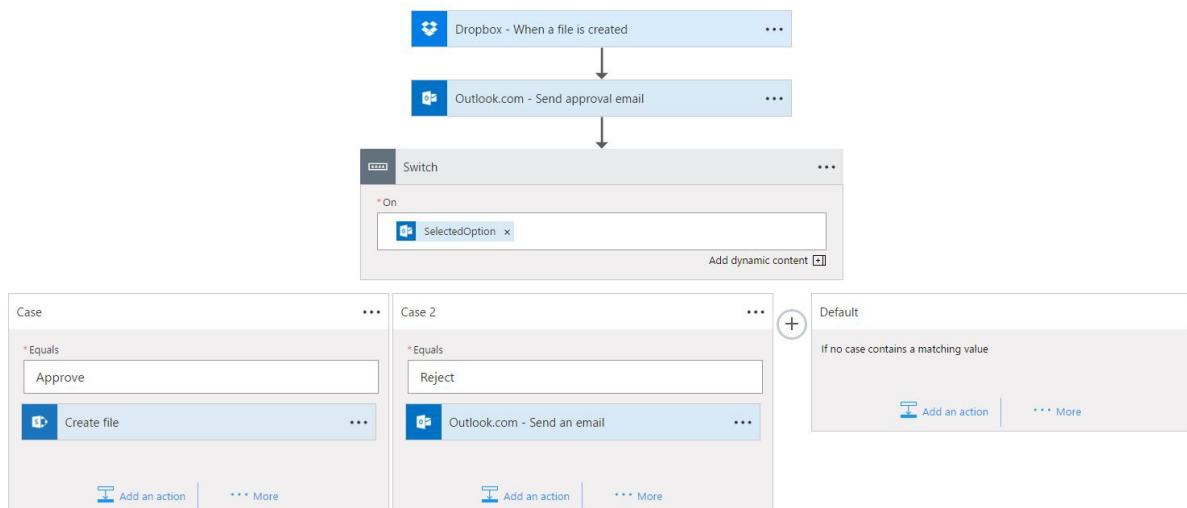
Logic Apps also supports approval email scenario from an Office 365 Outlook account.

- If you don't have an existing connection, you will be prompted to create one.
- Fill in required fields, we will send email to approvers@contoso.com.
- Under *User Options*, enter **Approve, Reject**.



### 3. Add a switch statement.

- Select **+ New step, ... More, Add a switch statement**.
- We want to select what to execute based on **SelectedOptions** output of the *Send approval email* action, you can find it in the **Add dynamic content** selector.
- Use **Case 1** to handle when user selected **Approve**.
  - If approved, copy the original file to SharePoint Online with **SharePoint Online - Create file** action.
  - Add another action within the case to notify users that a new file is available on SharePoint.
- Add another case to handle when user selected **Reject**.
  - If rejected, send a notification email informing other approvers that the file is rejected and no further action is required.
- We know **SelectedOptions** only has two provided options, *default* case can be left empty.



#### NOTE

Switch statement needs at least one case in addition to the default case.

4. After the switch statement, delete the original file uploaded to Dropbox with **Dropbox - Delete file** action.
5. Save your Logic App, and test it by uploading a file to Dropbox. You should receive an approval email shortly after, select an option, and observe the behavior.

#### TIP

Check out how to [monitor your Logic Apps](#).

## Understanding code behind

Now you have successfully created a Logic App using switch statement. Let's look at the code behind as follows.

```
"Switch": {  
    "type": "Switch",  
    "expression": "@body('Send_approval_email')?['SelectedOption']",  
    "cases": {  
        "Case 1" : {  
            "case" : "Approved",  
            "actions" : {}  
        },  
        "Case 2" : {  
            "case" : "Rejected",  
            "actions" : {}  
        }  
    },  
    "default": {  
        "actions": {}  
    },  
    "runAfter": {  
        "Send_approval_email": [  
            "Succeeded"  
        ]  
    }  
}
```

"Switch" is the name of the switch statement, it can be renamed for readability. "type": "Switch" indicates the action is a switch statement. "expression", in this case, user's selected option, is evaluated against each case declared later in the definition. "cases" can contain any number of cases, and if none of the cases match the switch expression, actions within "default" is executed.

There can be any number of cases inside "cases". For each case, "Case 1" is the name of the case, it can be renamed for readability. "case" specifies the case label, which the switch expression compares with, that must be a constant and unique value.

## Next steps

- Try other [Logic Apps features](#).
- Learn about [error and exception handling](#).
- Explore more [workflow language capabilities](#).
- Leave a comment with your questions or feedback, or [tell us how can we improve Logic Apps](#).

# Add and run custom code for logic apps through Azure Functions

3/9/2017 • 2 min to read • [Edit Online](#)

To run custom snippets of C# or nodejs in logic apps, you can create custom functions through Azure Functions. [Azure Functions](#) offers server-free computing in Microsoft Azure and are useful for performing these tasks:

- Advanced formatting or compute of fields in logic apps
- Perform calculations in a workflow.
- Extend the logic app functionality with functions that are supported in C# or node.js

## Create custom functions for your logic apps

We recommend that you create a function in the Azure Functions portal, from the [Generic Webhook - Node](#) or [Generic Webhook - C#](#) templates. The result creates an auto-populated template that accepts `application/json` from a logic app. Functions that you create from these templates are automatically detected and appear in the Logic App Designer under **Azure Functions in my region**.

In the Azure portal, on the **Integrate** pane for your function, your template should show that **Mode** set to **Webhook** and **Webhook type** is set to **Generic JSON**.

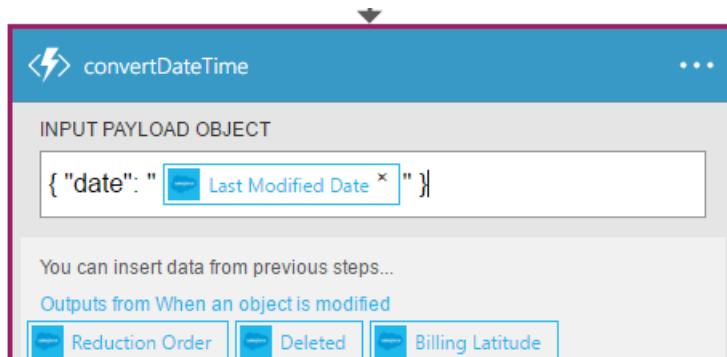
Webhook functions accept a request and pass it into the method via a `data` variable. You can access the properties of your payload by using dot notation like `data.function-name`. For example, a simple JavaScript function that converts a DateTime value into a date string looks like the following example:

```
function start(req, res){  
    var data = req.body;  
    res = {  
        body: data.date.ToString();  
    }  
}
```

## Call Azure Functions from logic apps

To list the containers in your subscription and select the function that you want to call, in Logic App Designer, click the **Actions** menu, and select from **Azure Functions in my Region**.

After you select the function, you are asked to specify an input payload object. This object is the message that the logic app sends to the function and must be a JSON object. For example, if you want to pass in the **Last Modified** date from a Salesforce trigger, the function payload might look like this example:



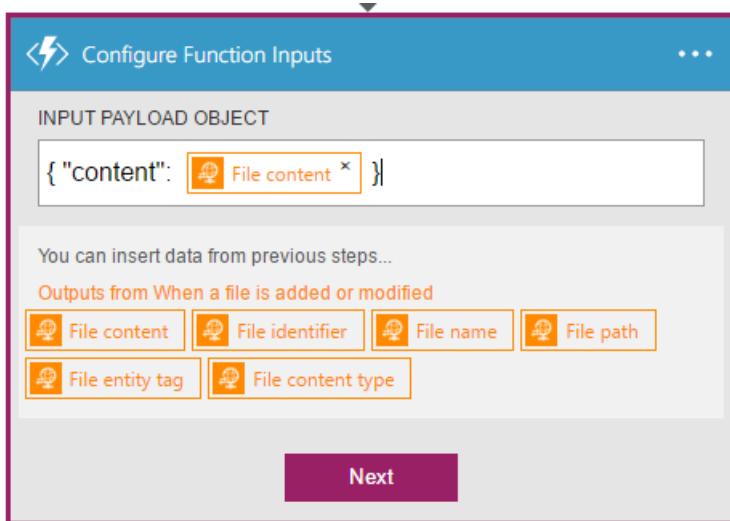
# Trigger logic apps from a function

You can trigger a logic app from inside a function. See [Logic apps as callable endpoints](#). Create a logic app that has a manual trigger, then from inside your function, generate an HTTP POST to the manual trigger URL with the payload that you want sent to the logic app.

## Create a function from Logic App Designer

You can also create a node.js webhook function from the designer. First, select **Azure Functions in my Region**, and then choose a container for your function. If you don't yet have a container, you need to create one from the [Azure Functions portal](#). Then select **Create New**.

To generate a template based on the data that you want to compute, specify the context object that you plan to pass into a function. This object must be a JSON object. For example, if you pass in the file content from an FTP action, the context payload looks like this example:



### NOTE

Because this object wasn't cast as a string, the content is added directly to the JSON payload. However, an error occurs if the object is not a JSON token (that is, a string or a JSON object/array). To cast the object as a string, add quotes as shown in the first illustration in this article.

The designer then generates a function template that you can create inline. Variables are pre-created based on the context that you plan to pass into the function.

# Logic Apps Loops, Scopes, and Debatching

1/27/2017 • 3 min to read • [Edit Online](#)

Logic Apps provides a number of ways to work with arrays, collections, batches, and loops within a workflow.

## ForEach loop and arrays

Logic Apps allows you to loop over a set of data and perform an action for each item. This is possible via the `foreach` action. In the designer, you can specify to add a for each loop. After selecting the array you wish to iterate over, you can begin adding actions. Currently you are limited to only one action per foreach loop, but this restriction will be lifted in the coming weeks. Once within the loop you can begin to specify what should occur at each value of the array.

If using code-view, you can specify a for each loop like below. This is an example of a for each loop that sends an email for each email address that contains 'microsoft.com':

```
{
  "email_filter": {
    "type": "query",
    "inputs": {
      "from": "@triggerBody()['emails']",
      "where": "@contains(item()['email'], 'microsoft.com')"
    }
  },
  "forEach_email": {
    "type": "foreach",
    "foreach": "@body('email_filter')",
    "actions": {
      "send_email": {
        "type": "ApiConnection",
        "inputs": {
          "body": {
            "to": "@item()",
            "from": "me@contoso.com",
            "message": "Hello, thank you for ordering"
          },
          "host": {
            "connection": {
              "id": "@parameters('$connections')['office365']['connection']['id']"
            }
          }
        }
      }
    }
  },
  "runAfter": {
    "email_filter": [ "Succeeded" ]
  }
}
```

A `foreach` action can iterate over arrays up to 5,000 rows. Each iteration will execute in parallel by default.

### Sequential ForEach loops

To enable a foreach loop to execute sequentially, the `Sequential` operation option should be added.

```

"forEach_email": {
    "type": "foreach",
    "foreach": "@body('email_filter')",
    "operationOptions": "Sequential",
    "..."
}

```

## Until loop

You can perform an action or series of actions until a condition is met. The most common scenario for this is calling an endpoint until you get the response you are looking for. In the designer, you can specify to add an until loop. After adding actions inside the loop, you can set the exit condition, as well as the loop limits. There is a 1 minute delay between loop cycles.

If using code-view, you can specify an until loop like below. This is an example of calling an HTTP endpoint until the response body has the value 'Completed'. It will complete when either

- HTTP Response has status of 'Completed'
- It has tried for 1 hour
- It has looped 100 times

```

{
    "until_successful": {
        "type": "until",
        "expression": "@equals(actions('http')['status'], 'Completed')",
        "limit": {
            "count": 100,
            "timeout": "PT1H"
        },
        "actions": {
            "create_resource": {
                "type": "http",
                "inputs": {
                    "url": "http://provisionRsource.com",
                    "body": {
                        "resourceId": "@triggerBody()"
                    }
                }
            }
        }
    }
}

```

## SplitOn and debatching

Sometimes a trigger may receive an array of items that you want to debatch and start a workflow per item. This can be accomplished via the `spliton` command. By default, if your trigger swagger specifies a payload that is an array, a `spliton` will be added and start a run per item. SplitOn can only be added to a trigger. This can be manually configured or overridden in definition code-view. Currently SplitOn can debatch arrays up to 5,000 items. You cannot have a `spliton` and also implement the synchronous response pattern. Any workflow called that has a `response` action in addition to `spliton` will run asynchronously and send an immediate `202 Accepted` response.

SplitOn can be specified in code-view as the following example. This receives an array of items and debatches on each row.

```
{  
    "myDebatchTrigger": {  
        "type": "Http",  
        "inputs": {  
            "url": "http://getNewCustomers",  
        },  
        "recurrence": {  
            "frequency": "Second",  
            "interval": 15  
        },  
        "spliton": "@triggerBody()['rows']"  
    }  
}
```

## Scopes

It is possible to group a series of actions together using a scope. This is particularly useful for implementing exception handling. In the designer you can add a new scope, and begin adding any actions inside of it. You can define scopes in code-view like the following:

```
{  
    "myScope": {  
        "type": "scope",  
        "actions": {  
            "call_bing": {  
                "type": "http",  
                "inputs": {  
                    "url": "http://www.bing.com"  
                }  
            }  
        }  
    }  
}
```

# Create workflow definitions for logic apps using JSON

3/29/2017 • 6 min to read • [Edit Online](#)

You can create workflow definitions for [Azure Logic Apps](#) with simple, declarative JSON language. If you haven't already, first review [how to create your first logic app with Logic App Designer](#). Also, see the [full reference for the Workflow Definition Language](#).

## Repeat steps over a list

To iterate through an array that has up to 10,000 items and perform an action for each item, use the [foreach type](#).

## Handle failures if something goes wrong

Usually, you want to include a *remediation step* — some logic that executes *if and only if* one or more of your calls fail. This example gets data from various places, but if the call fails, we want to POST a message somewhere so we can track down that failure later:

```
{  
  "$schema": "https://schema.management.azure.com/providers/Microsoft.Logic/schemas/2016-06-01/workflowdefinition.json#",  
  "contentVersion": "1.0.0.0",  
  "parameters": {},  
  "triggers": {  
    "Request": {  
      "type": "request",  
      "kind": "http"  
    }  
  },  
  "actions": {  
    "readData": {  
      "type": "Http",  
      "inputs": {  
        "method": "GET",  
        "uri": "http://myurl"  
      }  
    },  
    "postToErrorMessageQueue": {  
      "type": "ApiConnection",  
      "inputs": "...",  
      "runAfter": {  
        "readData": [  
          "Failed"  
        ]  
      }  
    },  
    "outputs": {}  
  }  
}
```

To specify that `postToErrorMessageQueue` only runs after `readData` has `Failed`, use the `runAfter` property, for example, to specify a list of possible values, so that `runAfter` could be `["Succeeded", "Failed"]`.

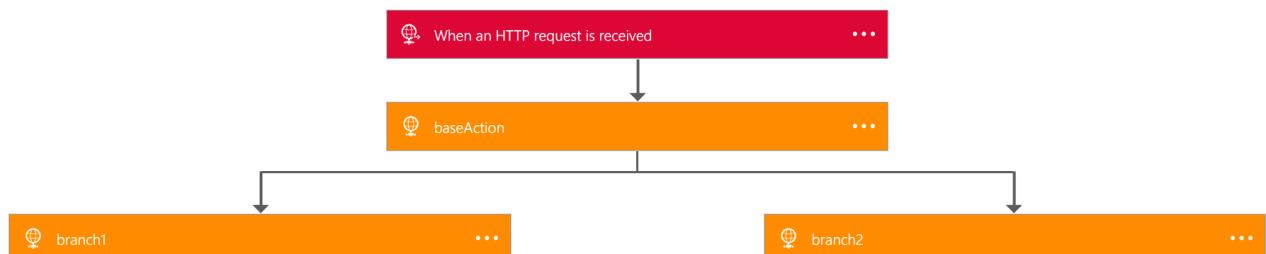
Finally, because this example now handles the error, we no longer mark the run as `Failed`. Because we added the step for handling this failure in this example, the run has `Succeeded` although one step `Failed`.

## Execute two or more steps in parallel

To run multiple actions in parallel, the `runAfter` property must be equivalent at runtime.

```
{  
    "$schema": "https://schema.management.azure.com/providers/Microsoft.Logic/schemas/2016-06-01/workflowdefinition.json#",  
    "contentVersion": "1.0.0.0",  
    "parameters": {},  
    "triggers": {  
        "Request": {  
            "kind": "http",  
            "type": "Request"  
        }  
    },  
    "actions": {  
        "readData": {  
            "type": "Http",  
            "inputs": {  
                "method": "GET",  
                "uri": "http://myurl"  
            }  
        },  
        "branch1": {  
            "type": "Http",  
            "inputs": {  
                "method": "GET",  
                "uri": "http://myurl"  
            }  
        },  
        "runAfter": {  
            "readData": [  
                "Succeeded"  
            ]  
        }  
    },  
    "branch2": {  
        "type": "Http",  
        "inputs": {  
            "method": "GET",  
            "uri": "http://myurl"  
        }  
    },  
    "runAfter": {  
        "readData": [  
            "Succeeded"  
        ]  
    }  
},  
"outputs": {}  
}
```

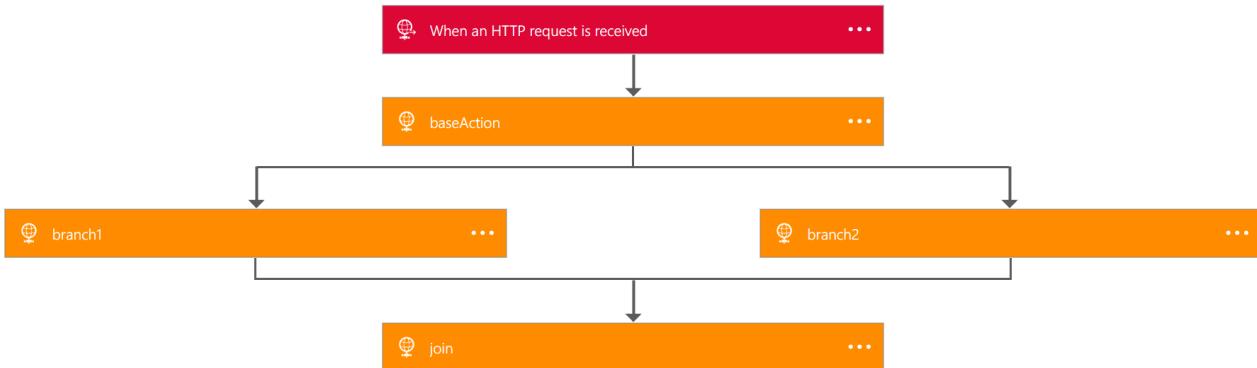
In this example, both `branch1` and `branch2` are set to run after `readData`. As a result, both branches run in parallel. The timestamp for both branches is identical.



Join two parallel branches

You can join two actions that are set to run in parallel by adding items to the `runAfter` property as in the previous example.

```
{
    "$schema": "https://schema.management.azure.com/providers/Microsoft.Logic/schemas/2016-04-01-preview/workflowdefinition.json#",
    "actions": {
        "readData": {
            "type": "Http",
            "inputs": {
                "method": "GET",
                "uri": "http://myurl"
            },
            "runAfter": {}
        },
        "branch1": {
            "type": "Http",
            "inputs": {
                "method": "GET",
                "uri": "http://myurl"
            },
            "runAfter": {
                "readData": [
                    "Succeeded"
                ]
            }
        },
        "branch2": {
            "type": "Http",
            "inputs": {
                "method": "GET",
                "uri": "http://myurl"
            },
            "runAfter": {
                "readData": [
                    "Succeeded"
                ]
            }
        },
        "join": {
            "type": "Http",
            "inputs": {
                "method": "GET",
                "uri": "http://myurl"
            },
            "runAfter": {
                "branch1": [
                    "Succeeded"
                ],
                "branch2": [
                    "Succeeded"
                ]
            }
        }
    },
    "parameters": {},
    "triggers": {
        "Request": {
            "type": "Request",
            "kind": "Http",
            "inputs": {
                "schema": {}
            }
        }
    },
    "contentVersion": "1.0.0.0",
    "outputs": {}
}
```



## Map list items to a different configuration

Next, let's say that we want to get different content based on the value of a property. We can create a map of values to destinations as a parameter:

```

{
    "$schema": "https://schema.management.azure.com/providers/Microsoft.Logic/schemas/2016-06-01/workflowdefinition.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "specialCategories": {
            "defaultValue": [
                "science",
                "google",
                "microsoft",
                "robots",
                "NSA"
            ],
            "type": "Array"
        },
        "destinationMap": {
            "defaultValue": {
                "science": "http://www.nasa.gov",
                "microsoft": "https://www.microsoft.com/en-us/default.aspx",
                "google": "https://www.google.com",
                "robots": "https://en.wikipedia.org/wiki/Robot",
                "NSA": "https://www.nsa.gov/"
            },
            "type": "Object"
        }
    },
    "triggers": {
        "Request": {
            "type": "Request",
            "kind": "http"
        }
    },
    "actions": {
        "getArticles": {
            "type": "Http",
            "inputs": {
                "method": "GET",
                "uri": "https://ajax.googleapis.com/ajax/services/feed/load?v=1.0&q=http://feeds.wired.com/wired/index"
            }
        },
        "forEachArticle": {
            "type": "foreach",
            "foreach": "@body('getArticles').responseData.feed.entries",
            "actions": {
                "ifGreater": {
                    "type": "if",
                    "expression": "@greater(length(intersection(item().categories, parameters('specialCategories'))),"
                }
            }
        }
    }
}

```

```

      },
      "actions": {
        "getSpecialPage": {
          "type": "Http",
          "inputs": {
            "method": "GET",
            "uri": "@parameters('destinationMap')[first(intersection(item().categories,
parameters('specialCategories')))]"
          }
        }
      }
    },
    "runAfter": {
      "getArticles": [
        "Succeeded"
      ]
    }
  }
}

```

In this case, we first get a list of articles. Based on the category that was defined as a parameter, the second step uses a map to look up the URL for getting the content.

Some times to note here:

- The `intersection()` function checks whether the category matches one of the known defined categories.
- After we get the category, we can pull the item from the map using square brackets: `parameters[...]`

## Process strings

You can use various functions to manipulate strings. For example, suppose we have a string that we want to pass to a system, but we aren't confident about proper handling for character encoding. One option is to base64 encode this string. However, to avoid escaping in a URL, we are going to replace a few characters.

We also want a substring of the order's name because the first five characters are not used.

```
{
    "$schema": "https://schema.management.azure.com/providers/Microsoft.Logic/schemas/2016-06-01/workflowdefinition.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "order": {
            "defaultValue": {
                "quantity": 10,
                "id": "myorder1",
                "orderer": "NAME=Stephon_Ian"
            },
            "type": "Object"
        }
    },
    "triggers": {
        "request": {
            "type": "request",
            "kind": "http"
        }
    },
    "actions": {
        "order": {
            "type": "Http",
            "inputs": {
                "method": "GET",
                "uri": "http://www.example.com/?id=@{replace(replace(base64(substring(parameters('order').orderer,5,sub(length(parameters('order').orderer),5)),'+','-'), '/','_'))}"
            }
        },
        "outputs": {}
    }
}
```

Working from inside to outside:

1. Get the `length()` for the orderer's name, so we get back the total number of characters.
2. Subtract 5 because we want a shorter string.
3. Actually, take the `substring()`. We start at index `5` and go the remainder of the string.
4. Convert this substring to a `base64()` string.
5. `replace()` all the `+` characters with `-` characters.
6. `replace()` all the `/` characters with `_` characters.

## Work with Date Times

Date Times can be useful, particularly when you are trying to pull data from a data source that doesn't naturally support *triggers*. You can also use Date Times for finding how long various steps are taking.

```
{
    "$schema": "https://schema.management.azure.com/providers/Microsoft.Logic/schemas/2016-06-01/workflowdefinition.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "order": {
            "defaultValue": {
                "quantity": 10,
                "id": "myorder1"
            },
            "type": "Object"
        }
    },
    "triggers": {
        "Request": {
            "type": "request",
            "kind": "http"
        }
    },
    "actions": {
        "order": {
            "type": "Http",
            "inputs": {
                "method": "GET",
                "uri": "http://www.example.com/?id=@{parameters('order').id}"
            }
        },
        "ifTimingWarning": {
            "type": "If",
            "expression": "@less(actions('order').startTime,addseconds(utcNow(),-1))",
            "actions": {
                "timingWarning": {
                    "type": "Http",
                    "inputs": {
                        "method": "GET",
                        "uri": "http://www.example.com/?recordLongOrderTime=@{parameters('order').id}&currentTime=@{utcNow('r')}"
                    }
                }
            }
        },
        "runAfter": {
            "order": [
                "Succeeded"
            ]
        }
    },
    "outputs": {}
}
```

In this example, we extract the `startTime` from the previous step. Then we get the current time, and subtract one second:

`addseconds(..., -1)`

You can use other units of time, like `minutes` or `hours`. Finally, we can compare these two values. If the first value is less than the second value, then more than one second has passed since the order was first placed.

To format dates, we can use string formatters. For example, to get the RFC1123, we use `utcnow('r')`. To learn about date formatting, see [Workflow Definition Language](#).

## Deployment parameters for different environments

Commonly, deployment lifecycles have a development environment, a staging environment, and a production

environment. For example, you might use the same definition in all these environments but use different databases. Likewise, you might want to use the same definition across different regions for high availability but want each logic app instance to talk to that region's database. This scenario differs from taking parameters at *runtime* where instead, you should use the `trigger()` function as in the previous example.

You can start with a basic definition like this example:

```
{  
    "$schema": "https://schema.management.azure.com/providers/Microsoft.Logic/schemas/2016-06-  
01/workflowdefinition.json#",  
    "contentVersion": "1.0.0.0",  
    "parameters": {  
        "uri": {  
            "type": "string"  
        }  
    },  
    "triggers": {  
        "request": {  
            "type": "request",  
            "kind": "http"  
        }  
    },  
    "actions": {  
        "readData": {  
            "type": "Http",  
            "inputs": {  
                "method": "GET",  
                "uri": "@parameters('uri')"  
            }  
        }  
    },  
    "outputs": {}  
}
```

In the actual `PUT` request for the logic apps, you can provide the parameter `uri`. Because a default value no longer exists, the logic app payload requires this parameter:

```
{  
    "properties": {},  
    "definition": {  
        // Use the definition from above here  
    },  
    "parameters": {  
        "connection": {  
            "value": "https://my.connection.that.is.per.enviornment"  
        }  
    },  
    "location": "westus"  
}
```

In each environment, you can provide a different value for the `connection` parameter.

For all the options that you have for creating and managing logic apps, see the [REST API documentation](#).

# Call, trigger, or nest workflows with HTTP endpoints in logic apps

4/3/2017 • 8 min to read • [Edit Online](#)

You can natively expose synchronous HTTP endpoints as triggers on logic apps so that you can trigger or call your logic apps through a URL. You can also nest workflows in your logic apps by using a pattern of callable endpoints.

To create HTTP endpoints, you can add these triggers so that your logic apps can receive incoming requests:

- [Request](#)
- [API Connection Webhook](#)
- [HTTP Webhook](#)

#### NOTE

Although our examples use the **Request** trigger, you can use any of the listed HTTP triggers, and all principles identically apply to the other trigger types.

## Set up an HTTP endpoint for your logic app

To create an HTTP endpoint, add a trigger that can receive incoming requests.

1. Sign in to the [Azure portal](#). Go to your logic app, and open Logic App Designer.
2. Add a trigger that lets your logic app receive incoming requests. For example, add the **Request** trigger to your logic app.
3. Under **Request Body JSON Schema**, you can optionally enter a JSON schema for the payload (data) that you expect the trigger to receive.

The designer uses this schema for generating tokens that your logic app can use to consume, parse, and pass data from the trigger through your workflow. More about [tokens generated from JSON schemas](#).

For this example, enter the schema shown in the designer:

```
{  
  "type": "object",  
  "properties": {  
    "address": {  
      "type": "string"  
    }  
  },  
  "required": [  
    "address"  
  ]  
}
```

The screenshot shows the 'Request' trigger configuration in the Azure Logic Apps designer. At the top, there's a red header bar with the 'Request' trigger name. Below it, a pink header bar says 'HTTP POST to this URL'. A grey input field contains the placeholder 'URL will be generated after save' with a copy icon to its right. Underneath, a section titled 'Request Body JSON Schema' displays the following JSON schema:

```
{ "type": "object", "properties": { "address": { "type": "string" } }, "required": [ "address" ] }
```

Below the schema, a purple link says 'Use sample payload to generate schema'. At the bottom left, there's a link 'Show advanced options ▾'.

**TIP**

You can generate a schema for a sample JSON payload from a tool like [jsonschema.net](#), or in the **Request** trigger by choosing **Use sample payload to generate schema**. Enter your sample payload, and choose **Done**.

For example, this sample payload:

```
{ "address": "21 2nd Street, New York, New York" }
```

generates this schema:

```
}
```

```
    "type": "object",
    "properties": {
        "address": {
            "type": "string"
        }
    }
}
```

4. Save your logic app. Under **HTTP POST to this URL**, you should now find a generated callback URL, like this example:

The screenshot shows the 'Request' trigger configuration page again, but now the 'HTTP POST to this URL' field contains a generated URL: <https://prod-00.southcentralus.logic.azure.com:443/workflows/f9...>. The rest of the interface is identical to the first screenshot.

This URL contains a Shared Access Signature (SAS) key in the query parameters that are used for authentication. You can also get the HTTP endpoint URL from your logic app overview in the Azure portal. Under **Trigger History**, select your trigger:

The screenshot shows the Azure Logic App Overview page for 'MyLogicApp'. The left sidebar has a red circle labeled '1' over the 'Overview' item. The main area shows the 'Essentials' section with details like Resource group (MyLogicAppResourceGroup), Location (South Central), Subscription (Visual Studio Enterprise), and Plan (Consumption). A red circle labeled '2' is over the 'Trigger History' section, which includes a dropdown for 'manual' and a 'Callback url [POST]' field containing 'https://prod-00.southcentralus...'. A red circle labeled '3' is over the URL field. Below the trigger history is a table with columns STATUS, START TIME, IDENTIFIER, and DURATION, showing 'No runs'.

Or you can get the URL by making this call:

```
POST https://management.azure.com/{logic-app-resourceID}/triggers/{myendpointtrigger}/listCallbackURL?  
api-version=2016-06-01
```

## Change the HTTP method for your trigger

By default, the **Request** trigger expects an HTTP POST request, but you can use a different HTTP method.

### NOTE

You can specify only one method type.

1. On your **Request** trigger, choose **Show advanced options**.
2. Open the **Method** list. For this example, select **GET** so that you can test your HTTP endpoint's URL later.

### NOTE

You can select any other HTTP method, or specify a custom method for your own logic app.

The screenshot shows the 'Request' trigger configuration in the Azure Logic Apps designer. At the top, there's a red header bar with a globe icon and the word 'Request'. To the right of the header is a '...' button. Below the header, the interface is divided into several sections:

- HTTP POST to this URL:** A text input field containing the URL `https://prod-00.southcentralus.logic.azure.com:443/workflows/f9...`. To the right of the URL is a small icon of a clipboard with a checkmark.
- Request Body JSON Schema:** A code editor showing a JSON schema definition:

```
{
  "type": "object",
  "properties": {
    "address": {
      "type": "string"
    }
  },
  "required": [
    "address"
  ]
}
```
- Use sample payload to generate schema**: A purple link below the schema editor.
- Method:** A dropdown menu where the 'GET' option is selected and highlighted with a red border. Other options include PUT, POST, PATCH, and DELETE, along with a 'Enter custom value' option.
- Relative path:** An empty text input field for specifying a relative path for the trigger.
- Hide advanced options ↗**: A link at the bottom of the configuration area.

## Accept parameters through your HTTP endpoint URL

When you want your HTTP endpoint URL to accept parameters, customize your trigger's relative path.

1. On your **Request** trigger, choose **Show advanced options**.
2. Under **Method**, specify the HTTP method that you want your request to use. For this example, select the **GET** method, if you haven't already, so that you can test your HTTP endpoint's URL.

### NOTE

When you specify a relative path for your trigger, you must also explicitly specify an HTTP method for your trigger.

3. Under **Relative path**, specify the relative path for the parameter that your URL should accept, for example, `customers/{customerID}`.

Request

HTTP POST to this URL

<https://prod-00.southcentralus.logic.azure.com:443/workflows/f9...>

Request Body JSON Schema

```
{
  "type": "object",
  "properties": {
    "address": {
      "type": "string"
    }
  },
  "required": [
    "address"
  ]
}
```

Use sample payload to generate schema

Method

GET

Relative path

customers/{customerID}

Hide advanced options ^

4. To use the parameter, add a **Response** action to your logic app. (Under your trigger, choose **New step > Add an action > Response**)
5. In your response's **Body**, include the token for the parameter that you specified in your trigger's relative path.

For example, to return `Hello {customerID}`, update your response's **Body** with `Hello {customerID token}`. The dynamic content list should appear and show the `customerID` token for you to select.

Relative path

customers/{customerID}

Hide advanced options ^

Response

\* Status Code

200

Headers

Enter JSON object of response headers

Body

1 Hello

Add dynamic content

+ New step

Add dynamic content from the apps and services used in this flow. Hide

Search dynamic content

Request

2 Body

customerID  
This is a path parameter.

Headers

Path Parameters

Queries

Your **Body** should look like this example:

The screenshot shows a 'Body' step in the Logic App Designer. Inside the step, the text 'Hello' is followed by a red placeholder box containing 'customerID'. Below the step, there is a button labeled 'Add dynamic content' with a plus sign icon.

## 6. Save your logic app.

Your HTTP endpoint URL now includes the relative path, for example:

`https://prod-00.southcentralus.logic.azure.com/workflows/f90cb66c52ea4e9cabe0abf4e197deff/triggers/manual/paths/invoke/customers/{customerID}...`

## 7. To test your HTTP endpoint, copy and paste the updated URL into another browser window, but replace `{customerID}` with `123456`, and press Enter.

Your browser should show this text:

`Hello 123456`

### Tokens generated from JSON schemas for your logic app

When you provide a JSON schema in your **Request** trigger, the Logic App Designer generates tokens for properties in that schema. You can then use those tokens for passing data through your logic app workflow.

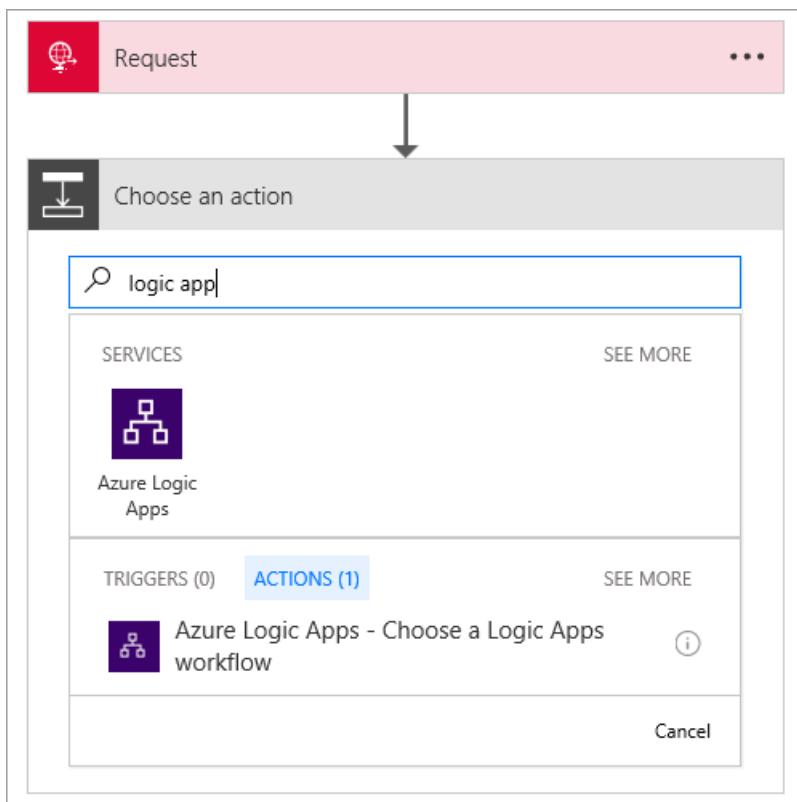
For this example, if you add the `title` and `name` properties to your JSON schema, their tokens are now available to use in later workflow steps.

Here is the complete JSON schema:

```
{
  "type": "object",
  "properties": {
    "address": {
      "type": "string"
    },
    "title": {
      "type": "string"
    },
    "name": {
      "type": "string"
    }
  },
  "required": [
    "address",
    "title",
    "name"
  ]
}
```

## Create nested workflows for logic apps

You can nest workflows in your logic app by adding other logic apps that can receive requests. To include these logic apps, add the **Azure Logic Apps - Choose a Logic Apps workflow** action to your trigger. You can then select from eligible logic apps.



## Call or trigger logic apps through HTTP endpoints

After you create your HTTP endpoint, you can trigger your logic app through a `POST` method to the full URL. Logic apps have built-in support for direct-access endpoints.

## Reference content from an incoming request

If the content's type is `application/json`, you can reference properties from the incoming request. Otherwise, content is treated as a single binary unit that you can pass to other APIs. You can't reference this content inside the workflow without converting that content. For example, if you pass `application/xml` content, you can use `@xpath()` for an XPath extraction, or `@json()` for converting XML to JSON. Learn about [working with content types](#).

To get the output from an incoming request, you can use the `@triggerOutputs()` function. The output might look like this example:

```
{  
  "headers": {  
    "content-type" : "application/json"  
  },  
  "body": {  
    "myProperty" : "property value"  
  }  
}
```

To access the `body` property specifically, you can use the `@triggerBody()` shortcut.

## Respond to requests

You might want to respond to certain requests that start a logic app by returning content to the caller. To construct the status code, header, and body for your response, you can use the **Response** action. This action can appear anywhere in your logic app, not just at the end of your workflow.

## NOTE

If your logic app doesn't include a **Response**, the HTTP endpoint responds *immediately* with a **202 Accepted** status. Also, for the original request to get the response, all steps required for the response must finish within the [request timeout limit](#) unless you call the workflow as a nested logic app. If no response happens within this limit, the incoming request times out and receives the HTTP response **408 Client timeout**. For nested logic apps, the parent logic app continues to wait for a response until completed, regardless of how much time is required.

## Construct the response

You can include more than one header and any type of content in the response body. In our example response, the header specifies that the response has content type `application/json` , and the body contains `title` and `name` , based on the JSON schema updated previously for the **Request** trigger.

The screenshot shows the Logic App designer interface with a 'Response' action selected. The configuration pane is divided into sections:

- Status Code:** Set to 200.
- Headers:** Contains the following JSON:

```
{  
  "content-type": "application/json"  
}
```
- Body:** Contains the following JSON:

```
{  
  "name": "name",  
  "title": "title"  
}
```

A button labeled "Add dynamic content" with a plus sign is located at the bottom right of the body section.

Responses have these properties:

PROPERTY	DESCRIPTION
statusCode	Specifies the HTTP status code for responding to the incoming request. This code can be any valid status code that starts with 2xx, 4xx, or 5xx. However, 3xx status codes are not permitted.
headers	Defines any number of headers to include in the response.
body	Specifies a body object that can be a string, a JSON object, or even binary content referenced from a previous step.

Here's what the JSON schema looks like now for the **Response** action:

```

"Response": {
    "inputs": {
        "body": {
            "title": "@{triggerBody()['title']}",
            "name": "@{triggerBody()['name']}"
        },
        "headers": {
            "content-type": "application/json"
        },
        "statusCode": 200
    },
    "runAfter": {},
    "type": "Response"
}

```

### TIP

To view the complete JSON definition for your logic app, on the Logic App Designer, choose **Code view**.

## Q & A

**Q: What about URL security?**

A: Azure securely generates logic app callback URLs using a Shared Access Signature (SAS). This signature passes through as a query parameter and must be validated before your logic app can fire. Azure generates the signature using a unique combination of a secret key per logic app, the trigger name, and the operation that's performed. So unless someone has access to the secret logic app key, they cannot generate a valid signature.

**Q: Can I configure HTTP endpoints further?**

A: Yes, HTTP endpoints support more advanced configuration through [API Management](#). This service also offers the capability for you to consistently manage all your APIs, including logic apps, set up custom domain names, use more authentication methods, and more, for example:

- [Change the request method](#)
- [Change the URL segments of the request](#)
- Set up your API Management domains in the [Azure portal](#)
- Set up policy to check for Basic authentication

**Q: What changed when the schema migrated from the December 1, 2014 preview?**

A: Here's a summary about these changes:

DECEMBER 1, 2014 PREVIEW	JUNE 1, 2016
Click <b>HTTP Listener</b> API App	Click <b>Manual trigger</b> (no API App required)
HTTP Listener setting " <i>Sends response automatically</i> "	Either include a <b>Response</b> action or not in the workflow definition
Configure Basic or OAuth authentication	via API Management
Configure HTTP method	Under <b>Show advanced options</b> , choose an HTTP method
Configure relative path	Under <b>Show advanced options</b> , add a relative path
Reference the incoming body through <code>@triggerOutputs().body.Content</code>	Reference through <code>@triggerOutputs().body</code>

DECEMBER 1, 2014 PREVIEW

JUNE 1, 2016

Send **HTTP response** action on the HTTP Listener

Click **Respond to HTTP request** (no API App required)

## Get help

To ask questions, answer questions, and learn what other Azure Logic Apps users are doing, visit the [Azure Logic Apps forum](#).

To help improve Azure Logic Apps and connectors, vote on or submit ideas at the [Azure Logic Apps user feedback site](#).

## Next steps

- [Author logic app definitions](#)
- [Handle errors and exceptions](#)

# Creating a custom API to use with Logic Apps

1/20/2017 • 6 min to read • [Edit Online](#)

If you want to extend the Logic Apps platform, there are many ways you can call into APIs or systems that aren't available as one of our many out-of-the-box connectors. One of those ways to create an API App you can call from within a Logic App workflow.

## Helpful Tools

For APIs to work best with Logic Apps, we recommend generating a [swagger](#) doc detailing the supported operations and parameters for your API. There are many libraries (like [Swashbuckle](#)) that will automatically generate the swagger for you. You can also use [TREx](#) to help annotate the swagger to work well with Logic Apps (display names, property types, etc.). For some samples of API Apps built for Logic Apps, be sure to check out our [GitHub repository](#) or [blog](#).

## Actions

The basic action for a Logic App is a controller that will accept an HTTP Request and return a response (usually 200). However there are different patterns you can follow to extend actions based on your needs.

By default the Logic App engine will timeout a request after 1 minute. However, you can have your API execute on actions that take longer, and have the engine wait for completion, by following either an async or webhook pattern detailed below.

For standard actions, simply write an HTTP request method in your API which is exposed via swagger. You can see samples of API apps that work with Logic Apps in our [GitHub repository](#). Below are ways to accomplish common patterns with a custom connector.

### Long Running Actions - Async Pattern

When running a long step or task, the first thing you need to do is make sure the engine knows you haven't timed out. You also need to communicate with the engine how it will know when you are finished with the task, and finally, you need to return relevant data to the engine so it can continue with the workflow. You can complete that via an API by following the flow below. These steps are from the point-of-view of the custom API:

1. When a request is received, immediately return a response (before work is done). This response will be a

`202 ACCEPTED` response, letting the engine know you got the data, accepted the payload, and are now processing. The 202 response should contain the following headers:

- `location` header (required): This is an absolute path to the URL Logic Apps can use to check the status of the job.
- `retry-after` (optional, will default to 20 for actions). This is the number of seconds the engine should wait before polling the location header URL to check status.

2. When a job status is checked, perform the following checks:

- If the job is done: return a `200 OK` response, with the response payload.
- If the job is still processing: return another `202 ACCEPTED` response, with the same headers as the initial response

This pattern allows you to run extremely long tasks within a thread of your custom API, but keep an active connection alive with the Logic Apps engine so it doesn't timeout or continue before work is completed. When adding this into your Logic App, it's important to note you do not need do anything in your definition for the Logic

App to continue to poll and check the status. As soon as the engine sees a 202 ACCEPTED response with a valid location header, it will honor the async pattern and continue to poll the location header until a non-202 is returned.

You can see a sample of this pattern in GitHub [here](#)

## Webhook Actions

During your workflow, you can have the Logic App pause and wait for a "callback" to continue. This callback comes in the form of an HTTP POST. To implement this pattern, you need to provide two endpoints on your controller: subscribe and unsubscribe.

On 'subscribe', the Logic App will create and register a callback URL which your API can store and callback with ready as an HTTP POST. Any content/headers will be passed into the Logic App and can be used within the remainder of the workflow. The Logic App engine will call the subscribe point on execution as soon as it hits that step.

If the run was cancelled, the Logic App engine will make a call to the 'unsubscribe' endpoint. Your API can then unregister the callback URL as needed.

Currently the Logic App Designer doesn't support discovering a webhook endpoint through swagger, so to use this type of action you must add the "Webhook" action and specify the URL, headers, and body of your request.

You can use the `@listCallbackUrl()` workflow function in any of those fields as needed to pass in the callback URL.

You can see a sample of a webhook pattern in GitHub [here](#)

# Triggers

In addition to actions, you can have your custom API act as a trigger to a Logic App. There are two patterns you can follow below to trigger a Logic App:

## Polling Triggers

Polling triggers act much like the Long Running Async actions above. The Logic App engine will call the trigger endpoint after a certain period of time elapsed (dependent on SKU, 15 seconds for Premium, 1 minute for Standard, and 1 hour for Free).

If there is no data available, the trigger returns a `202 ACCEPTED` response, with a `location` and `retry-after` header. However, for triggers it is recommended the `location` header contains a query parameter of `triggerState`. This is some identifier for your API to know when the last time the Logic App fired. If there is data available, the trigger returns a `200 OK` response with the content payload. This will fire the Logic App.

For example if I was polling to see if a file was available, you could build a polling trigger that would do the following:

- If a request was received with no triggerState the API would return a `202 ACCEPTED` with a `location` header that has a triggerState of the current time and a `retry-after` of 15.
- If a request was received with a triggerState:
  - Check to see if any files were added after the triggerState DateTime.
  - If there is 1 file, return a `200 OK` response with the content payload, increment the triggerState to the DateTime of the file I returned, and set the `retry-after` to 15.
  - If there are multiple files, I can return 1 at a time with a `200 OK`, increment my triggerState in the `location` header, and set `retry-after` to 0. This will let the engine know there is more data available and it will immediately request it at the `location` header specified.
  - If there are no files available, return a `202 ACCEPTED` response, and leave the `location` triggerState the same. Set `retry-after` to 15.

You can see a sample of a polling trigger in GitHub [here](#)

## Webhook Triggers

Webhook triggers act much like Webhook Actions above. The Logic App engine will call the 'subscribe' endpoint whenever a webhook trigger is added and saved. Your API can register the webhook URL and call it via HTTP POST whenever data is available. The content payload and headers will be passed into the Logic App run.

If a webhook trigger is ever deleted (either the Logic App entirely, or just the webhook trigger), the engine will make a call to the 'unsubscribe' URL where your API can unregister the callback URL and stop any processes as needed.

Currently the Logic App Designer doesn't support discovering a webhook trigger through swagger, so to use this type of action you must add the "Webhook" trigger and specify the URL, headers, and body of your request. You can use the `@listCallbackUrl()` workflow function in any of those fields as needed to pass in the callback URL.

You can see a sample of a webhook trigger in GitHub [here](#)

# Call custom APIs hosted on Azure App Service with Azure Logic Apps

2/16/2017 • 6 min to read • [Edit Online](#)

Although Azure Logic Apps offers 40+ connectors for various services, you might want to call into your own custom API that can run your own code. Azure App Service provides one of the easiest and most scalable ways for hosting your own custom web APIs. This article covers how to call into any web API hosted in an App Service API app, web app, or mobile app. Learn [how to build APIs as a trigger or action in logic apps](#).

## Deploy your web app

First, you must deploy your API as a web app in Azure App Service. Learn about [basic deployment when you create an ASP.NET web app](#). While you can call into any API from a logic app, for the best experience, we recommend that you add Swagger metadata to integrate easily with logic app actions. Learn about [adding Swagger metadata](#).

### API settings

For Logic App Designer to parse your Swagger, you must enable CORS and set the API Definition properties for your web app.

1. In the Azure portal, select your web app.
2. In the blade that opens, find **API**, and select **API definition**. Set the **API definition location** to your `swagger.json` file's URL.

Usually, this URL is `https://[name].azurewebsites.net/swagger/docs/v1`.

3. To allow requests from Logic App Designer, under **API**, select **CORS**, and set a CORS policy for '\*'.

## Call into your custom API

If you set up CORS and the API Definition properties, you should be able to easily add Custom API actions to your workflow in the Logic Apps portal. In the Logic Apps Designer, you can browse your subscription websites to list the websites that have a defined Swagger URL. You can also point to a Swagger and list the available actions and inputs by using the HTTP + Swagger action. Finally, you can always create a request using the HTTP action to call any API, even APIs that don't have or expose a Swagger doc.

To secure your API, you have a couple different ways to do that:

- No code changes required. You can use Azure Active Directory to protect your API without requiring any code changes or redeployment.
- In your API's code, enforce Basic Authentication, Azure Active Directory authentication, or Certificate Authentication.

## Secure calls to your API without changing code

In this section, you will create two Azure Active Directory applications – one for your logic app and one for your web app. Authenticate calls to your web app by using the service principal (client id and secret) associated with your logic app's Azure Active Directory app. Finally, include the application IDs in your logic app definition.

### Part 1: Set up an application identity for your logic app

Your logic app uses this application identity to authenticate against Azure Active Directory. You only have to set up this identity once for your directory. For example, you can choose to use the same identity for all your logic apps,

although you can also create unique identities per logic app. You can set up these identities either in the Azure portal or use PowerShell.

#### Create the application identity in the Azure classic portal

1. In the Azure classic portal, go to your [Azure Active Directory](#).
2. Select the directory that you use for your web app.
3. Choose the **Applications** tab. In the command bar at the bottom of the page, choose **Add**.
4. Give your app identity a name, and click the next arrow.
5. Under **App properties**, put in a unique string formatted as a domain, and click the checkmark.
6. Choose the **Configure** tab. Go to **Client ID**, and copy the client ID for use in your logic app.
7. Under **Keys**, open the **Select duration** list, and select the duration time for your key.
8. At the bottom of the page, click **Save**. You might have to wait a few seconds.
9. Make sure to copy the key that now appears under **Keys**, so you can use this key in your logic app.

#### Create the application identity using PowerShell

1. `Switch-AzureMode AzureResourceManager`
2. `Add-AzureAccount`
3. 

```
New-AzureADApplication -DisplayName "MyLogicAppID" -HomePage "http://somerandomdomain.tld" -IdentifierUris "http://somerandomdomain.tld" -Password "Pass@word1!"
```
4. Make sure to copy the **Tenant ID**, the **Application ID**, and the password you used.

### Part 2: Protect your web app with an Azure Active Directory app identity

If your web app is already deployed, you can enable authorization in the Azure portal. Otherwise, you can make enabling authorization part of your Azure Resource Manager deployment.

#### Enable authorization in the Azure portal

1. Find and select your web app. Under **Settings**, choose **Authentication/Authorization**.
2. Under **App Service Authentication**, turn authentication **On**, and choose **Save**.

At this point, an Application is automatically created for you. You need this Application's Client ID for Part 3, so you must follow these steps:

1. In the Azure classic portal, go to your [Azure Active Directory](#).
2. Select your directory.
3. In the search box, find your app.
4. In the list, select your app.
5. Choose the **Configure** tab where you should see the **Client ID**.

#### Deploy your web app using an Azure Resource Manager template

First, you must create an Application for your web app that's different from the Application that you use for your logic app. Start by following the previous steps in Part 1, but for **HomePage** and **IdentifierUris**, use your web app's actual <https://URL>.

#### NOTE

When you create the Application for your web app, you must use the [Azure classic portal](#). The PowerShell commandlet doesn't set up the required permissions to sign users into a website.

After you have the client ID and tenant ID, include this part as a sub resource of your web app in your deployment template:

```

"resources": [
    {
        "apiVersion": "2015-08-01",
        "name": "web",
        "type": "config",
        "dependsOn": ["[concat('Microsoft.Web/sites/', 'parameters('webAppName')))]"],
        "properties": {
            "siteAuthEnabled": true,
            "siteAuthSettings": {
                "clientId": "<<clientID>>",
                "issuer": "https://sts.windows.net/<<tenantID>>/",
            }
        }
    }
]

```

To automatically run a deployment that deploys a blank web app and logic app together that use Azure Active Directory, click **Deploy to Azure**:



For the complete template, see [Logic app calls into a custom API hosted on App Service and protected by Azure Active Directory](#).

### Part 3: Populate the Authorization section in your logic app

In the **Authorization** section of the **HTTP** action:

```
{"tenant": "<<tenantId>>", "audience": "<<clientID from Part 2>>", "clientId": "<<clientID from Part 1>>", "secret": "<<Password or Key from Part 1>>", "type": "ActiveDirectoryOAuth" }
```

ELEMENT	DESCRIPTION
type	Type of authentication. For ActiveDirectoryOAuth authentication, the value is <code>ActiveDirectoryOAuth</code> .
tenant	The tenant identifier used to identify the AD tenant.
audience	Required. The resource you are connecting to.
clientID	The client identifier for the Azure AD application.
secret	Required. Secret of the client that is requesting the token.

The previous template has this authorization section set up already, but if you are authoring the logic app directly, you must include the full authorization section.

## Secure your API in code

### Certificate authentication

You can use client certificates to validate the incoming requests to your web app. For how to set up your code, see [How to configure TLS mutual authentication for web app](#).

In the **Authorization** section, you should include:

```
{"type": "clientcertificate", "password": "test", "pfx": "long-pfx-key"}
```

ELEMENT	DESCRIPTION
type	Required. Type of authentication. For SSL client certificates, the value must be <code>clientCertificate</code> .
pfx	Required. Base64-encoded contents of the PFX file.
password	Required. Password to access the PFX file.

## Basic authentication

To validate the incoming requests, you can use basic authentication, such as username and password. Basic authentication is a common pattern, and you can use this authentication in any language used to build your app.

In the **Authorization** section, you should include:

```
{"type": "basic", "username": "test", "password": "test"}.
```

ELEMENT	DESCRIPTION
type	Required. Type of authentication. For Basic authentication, the value must be <code>Basic</code> .
username	Required. Username to authenticate.
password	Required. Password to authenticate.

## Handle Azure Active Directory authentication in code

By default, the Azure Active Directory authentication that you enable in the Azure portal doesn't provide fine-grained authorization. For example, this authentication doesn't lock your API to a specific user or app, but just to a particular tenant.

To restrict your API to your logic app, for example, in code, extract the header that has the JWT. Check the caller's identity, and reject requests that don't match.

Going further, to implement this authentication entirely in your own code, and not use the Azure portal feature, see [Authenticate with on-premises Active Directory in your Azure app](#). To create an Application identity for your logic app and use that identity to call your API, you must follow the previous steps.

# Handle errors and exceptions in Azure Logic Apps

3/9/2017 • 6 min to read • [Edit Online](#)

Azure Logic Apps provides rich tools and patterns to help you make sure your integrations are robust and resilient against failures. Any integration architecture poses the challenge of making sure to appropriately handle downtime or issues from dependent systems. Logic Apps makes handling errors a first-class experience, giving you the tools you need to act on exceptions and errors in your workflows.

## Retry policies

A retry policy is the most basic type of exception and error handling. If an initial request timed out or failed (any request that results in a 429 or 5xx response), this policy defines whether the action should retry. By default, all actions retry 4 additional times over 20-second intervals. So if the first request receives a `500 Internal Server Error` response, the workflow engine pauses for 20 seconds, and attempts the request again. If after all retries, the response is still an exception or failure, the workflow continues and marks the action status as `Failed`.

You can configure retry policies in the **inputs** for a particular action. For example, you can configure a retry policy to try as many as 4 times over 1-hour intervals. For full details about input properties, see [Workflow Actions and Triggers](#).

```
"retryPolicy" : {  
    "type": "<type-of-retry-policy>",  
    "interval": "<retry-interval>",  
    "count": "<number-of-retry-attempts>"  
}
```

If you wanted your HTTP action to retry 4 times and wait 10 minutes between each attempt, you would use the following definition:

```
"HTTP":  
{  
    "inputs": {  
        "method": "GET",  
        "uri": "http://myAPIendpoint/api/action",  
        "retryPolicy" : {  
            "type": "fixed",  
            "interval": "PT10M",  
            "count": 4  
        }  
    },  
    "runAfter": {},  
    "type": "Http"  
}
```

For more information on supported syntax, see the [retry-policy section in Workflow Actions and Triggers](#).

## Catch failures with the RunAfter property

Each logic app action declares which actions must finish before the action starts, like ordering the steps in your workflow. In the action definition, this ordering is known as the `runAfter` property. This property is an object that describes which actions and action statuses execute the action. By default, all actions added through the Logic App

Designer are set to `runAfter` the previous step if the previous step `Succeeded`. However, you can customize this value to fire actions when previous actions have `Failed`, `Skipped`, or a possible set of these values. If you wanted to add an item to a designated Service Bus topic after a specific action `Insert_Row` fails, you could use the following `runAfter` configuration:

```
"Send_message": {
    "inputs": {
        "body": {
            "ContentData": "@{encodeBase64(body('Insert_Row'))}",
            "ContentType": "{ \"content-type\" : \"application/json\" }"
        },
        "host": {
            "api": {
                "runtimeUrl": "https://logic-apis-westus.azure-apim.net/apim/servicebus"
            },
            "connection": {
                "name": "@parameters('$connections')['servicebus']['connectionId']"
            }
        },
        "method": "post",
        "path": "/@{encodeURIComponent('failures')}/messages"
    },
    "runAfter": {
        "Insert_Row": [
            "Failed"
        ]
    }
}
```

Notice the `runAfter` property is set to fire if the `Insert_Row` action is `Failed`. To run the action if the action status is `Succeeded`, `Failed`, or `Skipped`, use this syntax:

```
"runAfter": {
    "Insert_Row": [
        "Failed", "Succeeded", "Skipped"
    ]
}
```

#### TIP

Actions that run and complete successfully after a preceding action has failed, are marked as `Succeeded`. This behavior means that if you successfully catch all failures in a workflow, the run itself is marked as `Succeeded`.

## Scopes and results to evaluate actions

Similar to how you can run after individual actions, you can also group actions together inside a `scope`, which act as a logical grouping of actions. Scopes are useful both for organizing your logic app actions, and for performing aggregate evaluations on the status of a scope. The scope itself receives a status after all actions in a scope have finished. The scope status is determined with the same criteria as a run. If the final action in an execution branch is `Failed` or `Aborted`, the status is `Failed`.

To fire specific actions for any failures that happened within the scope, you can use `runAfter` with a scope that is marked `Failed`. If *any* actions in the scope fail, running after a scope fails lets you create a single action to catch failures.

### Getting the context of failures with results

Although catching failures from a scope is useful, you might also want context to help you understand exactly

which actions failed, and any errors or status codes that were returned. The `@result()` workflow function provides context about the result of all actions in a scope.

`@result()` takes a single parameter, scope name, and returns an array of all the action results from within that scope. These action objects include the same attributes as the `@actions()` object, including action start time, action end time, action status, action inputs, action correlation IDs, and action outputs. To send context of any actions that failed within a scope, you can easily pair an `@result()` function with a `runAfter`.

To execute an action *for each* action in a scope that `Failed`, filter the array of results to actions that failed, you can pair `@result()` with a **Filter Array** action and a **ForEach** loop. You can take the filtered result array and perform an action for each failure using the **ForEach** loop. Here's an example, followed by a detailed explanation, that sends an HTTP POST request with the response body of any actions that failed within the scope `My_Scope`.

```
"Filter_array": {
    "inputs": {
        "from": "@result('My_Scope')",
        "where": "@equals(item()['status'], 'Failed')"
    },
    "runAfter": {
        "My_Scope": [
            "Failed"
        ]
    },
    "type": "Query"
},
"For_each": {
    "actions": {
        "Log_Exception": {
            "inputs": {
                "body": "@item()['outputs']['body']",
                "method": "POST",
                "headers": {
                    "x-failed-action-name": "@item()['name']",
                    "x-failed-tracking-id": "@item()['clientTrackingId']"
                },
                "uri": "http://requestb.in/"
            },
            "runAfter": {},
            "type": "Http"
        }
    },
    "foreach": "@body('Filter_array')",
    "runAfter": {
        "Filter_array": [
            "Succeeded"
        ]
    },
    "type": "Foreach"
}
```

Here's a detailed walkthrough to describe what happens:

1. To get the result of all actions within `My_Scope`, the **Filter Array** action filters `@result('My_Scope')`.
2. The condition for **Filter Array** is any `@result()` item that has status equal to `Failed`. This condition filters the array with all action results from `My_Scope` to an array with only failed action results.
3. Perform a **For Each** action on the **Filtered Array** outputs. This step performs an action *for each* failed action result that was previously filtered.

If a single action in the scope failed, the actions in the `foreach` run only once. Many failed actions cause one action per failure.

- Send an HTTP POST on the `foreach` item response body, or `@item()['outputs']['body']`. The `@result()` item shape is the same as the `@actions()` shape, and can be parsed the same way.
- Include two custom headers with the failed action name `@item()['name']` and the failed run client tracking ID `@item()['clientTrackingId']`.

For reference, here's an example of a single `@result()` item, showing the `name`, `body`, and `clientTrackingId` properties that are parsed in the previous example. Outside of a `foreach`, `@result()` returns an array of these objects.

```
{
  "name": "Example_Action_That_Failed",
  "inputs": {
    "uri": "https://myfailedaction.azurewebsites.net",
    "method": "POST"
  },
  "outputs": {
    "statusCode": 404,
    "headers": {
      "Date": "Thu, 11 Aug 2016 03:18:18 GMT",
      "Server": "Microsoft-IIS/8.0",
      "X-Powered-By": "ASP.NET",
      "Content-Length": "68",
      "Content-Type": "application/json"
    },
    "body": {
      "code": "ResourceNotFound",
      "message": "/docs/folder-name/resource-name does not exist"
    }
  },
  "startTime": "2016-08-11T03:18:19.7755341Z",
  "endTime": "2016-08-11T03:18:20.2598835Z",
  "trackingId": "bdd82e28-ba2c-4160-a700-e3a8f1a38e22",
  "clientTrackingId": "08587307213861835591296330354",
  "code": "NotFound",
  "status": "Failed"
}
```

To perform different exception handling patterns, you can use the expressions shown previously. You might choose to execute a single exception handling action outside the scope that accepts the entire filtered array of failures, and remove the `foreach`. You can also include other useful properties from the `@result()` response shown previously.

## Azure Diagnostics and telemetry

The previous patterns are great way to handle errors and exceptions within a run, but you can also identify and respond to errors independent of the run itself. [Azure Diagnostics](#) provides a simple way to send all workflow events (including all run and action statuses) to an Azure Storage account or an Azure Event Hub. To evaluate run statuses, you can monitor the logs and metrics, or publish them into any monitoring tool you prefer. One potential option is to stream all the events through Azure Event Hub into [Stream Analytics](#). In Stream Analytics, you can write live queries off any anomalies, averages, or failures from the diagnostic logs. Stream Analytics can easily output to other data sources like queues, topics, SQL, DocumentDB, and Power BI.

## Next Steps

- [See how a customer builds error handling with Azure Logic Apps](#)
- [Find more Logic Apps examples and scenarios](#)
- [Learn how to create automated deployments for logic apps](#)

- Build and deploy logic apps with Visual Studio

# Handle content types in logic apps

3/24/2017 • 4 min to read • [Edit Online](#)

Many different types of content can flow through a logic app, including JSON, XML, flat files, and binary data. While the Logic Apps Engine supports all content types, some are natively understood by the Logic Apps Engine. Others might require casting or conversions as necessary. This article describes how the engine handles different content types and how to correctly handle these types when necessary.

## Content-Type Header

To start basically, let's look at the two `Content-Types` that don't require conversion or casting that you can use in a logic app: `application/json` and `text/plain`.

## Application/JSON

The workflow engine relies on the `Content-Type` header from HTTP calls to determine the appropriate handling. Any request with the content type `application/json` is stored and handled as a JSON Object. Also, JSON content can be parsed by default without needing any casting.

For example, you could parse a request that has the content type header `application/json` in a workflow by using an expression like `@body('myAction')['foo'][0]` to get the value `bar` in this case:

```
{  
    "data": "a",  
    "foo": [  
        "bar"  
    ]  
}
```

No additional casting is needed. If you are working with data that is JSON but didn't have a header specified, you can manually cast it to JSON using the `@json()` function, for example: `@json(triggerBody())['foo']`.

## Schema and schema generator

The Request trigger lets you enter a JSON schema for the payload you expect to receive. This schema lets the designer generate tokens so you can consume the content of the request. If you don't have a schema ready, select **Use sample payload to generate schema**, so you can generate a JSON schema from a sample payload.

The screenshot shows the 'Request' configuration interface. At the top, there's a red header bar with a globe icon, the word 'Request', and three dots for more options. Below the header, it says 'HTTP POST to this URL'. A grey bar contains the placeholder 'URL will be generated after save' with a copy icon. The main area is titled 'Request Body JSON Schema' and contains a JSON schema editor. The schema is defined as:

```
{
  "type": "object",
  "properties": {
    "address": {
      "type": "string"
    }
  },
  "required": [
    "address"
  ]
}
```

Below the schema editor, there's a link 'Use sample payload to generate schema' and a button 'Show advanced options ▾'.

### 'Parse JSON' action

The `Parse JSON` action lets you parse JSON content into friendly tokens for logic app consumption. Similar to the Request trigger, this action lets you enter or generate a JSON schema for the content you want to parse. This tool makes consuming data from Service Bus, DocumentDB, and so on, much easier.

The screenshot shows the 'Parse JSON' action configuration interface. It has a dark header bar with a file icon, the action name 'Parse JSON', and three dots for more options. The main area is divided into two sections: 'Content' and 'Schema'. The 'Content' section has a 'Content' input field with a file icon and an 'x' button. The 'Schema' section contains a JSON schema editor with the same schema as the previous screenshot:

```
{
  "type": "object",
  "properties": {
    "Name": {
      "type": "string"
    },
    "Address": {
      "type": "string"
    }
  }
}
```

Below the schema editor, there's a link 'Use sample payload to generate schema'.

## Text/plain

Similar to `application/json`, HTTP messages received with the `Content-Type` header of `text/plain` are stored in raw form. Also, if those messages are included in subsequent actions without casting, those requests go out with `Content-Type : text/plain` header. For example, when working with a flat file, you might get this HTTP content as `text/plain :`

```
Date,Name,Address
Oct-1,Frank,123 Ave.
```

If in the next action, you send the request as the body of another request (`@body('flatfile')`), the request would have a `text/plain` Content-Type header. If you are working with data that is plain text but didn't have a header specified, you can manually cast the data to text using the `@string()` function, for example:

```
@string(triggerBody()) .
```

## Application/xml and Application/octet-stream and converter functions

The Logic Apps Engine always preserves the `Content-Type` that was received on the HTTP request or response. So if the engine receives content with the `Content-Type` of `application/octet-stream`, and you include that content in a subsequent action without casting, the outgoing request has `Content-Type : application/octet-stream`. This way, the engine can guarantee data isn't lost while moving through the workflow. However, the action state (inputs and outputs) is stored in a JSON object as the state moves through the workflow. So to preserve some data types, the engine converts the content to a binary base64 encoded string with appropriate metadata that preserves both `$content` and `$content-type`, which are automatically be converted.

- `@json()` - casts data to `application/json`
- `@xml()` - casts data to `application/xml`
- `@binary()` - casts data to `application/octet-stream`
- `@string()` - casts data to `text/plain`
- `@base64()` - converts content to a base64 string
- `@base64ToString()` - converts a base64 encoded string to `text/plain`
- `@base64toBinary()` - converts a base64 encoded string to `application/octet-stream`
- `@encodeDataUri()` - encodes a string as a dataUri byte array
- `@decodeDataUri()` - decodes a dataUri into a byte array

For example, if you received an HTTP request with `Content-Type : application/xml`:

```
<?xml version="1.0" encoding="UTF-8" ?>
<CustomerName>Frank</CustomerName>
```

You could cast and use later with something like `@xml(triggerBody())`, or in a function like `@xpath(xml(triggerBody()), '/CustomerName')`.

## Other content types

Other content types are supported and work with logic apps, but might require manually retrieving the message body by decoding the `$content`. For example, suppose you trigger an `application/x-www-url-formencoded` request where `$content` is the payload encoded as a base64 string to preserve all data:

```
CustomerName=Frank&Address=123+Avenue
```

Because the request isn't plain text or JSON, the request is stored in the action as follows:

```
...
"body": {
    "$content-type": "application/x-www-url-formencoded",
    "$content": "AAB1241BACDFA=="
}
```

Currently, there isn't a native function for form data, so you could still use this data in a workflow by manually accessing the data with a function like `@string(body('formdataAction'))`. If you wanted the outgoing request to also have the `application/x-www-url-formencoded` content type header, you could add the request to the action body without any casting like `@body('formdataAction')`. However, this method only works if the body is the only parameter in the `body` input. If you try to use `@body('formdataAction')` in an `application/json` request, you get a

runtime error because the encoded body is sent.

# Secure access to your logic apps

2/15/2017 • 9 min to read • [Edit Online](#)

There are many tools available to help you secure your logic app.

- Securing access to trigger a logic app (HTTP Request Trigger)
- Securing access to manage, edit, or read a logic app
- Securing access to contents of inputs and outputs for a run
- Securing parameters or inputs within actions in a workflow
- Securing access to services that receive requests from a workflow

## Secure access to trigger

When you work with a logic app that fires on an HTTP Request ([Request](#) or [Webhook](#)), you can restrict access so that only authorized clients can fire the logic app. All requests into a logic app are encrypted and secured via SSL.

### Shared Access Signature

Every request endpoint for a logic app includes a [Shared Access Signature \(SAS\)](#) as part of the URL. Each URL contains a `sp`, `sv`, and `sig` query parameter. Permissions are specified by `sp`, and correspond to HTTP methods allowed, `sv` is the version used to generate, and `sig` is used to authenticate access to trigger. The signature is generated using the SHA256 algorithm with a secret key on all the URL paths and properties. The secret key is never exposed and published, and is kept encrypted and stored as part of the logic app. Your logic app only authorizes triggers that contain a valid signature created with the secret key.

#### Regenerate access keys

You can regenerate a new secure key at anytime through the REST API or Azure portal. All current URLs that were generated previously using the old key are invalidated and no longer authorized to fire the logic app.

1. In the Azure portal, open the logic app you want to regenerate a key
2. Click the **Access Keys** menu item under **Settings**
3. Choose the key to regenerate and complete the process

URLs you retrieve after regeneration are signed with the new access key.

#### Creating callback URLs with an expiration date

If you are sharing the URL with other parties, you can generate URLs with specific keys and expiration dates as needed. You can then seamlessly roll keys, or ensure access to fire an app is restricted to a certain timespan. You can specify an expiration date for a URL through the [logic apps REST API](#):

```
POST  
/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/providers/Microsoft.Logic/workflows/{workflo  
wName}/triggers/{triggerName}/listCallbackUrl?api-version=2016-06-01
```

In the body, include the property `NotAfter` as a JSON date string, which returns a callback URL that is only valid until the `NotAfter` date and time.

#### Creating URLs with primary or secondary secret key

When you generate or list callback URLs for request-based triggers, you can also specify which key to use to sign the URL. You can generate a URL signed by a specific key through the [logic apps REST API](#) as follows:

```
POST  
/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/providers/Microsoft.Logic/workflows/{workflo  
wName}/triggers/{triggerName}/listCallbackUrl?api-version=2016-06-01
```

In the body, include the property `KeyType` as either `Primary` or `Secondary`. This returns a URL signed by the secure key specified.

### Restrict incoming IP addresses

In addition to the Shared Access Signature, you may wish to restrict calling a logic app only from specific clients. For example, if you manage your endpoint through Azure API Management, you can restrict the logic app to only accept the request when the request comes from the API Management instance IP address.

This setting can be configured within the logic app settings:

1. In the Azure portal, open the logic app you want to add IP address restrictions
2. Click the **Access control configuration** menu item under **Settings**
3. Specify the list of IP address ranges to be accepted by the trigger

A valid IP range takes the format `192.168.1.1/255`. If you want the logic app to only fire as a nested logic app, select the **Only other logic apps** option. This option writes an empty array to the resource, meaning only calls from the service itself (parent logic apps) fire successfully.

#### NOTE

You can still run a logic app with a request trigger through the REST API / Management `/triggers/{triggerName}/run` regardless of IP. This scenario requires authentication against the Azure REST API, and all events would appear in the Azure Audit Log. Set access control policies accordingly.

### Setting IP ranges on the resource definition

If you are using a [deployment template](#) to automate your deployments, the IP range settings can be configured on the resource template.

```
{  
    "properties": {  
        "definition": {  
        },  
        "parameters": {},  
        "accessControl": {  
            "triggers": {  
                "allowedCallerIpAddresses": [  
                    {  
                        "addressRange": "192.168.12.0/23"  
                    },  
                    {  
                        "addressRange": "2001:0db8::/64"  
                    }  
                ]  
            }  
        },  
        "type": "Microsoft.Logic/workflows"  
    }  
}
```

### Adding Azure Active Directory, OAuth, or other security

To add more authorization protocols on top of a logic app, [Azure API Management](#) offers rich monitoring, security, policy, and documentation for any endpoint with the capability to expose a logic app as an API. Azure API Management can expose a public or private endpoint for the logic app, which could use Azure Active Directory,

certificate, OAuth, or other security standards. When a request is received, Azure API Management forwards the request to the logic app (performing any needed transformations or restrictions in-flight). You can use the incoming IP range settings on the logic app to only allow the logic app to be triggered from API Management.

## Secure access to manage or edit logic apps

You can restrict access to management operations on a logic app so that only specific users or groups are able to perform operations on the resource. Logic apps use the Azure [Role-Based Access Control \(RBAC\)](#) feature, and can be customized with the same tools. There are a few built-in roles you can assign members of your subscription to as well:

- **Logic App Contributor** - Provides access to view, edit, and update a logic app. Cannot remove the resource or perform admin operations.
- **Logic App Operator** - Can view the logic app and run history, and enable/disable. Cannot edit or update the definition.

You can also use [Azure Resource Lock](#) to prevent changing or deleting logic apps. This capability is valuable to prevent production resources from changes or deletions.

## Secure access to contents of the run history

You can restrict access to contents of inputs or outputs from previous runs to specific IP address ranges.

All data within a workflow run is encrypted in transit and at rest. When a call to run history is made, the service authenticates the request and provides links to the request and response inputs and outputs. This link can be protected so only requests to view content from a designated IP address range return the contents. You can use this capability for additional access control. You could even specify an IP address like `0.0.0.0` so no one could access inputs/outputs. Only someone with admin permissions could remove this restriction, providing the possibility for 'just-in-time' access to workflow contents.

This setting can be configured within the resource settings of the Azure portal:

1. In the Azure portal, open the logic app you want to add IP address restrictions
2. Click the **Access control configuration** menu item under **Settings**
3. Specify the list of IP address ranges for access to content

### Setting IP ranges on the resource definition

If you are using a [deployment template](#) to automate your deployments, the IP range settings can be configured on the resource template.

```
{
  "properties": {
    "definition": {
      "parameters": {},
      "accessControl": {
        "contents": [
          "allowedCallerIpAddresses": [
            {
              "addressRange": "192.168.12.0/23"
            },
            {
              "addressRange": "2001:0db8::/64"
            }
          ]
        }
      }
    },
    "type": "Microsoft.Logic/workflows"
  }
}
```

## Secure parameters and inputs within a workflow

You might want to parameterize some aspects of a workflow definition for deployment across environments. Also, some parameters might be secure parameters you don't want to appear when editing a workflow, such as a client ID and client secret for [Azure Active Directory authentication](#) of an HTTP action.

### Using parameters and secure parameters

To access the value of a resource parameter at runtime, the [workflow definition language](#) provides a `@parameters()` operation. Also, you can [specify parameters in the resource deployment template](#). But if you specify the parameter type as `securestring`, the parameter won't be returned with the rest of the resource definition, and won't be accessible by viewing the resource after deployment.

#### NOTE

If your parameter is used in the headers or body of a request, the parameter might be visible by accessing the run history and outgoing HTTP request. Make sure to set your content access policies accordingly. Authorization headers are never visible through inputs or outputs. So if the secret is being used there, the secret is not retrievable.

### Resource deployment template with secrets

The following example shows a deployment that references a secure parameter of `secret` at runtime. In a separate parameters file, you could specify the environment value for the `secret`, or use [Azure Resource Manager KeyVault](#) to retrieve secrets at deploy time.

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "secretDeploymentParam": {
            "type": "securestring"
        }
    },
    "variables": {},
    "resources": [
        {
            "name": "secret-deploy",
            "type": "Microsoft.Logic/workflows",
            "location": "westus",
            "tags": {
                "displayName": "LogicApp"
            },
            "apiVersion": "2016-06-01",
            "properties": {
                "definition": {
                    "$schema": "https://schema.management.azure.com/providers/Microsoft.Logic/schemas/2016-06-01/workflowdefinition.json#",
                    "actions": {
                        "Call_External_API": {
                            "type": "http",
                            "inputs": {
                                "headers": {
                                    "Authorization": "@parameters('secret')"
                                },
                                "body": "This is the request"
                            },
                            "runAfter": {}
                        }
                    },
                    "parameters": {
                        "secret": {
                            "type": "SecureString"
                        }
                    },
                    "triggers": {
                        "manual": {
                            "type": "Request",
                            "kind": "Http",
                            "inputs": {
                                "schema": {}
                            }
                        }
                    },
                    "contentVersion": "1.0.0.0",
                    "outputs": {}
                },
                "parameters": {
                    "secret": {
                        "value": "[parameters('secretDeploymentParam')]"
                    }
                }
            }
        }
    ],
    "outputs": {}
}
```

## Secure access to services receiving requests from a workflow

There are many ways to help secure any endpoint the logic app needs to access.

## Using authentication on outbound requests

When working with an HTTP, HTTP + Swagger (Open API), or Webhook action, you can add authentication to the request being sent. You could include basic authentication, certificate authentication, or Azure Active Directory authentication. Details on how to configure this authentication can be found [in this article](#).

## Restricting access to logic app IP addresses

All calls from logic apps come from a specific set of IP addresses per region. You can add additional filtering to only accept requests from those designated IP addresses. For a list of those IP addresses, see [logic app limits and configuration](#).

## On-premises connectivity

Logic apps provide integration with several services to provide secure and reliable on-premises communication.

### On-premises data gateway

Many of the managed connectors from logic apps provide secure connectivity to on-premises systems, including File System, SQL, SharePoint, DB2, and more. The gateway uses encrypted channels via Azure Service Bus to relay data on-premises, and all traffic originates from secure outbound traffic from the gateway agent. More details on how the gateway works [in this article](#).

### Azure API Management

[Azure API Management](#) has on-premises connectivity options, including site-to-site VPN and ExpressRoute integration for secured proxy and communication to on-premises systems. In the Logic App Designer, you can quickly select an API exposed from Azure API Management within a workflow, providing quick access to on-premises systems.

### Hybrid connections from Azure App Service

You can use the on-premises hybrid connection feature for Azure API and Web apps to communicate on-premises. Details on hybrid connections and how to configure can be found [in this article](#).

## Next steps

[Create a deployment template](#)

[Exception handling](#)

[Monitor your logic apps](#)

[Diagnosing logic app failures and issues](#)

# Diagnose logic app failures

3/9/2017 • 3 min to read • [Edit Online](#)

If you experience issues or failures with your logic apps, there are a few approaches can help you better understand where the failures are coming from.

## Azure portal tools

The Azure portal provides many tools to diagnose each logic app at each step.

### Trigger history

Each logic app has at least one trigger. If you notice that apps aren't firing, look first at the trigger history for more information. You can access the trigger history on the logic app's main blade.

The screenshot shows the Azure Logic App main blade for a logic app named "SalesforceToYammer". The left pane displays the "Essentials" section with tabs for "Summary" and "Monitoring". The "Summary" tab shows a table of recent runs, with three entries: 5/18/2016, 9:53 AM (2.39 Seconds), 4/26/2016, 11:54 AM (3 Seconds), and 4/26/2016, 10:33 AM (3.16 Seconds). A red circle labeled "1" highlights the "All triggers" link next to the summary table. The right pane shows the "Trigger histories" section, which lists trigger attempts with columns for "Fired", "Start Time", "Trigger Name", and "Call Status". There are four entries: 5/18/2016, 3:48 PM (Skipped), 5/18/2016, 3:48 PM (Skipped), 5/18/2016, 9:53 AM (Skipped), and a recent entry labeled "2" (Fired) at 5/18/2016, 9:53 AM with a status of "Succeeded".

The trigger history lists all trigger attempts that your logic app made. You can click each trigger attempt to drill into the details, specifically, any inputs or outputs that the trigger attempt generated. If you find failed triggers, select the trigger attempt and choose the **Outputs** link to review any generated error messages, for example, for FTP credentials that aren't valid.

The different statuses you might see are:

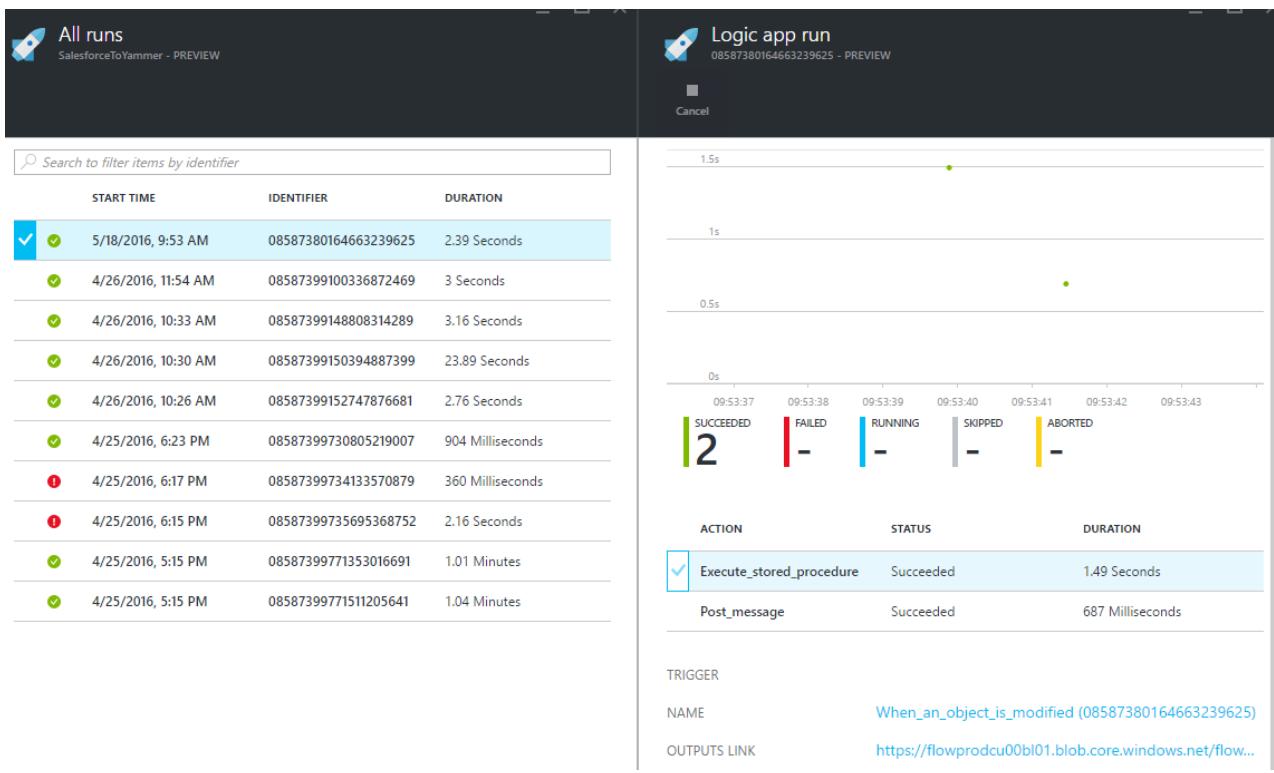
- **Skipped.** The endpoint was polled to check for data and received a response that no data was available.
- **Succeeded.** The trigger received a response that data was available. This status might result from a manual trigger, a recurrence trigger, or a polling trigger. This status is usually accompanied by the **Fired** status, but might not be if you have a condition or SplitOn command in code view that wasn't satisfied.
- **Failed.** An error was generated.

### Start a trigger manually

If you want the logic app to check for an available trigger immediately without waiting for the next recurrence, click **Select Trigger** on the main blade to force a check. For example, clicking this link with a Dropbox trigger causes the workflow to immediately poll Dropbox for new files.

### Run history

Every fired trigger results in a run. You can access run information from the main blade, which contains many details that can help you understand what happened during the workflow.



A run displays one of the following statuses:

- **Succeeded.** All actions succeeded. If a failure happened, that failure was handled by an action that occurred later in the workflow. That is, the failure was handled by an action that was set to run after a failed action.
- **Failed.** At least one action had a failure that was not handled by an action later in the workflow.
- **Cancelled.** The workflow was running but received a cancel request.
- **Running.** The workflow is currently running. This status might occur for throttled workflows, or because of the current pricing plan. For details, see [action limits on the pricing page](#). Configuring diagnostics (the charts that appear under the run history) also can provide information about any throttle events that happen.

When you are looking at a run history, you can drill in for more details.

#### Trigger outputs

Trigger outputs show the data that came from the trigger. These outputs can help you determine whether all properties returned as expected.

**NOTE**

If you see any content that you don't understand, learn how Azure Logic Apps [handles different content types](#).

ACTION	STATUS	DURATION
Execute_stored_procedure	Succeeded	1.49 Seconds
Post_message	Succeeded	687 Milliseconds

**TRIGGER**

NAME	When_an_object_is_modified (08587380164663239625)
OUTPUTS LINK	https://flowprod00bl01.blob.core.windows.net/flow...

#### Action inputs and outputs

You can drill into the inputs and outputs that an action received. This data is useful for understanding the size and

shape of the outputs, and also for finding any error messages that might have been generated.

The screenshot shows the run details for a logic app. It includes a table of actions, trigger information, and execution details. A red circle with the number 1 points to the first action in the table. A red circle with the number 2 points to the outputs link in the details section.

ACTION	STATUS	DURATION
Execute_stored_procedure	Succeeded	1.49 Seconds
Post_message	Succeeded	687 Milliseconds

TRIGGER

NAME When\_an\_object\_is\_modified (08587380164663239625)

OUTPUTS LINK <https://flowprodmcu00bl01.blob.core.windows.net/flow...>

DETAILS

START DATE Wednesday, May 18, 2016, 9:53:39 AM

END DATE Wednesday, May 18, 2016, 9:53:42 AM

INPUTS  
88abf944-f349-4498-918d-4d5b171808a4

INPUTS LINK <https://flowprodmcu00bl01.blob.core.windows.net/flow...>

SIZE 360 bytes

OUTPUTS  
2 OUTPUTS LINK <https://flowprodmcu00bl01.blob.core.windows.net/flow...>

## Debug workflow runtime

Along with monitoring the inputs, outputs, and triggers of a run, you could add some steps to a workflow that help with debugging. [RequestBin](#) is a powerful tool that you can add as a step in a workflow. By using RequestBin, you can set up an HTTP request inspector to determine the exact size, shape, and format of an HTTP request. You can create a RequestBin and paste the URL in a logic app HTTP POST action with body content that you want to test, for example, an expression or another step output. After you run the logic app, you can refresh your RequestBin to see how the request was formed when generated from the Logic Apps engine.

# Logic App limits and configuration

3/28/2017 • 3 min to read • [Edit Online](#)

Below are information on the current limits and configuration details for Azure Logic Apps.

## Limits

### HTTP request limits

These are limits for a single HTTP request and/or connector call

#### Timeout

NAME	LIMIT	NOTES
Request Timeout	120 Seconds	An <a href="#">async pattern</a> or <a href="#">until loop</a> can compensate as needed

#### Message size

NAME	LIMIT	NOTES
Message size	100 MB	Some connectors and APIs may not support 100MB
Expression evaluation limit	131,072 characters	<code>@concat()</code> , <code>@base64()</code> , <code>string</code> cannot be longer than this

#### Retry policy

NAME	LIMIT	NOTES
Retry attempts	4	Can configure with the <a href="#">retry policy parameter</a>
Retry max delay	1 hour	Can configure with the <a href="#">retry policy parameter</a>
Retry min delay	5 sec	Can configure with the <a href="#">retry policy parameter</a>

### Run duration and retention

These are the limits for a single logic app run.

NAME	LIMIT	NOTES
Run duration	90 days	
Storage retention	90 days	This is from the run start time
Min recurrence interval	1 sec	
Max recurrence interval	500 days	

NAME	LIMIT	NOTES
------	-------	-------

## Looping and debatching limits

These are limits for a single logic app run.

NAME	LIMIT	NOTES
ForEach items	100,000	You can use the <a href="#">query action</a> to filter larger arrays as needed
Until iterations	5,000	
SplitOn items	100,000	
ForEach Parallelism	20	You can set to a sequential foreach by adding "operationOptions": "Sequential" to the <code>foreach</code> action

## Throughput limits

These are limits for a single logic app instance.

NAME	LIMIT	NOTES
Actions executions per 5 minutes	100,000	Can distribute workload across multiple apps as needed

If you expect to exceed this limit in normal processing or wish to run load testing that may exceed this limit for a period of time please [contact us](#) so that we can help with your requirements.

## Definition limits

These are limits for a single logic app definition.

NAME	LIMIT	NOTES
Actions per workflow	250	You can add nested workflows to extend this as needed
Allowed action nesting depth	5	You can add nested workflows to extend this as needed
Workflows per region per subscription	1000	
Triggers per workflow	10	
Max characters per expression	8,192	
Max <code>trackedProperties</code> size in characters	16,000	
<code>action / trigger</code> name limit	80	

NAME	LIMIT	NOTES
<code>description</code> length limit	256	
<code>parameters</code> limit	50	
<code>outputs</code> limit	10	

## Integration Account limits

These are limits for artifacts added to integration Account

NAME	LIMIT	NOTES
Schema	8MB	You can use <a href="#">blob URI</a> to upload files larger than 2 MB
Map (XSLT file)	2MB	

## B2B protocols (AS2, X12, EDIFACT) message size

These are the limits for B2B protocols

NAME	LIMIT	NOTES
AS2	50MB	Applicable to decode and encode
X12	50MB	Applicable to decode and encode
EDIFACT	50MB	Applicable to decode and encode

## Configuration

### IP Address

#### Logic App Service

Calls made from a logic app directly (i.e. via [HTTP](#) or [HTTP + Swagger](#)) or other HTTP requests will come from the IP Address specified below:

LOGIC APP REGION	OUTBOUND IP
Australia East	13.75.153.66, 104.210.89.222, 104.210.89.244, 13.75.149.4, 104.210.91.55, 104.210.90.241
Australia Southeast	13.73.115.153, 40.115.78.70, 40.115.78.237, 13.73.114.207, 13.77.3.139, 13.70.159.205
Brazil South	191.235.86.199, 191.235.95.229, 191.235.94.220, 191.235.82.221, 191.235.91.7, 191.234.182.26
Canada Central	52.233.29.92, 52.228.39.241, 52.228.39.244
Canada East	52.232.128.155, 52.229.120.45, 52.229.126.25

LOGIC APP REGION	OUTBOUND IP
Central India	52.172.157.194, 52.172.184.192, 52.172.191.194, 52.172.154.168, 52.172.186.159, 52.172.185.79
Central US	13.67.236.76, 40.77.111.254, 40.77.31.87, 13.67.236.125, 104.208.25.27, 40.122.170.198
East Asia	168.63.200.173, 13.75.89.159, 23.97.68.172, 13.75.94.173, 40.83.127.19, 52.175.33.254
East US	137.135.106.54, 40.117.99.79, 40.117.100.228, 13.92.98.111, 40.121.91.41, 40.114.82.191
East US 2	40.84.25.234, 40.79.44.7, 40.84.59.136, 40.84.30.147, 104.208.155.200, 104.208.158.174
Japan East	13.71.146.140, 13.78.84.187, 13.78.62.130, 13.71.158.3, 13.73.4.207, 13.71.158.120
Japan West	40.74.140.173, 40.74.81.13, 40.74.85.215, 40.74.140.4, 104.214.137.243, 138.91.26.45
North Central US	168.62.249.81, 157.56.12.202, 65.52.211.164, 168.62.248.37, 157.55.210.61, 157.55.212.238
North Europe	13.79.173.49, 52.169.218.253, 52.169.220.174, 40.113.12.95, 52.178.165.215, 52.178.166.21
South Central US	13.65.98.39, 13.84.41.46, 13.84.43.45, 104.210.144.48, 13.65.82.17, 13.66.52.232
Southeast Asia	52.163.93.214, 52.187.65.81, 52.187.65.155, 13.76.133.155, 52.163.228.93, 52.163.230.166
South India	52.172.9.47, 52.172.49.43, 52.172.51.140, 52.172.50.24, 52.172.55.231, 52.172.52.0
West Europe	13.95.155.53, 52.174.54.218, 52.174.49.6, 40.68.222.65, 40.68.209.23, 13.95.147.65
West India	104.211.164.112, 104.211.165.81, 104.211.164.25, 104.211.164.80, 104.211.162.205, 104.211.164.136
West US	52.160.90.237, 138.91.188.137, 13.91.252.184, 52.160.92.112, 40.118.244.241, 40.118.241.243

#### Connectors

Calls made from a [connector](#) will come from the IP Address specified below:

LOGIC APP REGION	OUTBOUND IP
Australia East	40.126.251.213
Australia Southeast	40.127.80.34

LOGIC APP REGION	OUTBOUND IP
Brazil South	191.232.38.129
Canada Central	52.233.31.197,52.228.42.205,52.228.33.76,52.228.34.13
Canada East	52.229.123.98,52.229.120.178,52.229.126.202,52.229.120.52
Central India	104.211.98.164
Central US	40.122.49.51
East Asia	23.99.116.181
East US	191.237.41.52
East US 2	104.208.233.100
Japan East	40.115.186.96
Japan West	40.74.130.77
North Central US	65.52.218.230
North Europe	104.45.93.9
South Central US	104.214.70.191
Southeast Asia	13.76.231.68
South India	104.211.227.225
West Europe	40.115.50.13
West India	104.211.161.203
West US	104.40.51.248

## Next Steps

- To get started with Logic Apps, follow the [create a Logic App](#) tutorial.
- [View common examples and scenarios](#)
- [You can automate business processes with Logic Apps](#)
- [Learn How to Integrate your systems with Logic Apps](#)

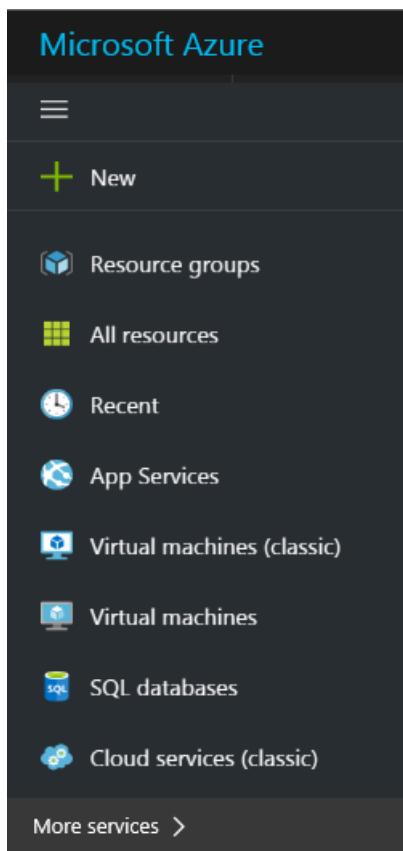
# What is an integration account?

3/9/2017 • 2 min to read • [Edit Online](#)

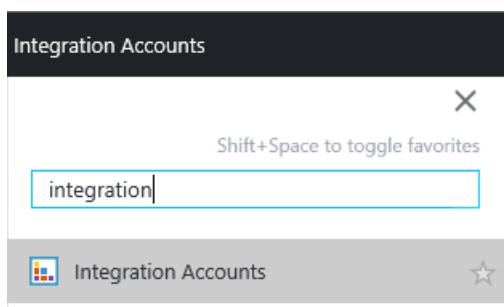
An integration account allows enterprise integration apps to manage artifacts, including schemas, maps, certificates, partners and agreements. Any integration app you create uses an integration account to access these schemas, maps, certificates, and so on.

## Create an integration account

1. Sign in to the [Azure portal](#). From the left menu, select **More services**.



2. In the search box, type "integration" for your filter. In the results list, select **Integration Accounts**.



3. At the top of the page, choose **Add**.

**Integration Accounts**

Microsoft

**Add** **Columns** **Refresh**

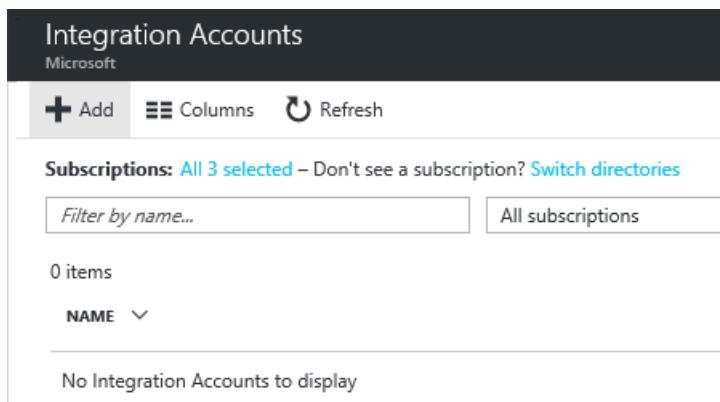
**Subscriptions:** All 3 selected – Don't see a subscription? [Switch directories](#)

Filter by name... [All subscriptions](#)

0 items

NAME ▾

No Integration Accounts to display



4. Name your integration account and select the Azure subscription that you want to use. You can either create a new **Resource group** or select an existing resource group. Then select a **Location** for hosting your integration account and a **Pricing Tier**.

When you're ready, choose **Create**.

**Integration Account** X

\* Name  
 ✓

\* Subscription  
 ▾

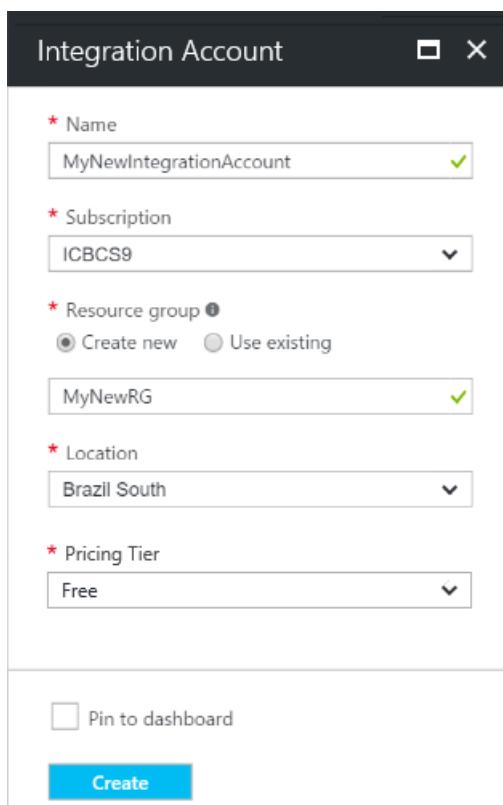
\* Resource group ⓘ  
 Create new  Use existing  
 ✓

\* Location  
 ▾

\* Pricing Tier  
 ▾

Pin to dashboard

**Create**



Azure provisions your integration account in the selected location, which should complete within 1 minute.

5. Refresh the page. You see your new integration account listed.

**Integration Accounts**

Microsoft

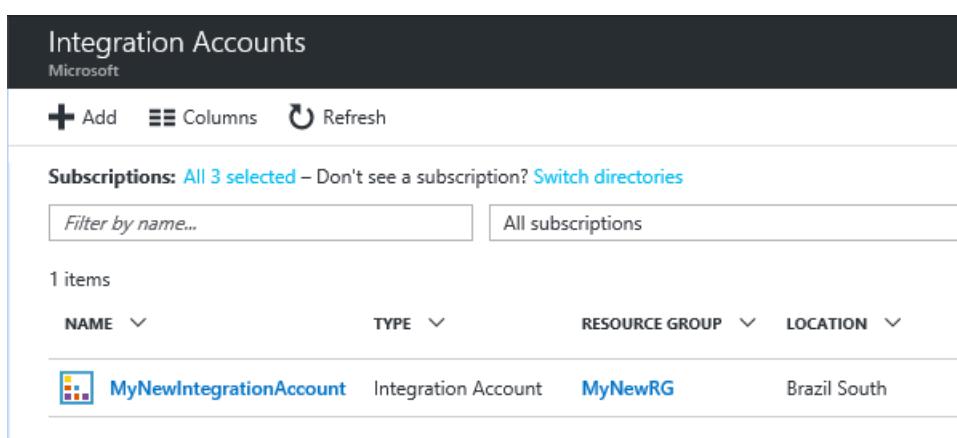
**Add** **Columns** **Refresh**

**Subscriptions:** All 3 selected – Don't see a subscription? [Switch directories](#)

Filter by name... [All subscriptions](#)

1 items

NAME ▾	TYPE ▾	RESOURCE GROUP ▾	LOCATION ▾
 MyNewIntegrationAccount	Integration Account	MyNewRG	Brazil South



Next, link the integration account that you created to your logic app.

## Link an integration account to a logic app

To give your logic apps access to maps, schemas, agreements, and other artifacts in your integration account, link the integration account to your logic app.

### Prerequisites

- An integration account
- A logic app

#### NOTE

Make sure your integration account and logic app are in the *same Azure location* before you begin.

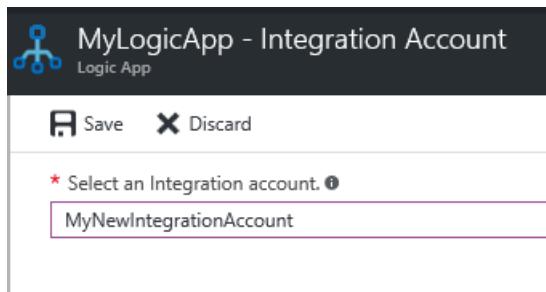
1. In the Azure portal, select your logic app, and check your logic app's location.

The screenshot shows the Azure Logic Apps blade. On the left, there's a sidebar with 'Subscriptions' (All 3 selected), a search bar, and a list of logic apps. The main area is titled 'MyLogicApp' and shows the 'Overview' page. On the right, there's a 'Run Trigger' button and a summary card for the logic app, including its resource group ('TestLogicApp-ResourceGroup'), location ('Brazil South'), subscription ('Visual Studio Enterprise'), and subscription ID. The 'Location' field is specifically highlighted with a red box.

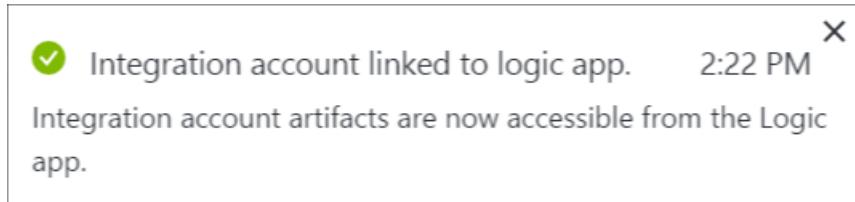
2. Under **Settings**, select **Integration Account**.

The screenshot shows the 'MyLogicApp' settings blade. It has sections for 'Access control (IAM)', 'Tags', 'Development Tools' (including 'Logic App Designer', 'Logic App Code View', 'API Connections', and 'Quick Start Guides'), and 'Settings'. The 'Integration Account' option is located at the bottom of the settings section and is highlighted with a red box.

3. From the **Select an Integration account** list, select the integration account you want to link to your logic app. To finish linking, choose **Save**.



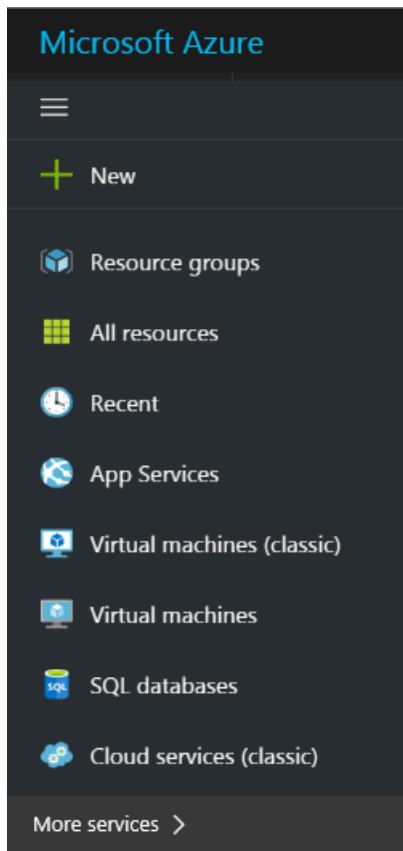
You get a notification that shows your integration account is linked to your logic app, and that all artifacts in your integration account are now available to your logic app.



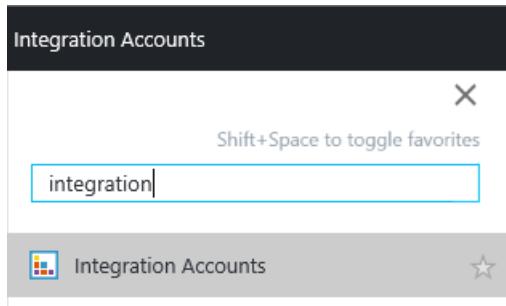
Now that your integration account is linked to your logic app, you can use the B2B connectors in your logic apps. Some common B2B connectors include XML validation and flat file encode/decode.

## Delete your integration account

1. Select **More services**.



2. In the search box, type "integration" for your filter. In the results list, select **Integration Accounts**.



3. Select the integration account that you want to delete.

A screenshot of the 'Integration Accounts' list page in the Azure portal. It shows one item: 'MyNewIntegrationAccount' (Integration Account, MyNewRG, Brazil South). The page includes filters for 'NAME', 'TYPE', 'RESOURCE GROUP', and 'LOCATION'.

4. On the menu, choose **Delete**.

A screenshot of the 'MyNewIntegrationAccount' details page. The 'Delete' button in the top right corner is highlighted with a red box. The page also shows the resource group as 'MyNewRG'.

5. Confirm your choice to delete the integration account.

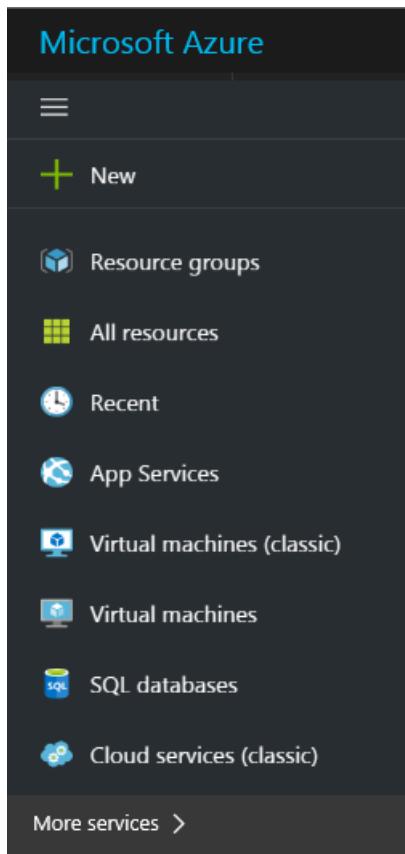
## Move your integration account

To move an integration account to another Azure subscription or resource group, follow these steps.

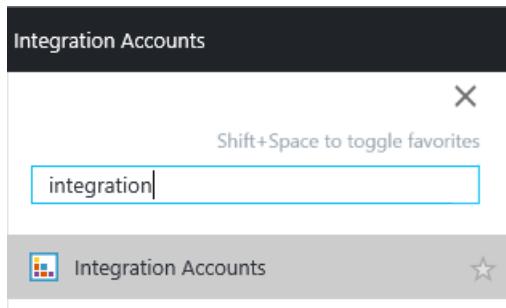
### IMPORTANT

You must update all scripts to use the new resource IDs after you move an integration account.

1. Select **More services**.



2. In the search box, type "integration" for your filter. In the results list, select **Integration Accounts**.



3. Select the integration account that you want to move. Under **Settings**, choose **Properties**.

The screenshot shows two side-by-side views in the Azure portal. On the left, the 'Integration Accounts' blade lists 'Subscriptions: All 3 selected – Don't see a subscription? Switch directories'. It has a search bar ('Filter by name...'), a dropdown for 'All subscriptions', and a table with one item: 'MyNewIntegrationAccount' (selected). On the right, the 'MyNewIntegrationAccount - Integration Account' blade shows a navigation menu with 'Overview', 'Access control (IAM)', 'Tags', 'SETTINGS' (selected), 'Callback URL', 'Schemas', 'Maps', 'Certificates', 'Partners', 'Agreements', and 'Properties' (selected).

4. Change the resource group or Azure subscription that's associated with your integration account.

This screenshot shows the 'Properties' blade for the integration account. It includes fields for 'NAME' (MyNewIntegrationAccount), 'LOCATION' (brazilsouth), and 'Visual Studio Enterprise' under 'Subscription'. A red box highlights the 'Change resource group' link under 'Resource group' and the 'Change subscription' link under 'Subscription'.

NAME	MyNewIntegrationAccount
Resource group	MyNewRG
<a href="#">Change resource group</a>	
LOCATION	brazilsouth
Subscription	Visual Studio Enterprise
<a href="#">Change subscription</a>	

## Next Steps

- [Learn more about agreements](#)

# Add or update partners in business-to-business agreements in your workflow

2/28/2017 • 2 min to read • [Edit Online](#)

Partners are entities that participate in business-to-business (B2B) transactions and exchange messages between each other. Before you can create partners that represent you and another organization in these transactions, you must both share information that identifies and validates messages sent by each other. After you discuss these details and are ready to start your business relationship, you can create partners in your integration account to represent you both.

## What roles do partners have in your integration account?

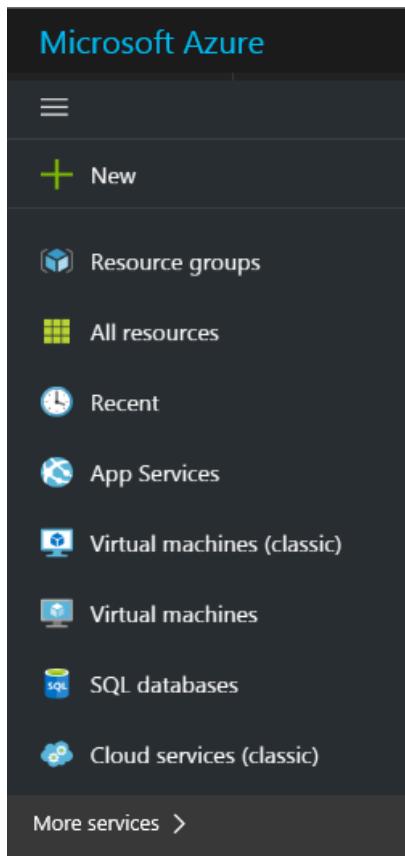
To define details about the messages exchanged between partners, you create agreements between those partners. However, before you can create an agreement, you must have added at least two partners to your integration account. Your organization must be part of the agreement as the **host partner**. The other partner, or **guest partner** represents the organization that exchanges messages with your organization. The guest partner can be another company, or even a department in your own organization.

After you add these partners, you can create an agreement.

Receive and Send settings are oriented from the point of view of the Hosted Partner. For example, the receive settings in an agreement determine how the hosted partner receives messages sent from a guest partner. Likewise, the send settings on the agreement indicate how the hosted partner sends messages to the guest partner.

## How to create a partner?

1. In the Azure portal, select **Browse**.



2. In the filter search box, enter **integration**, then select **Integration Accounts** in the results list.

A screenshot of the 'Integration Accounts' search results. A search bar at the top contains the text 'integration'. Below it, a single result is shown: 'Integration Accounts' with a star icon to its right.

3. Select the integration account where you want to add your partners.

A screenshot of the 'Integration Accounts' blade. It shows a table with one item: 'MyNewIntegrationAccount' (Integration Account, MyNewRG, Brazil South). The table has columns for NAME, TYPE, RESOURCE GROUP, and LOCATION.

4. Select the **Partners** tile.

The screenshot shows the 'MyNewIntegrationAccount' integration account in preview mode. At the top, there are 'Move' and 'Delete' buttons. Below the header, there's a 'Essentials' section with a collapse arrow, followed by user and export icons. The 'Components' section contains four tiles: 'Schemas' (0), 'Maps' (0), 'Certificates' (0), and 'Partners' (0). The 'Partners' tile is highlighted with a blue border. Below the tiles is a button labeled 'Add a section +'. At the bottom right of the main area is a 'All settings →' link.

Resource group  
MyNewRG

Name  
MyNewIntegrationAccount

Location  
brazilsouth

Subscription  
ICBCS9

Subscription ID  
1217a102-55fc-461a-9e2d-56a0aacc2972

All settings →

Components

Add tiles +

Schemas 0

Maps 0

Certificates 0

Partners 0

Agreements 0

Add a section +

5. In the Partners blade, choose **Add**.

The screenshot shows the 'Partners' blade for the 'MyNewIntegrationAccount'. It features a toolbar with 'Add' and 'Delete' buttons. Below the toolbar, the word 'Add' is highlighted with a blue box. A search bar labeled 'TYPE' is present. The main content area displays the message 'There are no partners to display.'

Partners

MyNewIntegrationAccount - PREVIEW

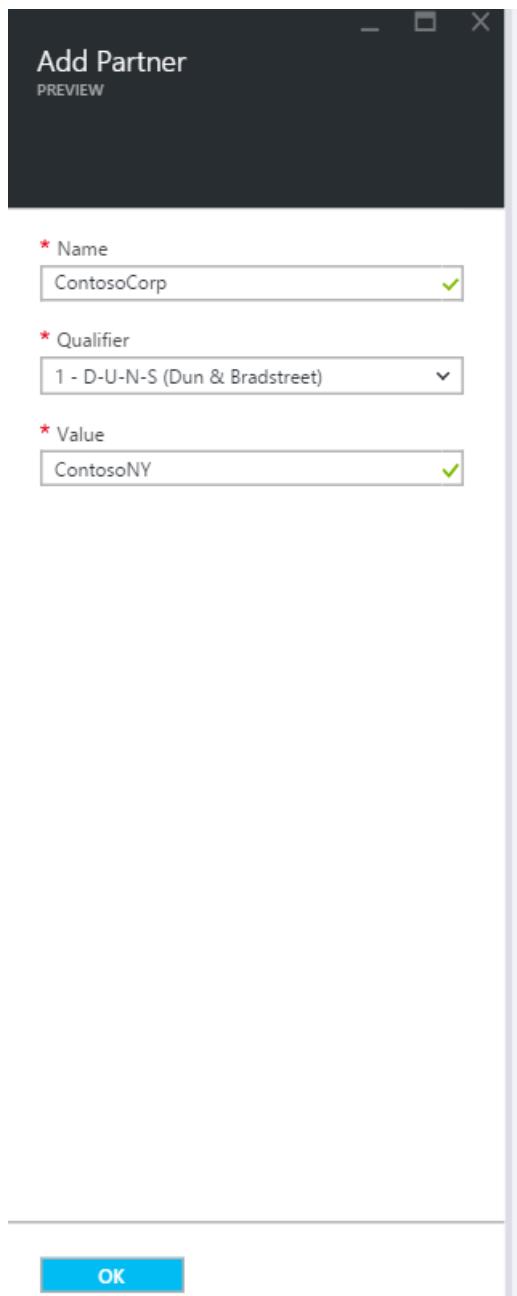
Add Delete

N Add

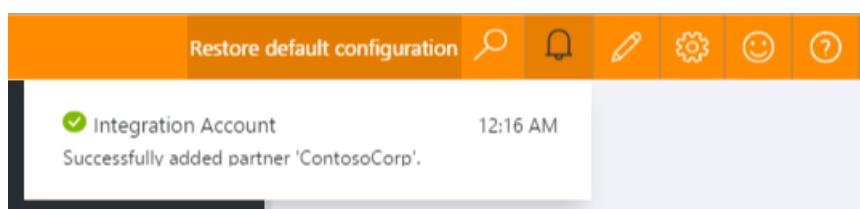
TYPE

There are no partners to display.

6. Enter a name for your partner, then select a **Qualifier**. Finally, enter a **Value** to help identify documents that come into your apps.



7. To see the progress for your partner creation process, select the *bell* notification icon.



8. To confirm that your new partners were successfully added, select the **Partners** tile.

The screenshot shows the 'MyNewIntegrationAccount' integration account preview. At the top, there are 'Move' and 'Delete' buttons. Below the title, there's a 'Resource group' section with 'MyNewRG' and a location 'brazilsouth'. The 'Name' is 'MyNewIntegrationAccount' and the 'Subscription' is 'ICBCS9'. A 'Subscription ID' is also listed. An 'All settings' button is at the bottom right. The main area is titled 'Components' and contains tiles for 'Schemas', 'Maps', 'Certificates', 'Partners' (which is highlighted with a blue border), 'Agreements', and a button to 'Add a section'. Each tile displays a count (0 or 1) and a small icon.

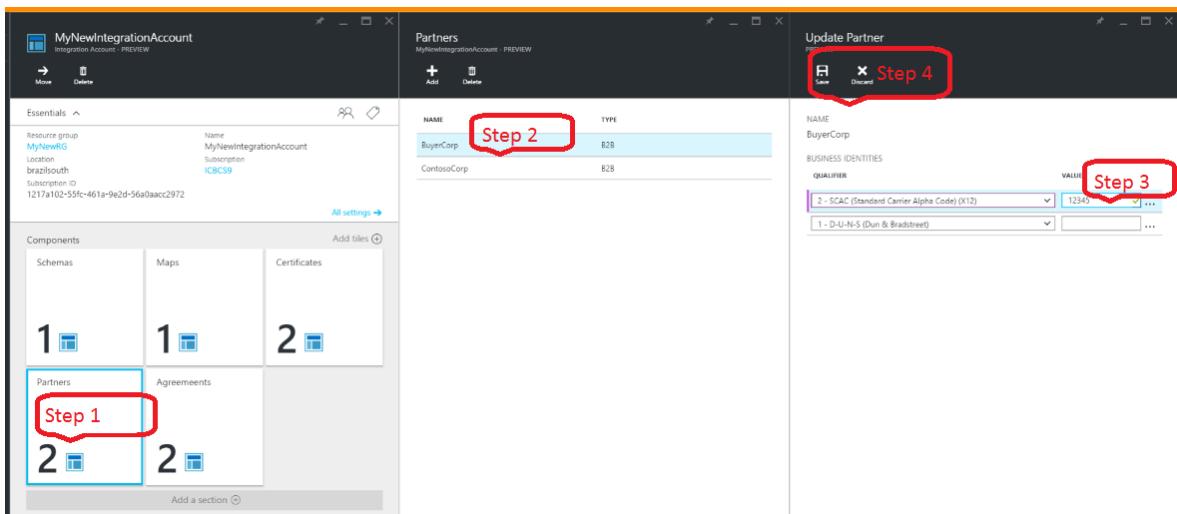
After you select the Partners tile, you'll also see newly added partners in the Partners blade.

The screenshot shows the 'Partners' blade for the 'MyNewIntegrationAccount' integration account. It has 'Add' and 'Delete' buttons at the top. The table below lists one partner:

NAME	TYPE
ContosoCorp	B2B

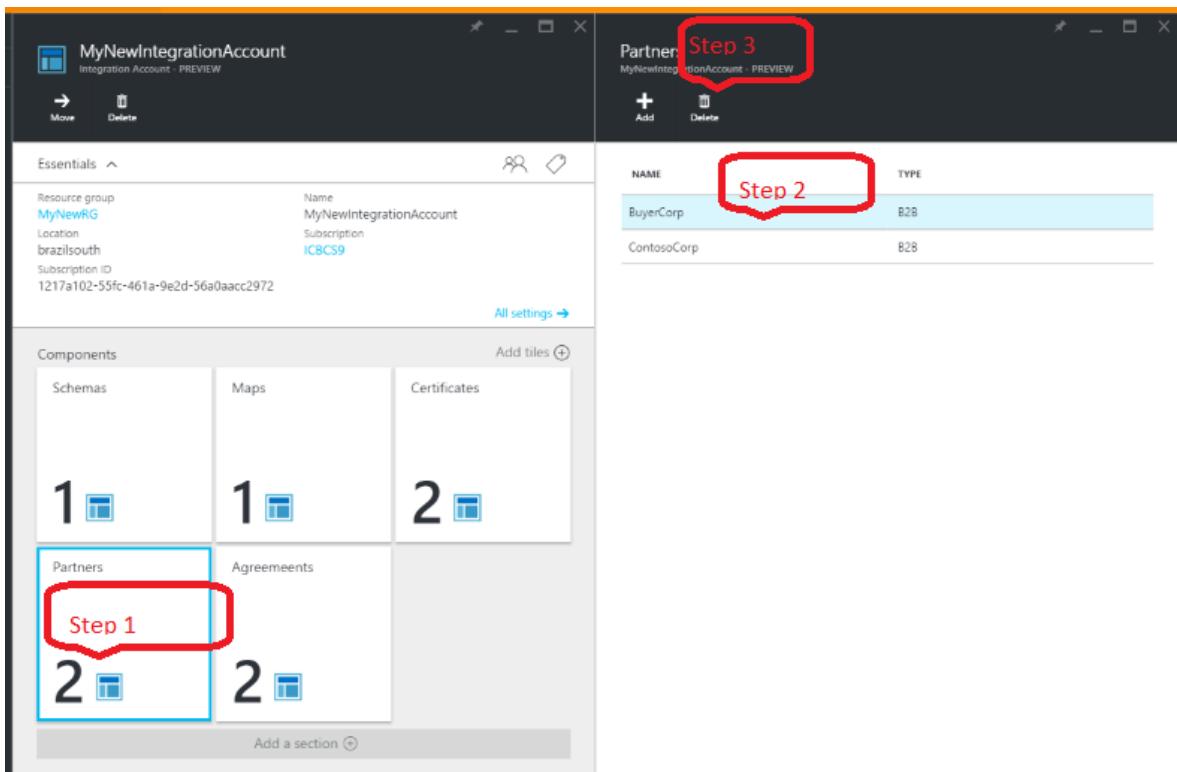
## How to edit existing partners in your integration account

1. Select the **Partners** tile.
2. After the Partners blade opens, select the partner you want to edit.
3. On the **Update Partner** tile, make your changes.
4. After you're done, choose **Save**, or to cancel your changes, select **Discard**.



## How to delete a partner

1. Select the **Partners** tile.
2. After the Partner blade opens, select the partner that you want to delete.
3. Choose **Delete**.



## Next steps

- Learn more about agreements

# Partner agreements for B2B communication with Azure Logic Apps and Enterprise Integration Pack

2/7/2017 • 1 min to read • [Edit Online](#)

Agreements let business entities communicate seamlessly using industry standard protocols and are the cornerstone for business-to-business (B2B) communication. When enabling B2B scenarios for logic apps with the Enterprise Integration Pack, an agreement is a communications arrangement between B2B trading partners. This agreement is based on the communications that the partners want to establish and is protocol or transport-specific.

Enterprise integration supports these protocol or transport standards:

- [AS2](#)
- [X12](#)
- [EDIFACT](#)

## Why use agreements

Here are some common benefits when using agreements:

- Enables different organizations and businesses to exchange information in a well-known format.
- Improves efficiency when conducting B2B transactions
- Easy to create, manage, and use when creating enterprise integration apps

## How to create agreements

- [Create an AS2 agreement](#)
- [Create an X12 agreement](#)
- [Create an EDIFACT agreement](#)

## How to use an agreement

You can create [logic apps](#) with B2B capabilities by using an agreement that you created.

## How to edit an agreement

You can edit any agreement by following these steps:

1. Select the integration account that has the agreement you want to update.
2. Choose the **Agreements** tile.
3. On the **Agreements** blade, select the agreement.
4. Choose **Edit**. Make your changes.
5. To save your changes, choose **OK**.

## How to delete an agreement

You can delete any agreement by following these steps:

1. Select the integration account that has the agreement you want to delete.
2. Choose the **Agreements** tile.
3. On the **Agreements** blade, select the agreement.
4. Choose **Delete**.
5. Confirm that you want to delete the selected agreement.

The Agreements blade no longer shows the deleted agreement.

## Next steps

- [Create an AS2 agreement](#)

# Receive data in logic apps with the B2B features in the Enterprise Integration Pack

2/7/2017 • 2 min to read • [Edit Online](#)

After you create an integration account that has partners and agreements, you are ready to create a business to business (B2B) workflow for your logic app with the [Enterprise Integration Pack](#).

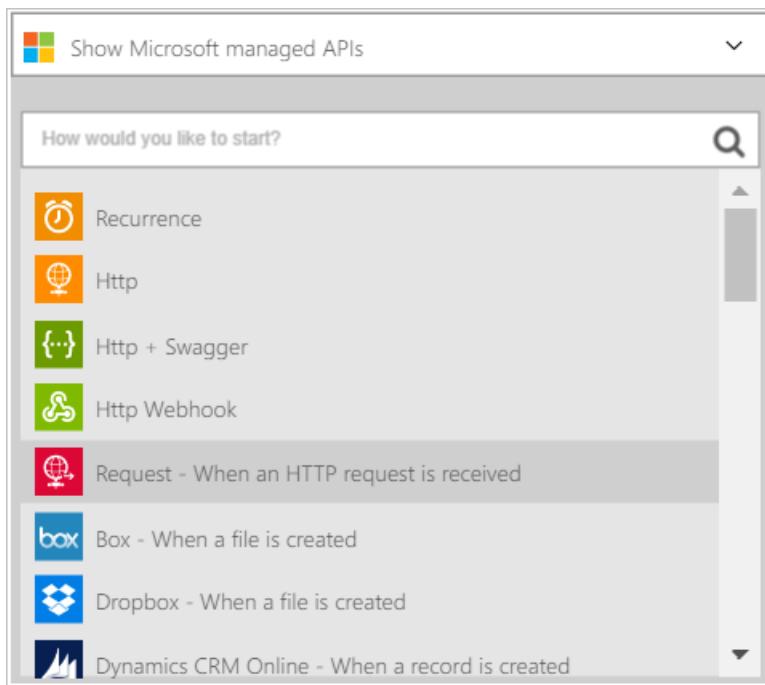
## Prerequisites

To use the AS2 and X12 actions, you must have an Enterprise Integration Account. Learn [how to create an Enterprise Integration Account](#).

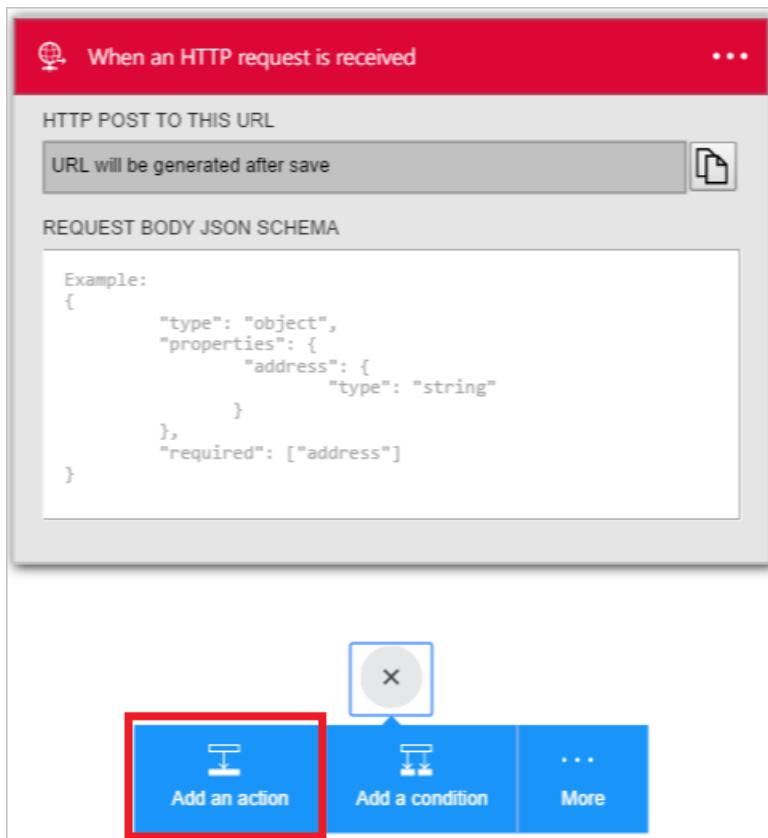
## Create a logic app with B2B connectors

Follow these steps to create a B2B logic app that uses the AS2 and X12 actions to receive data from a trading partner:

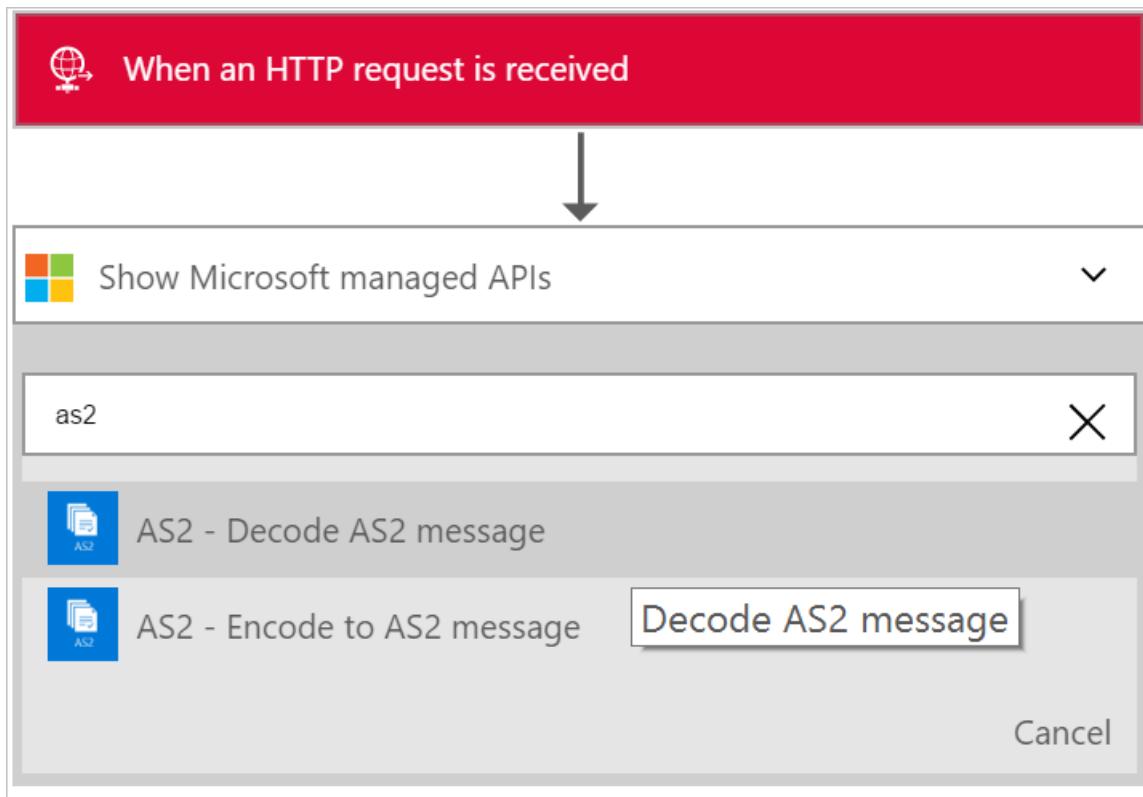
1. Create a logic app, then [link your app to your integration account](#).
2. Add a **Request - When an HTTP request is received** trigger to your logic app.



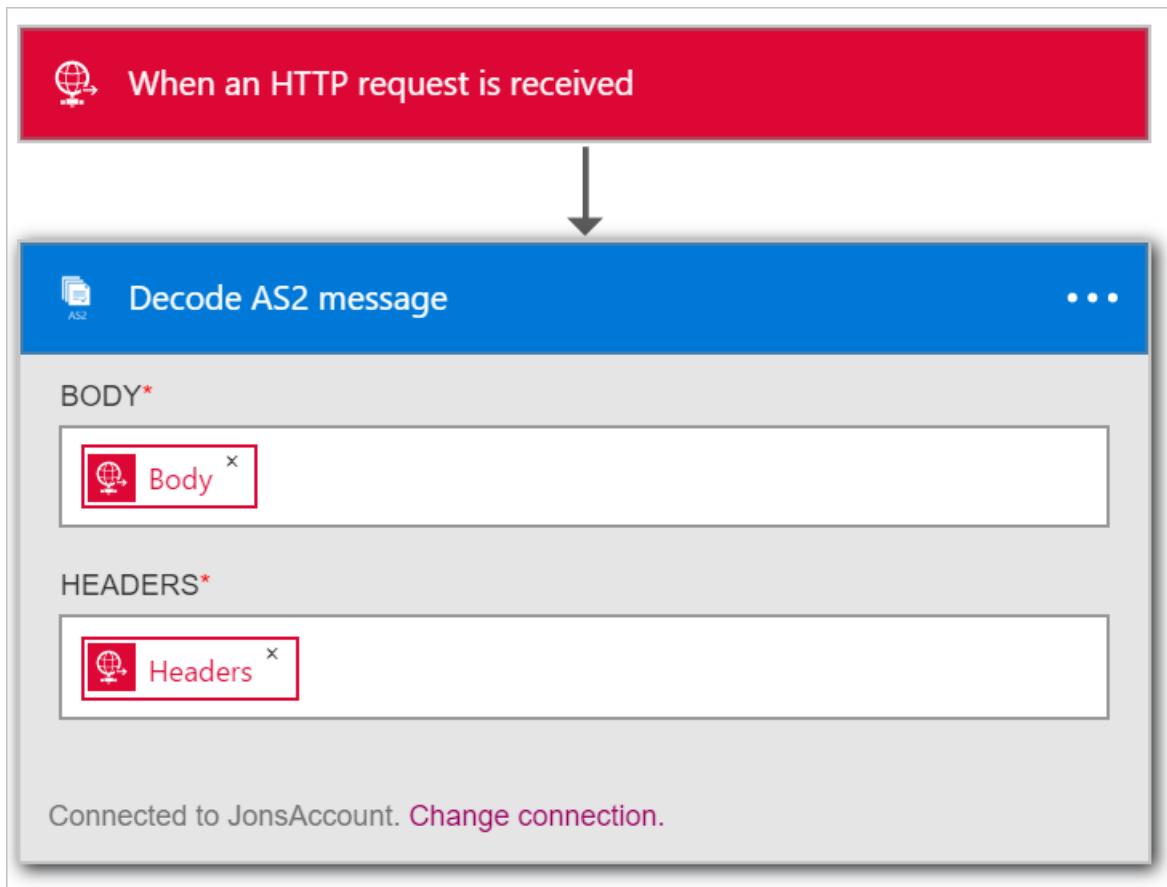
3. To add the **Decode AS2** action, select **Add an action**.



4. To filter all actions to the one that you want, enter the word **as2** in the search box.



5. Select the **AS2 - Decode AS2 message** action.



6. Add the **Body** that you want to use as input. In this example, select the body of the HTTP request that triggers the logic app. Or enter an expression that inputs the headers in the **HEADERS** field:  
`@triggerOutputs()['headers']`
7. Add the required **Headers** for AS2, which you can find in the HTTP request headers. In this example, select the headers of the HTTP request that trigger the logic app.
8. Now add the Decode X12 message action. Select **Add an action**.



When an HTTP request is received



Decode AS2 message

...

BODY\*



Body ×

HEADERS\*



Headers ×

Connected to JonsAccount. Change connection.



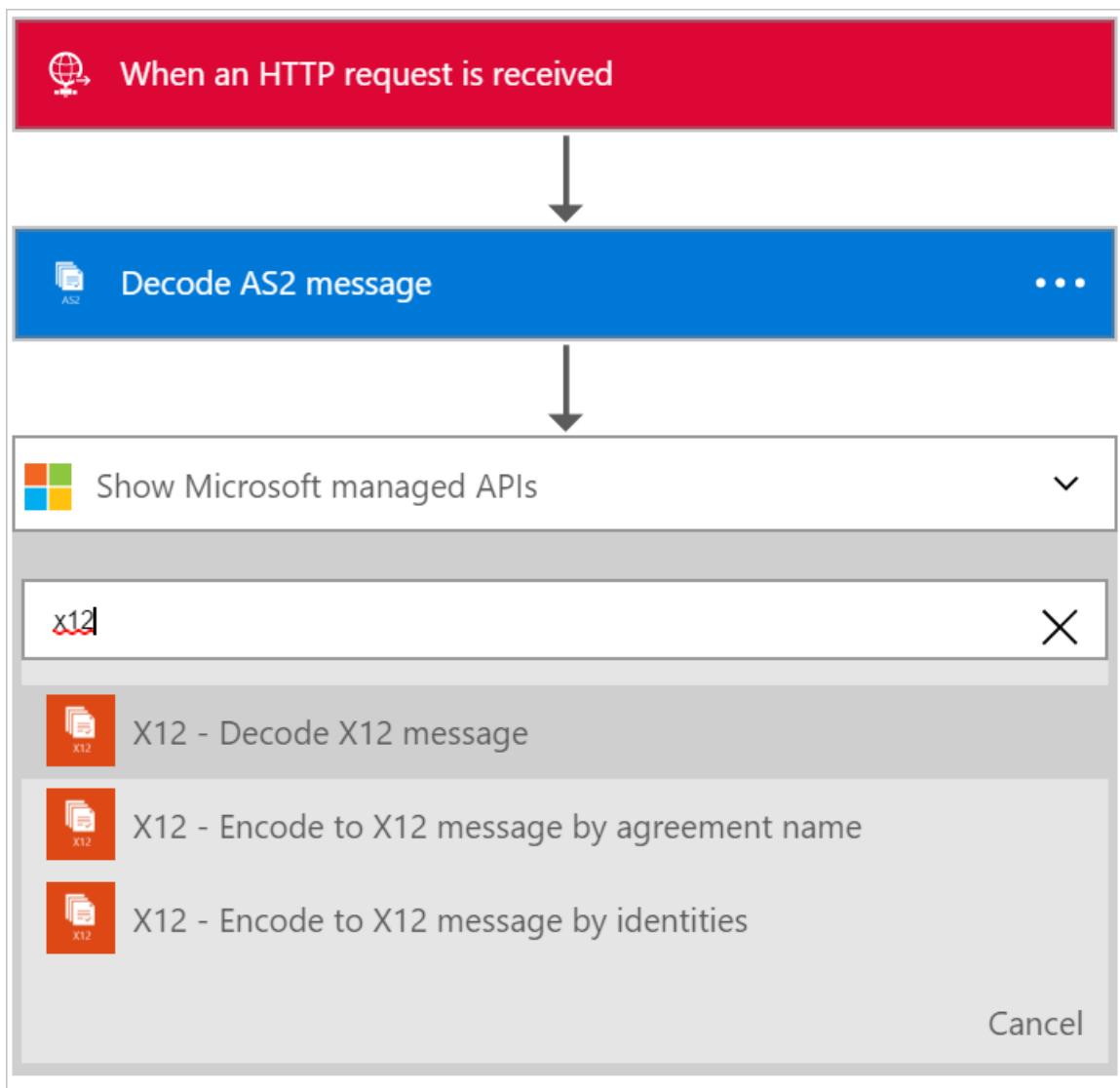
Add an action

Add a condition

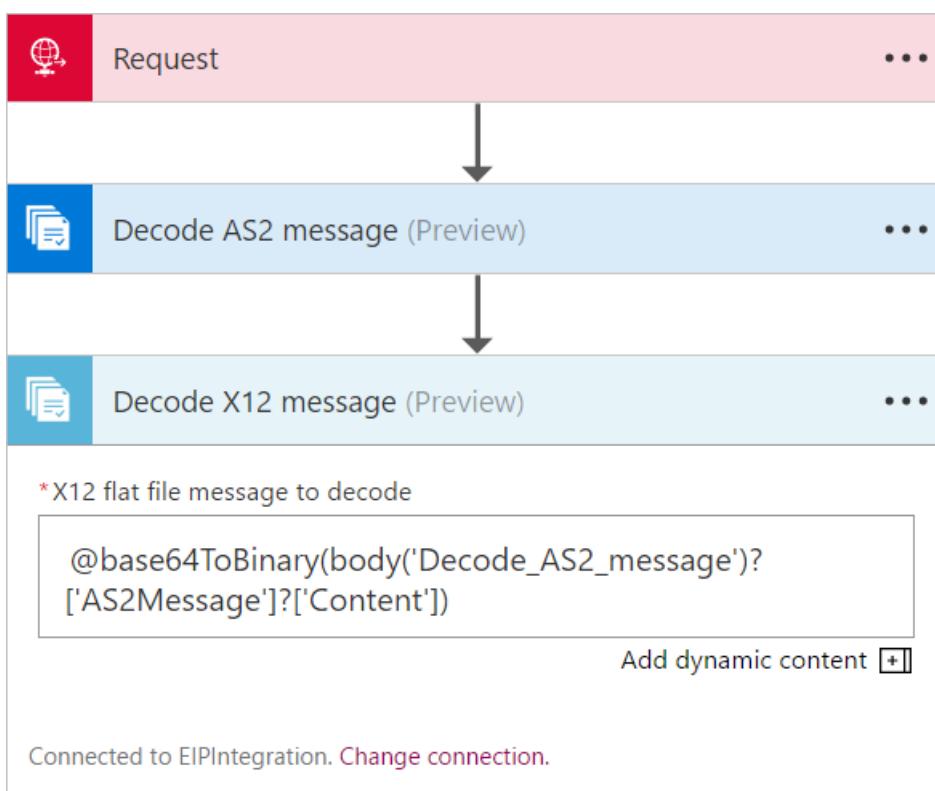
...

More

9. To filter all actions to the one that you want, enter the word **x12** in the search box.



10. Select the **X12 - Decode X12 message** action.



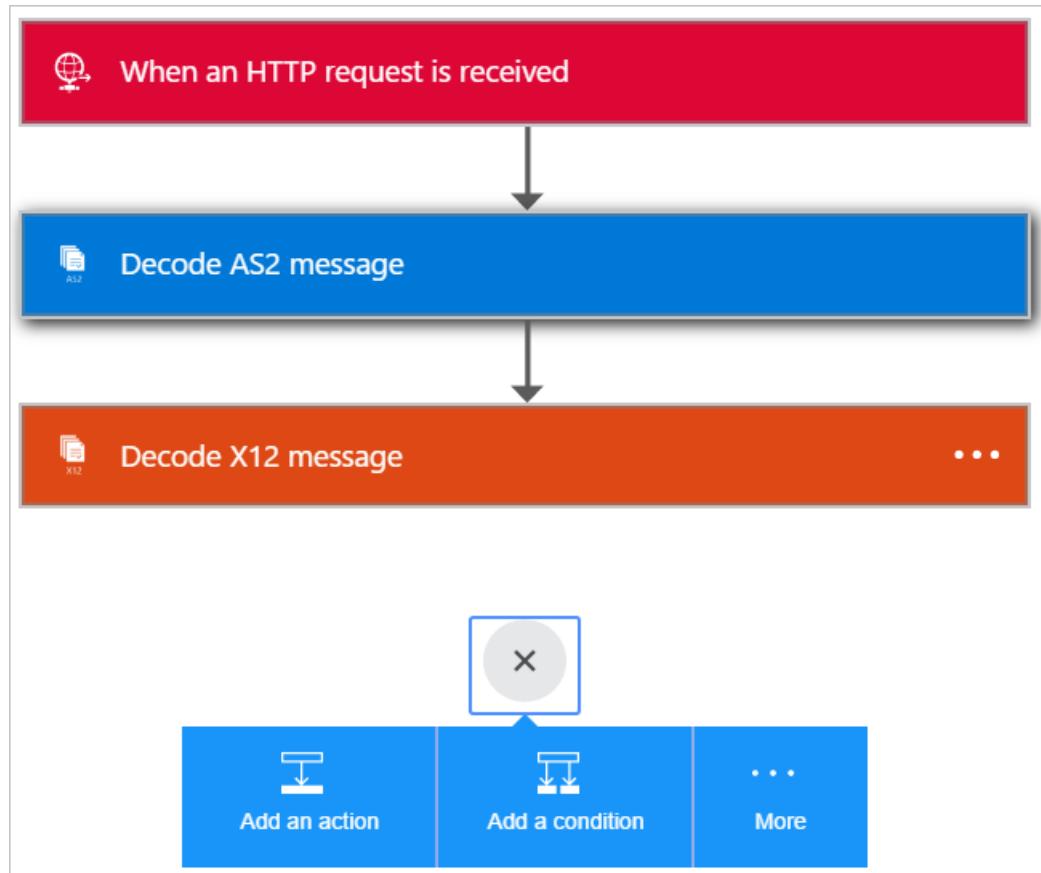
11. Now you must specify the input to this action. This input is the output from the previous AS2 action.

The actual message content is in a JSON object and is base64 encoded, so you must specify an expression as the input. Enter the following expression in the **X12 FLAT FILE MESSAGE TO DECODE** input field:

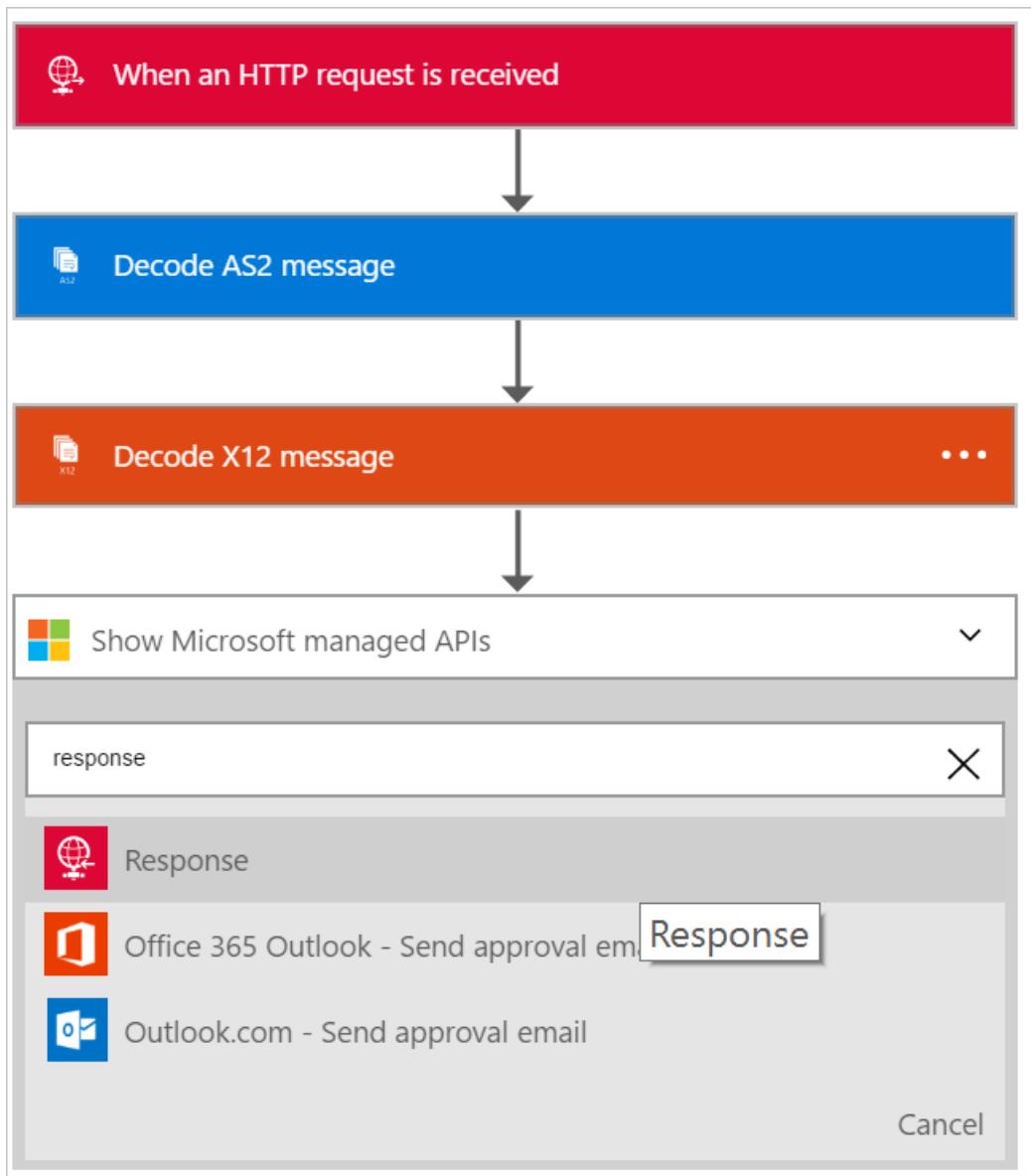
```
@base64ToString(body('Decode_AS2_message')?['AS2Message']?['Content'])
```

Now add steps to decode the X12 data received from the trading partner and output items in a JSON object. To notify the partner that the data was received, you can send back a response containing the AS2 Message Disposition Notification (MDN) in an HTTP Response Action.

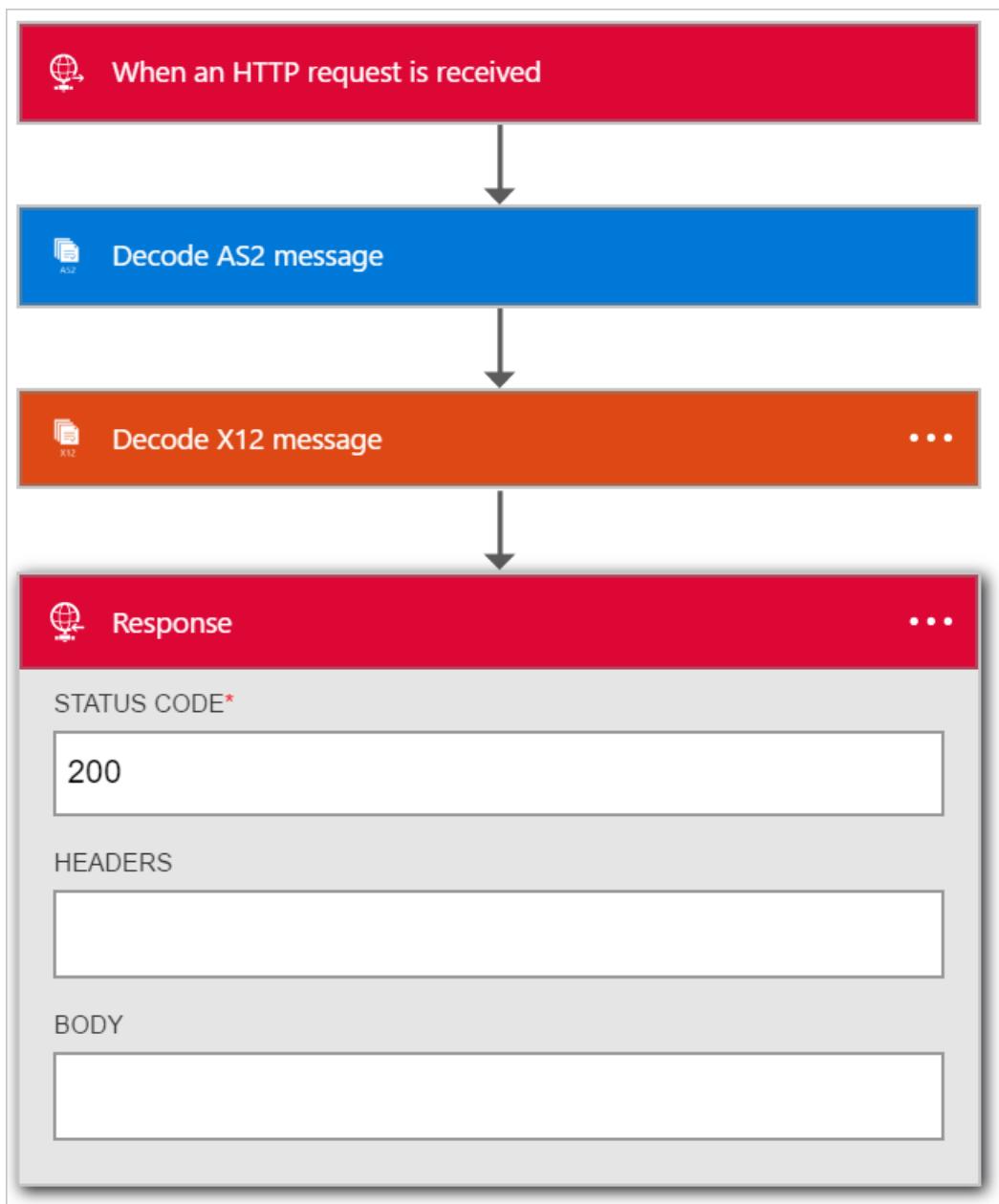
12. To add the **Response** action, choose **Add an action**.



13. To filter all actions to the one that you want, enter the word **response** in the search box.

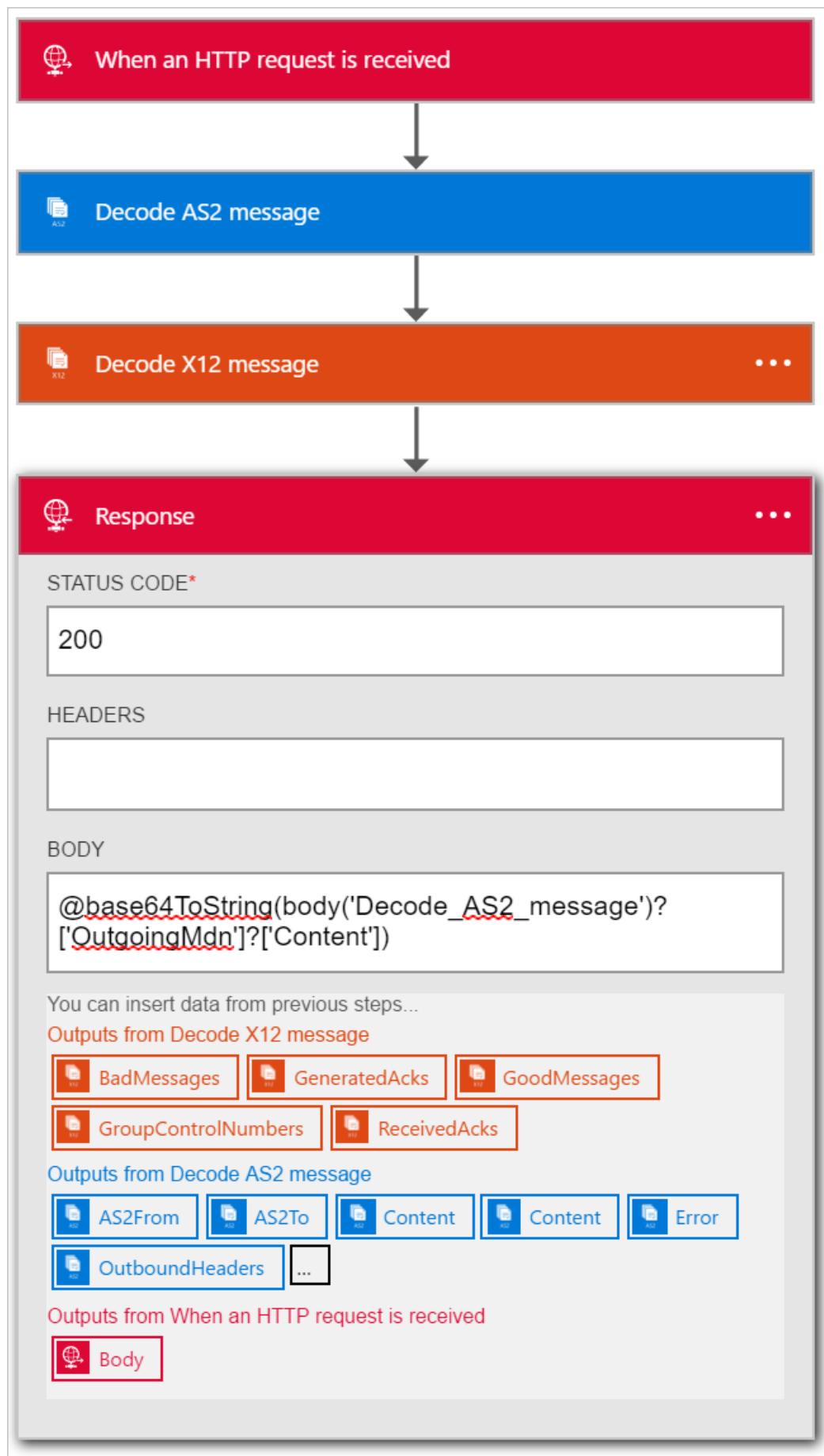


14. Select the **Response** action.

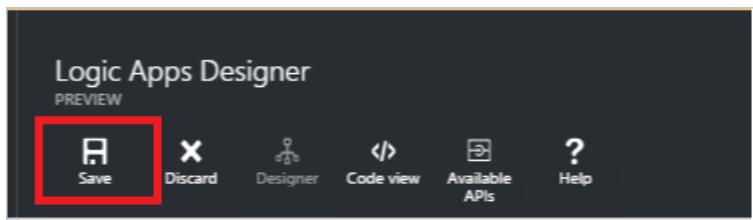


15. To access the MDN from the output of the **Decode X12 message** action, set the response **BODY** field with this expression:

```
@base64ToString(body('Decode_X12_message')?['OutgoingMdn']?['Content'])
```



16. Save your work.



You are now done setting up your B2B logic app. In a real world application, you might want to store the decoded X12 data in a line-of-business (LOB) app or data store. To connect your own LOB apps and use these APIs in your logic app, you can add further actions or write custom APIs.

## Features and use cases

- The AS2 and X12 decode and encode actions let you exchange data between trading partners by using industry standard protocols in logic apps.
- To exchange data with trading partners, you can use AS2 and X12 with or without each other.
- The B2B actions help you create partners and agreements easily in your integration account and consume them in a logic app.
- When you extend your logic app with other actions, you can send and receive data between other apps and services like SalesForce.

## Learn more

[Learn more about the Enterprise Integration Pack](#)

# Validate and transform XML, encode and decode flat files, and enrich messages features in logic apps

2/28/2017 • 1 min to read • [Edit Online](#)

Using logic apps, you have the ability to process XML messages that you send and receive. This feature is included with the Enterprise Integration Pack. For those users with a BizTalk Server background, the Enterprise Integration Pack gives you similar abilities to transform and validate messages, work with flat files, and even use XPath to enrich or extract specific properties from a message.

For those users who are new to this space, these features expand how you process messages within your workflow. For example, if you are in a business-to-business scenario, and work with specific XML schemas, then you can use the Enterprise Integration Pack to enhance how your company processes these messages.

The Enterprise Integration Pack includes:

- [XML validation](#) - Validate an incoming or outgoing XML message against a specific schema.
- [XML transform](#) - Convert or customize an XML message based on your requirements, or the requirements of a partner.
- [Flat file encoding and flat file decoding](#) - Encode or decode a flat file. For example, SAP accepts and sends IDOC files in flat file format. Many integration platforms create XML messages, including Logic Apps. So, you can create a logic app that uses the flat file encoder to "convert" XML files to flat files.
- [XPath](#) - Enrich a message and extract specific properties from the message. You can then use the extracted properties to route the message to a destination, or an intermediary endpoint.

## Try it out

[Deploy a fully operational logic app](#) (GitHub sample) by using the XML features in Azure Logic Apps.

## Learn more

[Learn more about the Enterprise Integration Pack](#)

# Overview of enterprise integration with flat files

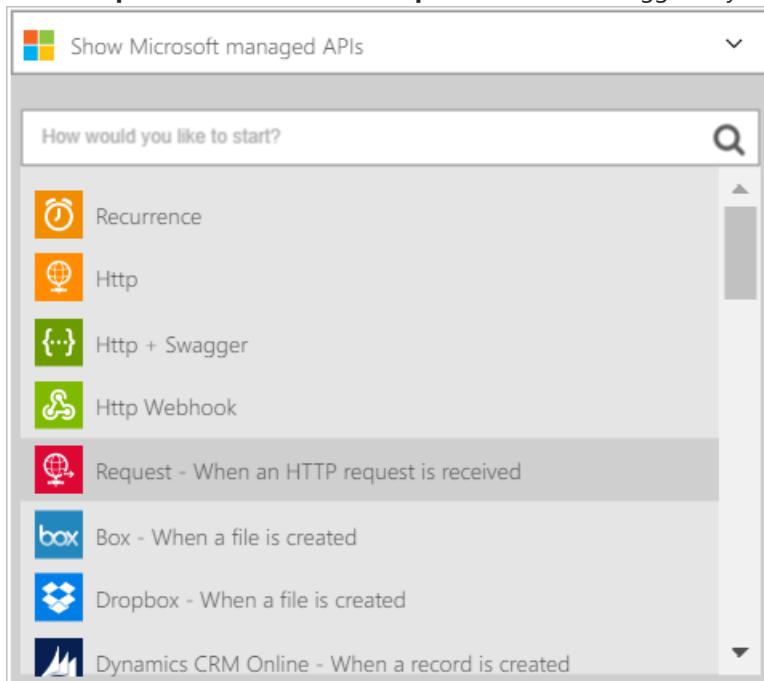
1/31/2017 • 3 min to read • [Edit Online](#)

You may want to encode XML content before you send it to a business partner in a business-to-business (B2B) scenario. In a logic app, you can use the flat file encoding connector to do this. The logic app that you create can get its XML content from a variety of sources, including from an HTTP request trigger, from another application, or even from one of the many [connectors](#). For more information about logic apps, see the [logic apps documentation](#).

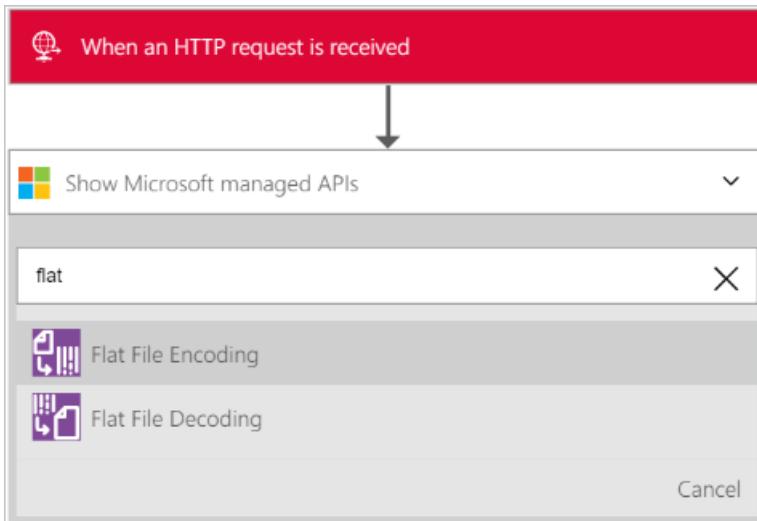
## Create the flat file encoding connector

Follow these steps to add a flat file encoding connector to your logic app.

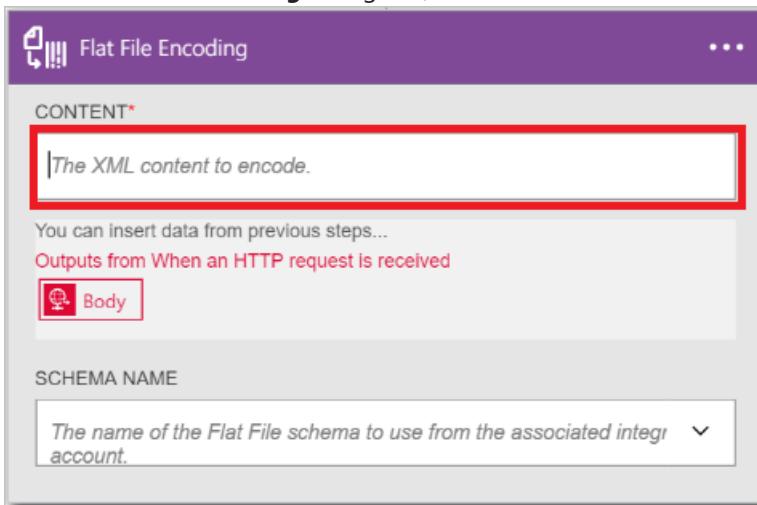
1. Create a logic app and [link it to your integration account](#). This account contains the schema you will use to encode the XML data.
2. Add a **Request - When an HTTP request is received** trigger to your logic app.



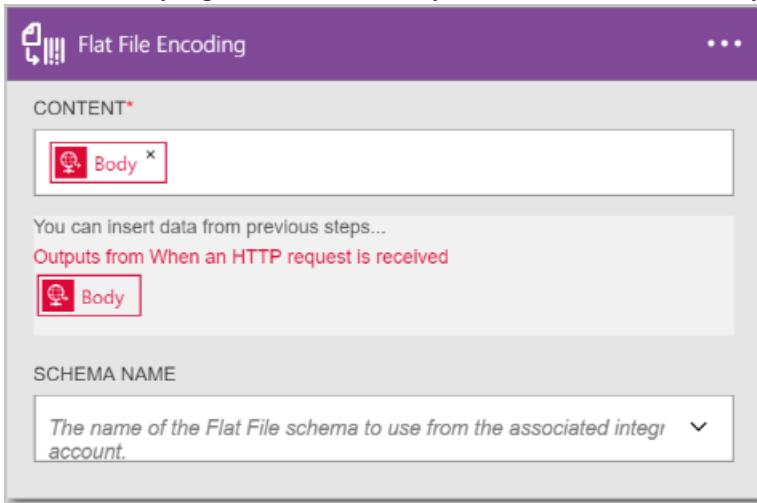
3. Add the flat file encoding action, as follows:
  - a. Select the **plus** sign.
  - b. Select the **Add an action** link (appears after you have selected the plus sign).
  - c. In the search box, enter *Flat* to filter all the actions to the one that you want to use.
  - d. Select the **Flat File Encoding** option from the list.



4. On the **Flat File Encoding** dialog box, select the **Content** text box.



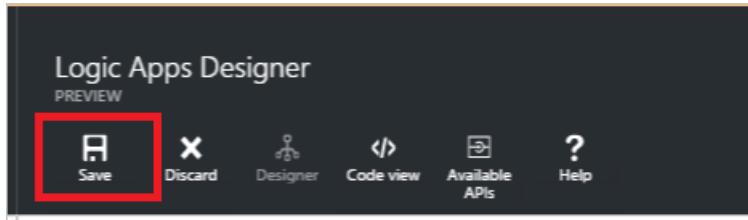
5. Select the body tag as the content that you want to encode. The body tag will populate the content field.



6. Select the **Schema Name** list box, and choose the schema you want to use to encode the input content.

The screenshot shows the 'Flat File Encoding' configuration page. At the top, there's a 'CONTENT' section containing a red box labeled 'Body'. Below it, a note says 'You can insert data from previous steps...' followed by 'Outputs from When an HTTP request is received' with another red box labeled 'Body'. Under 'SCHEMA NAME', there's a dropdown menu with the placeholder text 'The name of the Flat File schema to use from the associated integration account.' A red box highlights this dropdown.

7. Save your work.



At this point, you are finished setting up your flat file encoding connector. In a real world application, you may want to store the encoded data in a line-of-business application, such as Salesforce. Or you can send that encoded data to a trading partner. You can easily add an action to send the output of the encoding action to Salesforce, or to your trading partner, by using any one of the other connectors provided.

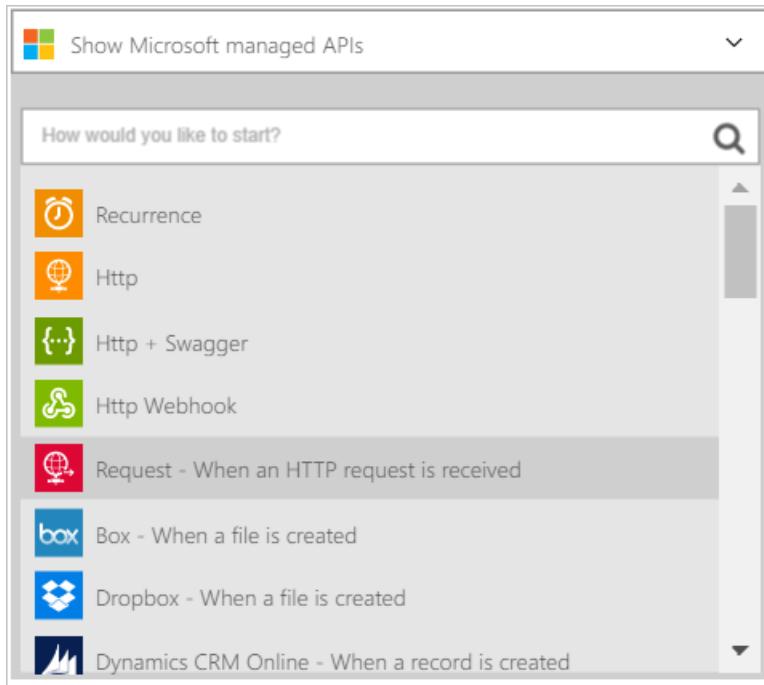
You can now test your connector by making a request to the HTTP endpoint, and including the XML content in the body of the request.

## Create the flat file decoding connector

### NOTE

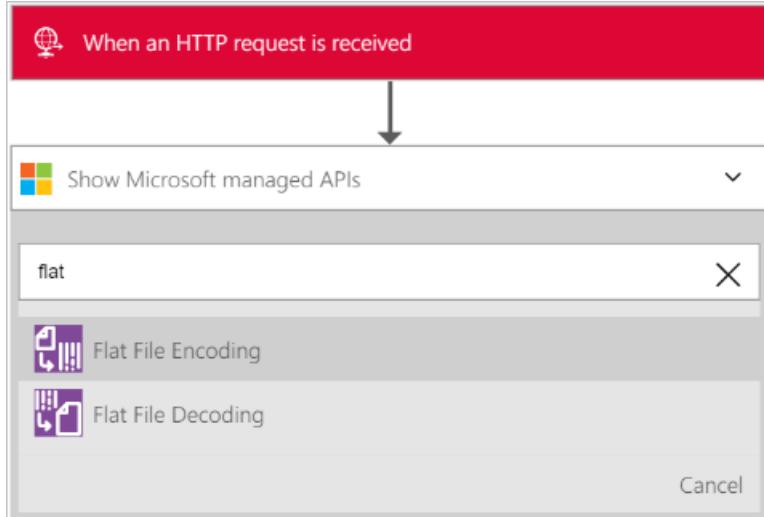
To complete these steps, you need to have a schema file already uploaded into your integration account.

1. Add a **Request - When an HTTP request is received** trigger to your logic app.

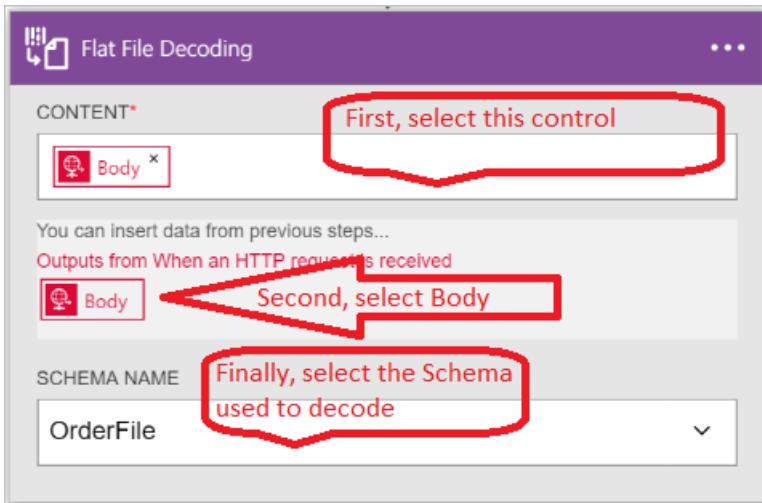


2. Add the flat file decoding action, as follows:

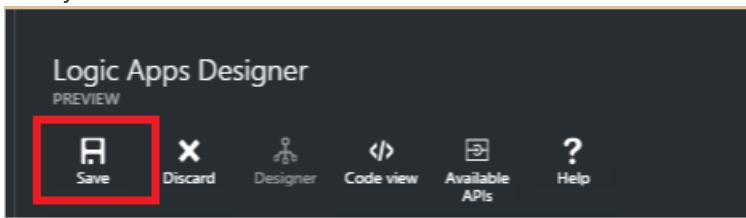
- Select the **plus** sign.
- Select the **Add an action** link (appears after you have selected the plus sign).
- In the search box, enter *Flat* to filter all the actions to the one that you want to use.
- Select the **Flat File Decoding** option from the list.



- Select the **Content** control. This produces a list of the content from earlier steps that you can use as the content to decode. Notice that the *Body* from the incoming HTTP request is available to be used as the content to decode. You can also enter the content to decode directly into the **Content** control.
- Select the *Body* tag. Notice the body tag is now in the **Content** control.
- Select the name of the schema that you want to use to decode the content. The following screenshot shows that *OrderFile* is the selected schema name. This schema name had been uploaded into the integration account previously.



6. Save your work.



At this point, you are finished setting up your flat file decoding connector. In a real world application, you may want to store the decoded data in a line-of-business application such as Salesforce. You can easily add an action to send the output of the decoding action to Salesforce.

You can now test your connector by making a request to the HTTP endpoint and including the XML content you want to decode in the body of the request.

## Next steps

- [Learn more about the Enterprise Integration Pack.](#)

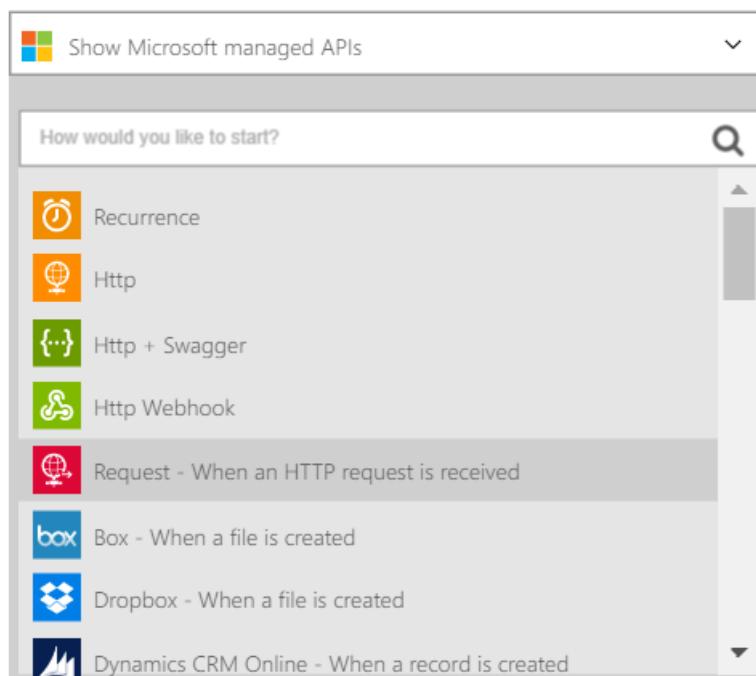
# Validate XML for enterprise integration

2/7/2017 • 1 min to read • [Edit Online](#)

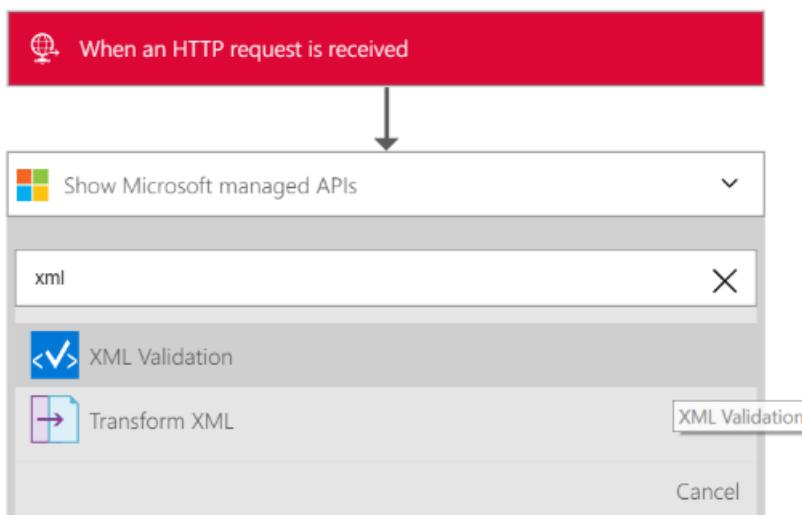
Often in B2B scenarios, the partners in an agreement must make sure that the messages they exchange are valid before data processing can start. You can validate documents against a predefined schema by using the use the XML Validation connector in the Enterprise Integration Pack.

## Validate a document with the XML Validation connector

1. Create a logic app, and [link the app to the integration account](#) that has the schema you want to use for validating XML data.
2. Add a **Request - When an HTTP request is received** trigger to your logic app.



3. To add the **XML Validation** action, choose **Add an action**.
4. To filter all the actions to the one that you want, enter *xml* in the search box. Choose **XML Validation**.



5. To specify the XML content that you want to validate, select **CONTENT**.

**<✓> XML Validation**

**CONTENT\***

The XML content to validate.

You can insert data from previous steps...  
Outputs from When an HTTP request is received

 Body

**SCHEMA NAME**

The name of the XML schema to use from the associated integrat... ▾

6. Select the body tag as the content that you want to validate.

**<✓> When an HTTP request is received**

**<✓> XML Validation**

**CONTENT\***

 Body

You can insert data from previous steps...  
Outputs from When an HTTP request is received

 Body

**SCHEMA NAME**

The name of the XML schema to use from the associated integrat... ▾

7. To specify the schema you want to use for validating the previous *content* input, choose **SCHEMA NAME**.

**<✓> When an HTTP request is received**

**<✓> XML Validation**

**CONTENT\***

 Body

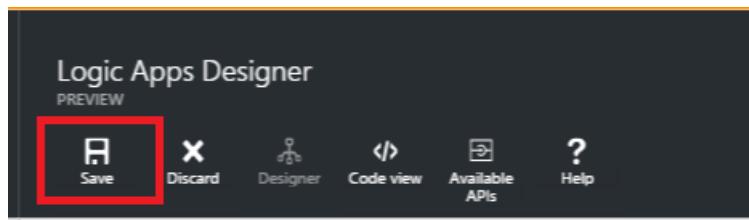
You can insert data from previous steps...  
Outputs from When an HTTP request is received

 Body

**SCHEMA NAME**

The name of the XML schema to use from the associated integrat... ▾

8. Save your work



You are now done with setting up your validation connector. In a real world application, you might want to store the validated data in a line-of-business (LOB) app like SalesForce. To send the validated output to Salesforce, add an action.

To test your validation action, make a request to the HTTP endpoint.

## Next steps

[Learn more about the Enterprise Integration Pack](#)

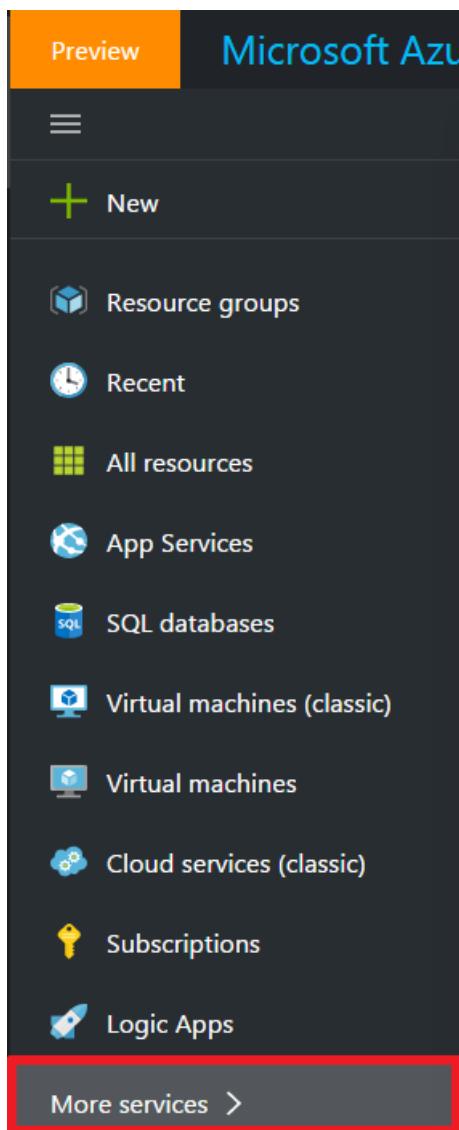
# Validate XML with schemas for Azure Logic Apps and the Enterprise Integration Pack

2/7/2017 • 1 min to read • [Edit Online](#)

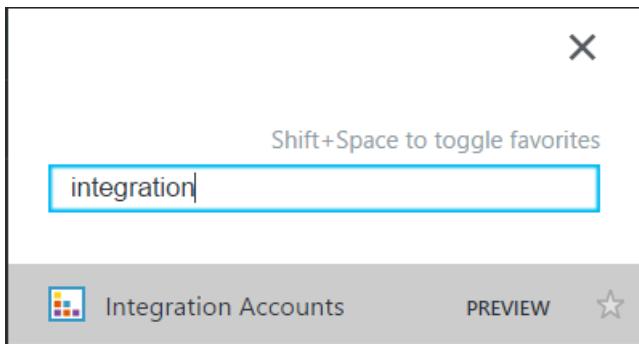
Schemas confirm that the XML documents you receive are valid and have the expected data in a predefined format. Schemas also help validate messages that are exchanged in a B2B scenario.

## Add a schema

1. In the Azure portal, select **More services**.



2. In the filter search box, enter **integration**, and select **Integration Accounts** from the results list.



3. Select the **integration account** where you want to add the schema.

A screenshot of the "Integration Accounts" page in Microsoft Power BI. The title bar says "Integration Accounts" and "Microsoft - PREVIEW". It includes buttons for "Add", "Columns", and "Refresh". Below the title, it says "Subscriptions: 1 of 5 selected". There are two filter boxes: "Filter items..." and "BTS4". A table lists five integration accounts:

NAME	TYPE	RESOURCE GROUP
EIPIntegration	Integration Account	EIPTemplates
FabrikamIntegrationAccount	Integration Account	as2tryit
FabrikamIntegrationAccount	Integration Account	JonsIARG
FabrikamIntegrationAccount	Integration Account	padas2try

4. Choose the **Schemas** tile.

The screenshot shows the 'Components' section of the EIPIntegration Integration Account. The 'Schemas' component is highlighted with a red border. It displays 0 schemas, 2 maps, and 0 certificates. Below this, there are other components: Partners (5), Agreements (3), and another Schemas component.

Component	Count
Schemas	0
Maps	2
Certificates	0
Partners	5
Agreements	3

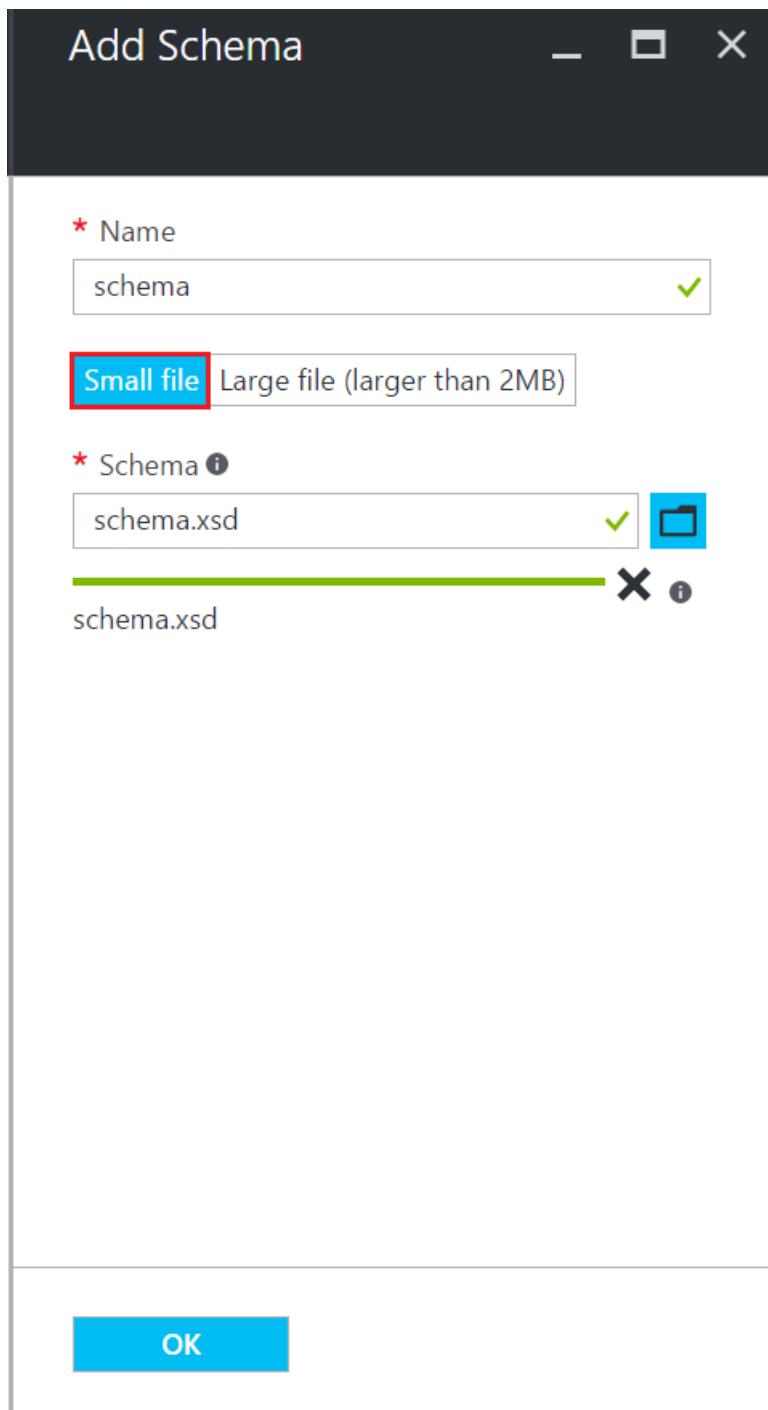
#### Add a schema file smaller than 2 MB

1. In the **Schemas** blade that opens (from the preceding steps), choose **Add**.

The 'Schemas' blade is open, showing a list of existing schemas. The 'Add' button is highlighted with a red box. The table below lists three existing schemas: EFACT\_d10b\_CUSDEC, EFACT\_D97A\_ORDERS, and X1200401850.

NAME	TYPE	CONTENT SIZE	CHANGED TIME
EFACT_d10b_CUSDEC	Xml	1.33 MiB	8/22/2016, 11:15 AM
EFACT_D97A_ORDERS	Xml	675.01 KiB	8/22/2016, 11:06 AM
X1200401850	Xml	1.34 MiB	7/27/2016, 10:53 AM

2. Enter a name for your schema. Upload the schema file by selecting the folder icon next to the **Schema** box. After the upload process completes, select **OK**.



#### Add a schema file larger than 2 MB (up to 8 MB maximum)

These steps differ based on the blob container access level: **Public** or **No anonymous access**.

##### To determine this access level

1. Open **Azure Storage Explorer**.
2. Under **Blob Containers**, select the blob container you want.
3. Select **Security, Access Level**.

If the blob security access level is **Public**, follow these steps.

The screenshot shows the Azure Storage Explorer interface. On the left, there's a sidebar with 'padstorageoms' selected. In the main area, under 'padstorageoms', there's a 'blob container' named 'aaa' (3 blobs, 6.37M) as of 9/16/2016 8:13:00 AM. A red box highlights the 'Security' button in the toolbar above the blob list. A modal window titled 'Blob Container Security' is open, showing the 'Access Level' tab selected. It asks 'Change the blob container's access level here.' The 'Blob Container Name' is set to 'aaa'. The 'Access Level' dropdown is open, with the first option, 'Public Container: Anonymous clients can read blob and container content/metadata.' (radio button selected), highlighted by a red box. The other two options ('Public Blob' and 'Off') are also shown. At the bottom of the modal are 'Update Access Level' and 'Close' buttons.

1. Upload the schema to your storage account, and copy the URI.

The screenshot shows the Azure Storage Explorer interface. Under 'padstorageoms', there's a 'blob container' named 'aaa' (1 blob, 3.35M) as of 9/13/2016 8:04:11 AM. A red box highlights the 'View' button in the toolbar. A modal window titled 'View Blob - X1200401855' is open, showing the 'Properties' tab selected. It lists properties like Blob Type (Block), Cache Control, Container (aaa), Content Disposition, Content Encoding (highlighted by a red box), Content Language, Last Modified (9/12/2016 9:51:01 PM +00:00), Length (3514650), Name (X1200401855.xsd), Parent (aaa), and Snapshot Qualified Storage Uri. The 'Content Encoding' field contains the Primary URI: https://padstorageoms.blob.core.windows.net/aaa/X1200401855. A secondary URI is also listed: https://padstorageoms-secondary.blob.core.windows.net/aaa/.

2. In **Add Schema**, select **Large file**, and provide the URI in the **Content URI** text box.

The screenshot shows the 'Schemas' and 'Add Schema' windows. The 'Schemas' window on the left has a 'NAME' column with entries: EFACT\_d10b\_CUSDEC, EFACT\_D97A\_ORDERS, and X1200401850. The 'Add' button is highlighted with a red box. The 'Add Schema' window on the right has fields for 'Name' (X1200401855) and 'Content URI' (https://padstorageoms.blob.core.windows...). The 'Content URI' field is highlighted with a red box. The 'Content URI' dropdown menu shows 'Small file' and 'Large file (larger than 2MB)', with 'Large file (larger than 2MB)' selected.

If the blob security access level is **No anonymous access**, follow these steps.

1. Upload the schema to your storage account.

2. Generate a shared access signature for the schema.

3. In **Add Schema**, select **Large file**, and provide the shared access signature URI in the **Content URI** text box.

The screenshot shows the 'Schemas' blade for an integration account named 'EIPIntegration'. It lists three existing schemas: 'EFACT\_d10b\_CUSDEC', 'EFACT\_D97A\_ORDERS', and 'X1200401850'. The 'Add' button is highlighted with a red box. To the right, a separate window titled 'Add Schema' is open, showing fields for 'Name' (X1200401855), 'Content URI' (https://padstorageoms.blob.core.windows...), and a note about it being a 'Large file (larger than 2MB)'.

- In the **Schemas** blade of your integration account, your newly added schema should appear.

The screenshot shows the 'Schemas' blade for the same 'EIPIntegration' account. The newly added schema 'X1200401855.xsd' is now listed in the table, highlighted with a red box. The table also includes other schemas like 'sapBinding\_Common' and 'schema'.

## Edit schemas

- Choose the **Schemas** tile.
- After the **Schemas** blade opens, select the schema that you want to edit.
- On the **Schemas** blade, choose **Edit**.

The screenshot shows the 'Schemas' blade for the 'EIPIntegration' account. A schema named 'schema' is selected and highlighted with a red box. The 'Edit' button at the top of the blade is also highlighted with a red box. The table lists two schemas: 'schema' and 'schema2'.

- Select the schema file that you want to edit, then select **Open**.

The screenshot shows two windows side-by-side. The left window is titled 'Schemas' and lists two items: 'schema' (Xml, 1.34 MiB, changed 9/13/2016, 11:02 AM) and 'X1200401850' (Xml, 1.34 MiB, changed 7/27/2016, 10:53 AM). The right window is titled 'Edit Schema' and contains a form with a 'Name' field set to 'schema'. Below it is a note about a 'Small file' (larger than 2MB). A dropdown menu shows 'schema.xsd' selected.

Azure shows a message that the schema uploaded successfully.

## Delete schemas

- Choose the **Schemas** tile.
- After the **Schemas** blade opens, select the schema you want to delete.
- On the **Schemas** blade, choose **Delete**.

The screenshot shows the 'Schemas' blade for the 'EIPIntegration' integration account. On the left, there's a summary section with resource group 'EIPTemplates', location 'West US', and subscription ID. Below it is a 'Components' section with cards for 'Schemas' (2), 'Maps' (2), 'Certificates' (0), 'Partners' (5), and 'Agreements' (3). The 'Schemas' card is highlighted with a red box. On the right, a table lists schemas: 'schema' (Xml, 1.34 MiB, changed 9/13/2016, 1:21 PM) and 'schema2' (Xml, 1.68 MiB, changed 9/13/2016, 1:23 PM). The 'Delete' button in the top right of the blade is also highlighted with a red box.

- To confirm that you want to delete the selected schema, choose **Yes**.

The screenshot shows a confirmation dialog box. It asks 'Delete schema' and 'Are you sure you want to delete the selected schema?'. At the bottom are two buttons: 'Yes' (highlighted) and 'No'.

In the **Schemas** blade, the schema list refreshes and no longer includes the schema that you deleted.

The screenshot shows the Azure portal interface for an Integration Account. On the left, the 'Essentials' tab is selected, displaying basic account information: Resource group (EIPTemplates), Name (EIPIntegration), Location (West US), Subscription name (BTS4), and a Subscription ID. A red box highlights the 'Schemas' component in the 'Components' section, which contains 1 item. The 'Maps' component contains 2 items, 'Certificates' contains 0, 'Partners' contains 5, and 'Agreements' contains 3.

NAME	TYPE	CONTENT SIZE	CHANGED TIME
X1200401850	Xml	1.34 MiB	7/27/2016, 10:53 AM

## Next steps

- Learn more about the Enterprise Integration Pack.

# Enterprise integration with XML transforms

2/7/2017 • 2 min to read • [Edit Online](#)

## Overview

The Enterprise integration Transform connector converts data from one format to another format. For example, you may have an incoming message that contains the current date in the YearMonthDay format. You can use a transform to reformat the date to be in the MonthDayYear format.

## What does a transform do?

A Transform, which is also known as a map, consists of a Source XML schema (the input) and a Target XML schema (the output). You can use different built-in functions to help manipulate or control the data, including string manipulations, conditional assignments, arithmetic expressions, date time formatters, and even looping constructs.

## How to create a transform?

You can create a transform/map by using the Visual Studio [Enterprise Integration SDK](#). When you are finished creating and testing the transform, you upload the transform into your integration account.

## How to use a transform

After you upload the transform/map into your integration account, you can use it to create a Logic app. The Logic app runs your transformations whenever the Logic app is triggered (and there is input content that needs to be transformed).

**Here are the steps to use a transform:**

### Prerequisites

- Create an integration account and add a map to it

Now that you've taken care of the prerequisites, it's time to create your Logic app:

1. Create a Logic app and [link it to your integration account](#) that contains the map.
2. Add a **Request** trigger to your Logic app

The screenshot shows the 'Show Microsoft managed APIs' dropdown open, displaying a list of triggers. The 'Request' trigger is highlighted with a red box.

- Recurrence
- HTTP
- HTTP + Swagger
- HTTP Webhook
- Request**
- appFigures - When an app is rated (Preview)
- appFigures - When an event occurs (Preview)
- appFigures - When there is a new review (Preview)

3. Add the **Transform XML** action by first selecting **Add an action**

The screenshot shows the Logic App builder interface. A red box highlights the 'Add an action' button in the central toolbar.

+ New step

Add an action    Add a condition    More

4. Enter the word *transform* in the search box to filter all the actions to the one that you want to use

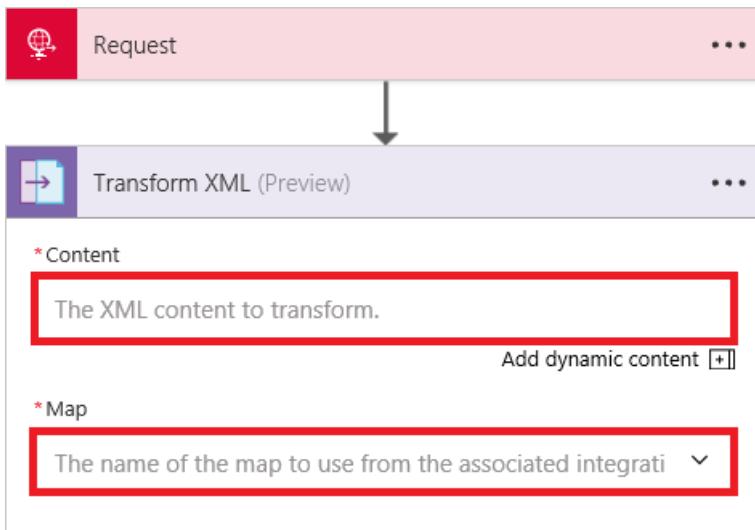
The screenshot shows the Logic App builder interface with a search bar containing 'trans'. A red box highlights the 'Transform XML (Preview)' action in the results list.

X trans

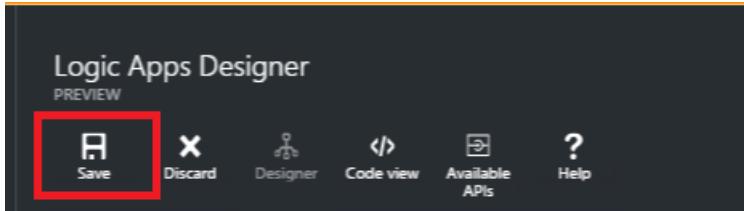
Transform XML (Preview)    Mandrill - Send mail

5. Select the **Transform XML** action

6. Add the XML **CONTENT** that you transform. You can use any XML data you receive in the HTTP request as the **CONTENT**. In this example, select the body of the HTTP request that triggered the Logic app.
7. Select the name of the **MAP** that you want to use to perform the transformation. The map must already be in your integration account. In an earlier step, you already gave your Logic app access to your integration account that contains your map.



#### 8. Save your work



At this point, you are finished setting up your map. In a real world application, you may want to store the transformed data in an LOB application such as SalesForce. You can easily do so as an action to send the output of the transform to Salesforce.

You can now test your transform by making a request to the HTTP endpoint.

## Features and use cases

- The transformation created in a map can be simple, such as copying a name and address from one document to another. Or, you can create more complex transformations using the out-of-the-box map operations.
- Multiple map operations or functions are readily available, including strings, date time functions, and so on.
- You can do a direct data copy between the schemas. In the Mapper included in the SDK, this is as simple as drawing a line that connects the elements in the source schema with their counterparts in the destination schema.
- When creating a map, you view a graphical representation of the map, which shows all the relationships and links you create.
- Use the Test Map feature to add a sample XML message. With a simple click, you can test the map you created, and see the generated output.
- Upload existing maps
- Includes support for the XML format.

## Learn more

- [Learn more about the Enterprise Integration Pack](#)
- [Learn more about maps](#)

# Add maps for XML data transform

2/7/2017 • 1 min to read • [Edit Online](#)

Enterprise integration uses maps to transform XML data between formats. A map is an XML document that defines the data in a document that should be transformed into another format.

## Why use maps?

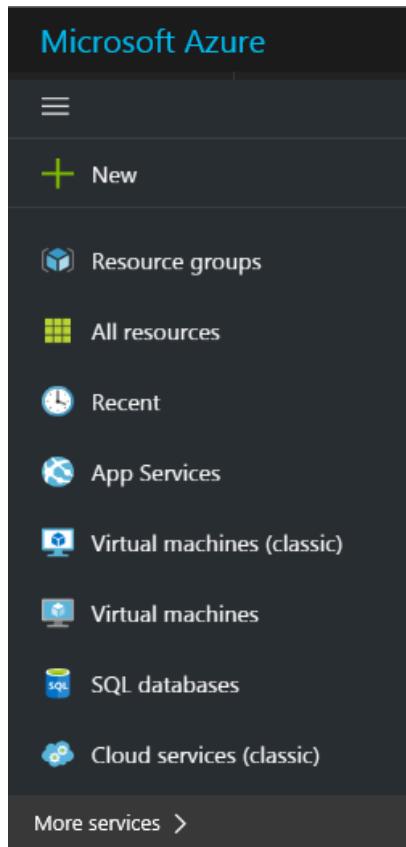
Suppose that you regularly receive B2B orders or invoices from a customer who uses the YYYYMMDD format for dates. However, in your organization, you store dates in the MMDDYYYY format. You can use a map to *transform* the YYYYMMDD date format into the MMDDYYYY before storing the order or invoice details in your customer activity database.

## How do I create a map?

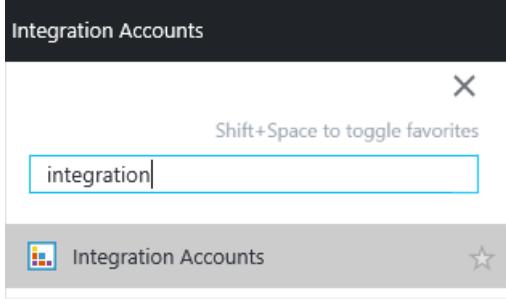
You can create BizTalk Integration projects with the [Enterprise Integration Pack](#) for Visual Studio 2015. You can then create an Integration Map file that lets you visually map items between two XML schema files. After you build this project, you will have an XSLT document.

## How do I add a map?

1. In the Azure portal, select **Browse**.



2. In the filter search box, enter **integration**, then select **Integration Accounts** from the results list.



3. Select the integration account where you want to add the map.

A screenshot of the 'Integration Accounts' blade in the Azure portal. It shows a list of one item: 'MyNewIntegrationAccount'. The list includes columns for NAME, TYPE, RESOURCE GROUP, and LOCATION. The account details are: NAME: MyNewIntegrationAccount, TYPE: Integration Account, RESOURCE GROUP: MyNewRG, and LOCATION: Brazil South. There are also buttons for 'Add', 'Columns', and 'Refresh' at the top, and a filter bar at the top.

4. Select the **Maps** tile.

MyNewIntegrationAccount  
Integration Account - PREVIEW

Move Delete

Essentials ▾

Resource group: MyNewRG  
Location: brazilsouth  
Subscription ID: 1217a102-55fc-461a-9e2d-56a0aacc2972

Name: MyNewIntegrationAccount  
Subscription: ICBCS9

All settings →

Components

Add tiles +

Schemas	Maps	Certificates
1	0	0
Partners	Agreements	
1	0	

Add a section +

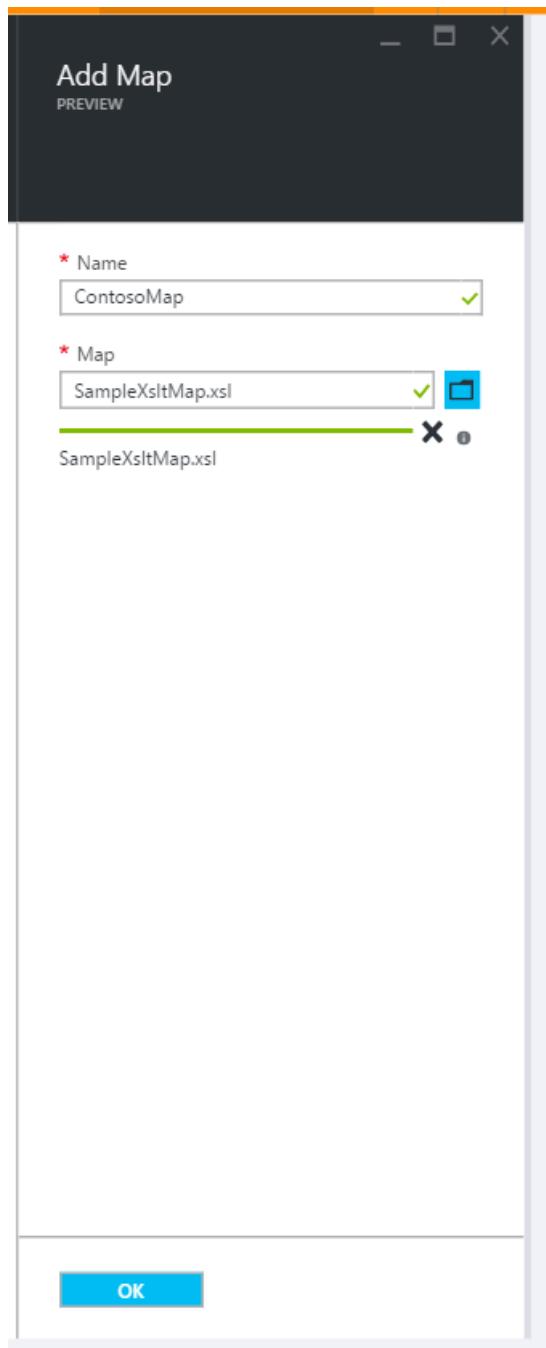
5. After the Maps blade opens, choose **Add**.

Maps  
MyNewIntegrationAccount - PREVIEW

Add Delete Update Download

NAME	TYPE	CONTENT SIZE
There are no maps to display.		

6. Enter a **Name** for your map. To upload the map file, choose the folder icon on the right side of the **Map** text box. After the upload process completes, choose **OK**.



7. After Azure adds the map to your integration account, you get an onscreen message that shows whether your map file was added or not. After you get this message, choose the **Maps** tile so you can view the newly added map.

**Essentials**

- Resource group: MyNewRG
- Location: brazilsouth
- Subscription ID: 1217a102-55fc-461a-9e2d-56a0aacc2972

**Components**

Schemas	Maps	Certificates
1	1	2
2	2	

**Properties**

NAME	ContosoMap
TYPE	Xslt
CONTENT SIZE	3056
CONTENT LINK	<a href="https://flowprodcu02cgsn01.blob.core.windows.net/">https://flowprodcu02cgsn01.blob.core.windows.net/</a>
CREATED TIME	6/25/2016, 1:13 AM
CHANGED TIME	6/25/2016, 1:13 AM

## How do I edit a map?

You must upload a new map file with the changes that you want. You can first download the map for editing.

To upload a new map that replaces the existing map, follow these steps.

1. Choose the **Maps** tile.
2. After the Maps blade opens, select the map that you want to edit.
3. On the **Maps** blade, choose **Update**.

**Essentials**

- Resource group: MyNewRG
- Location: brazilsouth
- Subscription ID: 1217a102-55fc-461a-9e2d-56a0aacc2972

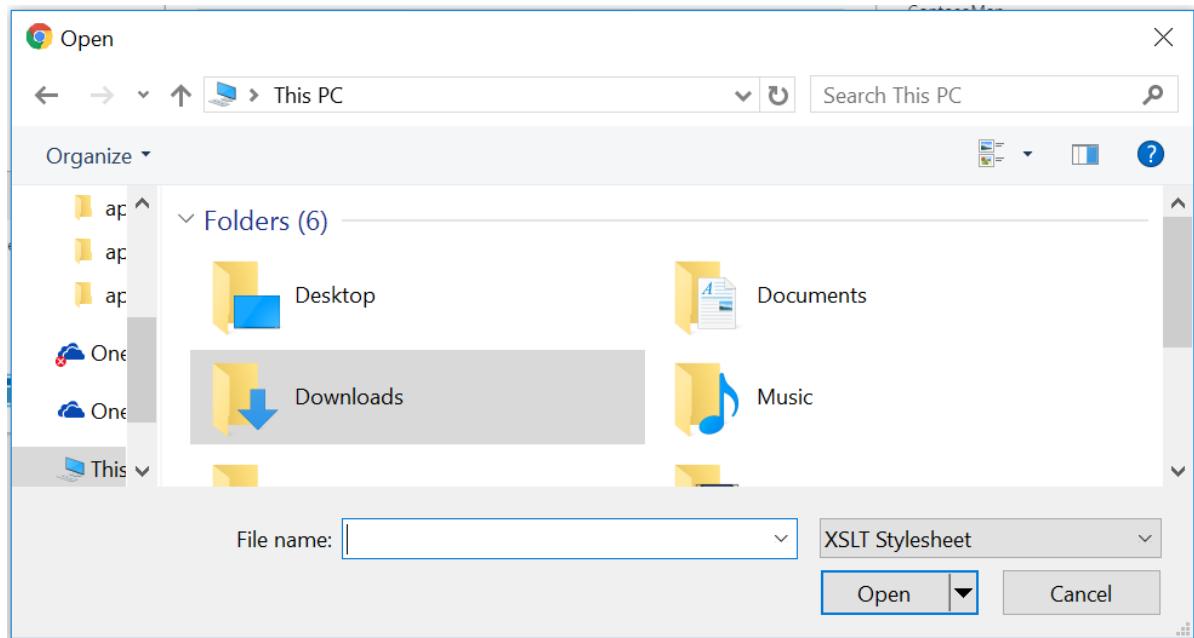
**Components**

Schemas	Maps	Certificates
1	Step 1	3
2	2	

**Properties**

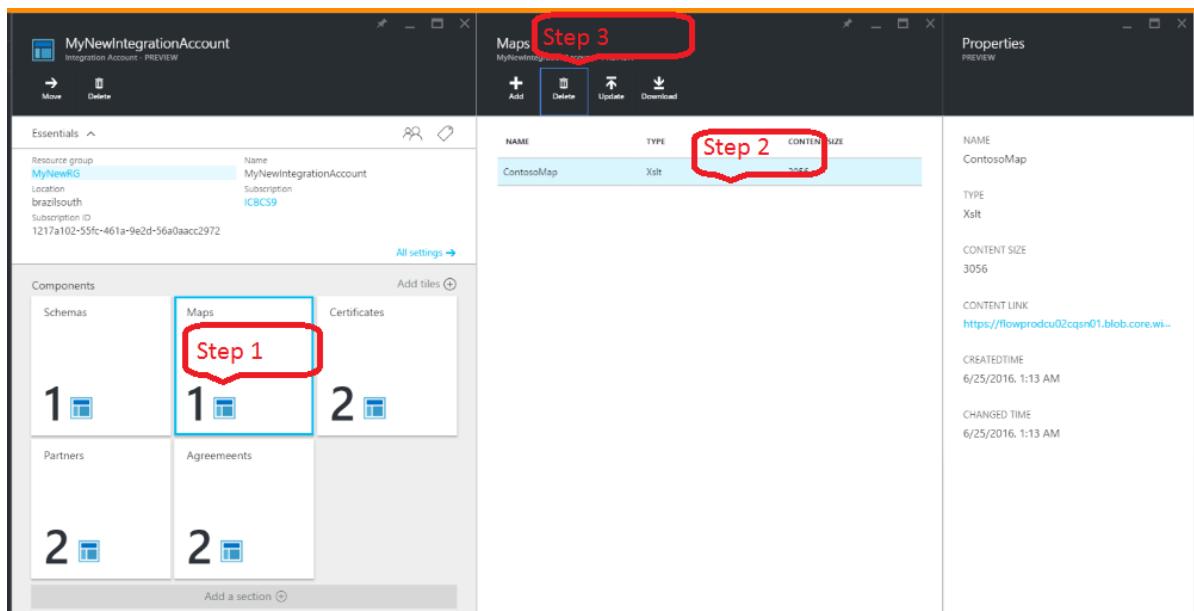
NAME	ContosoMap	CONTENT SIZE
TYPE	Xslt	3056

4. In the file picker, select the map file that you want to upload, then select **Open**.

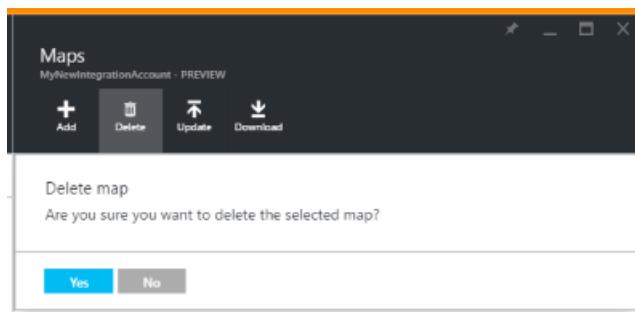


## How to delete a map?

1. Choose the **Maps** tile.
2. After the Maps blade opens, select the map you want to delete.
3. Choose **Delete**.



4. Confirm that you want to delete the map.



Next Steps

- [Learn more about the Enterprise Integration Pack](#)
- [Learn more about agreements](#)
- [Learn more about transforms](#)

# Learn about certificates and Enterprise Integration Pack

2/3/2017 • 2 min to read • [Edit Online](#)

## Overview

Enterprise integration uses certificates to secure B2B communications. You can use two types of certificates in your enterprise integration apps:

- Public certificates, which must be purchased from a certification authority (CA).
- Private certificates, which you can issue yourself. These certificates are sometimes referred to as self-signed certificates.

## What are certificates?

Certificates are digital documents that verify the identity of the participants in electronic communications and that also secure electronic communications.

## Why use certificates?

Sometimes B2B communications must be kept confidential. Enterprise integration uses certificates to secure these communications in two ways:

- By encrypting the contents of messages
- By digitally signing messages

## Upload a public certificate

To use a *public certificate* in your logic apps with B2B capabilities, you first need to upload the certificate into your integration account.

After you upload a certificate, it's available to help you secure your B2B messages when you define their properties in the [agreements](#) that you create.

Here are the detailed steps for uploading your public certificates into your integration account after you sign in to the Azure portal:

1. Select **More services** and enter **integration** in the filter search box. Select **Integration Accounts** from the results list

Preview Microsoft Azure Integration Accounts

New All resources Resource groups App Services SQL databases SQL data warehouses NoSQL (DocumentDB) Virtual machines Load balancers Storage accounts Virtual networks Azure Active Directory Monitor Azure Advisor Security Center Billing Help + support More services >

Shift+Space to toggle favorites

introduction

Integration Accounts

NAME	TYPE	RESOURCE GROUP	LOCATION
EIPIntegration	Integration Account	EIPTemplates	West US
FabrikamIntegrationAccount	Integration Account	as2tryit	West US
FabrikamIntegrationAccount	Integration Account	JonsIARG	Brazil South

2. Select the integration account to which you want to add the certificate.

Integration Accounts Microsoft

Add Columns Refresh

Subscriptions: 1 of 5 selected

Filter by name...

146 items

NAME	TYPE	RESOURCE GROUP	LOCATION
EIPIntegration	Integration Account	EIPTemplates	West US
FabrikamIntegrationAccount	Integration Account	as2tryit	West US
FabrikamIntegrationAccount	Integration Account	JonsIARG	Brazil South

3. Select the **Certificates** tile.

The screenshot shows the 'Overview' blade of the EIPIntegration Integration Account. On the left, there's a sidebar with 'SETTINGS' sections for 'Callback URL', 'Schemas', 'Maps', 'Certificates', 'Partners', and 'Agreements'. The 'Certificates' section is highlighted with a red box. The main area has a 'Components' section with five cards: 'Schemas' (11), 'Maps' (6), 'Certificates' (0, highlighted with a red box), 'Partners' (16), and 'Agreements' (3). At the top right, account details are shown: Name (EIPIntegration), Resource group (EIPTemplates), Location (West US), Subscription ID (redacted), and Subscription name (redacted).

4. In the **Certificates** blade that opens, select the **Add** button.

The screenshot shows the 'Certificates' blade for the EIPIntegration integration account. It has a header with a back arrow, a refresh icon, and a close icon. Below the header are three buttons: '+ Add' (highlighted with a red box), 'Edit', and 'Delete'. A table below the buttons has columns: NAME, TYPE, and CREATED TIME. A message at the top of the table says 'There are no certificate to display.'

5. Enter a **Name** for your certificate, and then select the certificate type as **public** from the dropdown.
6. Select the folder icon on the right side of the **Certificate** text box. When the file picker opens, find and select the certificate file that you want to upload to your integration account.
7. Select the certificate, and then select **OK** in the file picker. This validates and uploads the certificate to your integration account.
8. Finally, back on the **Add certificate** blade, select the **OK** button.

Add Certificate

\* Name  
publiccert

\* Certificate Type  
Public

\* Certificate  
aissit.cer

Resource Group

\* Key Vault

\* Key name  
Filter

**OK**

9. Select the **Certificates** tile. You should see the newly added certificate.

The screenshot shows the EIP Integration dashboard with two main sections: 'Essentials' and 'Components'.

**Essentials:** Displays resource group (EIPIntegration), location (West US), and subscription ID (redacted).

**Components:** A grid of tiles with the following counts:
 

- Schemas: 11
- Maps: 6
- Certificates:** 1 (this tile is highlighted with a red box)
- Partners: 16
- Agreements: 3

**Certificates Table:** A table titled 'Certificates EIPIntegration' showing one entry:
 

NAME	TYPE	CREATED TIME
publiccert	Public	1/24/2017 2:49 PM

# Upload a private certificate

To use a *private certificate* in your logic apps with B2B capabilities, You can upload a private certificate to your integration account by taking the following steps

1. Upload your private key to Key Vault and provide a **Key Name**

## TIP

You must authorize Logic Apps to perform operations on Key Vault. You can grant access to the Logic Apps service principal by using the following PowerShell command:

```
Set-AzureRmKeyVaultAccessPolicy -VaultName 'TestcertKeyVault' -ServicePrincipalName '7cd684f4-8a78-49b0-91ec-6a35d38739ba' -PermissionsToKeys decrypt, sign, get, list
```

After you've taken the previous step, add a private certificate to integration account.

Following are the detailed steps for uploading your private certificates into your integration account after you sign in to the Azure portal:

1. Select the integration account to which you want to add the certificate and select the **Certificates** tile.

The screenshot shows the 'Overview' tab selected in the left sidebar of an integration account named 'EIPIntegration'. The 'Certificates' tile in the main content area is highlighted with a red box. The tile displays a count of 0 certificates.

2. In the **Certificates** blade that opens, select the **Add** button.

The screenshot shows the 'Certificates' blade for the 'EIPIntegration' account. The 'Add' button in the top-left corner is highlighted with a red box. Below it, there are 'Edit' and 'Delete' buttons. The main area displays a message: 'There are no certificate to display.'

3. Enter a **Name** for your certificate, and select the certificate type as **private** from the dropdown.

4. select the folder icon on the right side of the **Certificate** text box. When the file picker opens, find the corresponding public certificate that you want to upload to your integration account.

## NOTE

While adding a private certificate it is important to add corresponding public certificate to show in [AS2 agreement](#) receive and send settings for signing and encrypting the messages.

5. Select the **Resource Group, Key Vault, Key Name** and select the **OK** button.

## Add Certificate

\* Name  
privatecert ✓

\* Certificate Type  
Private

Certificate  
aissit.cer   

aissit.cer

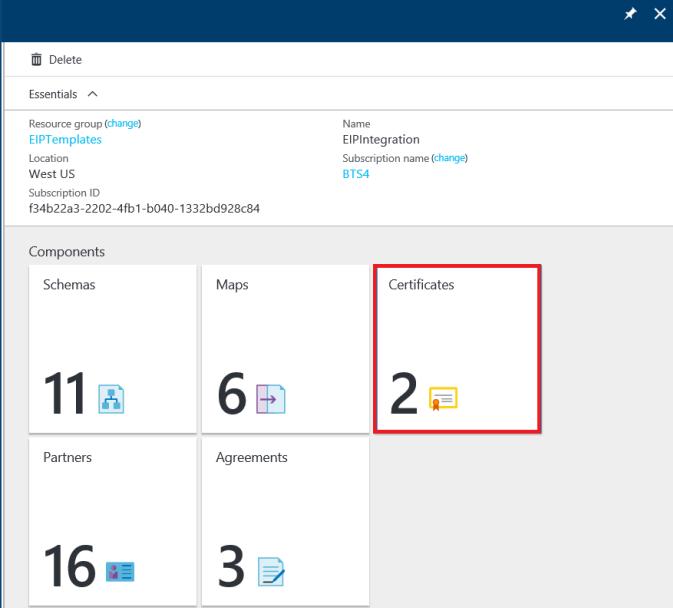
Resource Group  
EIPTemplates

\* Key Vault  
padmakv

\* Key name  
Filter

**OK**

6. Select the **Certificates** tile. You should see the newly added certificate.



NAME	TYPE	CREATED TIME
privatecert	Private	1/24/2017 3:16 PM
publiccert	Public	1/24/2017 2:49 PM

- [Create a B2B agreement](#)
- [Learn more about Key Vault](#)

# Manage artifact metadata in integration accounts for logic apps

3/1/2017 • 1 min to read • [Edit Online](#)

You can define custom metadata for artifacts in integration accounts and retrieve that metadata during runtime for your logic app. For example, you can specify metadata for artifacts like partners, agreements, schemas, and maps - all store metadata using key-value pairs. Currently, artifacts can't create metadata through UI, but you can use REST APIs to create metadata. To add metadata when you create or select a partner, agreement, or schema in the Azure portal, choose **Edit as JSON**. To retrieve artifact metadata in logic apps, you can use the Integration Account Artifact Lookup feature.

## Add metadata to artifacts in integration accounts

1. Create an [integration account](#).
2. Add an artifact to your integration account, for example, a [partner](#), [agreement](#), or [schema](#).
3. Select the artifact, choose **Edit as JSON**, and enter metadata details.

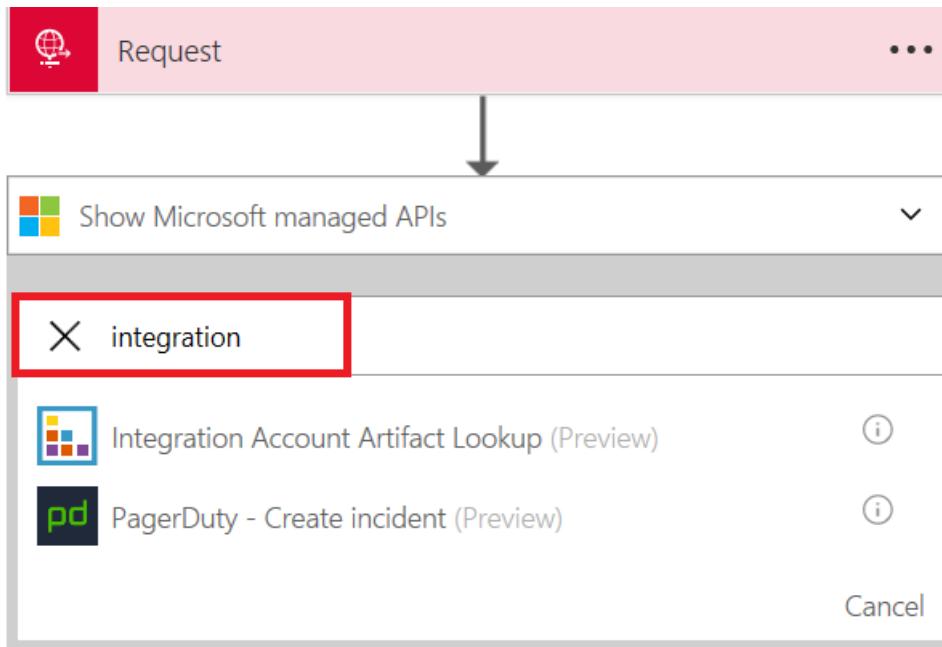
The screenshot shows the 'Edit as JSON' interface for an artifact named 'fabrikamtest1'. The JSON code is as follows:

```
1 {
  "properties": {
    "partnerType": "B2B",
    "content": {
      "b2b": {
        "businessIdentities": [
          {
            "qualifier": "AS2Identity",
            "value": "Fabrikamtest"
          }
        ]
      },
      "createdTime": "2016-11-08T17:39:50.5588065Z",
      "changedTime": "2016-11-18T18:19:27.1080365Z",
      "metadata": [
        {
          "name": "Padma",
          "SAPID": "1234567"
        }
      ],
      "id": "/subscriptions/[REDACTED]/resourceGroups/[REDACTED]/providers/Microsoft.Logic/integrationAccounts/[REDACTED]/partners/[REDACTED]",
      "name": "fabrikamtest1",
      "type": "Microsoft.Logic/integrationAccounts/partners"
    }
  }
}
```

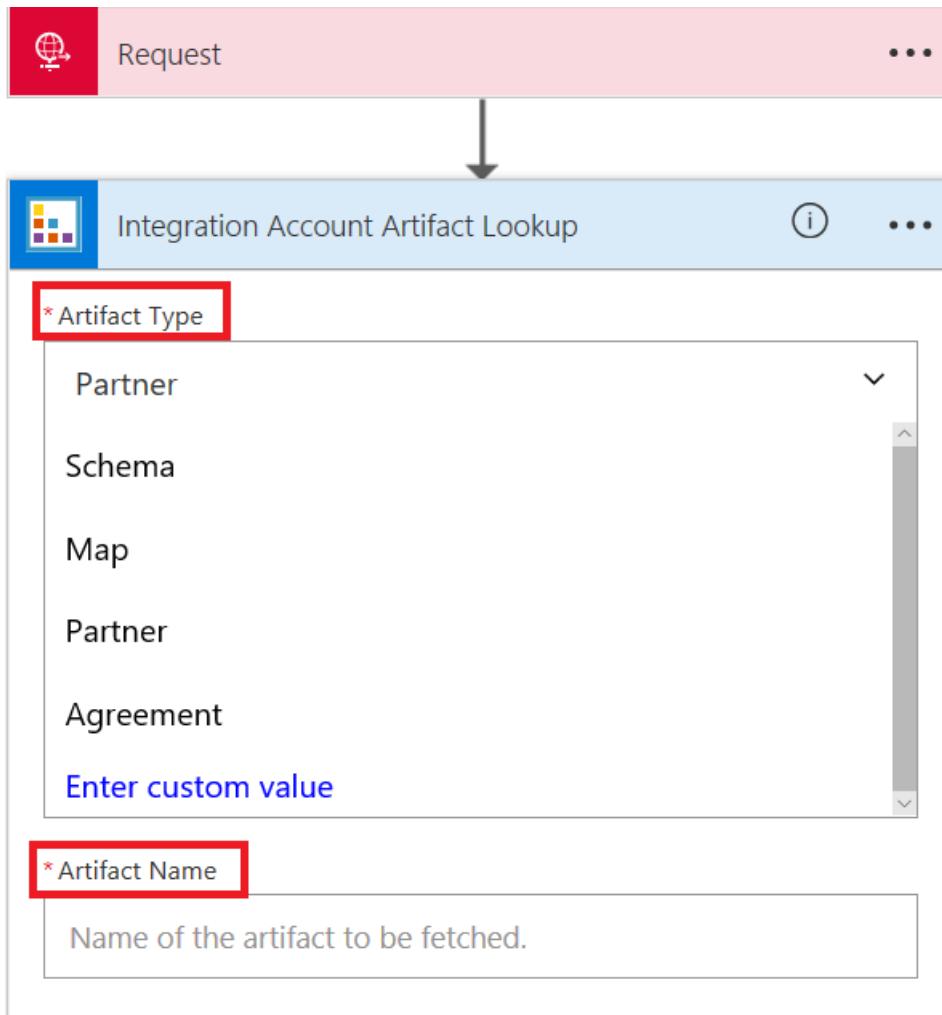
The 'Edit as JSON' button in the top navigation bar and the 'fabrikamtest1' row in the table are highlighted with red boxes.

## Retrieve metadata from artifacts for logic apps

1. Create a [logic app](#).
2. Create a [link from your logic app to your integration account](#).
3. In Logic App Designer, add a trigger like *Request* or *HTTP* to your logic app.
4. Choose **Next Step > Add an action**. Search for *integration* so you can find and then select **Integration Account - Integration Account Artifact Lookup**.



5. Select the **Artifact Type**, and provide the **Artifact Name**.



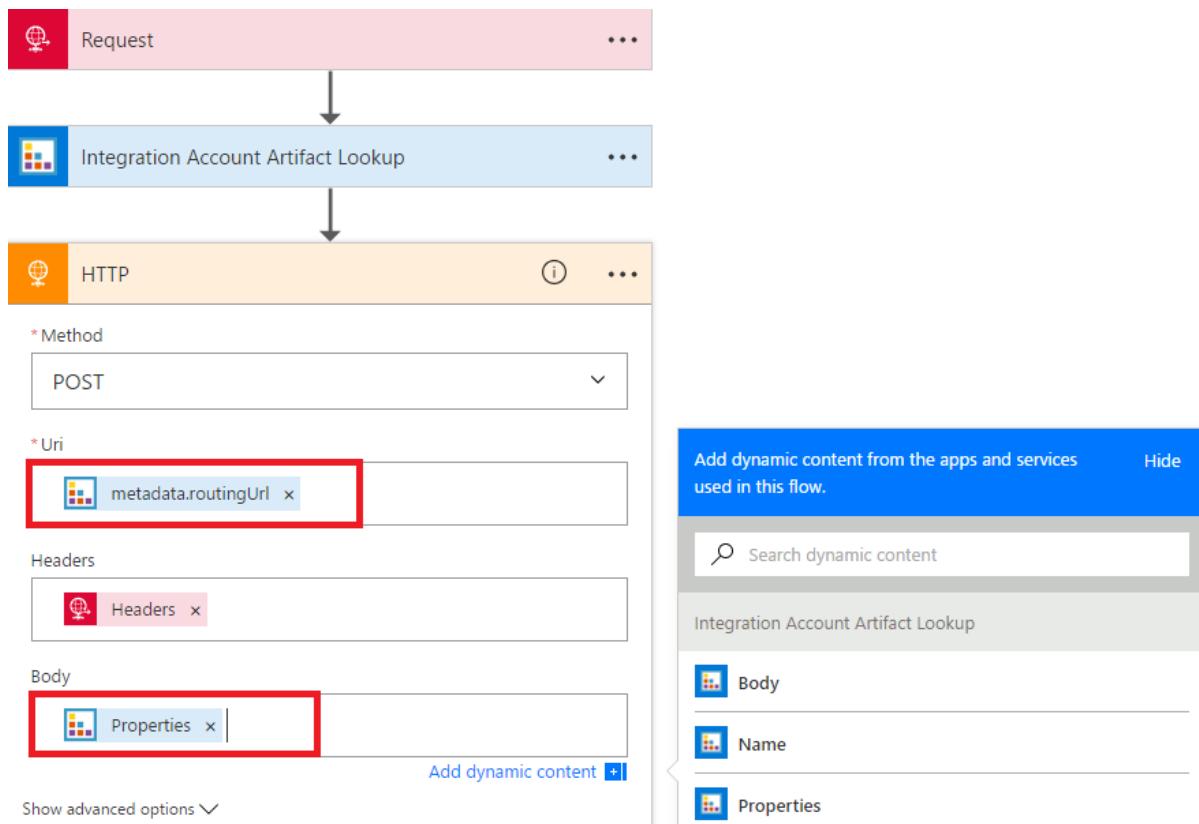
## Example: Retrieve partner metadata

Partner metadata has these `routingUrl` details:

## Edit as JSON

```
1 [
2     "properties": {
3         "partnerType": "B2B",
4         "content": {
5             "b2b": {
6                 "businessIdentities": [
7                     {
8                         "qualifier": "AS2Identity",
9                         "value": "Fabrikamtest"
10                    }
11                ]
12            }
13        },
14        "createdTime": "2016-11-08T17:39:50.5588065Z",
15        "changedTime": "2016-11-21T17:49:48.7499957Z",
16        "metadata": {
17            "name": "Padma",
18            "SAPID": "1234567",
19            "routingUrl": "http://[REDACTED]" [Redacted]
20        }
21    },
22    "id": "/subscriptions/[REDACTED]/resourceGroups/[REDACTED]",
23    "name": "fabrikamtest1",
24    "type": "Microsoft.Logic/integrationAccounts/partners"
25 }
```

1. In your logic app, add your trigger, an **Integration Account - Integration Account Artifact Lookup** action for your partner, and an **HTTP**.



- To retrieve the URI, go to Code View for your logic app. Your logic app definition should look like this example:

```

"actions": {
    "HTTP": {
        "inputs": {
            "body": "@outputs('Integration_Account_Artifact_Lookup')['properties']",
            "headers": "@triggerOutputs()['headers']",
            "method": "POST",
            "uri": "@{outputs('Integration_Account_Artifact_Lookup')['properties']['metadata']['routingUrl']}"
        },
        "runAfter": {
            "Integration_Account_Artifact_Lookup": [
                "Succeeded"
            ]
        }
    }
}

```

## Next steps

- Learn more about agreements

# Exchange AS2 messages for enterprise integration with logic apps

3/9/2017 • 6 min to read • [Edit Online](#)

Before you can exchange AS2 messages for Azure Logic Apps, you must create an AS2 agreement and store that agreement in your integration account. Here are the steps for how to create an AS2 agreement.

## Before you start

Here's the items you need:

- An [integration account](#) that's already defined and associated with your Azure subscription
- At least two [partners](#) that are already defined in your integration account and configured with the AS2 qualifier under **Business Identities**

### NOTE

When you create an agreement, the content in the agreement file must match the agreement type.

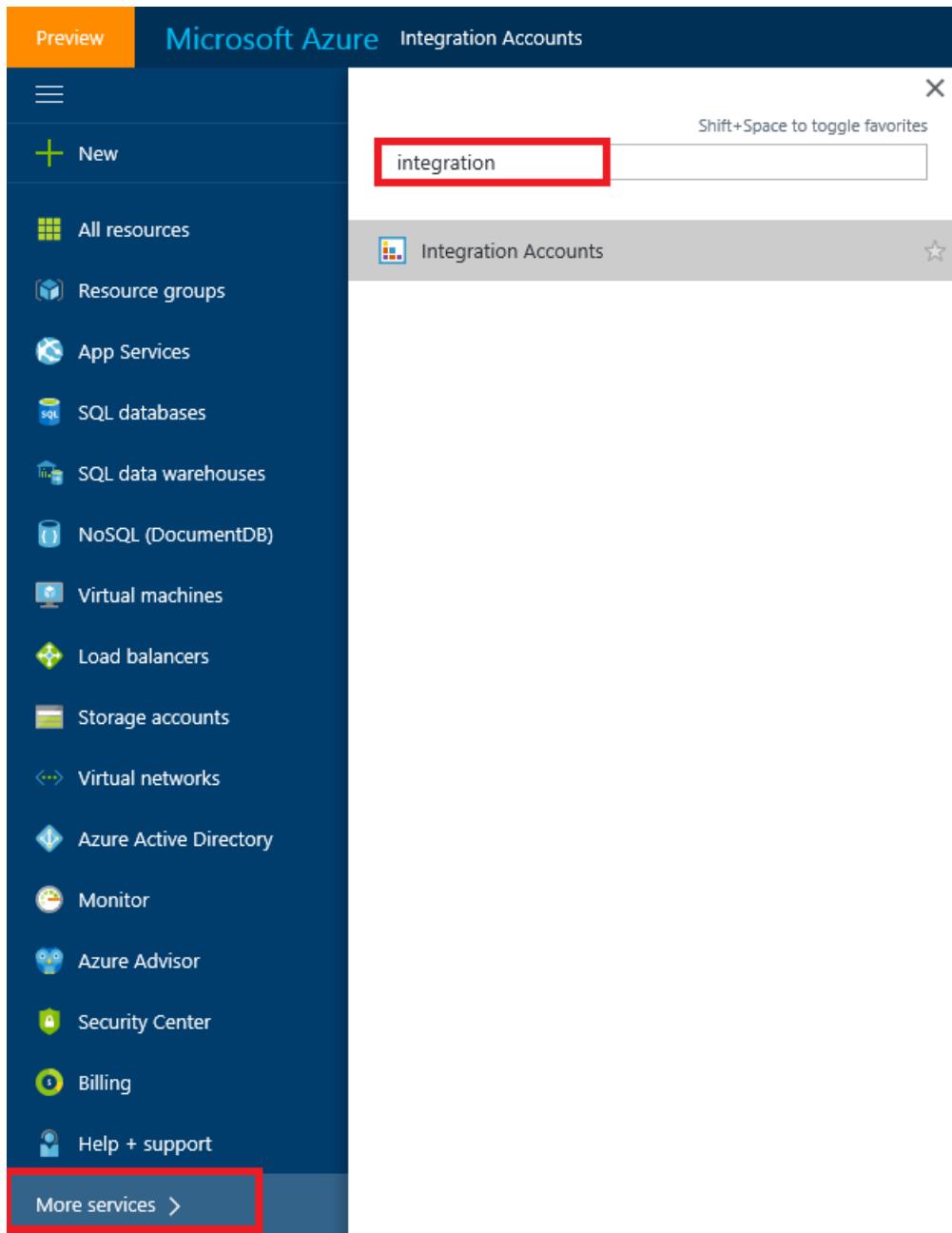
After you [create an integration account](#) and [add partners](#), you can create an AS2 agreement by following these steps.

## Create an AS2 agreement

1. Sign in to the [Azure portal](#).
2. From the left menu, select **More services**. In the search box, enter **integration** as your filter. In the results list, select **Integration Accounts**.

### TIP

If you don't see **More services**, you might have to expand the menu first. At the top of the collapsed menu, select **Show menu**.



3. In the **Integration Accounts** blade that opens, select the integration account where you want to create the agreement. If you don't see any integration accounts, [create one first](#).

The screenshot shows the 'Integration Accounts' blade in the Microsoft Azure portal. The title bar says 'Integration Accounts' and 'Microsoft'. The toolbar includes 'Add', 'Columns', and 'Refresh' buttons. A message 'Subscriptions: All 3 selected – Don't see a subscription? [Switch directories](#)' is displayed. Below is a search/filter section with 'Filter by name...' and 'All subscriptions' buttons. The main table shows 1 item: 'MyNewIntegrationAccount' (Integration Account) in 'MyNewRG' (Resource Group) located in 'Brazil South' (Location). The table has columns: NAME, TYPE, RESOURCE GROUP, and LOCATION.

NAME	TYPE	RESOURCE GROUP	LOCATION
MyNewIntegrationAccount	Integration Account	MyNewRG	Brazil South

4. Choose the **Agreements** tile. If you don't have an Agreements tile, add the tile first.

The screenshot shows the 'MyNewIntegrationAccount' blade in the Azure portal. On the left, there's a navigation menu with options like Overview, Access control (IAM), Tags, SETTINGS, Callback URL, Schemas, Maps, Certificates, Partners, and Agreements. The 'Agreements' option is selected and highlighted with a red box. The main area shows 'Essentials' information: Resource group (MyNewRG), Location (Brazil South), Subscription ID (redacted), and Name (MyNewIntegrationAccount). Below this is a 'Components' section with five categories: Schemas (11), Maps (6), Certificates (1), Partners (16), and Agreements (0, which is also highlighted with a red box).

5. In the Agreements blade that opens, choose **Add**.

The screenshot shows the 'MyNewIntegrationAccount - Agreements' blade. At the top, there are buttons for '+ Add', 'Edit', 'Edit as JSON', and 'Delete'. The main table below shows columns for NAME, TYPE, HOST PARTNER, and GUEST PARTNER. A message says 'There are no agreements to display.' To the right is the 'Add' form, which includes fields for Name (marked with a red asterisk), Agreement type (dropdown), Host Partner (dropdown), Host Identity (dropdown), Guest Partner (dropdown), and Guest Identity (dropdown). There are also 'Receive Settings' and 'Send Settings' sections with expand arrows.

6. Under **Add**, enter a **Name** for your agreement. For **Agreement type**, select **AS2**. Select the **Host Partner**, **Host Identity**, **Guest Partner**, and **Guest Identity** for your agreement.

Add

\* Name  
AS2agreement 

\* Agreement type  
AS2

\* Host Partner  
Contoso

\* Host Identity  
AS2Identity : Contoso

\* Guest Partner  
Fabrikam

\* Guest Identity  
AS2Identity : Fabrikam

---

Receive Settings >

---

Send Settings >

---

PROPERTY	DESCRIPTION
Name	Name of the agreement
Agreement Type	Should be AS2
Host Partner	An agreement needs both a host and guest partner. The host partner represents the organization that configures the agreement.
Host Identity	An identifier for the host partner
Guest Partner	An agreement needs both a host and guest partner. The guest partner represents the organization that's doing business with the host partner.
Guest Identity	An identifier for the guest partner

PROPERTY	DESCRIPTION
Receive Settings	These properties apply to all messages received by an agreement.
Send Settings	These properties apply to all messages sent by an agreement.

## Configure how your agreement handles received messages

Now that you've set the agreement properties, you can configure how this agreement identifies and handles incoming messages received from your partner through this agreement.

- Under **Add**, select **Receive Settings**. Configure these properties based on your agreement with the partner that exchanges messages with you. For property descriptions, see the table in this section.

The screenshot shows the 'Add' dialog with the 'Receive Settings' tab selected. On the left, there are two sections: 'Receive Settings' (highlighted with a red box) and 'Send Settings'. The 'Receive Settings' section contains fields for Name (AS2agreement), Agreement type (AS2), Host Partner (Contoso), Host Identity (AS2Identity : Contoso), Guest Partner (Fabrikam), and Guest Identity (AS2Identity : Fabrikam). On the right, the 'Receive Settings' configuration includes:

- Override message properties**: A checked checkbox.
- Message** section:
  - Certificate**: Set to publiccert.
  - Message should be encrypted**: A checked checkbox.
  - Certificate**: Set to privatecert.
  - Message should be compressed**: A checked checkbox.
- Acknowledgement** section:
  - MDN Text**: Contoso got your message.
  - Send MDN**: A checked checkbox.
  - Send signed MDN**: A checked checkbox.
  - MIC Algorithm**: Set to SHA2-256.
  - Send asynchronous MDN**: A checked checkbox.
  - URL**: Set to https://example.com/messages.

Both the 'Receive Settings' and 'Send Settings' sections have an 'OK' button at the bottom.

- Optionally, you can override the properties of incoming messages by selecting **Override message**.

## properties.

3. To require all incoming messages to be signed, select **Message should be signed**. From the **Certificate** list, select an existing [guest partner public certificate](#) for validating the signature on the messages. Or create the certificate, if you don't have one.
4. To require all incoming messages to be encrypted, select **Message should be encrypted**. From the **Certificate** list, select an existing [host partner private certificate](#) for decrypting incoming messages. Or create the certificate, if you don't have one.
5. To require messages to be compressed, select **Message should be compressed**.
6. To send a synchronous message disposition notification (MDN) for received messages, select **Send MDN**.
7. To send signed MDNs for received messages, select **Send signed MDN**.
8. To send asynchronous MDNs for received messages, select **Send asynchronous MDN**.
9. After you're done, make sure to save your settings by choosing **OK**.

Now your agreement is ready to handle incoming messages that conform to your selected settings.

PROPERTY	DESCRIPTION
Override message properties	Indicates that properties in received messages can be overridden.
Message should be signed	Requires messages to be digitally signed. Configure the guest partner public certificate for signature verification.
Message should be encrypted	Requires messages to be encrypted. Non-encrypted messages are rejected. Configure the host partner private certificate for decrypting the messages.
Message should be compressed	Requires messages to be compressed. Non-compressed messages are rejected.
MDN Text	The default message disposition notification (MDN) to be sent to the message sender.
Send MDN	Requires MDNs to be sent.
Send signed MDN	Requires MDNs to be signed.
MIC Algorithm	Select the algorithm to use for signing messages.
Send asynchronous MDN	Requires messages to be sent asynchronously.
URL	Specify the URL where to send the MDNs.

## Configure how your agreement sends messages

You can configure how this agreement identifies and handles outgoing messages that you send to your partners through this agreement.

1. Under **Add**, select **Send Settings**. Configure these properties based on your agreement with the partner that exchanges messages with you. For property descriptions, see the table in this section.

<h3>Add</h3> <p>* Name AS2agreement</p> <p>* Agreement type AS2</p> <p>* Host Partner Contoso</p> <p>* Host Identity AS2Identity : Contoso</p> <p>* Guest Partner Fabrikam</p> <p>* Guest Identity AS2Identity : Fabrikam</p> <p>Receive Settings &gt;</p> <p><b>Send Settings</b> &gt; <span style="border: 2px solid red; padding: 2px;">Send Settings</span></p>	<h3>Send Settings</h3> <p><b>Messages</b></p> <p><input checked="" type="checkbox"/> Enable message signing</p> <p>* MIC Algorithm SHA1</p> <p>* Certificate privatecert</p> <p><input checked="" type="checkbox"/> Enable message encryption</p> <p>* Encryption Algorithm DES3</p> <p>* Certificate</p> <p><input checked="" type="checkbox"/> Enable message compression</p> <p><input checked="" type="checkbox"/> Unfold HTTP headers</p> <p><b>Acknowledgement</b></p> <p><input checked="" type="checkbox"/> Request MDN</p> <p><input checked="" type="checkbox"/> Request signed MDN</p> <p><input checked="" type="checkbox"/> Request asynchronous MDN</p> <p>* URL https://example.com/messages</p> <p><b>NRR Status</b></p> <p><input checked="" type="checkbox"/> Enable NRR</p>
<input type="button" value="OK"/>	<input type="button" value="OK"/>

2. To send signed messages to your partner, select **Enable message signing**. For signing the messages, in the **MIC Algorithm** list, select the *host partner private certificate MIC Algorithm*. And in the **Certificate** list, select an existing *host partner private certificate*.
3. To send encrypted messages to the partner, select **Enable message encryption**. For encrypting the messages, in the **Encryption Algorithm** list, select the *guest partner public certificate algorithm*. And in the **Certificate** list, select an existing *guest partner public certificate*.
4. To compress the message, select **Enable message compression**.
5. To unfold the HTTP content-type header into a single line, select **Unfold HTTP headers**.
6. To receive synchronous MDNs for the sent messages, select **Request MDN**.
7. To receive signed MDNs for the sent messages, select **Request signed MDN**.
8. To receive asynchronous MDNs for the sent messages, select **Request asynchronous MDN**. If you select this option, enter the URL for where to send the MDNs.

9. To require non-repudiation of receipt, select **Enable NRR**.

10. After you're done, make sure to save your settings by choosing **OK**.

Now your agreement is ready to handle outgoing messages that conform to your selected settings.

PROPERTY	DESCRIPTION
Enable message signing	Requires all messages that are sent from the agreement to be signed.
MIC Algorithm	The algorithm to use for signing messages. Configures the host partner private certificate MIC Algorithm for signing the messages.
Certificate	Select the certificate to use for signing messages. Configures the host partner private certificate for signing the messages.
Enable message encryption	Requires encryption of all messages that are sent from this agreement. Configures the guest partner public certificate algorithm for encrypting the messages.
Encryption Algorithm	The encryption algorithm to use for message encryption. Configures the guest partner public certificate for encrypting the messages.
Certificate	The certificate to use to encrypt messages. Configures the guest partner private certificate for encrypting the messages.
Enable message compression	Requires compression of all messages that are sent from this agreement.
Unfold HTTP headers	Places the HTTP content-type header onto a single line.
Request MDN	Requires an MDN for all messages that are sent from this agreement.
Request signed MDN	Requires all MDNs that are sent to this agreement to be signed.
Request asynchronous MDN	Requires asynchronous MDNs to be sent to this agreement.
URL	Specify the URL where to send the MDNs.
Enable NRR	Requires non-repudiation of receipt (NRR), a communication attribute that provides evidence that the data was received as addressed.

## Find your created agreement

1. After you finish setting all your agreement properties, on the **Add** blade, choose **OK** to finish creating your agreement and return to your integration account blade.

Your newly added agreement now appears in your **Agreements** list.

2. You can also view your agreements in your integration account overview. On your integration account blade, choose **Overview**, then select the **Agreements** tile.

**Essentials**

Resource group ([change](#))  
EIPTemplates  
Location  
West US  
Subscription ID  
f34b22a3-2202-4fb1-b040-1332bd928c84

Name  
EIPIntegration  
Subscription name ([change](#))  
BTS4

**Components**

Schemas	Maps	Certificates
11	6	2
Partners	Agreements	
16	1	

**Agreements**

NAME	TYPE	HOST PARTNER	GUEST PARTNER
AS2agreement	AS2	Contoso	Fabrikam

## Next steps

- Learn more about the Enterprise Integration Pack

# Encode AS2 messages for Azure Logic Apps with the Enterprise Integration Pack

3/9/2017 • 1 min to read • [Edit Online](#)

To establish security and reliability while transmitting messages, use the Encode AS2 message connector. This connector provides digital signing, encryption, and acknowledgements through Message Disposition Notifications (MDN), which also leads to support for Non-Repudiation.

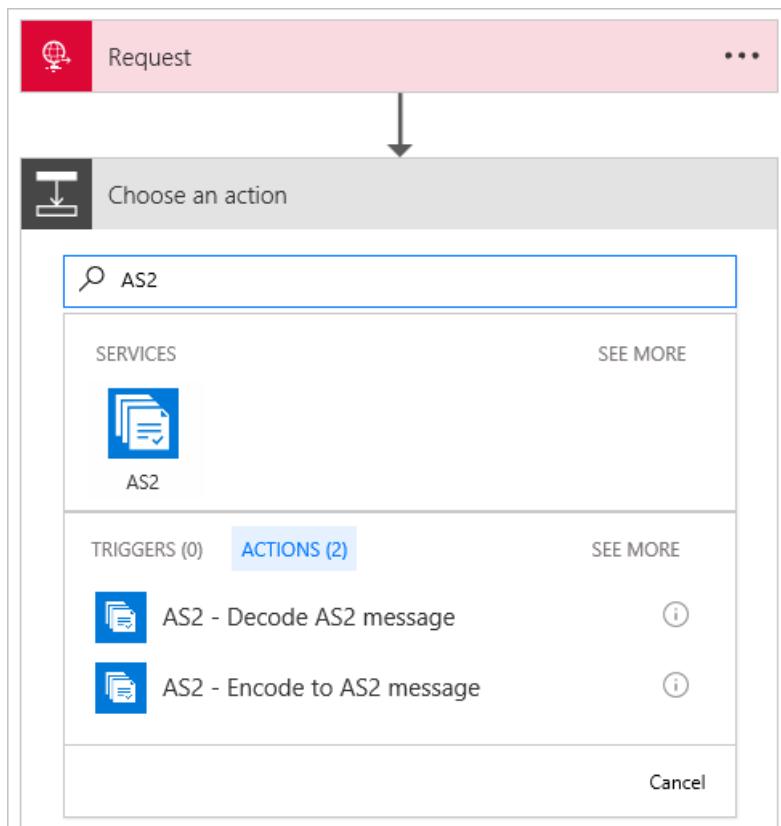
## Before you start

Here's the items you need:

- An Azure account; you can create a [free account](#)
- An [integration account](#) that's already defined and associated with your Azure subscription. You must have an integration account to use the Encode AS2 message connector.
- At least two [partners](#) that are already defined in your integration account
- An [AS2 agreement](#) that's already defined in your integration account

## Encode AS2 messages

1. [Create a logic app](#).
2. The Encode AS2 message connector doesn't have triggers, so you must add a trigger for starting your logic app, like a Request trigger. In the Logic App Designer, add a trigger, and then add an action to your logic app.
3. In the search box, enter "AS2" for your filter. Select **AS2 - Encode AS2 message**.



4. If you didn't previously create any connections to your integration account, you're prompted to create that connection now. Name your connection, and select the integration account that you want to connect.

AS2 - Encode to AS2 message

CONNECTION NAME\*

Enter name for connection

INTEGRATION ACCOUNT\*

EIPIntegration

Cancel Create

Properties with an asterisk are required.

PROPERTY	DETAILS
Connection Name *	Enter any name for your connection.
Integration Account *	Enter a name for your integration account. Make sure that your integration account and logic app are in the same Azure location.

5. When you're done, your connection details should look similar to this example. To finish creating your connection, choose **Create**.

AS2 - Encode to AS2 message

CONNECTION NAME\*

padmaconn

INTEGRATION ACCOUNT\*

EIPIntegration

Cancel Create

6. After your connection is created, as shown in this example, provide details for **AS2-From**, **AS2-To** identifiers as configured in your agreement, and **Body**, which is the message payload.

Encode to AS2 message

AS2-FROM\*

The AS2-from identifier.

AS2-TO\*

The AS2-to identifier.

BODY\*

Show advanced options

Connected to PadmaConn. Change connection.

## AS2 encoder details

The Encode AS2 connector performs these tasks:

- Applies AS2/HTTP headers
- Signs outgoing messages (if configured)
- Encrypts outgoing messages (if configured)
- Compresses the message (if configured)

## Try this sample

To try deploying a fully operational logic app and sample AS2 scenario, see the [AS2 logic app template and scenario](#).

## Next steps

[Learn more about the Enterprise Integration Pack](#)

# Decode AS2 messages for Azure Logic Apps with the Enterprise Integration Pack

3/9/2017 • 1 min to read • [Edit Online](#)

To establish security and reliability while transmitting messages, use the Decode AS2 message connector. This connector provides digital signing, decryption, and acknowledgements through Message Disposition Notifications (MDN).

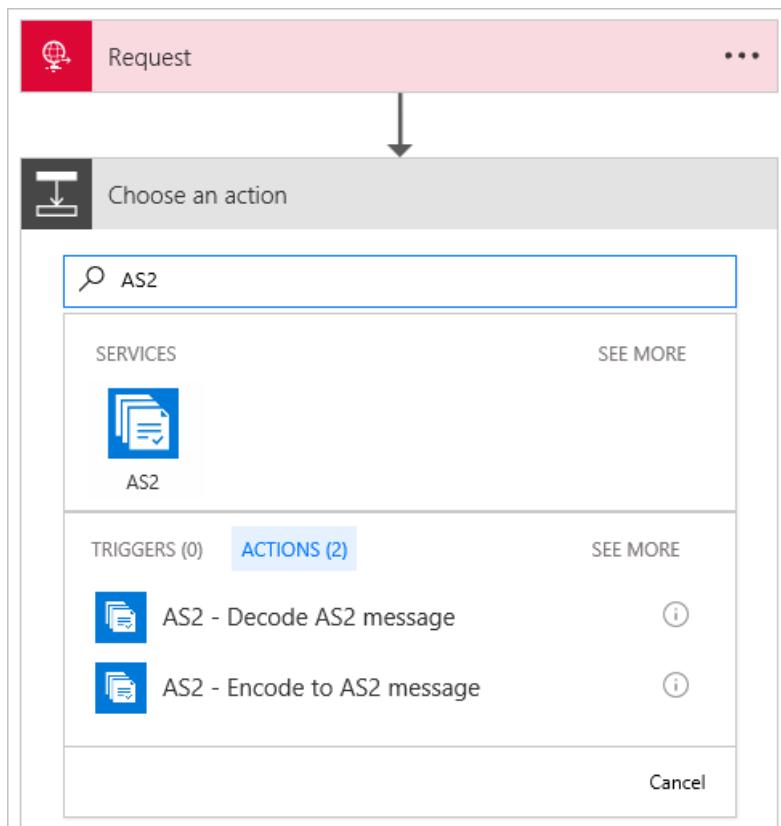
## Before you start

Here's the items you need:

- An Azure account; you can create a [free account](#)
- An [integration account](#) that's already defined and associated with your Azure subscription. You must have an integration account to use the Decode AS2 message connector.
- At least two [partners](#) that are already defined in your integration account
- An [AS2 agreement](#) that's already defined in your integration account

## Decode AS2 messages

1. [Create a logic app](#).
2. The Decode AS2 message connector doesn't have triggers, so you must add a trigger for starting your logic app, like a Request trigger. In the Logic App Designer, add a trigger, and then add an action to your logic app.
3. In the search box, enter "AS2" for your filter. Select **AS2 - Decode AS2 message**.



4. If you didn't previously create any connections to your integration account, you're prompted to create that connection now. Name your connection, and select the integration account that you want to connect.

AS2 - Decode AS2 message

CONNECTION NAME\*

Enter name for connection

INTEGRATION ACCOUNT\*

EIPIntegration

Cancel Create

Properties with an asterisk are required.

PROPERTY	DETAILS
Connection Name *	Enter any name for your connection.
Integration Account *	Enter a name for your integration account. Make sure that your integration account and logic app are in the same Azure location.

5. When you're done, your connection details should look similar to this example. To finish creating your connection, choose **Create**.

AS2 - Decode AS2 message

CONNECTION NAME\*

padmaconn

INTEGRATION ACCOUNT\*

EIPIntegration

Cancel Create

6. After your connection is created, as shown in this example, select **Body** and **Headers** from the Request outputs.

Decode AS2 message

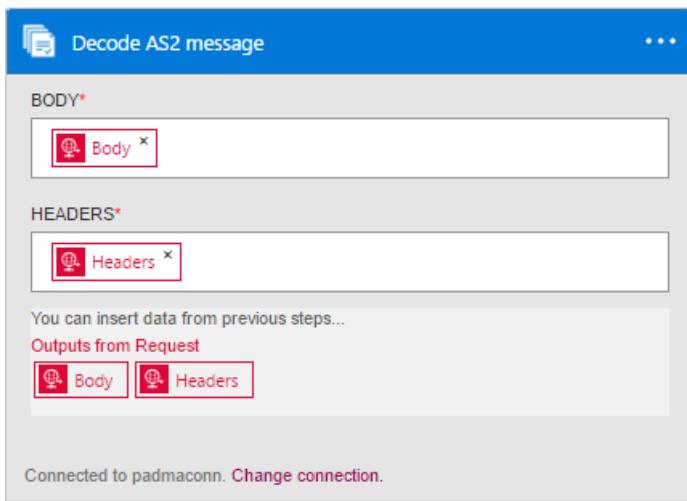
BODY\*

HEADERS\*

The AS2 request headers.

Connected to PadmaConn. Change connection.

For example:



## AS2 decoder details

The Decode AS2 connector performs these tasks:

- Processes AS2/HTTP headers
- Verifies the signature (if configured)
- Decrypts the messages (if configured)
- Decompresses the message (if configured)
- Reconciles a received MDN with the original outbound message
- Updates and correlates records in the non-repudiation database
- Writes records for AS2 status reporting
- The output payload contents are base64 encoded
- Determines whether an MDN is required, and whether the MDN should be synchronous or asynchronous based on configuration in AS2 agreement
- Generates a synchronous or asynchronous MDN (based on agreement configurations)
- Sets the correlation tokens and properties on the MDN

## Try this sample

To try deploying a fully operational logic app and sample AS2 scenario, see the [AS2 logic app template and scenario](#).

## Next steps

[Learn more about the Enterprise Integration Pack](#)

# Exchange EDIFACT messages for enterprise integration with logic apps

3/9/2017 • 12 min to read • [Edit Online](#)

Before you can exchange EDIFACT messages for Azure Logic Apps, you must create an EDIFACT agreement and store that agreement in your integration account. Here are the steps for how to create an EDIFACT agreement.

## NOTE

This page covers the EDIFACT features for Azure Logic Apps. For more information, see [X12](#).

## Before you start

Here's the items you need:

- An [integration account](#) that's already defined and associated with your Azure subscription
- At least two [partners](#) that are already defined in your integration account

## NOTE

When you create an agreement, the content in the messages that you receive or send to and from the partner must match the agreement type.

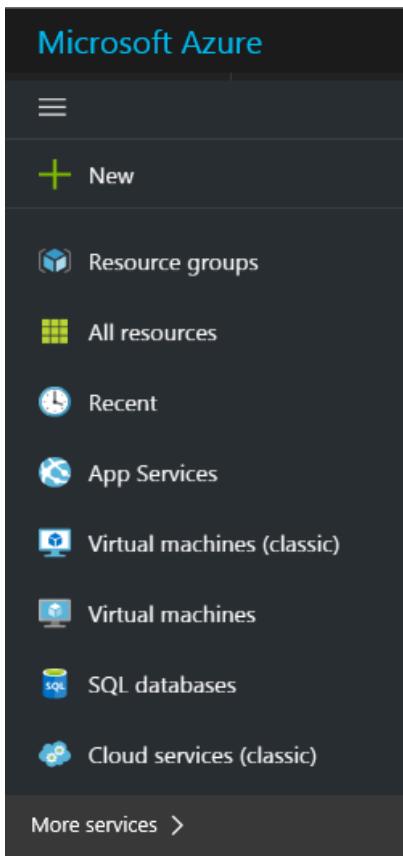
After you [create an integration account](#) and [add partners](#), you can create an EDIFACT agreement by following these steps.

## Create an EDIFACT agreement

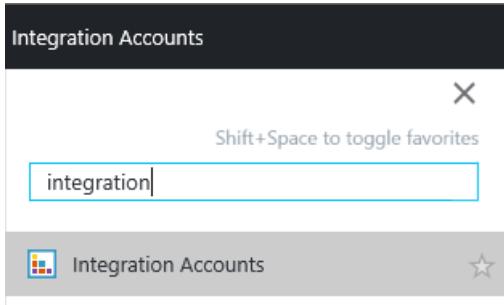
1. Sign in to the [Azure portal](#). From the left menu, select **More services**.

## TIP

If you don't see **More services**, you might have to expand the menu first. At the top of the collapsed menu, select **Show menu**.



2. In the search box, type "integration" for your filter. In the results list, select **Integration Accounts**.



3. In the **Integration Accounts** blade that opens, select the integration account where you want to create the agreement. If you don't see any integration accounts, [create one first](#).

The image shows the 'Integration Accounts' blade. At the top is a header with the title 'Integration Accounts' and the Microsoft logo. Below the header are buttons for 'Add', 'Columns', and 'Refresh'. A status message says 'Subscriptions: 1 of 3 selected'. There are two filter input fields: 'Filter items...' and 'ICBCS9'. The main area is a table with columns: NAME, TYPE, RESOURCE GROUP, LOCATION, and SUBSCRIPTION. It contains three rows of data:

NAME	TYPE	RESOURCE GROUP	LOCATION	SUBSCRIPTION
DeonheEIPNCentralIntegrationAccount	Integration Account	DeonheNCentralResGroyp	North Central US	ICBCS9
DeonheIntegrationAccount	Integration Account	DeonheResGroup	Brazil South	ICBCS9
MyNewIntegrationAccount	Integration Account	MyNewRG	Brazil South	ICBCS9

4. Choose the **Agreements** tile. If you don't have an Agreements tile, add the tile first.

Subscriptions: 1 of 3 selected  
Filter items...  
NAME  
ICBCS9

Essentials ^

Resource group: DeonheEIPNCentralResGroupt  
Name: DeonheEIPNCentralIntegrationAccount  
Location: North Central US  
Subscription ID: 1217a102-55fc-461a-9e2d-56a0acc2972  
All settings →

Components

Schemas	Maps	Certificates
2	1	1

Partners

Agreements	
2	1

Add a section ⓘ

NAME	TYPE	HOST PARTNER	GUEST PARTNER
AgreementWithNYTra...	X12	MyCompany	NYTradingPartner

5. In the Agreements blade that opens, choose **Add**.

Agreements  
MyNewIntegrationAccount - PREVIEW

Add Edit Delete

NAME	TYPE	HOST PARTNER	GUEST PARTNER
WarrantySalesAgreem...	AS2	ContosoCorp	BuyerCorp

6. Under **Add**, enter a **Name** for your agreement. For **Agreement type**, select **EDIFACT**. Select the **Host Partner**, **Host Identity**, **Guest Partner**, and **Guest Identity** for your agreement.

Add

\* Name  
WidgetAgreement ✓

\* Agreement type  
AS2  
AS2  
X12  
**EDIFACT**

\* Host Identity

\* Guest Partner

\* Guest Identity

Receive Settings >

Send Settings >

PROPERTY	DESCRIPTION
Name	Name of the agreement
Agreement Type	Should be EDIFACT
Host Partner	An agreement needs both a host and guest partner. The host partner represents the organization that configures the agreement.
Host Identity	An identifier for the host partner
Guest Partner	An agreement needs both a host and guest partner. The guest partner represents the organization that's doing business with the host partner.
Guest Identity	An identifier for the guest partner
Receive Settings	These properties apply to all messages received by an agreement.
Send Settings	These properties apply to all messages sent by an agreement.

## Configure how your agreement handles received messages

Now that you've set the agreement properties, you can configure how this agreement identifies and handles incoming messages received from your partner through this agreement.

- Under **Add**, select **Receive Settings**. Configure these properties based on your agreement with the partner that exchanges messages with you. For property descriptions, see the tables in this section.

**Receive Settings** is organized into these sections: Identifiers, Acknowledgment, Schemas, Control Numbers, Validation, and Internal Settings.

### Receive Settings

---

#### Identifiers

UNB6.1 (Recipient Reference Password)

UNB6.2 (Recipient Reference Qualifier)

#### Acknowledgement

Receipt of Message (CONTRL)  Acknowledgement (CONTRL)

#### Schemas

UNH2.1 (TYPE) UNH2.2 (VERSION) UNH2.3 (RELEASE) UNH2.5 (ASSOCIAT...) UNH2.1 (APP SEND...) UNH2.2 (APP SEND...) SCHEMA

No results

APERAK       No schema...  ...

#### Control Numbers

Disallow Interchange control number duplicates

Check for duplicate UNB5 every (days)

Disallow Group control number duplicates

Disallow Transaction set control number duplicates

EDIFACT Acknowledgement Control Number   to

#### Validations

MESSAGE TYPE	EDI VALIDATION	EXTENDED VALIDATION	ALLOW LEADING/TRAILING SPACES	TRIM LEADING/TRAILING SPACES	TRAILING SEPARATOR...
Default	true	false	false	false	NotAllowed <input type="button" value="..."/>
APERAK	<input type="button" value="▼"/> <input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Not Allowed <input type="button" value="..."/>

#### Internal Settings

Create empty XML tags if trailing separators are allowed

Inbound batch processing

Split Interchange as transaction sets - suspend transaction sets on error  
 Split Interchange as transaction sets - suspend interchange on error  
 Preserve Interchange - suspend transaction sets on error  
 Preserve Interchange - suspend interchange on error

- After you're done, make sure to save your settings by choosing **OK**.

Now your agreement is ready to handle incoming messages that conform to your selected settings.

## Identifiers

PROPERTY	DESCRIPTION
UNB6.1 (Recipient Reference Password)	Enter an alphanumeric value ranging between 1 and 14 characters.
UNB6.2 (Recipient Reference Qualifier)	Enter an alphanumeric value with a minimum of one character and a maximum of two characters.

## Acknowledgments

PROPERTY	DESCRIPTION
Receipt of Message (CTRL)	Select this checkbox to return a technical (CTRL) acknowledgment to the interchange sender. The acknowledgment is sent to the interchange sender based on the Send Settings for the agreement.
Acknowledgement (CTRL)	Select this checkbox to return a functional (CTRL) acknowledgment to the interchange sender. The acknowledgment is sent to the interchange sender based on the Send Settings for the agreement.

## Schemas

PROPERTY	DESCRIPTION
UNH2.1 (TYPE)	Select a transaction set type.
UNH2.2 (VERSION)	Enter the message version number. (Minimum, one character; maximum, three characters).
UNH2.3 (RELEASE)	Enter the message release number. (Minimum, one character; maximum, three characters).
UNH2.5 (ASSOCIATED ASSIGNED CODE)	Enter the assigned code. (Maximum, six characters. Must be alphanumeric).
UNG2.1 (APP SENDER ID)	Enter an alphanumeric value with a minimum of one character and a maximum of 35 characters.
UNG2.2 (APP SENDER CODE QUALIFIER)	Enter an alphanumeric value, with a maximum of four characters.
SCHEMA	Select the previously uploaded schema you want to use from your associated integration account.

## Control Numbers

PROPERTY	DESCRIPTION
Disallow Interchange Control Number duplicates	To block duplicate interchanges, select this property. If selected, the EDIFACT Decode Action checks that the interchange control number (UNB5) for the received interchange does not match a previously processed interchange control number. If a match is detected, then the interchange is not processed.

PROPERTY	DESCRIPTION
Check for duplicate UNB5 every (days)	If you chose to disallow duplicate interchange control numbers, you can specify the number of days when to perform the check by giving the appropriate value for this setting.
Disallow Group control number duplicates	To block interchanges with duplicate group control numbers (UNG5), select this property.
Disallow Transaction set control number duplicates	To block interchanges with duplicate transaction set control numbers (UNH1), select this property.
EDIFACT Acknowledgement Control Number	To designate the transaction set reference numbers for use in an acknowledgment, enter a value for the prefix, a range of reference numbers, and a suffix.

## Validations

When you complete each validation row, another is automatically added. If you don't specify any rules, then validation uses the "Default" row.

PROPERTY	DESCRIPTION
Message Type	Select the EDI message type.
EDI Validation	Perform EDI validation on data types as defined by the schema's EDI properties, length restrictions, empty data elements, and trailing separators.
Extended Validation	If the data type isn't EDI, validation is on the data element requirement and allowed repetition, enumerations, and data element length validation (min/max).
Allow Leading/Trailing Zeroes	Retain any additional leading or trailing zero and space characters. Don't remove these characters.
Trim Leading/Trailing Zeroes	Remove leading or trailing zero and space characters.
Trailing Separator Policy	Generate trailing separators. Select <b>Not Allowed</b> to prohibit trailing delimiters and separators in the received interchange. If the interchange has trailing delimiters and separators, the interchange is declared not valid. Select <b>Optional</b> to accept interchanges with or without trailing delimiters and separators. Select <b>Mandatory</b> when the received interchange must have trailing delimiters and separators.

## Internal Settings

PROPERTY	DESCRIPTION
Create empty XML tags if trailing separators are allowed	Select this check box to have the interchange sender include empty XML tags for trailing separators.

PROPERTY	DESCRIPTION
Split Interchange as transaction sets - suspend transaction sets on error	Parses each transaction set in an interchange into a separate XML document by applying the appropriate envelope to the transaction set. Suspend only the transaction sets that fail validation.
Split Interchange as transaction sets - suspend interchange on error	Parses each transaction set in an interchange into a separate XML document by applying the appropriate envelope. Suspend the entire interchange when one or more transaction sets in the interchange fail validation.
Preserve Interchange - suspend transaction sets on error	Leaves the interchange intact, creates an XML document for the entire batched interchange. Suspend only the transaction sets that fail validation, while continuing to process all other transaction sets.
Preserve Interchange - suspend interchange on error	Leaves the interchange intact, creates an XML document for the entire batched interchange. Suspend the entire interchange when one or more transaction sets in the interchange fail validation.

## Configure how your agreement sends messages

You can configure how this agreement identifies and handles outgoing messages that you send to your partners through this agreement.

- Under **Add**, select **Send Settings**. Configure these properties based on your agreement with your partner who exchanges messages with you. For property descriptions, see the tables in this section.

**Send Settings** is organized into these sections: Identifiers, Acknowledgment, Schemas, Envelopes, Character Sets and Separators, Control Numbers, and Validations.

### Send Settings

#### Identifiers

UNB1.2 (Syntax Version)

UNB2.3 (Sender Reverse Routing Address)

UNB3.3 (Recipient Reverse Routing Address)

UNB6.1 (Recipient Reference Password)

UNB6.2 (Recipient Reference Qualifier)

UNB7 (Application Reference ID)

#### Acknowledgement

Receipt of Message (CONTRL)

Acknowledgement (CONTRL)

Generate SG1/SG4 loop for accepted transaction sets

#### Schemas

UNH2.1 (TYPE)	UNH2.2 (VERSION)	UNH2.3 (RELEASE)	SCHEMA
---------------	------------------	------------------	--------

No results

APERAK



No schemas found



...

## Envelopes

UNB8 (Processing Priority Code)

UNB8 (Processing Priority Code)

UNB10 (Communication Agreement)

UNB10 (Communication Agreement)

UNB11 (Test Indicator)

Apply UNA Segment (Service String Advice)

Apply UNG Segments (Function Group Header)

## Character Sets and Separators

UNB1.1 (System Identifier)

UNOB



SCHEMA	INPUT TYPE	COMPONENT...	DATA ELEMENT...	UNA3 (DECIM...	UNA4 (RELEA...	UNA5 (REPETI...	SEGMENT TER...	SUFFIX
Default	:	+	Comma	?	*	'	None	...
	<input type="button"/> No sch...	<input type="button"/> Char	<input type="text"/>	<input type="text"/>	<input type="button"/> Decimal	<input type="text"/>	<input type="text"/>	<input type="text"/> None  ...

## Control Numbers

UNB5 (Interchange Control Number)

Prefix 1  to 999999999  Suffix

UNG5 (Group Control Number)

Prefix 1  to 999999999  Suffix

UNH1 (Message Header Reference Number)

Prefix 1  to 999999999  Suffix

## Validations

MESSAGE TYPE	EDI VALIDATION	EXTENDED VALIDATION	ALLOW LEADING/TRAII...	TRIM LEADING/TRAILL...	TRAILING SEPARATOR ...
Default	true	false	false	false	NotAllowed  ...
APERAK	<input type="button"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="button"/> Not Allowed  ...

2. After you're done, make sure to save your settings by choosing **OK**.

Now your agreement is ready to handle outgoing messages that conform to your selected settings.

## Identifiers

PROPERTY	DESCRIPTION
UNB1.2 (Syntax version)	Select a value between <b>1</b> and <b>4</b> .
UNB2.3 (Sender Reverse Routing Address)	Enter an alphanumeric value with a minimum of one character and a maximum of 14 characters.
UNB3.3 (Recipient Reverse Routing Address)	Enter an alphanumeric value with a minimum of one character and a maximum of 14 characters.
UNB6.1 (Recipient Reference Password)	Enter an alphanumeric value with a minimum of one and a maximum of 14 characters.
UNB6.2 (Recipient Reference Qualifier)	Enter an alphanumeric value with a minimum of one character and a maximum of two characters.

PROPERTY	DESCRIPTION
UNB7 (Application Reference ID)	Enter an alphanumeric value with a minimum of one character and a maximum of 14 characters

## Acknowledgment

PROPERTY	DESCRIPTION
Receipt of Message (CONTRL)	Select this checkbox if the hosted partner expects to receive a technical (CONTRL) acknowledgment. This setting specifies that the hosted partner, who is sending the message, requests an acknowledgement from the guest partner.
Acknowledgement (CONTRL)	Select this checkbox if the hosted partner expects to receive a functional (CONTRL) acknowledgment. This setting specifies that the hosted partner, who is sending the message, requests an acknowledgement from the guest partner.
Generate SG1/SG4 loop for accepted transaction sets	If you chose to request a functional acknowledgement, select this checkbox to force generation of SG1/SG4 loops in functional CONTRL acknowledgments for accepted transaction sets.

## Schemas

PROPERTY	DESCRIPTION
UNH2.1 (TYPE)	Select a transaction set type.
UNH2.2 (VERSION)	Enter the message version number.
UNH2.3 (RELEASE)	Enter the message release number.
SCHEMA	Select the schema to use. Schemas are located in your integration account. To access your schemas, first link your integration account to your Logic app.

## Envelopes

PROPERTY	DESCRIPTION
UNB8 (Processing Priority Code)	Enter an alphabetical value that is not more than one character long.
UNB10 (Communication Agreement)	Enter an alphanumeric value with a minimum of one character and a maximum of 40 characters.
UNB11 (Test Indicator)	Select this checkbox to indicate that the interchange generated is test data
Apply UNA Segment (Service String Advice)	Select this checkbox to generate a UNA segment for the interchange to be sent.

PROPERTY	DESCRIPTION
Apply UNG Segments (Function Group Header)	<p>Select this checkbox to create grouping segments in the functional group header in the messages sent to the guest partner. The following values are used to create the UNG segments:</p> <p>For <b>UNG1</b>, enter an alphanumeric value with a minimum of one character and a maximum of six characters.</p> <p>For <b>UNG2.1</b>, enter an alphanumeric value with a minimum of one character and a maximum of 35 characters.</p> <p>For <b>UNG2.2</b>, enter an alphanumeric value, with a maximum of four characters.</p> <p>For <b>UNG3.1</b>, enter an alphanumeric value with a minimum of one character and a maximum of 35 characters.</p> <p>For <b>UNG3.2</b>, enter an alphanumeric value, with a maximum of four characters.</p> <p>For <b>UNG6</b>, enter an alphanumeric value with a minimum of one and a maximum of three characters.</p> <p>For <b>UNG7.1</b>, enter an alphanumeric value with a minimum of one character and a maximum of three characters.</p> <p>For <b>UNG7.2</b>, enter an alphanumeric value with a minimum of one character and a maximum of three characters.</p> <p>For <b>UNG7.3</b>, enter an alphanumeric value with a minimum of 1 character and a maximum of 6 characters.</p> <p>For <b>UNG8</b>, enter an alphanumeric value with a minimum of one character and a maximum of 14 characters.</p>

## Character Sets and Separators

Other than the character set, you can enter a different set of delimiters to be used for each message type. If a character set is not specified for a given message schema, then the default character set is used.

PROPERTY	DESCRIPTION
UNB1.1 (System Identifier)	Select the EDIFACT character set to be applied on the outgoing interchange.
Schema	Select a schema from the drop-down list. After you complete each row, a new row is automatically added. For the selected schema, select the separators set that you want to use, based on the following separator descriptions.
Input Type	Select an input type from the drop-down list.
Component Separator	To separate composite data elements, enter a single character.
Data Element Separator	To separate simple data elements within composite data elements, enter a single character.

PROPERTY	DESCRIPTION
Segment Terminator	To indicate the end of an EDI segment, enter a single character.
Suffix	Select the character that is used with the segment identifier. If you designate a suffix, then the segment terminator data element can be empty. If the segment terminator is left empty, then you must designate a suffix.

## Control Numbers

PROPERTY	DESCRIPTION
UNB5 (Interchange Control Number)	Enter a prefix, a range of values for the interchange control number, and a suffix. These values are used to generate an outgoing interchange. The prefix and suffix are optional, while the control number is required. The control number is incremented for each new message; the prefix and suffix remain the same.
UNG5 (Group Control Number)	Enter a prefix, a range of values for the interchange control number, and a suffix. These values are used to generate the group control number. The prefix and suffix are optional, while the control number is required. The control number is incremented for each new message until the maximum value is reached; the prefix and suffix remain the same.
UNH1 (Message Header Reference Number)	Enter a prefix, a range of values for the interchange control number, and a suffix. These values are used to generate the message header reference number. The prefix and suffix are optional, while the reference number is required. The reference number is incremented for each new message; the prefix and suffix remain the same.

## Validations

When you complete each validation row, another is automatically added. If you don't specify any rules, then validation uses the "Default" row.

PROPERTY	DESCRIPTION
Message Type	Select the EDI message type.
EDI Validation	Perform EDI validation on data types as defined by the EDI properties of the schema, length restrictions, empty data elements, and trailing separators.
Extended Validation	If the data type isn't EDI, validation is on the data element requirement and allowed repetition, enumerations, and data element length validation (min/max).
Allow Leading/Trailing Zeros	Retain any additional leading or trailing zero and space characters. Don't remove these characters.
Trim Leading/Trailing Zeros	Remove leading or trailing zero characters.

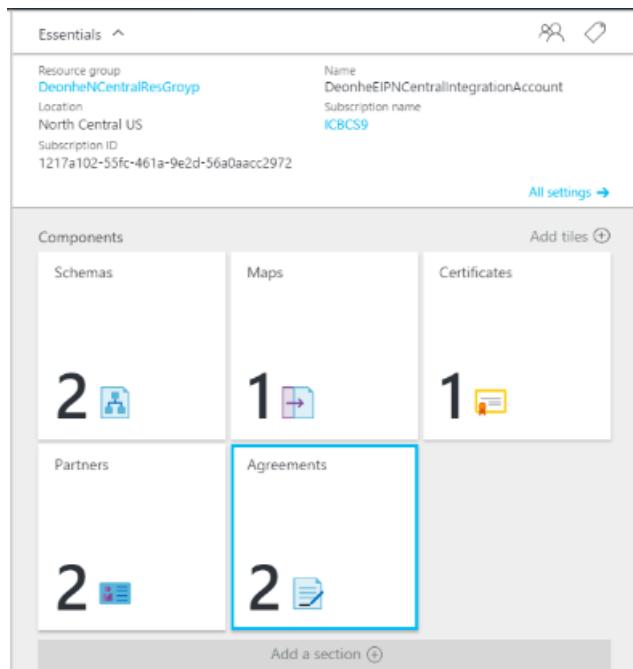
PROPERTY	DESCRIPTION
Trailing Separator Policy	<p>Generate trailing separators.</p> <p>Select <b>Not Allowed</b> to prohibit trailing delimiters and separators in the sent interchange. If the interchange has trailing delimiters and separators, the interchange is declared not valid.</p> <p>Select <b>Optional</b> to send interchanges with or without trailing delimiters and separators.</p> <p>Select <b>Mandatory</b> if the sent interchange must have trailing delimiters and separators.</p>

## Find your created agreement

- After you finish setting all your agreement properties, on the **Add** blade, choose **OK** to finish creating your agreement and return to your integration account blade.

Your newly added agreement now appears in your **Agreements** list.

- You can also view your agreements in your integration account overview. On your integration account blade, choose **Overview**, then select the **Agreements** tile.



## Learn more

- [Learn more about the Enterprise Integration Pack](#)

# Encode EDIFACT messages for Azure Logic Apps with the Enterprise Integration Pack

3/9/2017 • 2 min to read • [Edit Online](#)

With the Encode EDIFACT message connector, you can validate EDI and partner-specific properties, generate an XML document for each transaction set, and request a Technical Acknowledgement, Functional Acknowledgment, or both. To use this connector, you must add the connector to an existing trigger in your logic app.

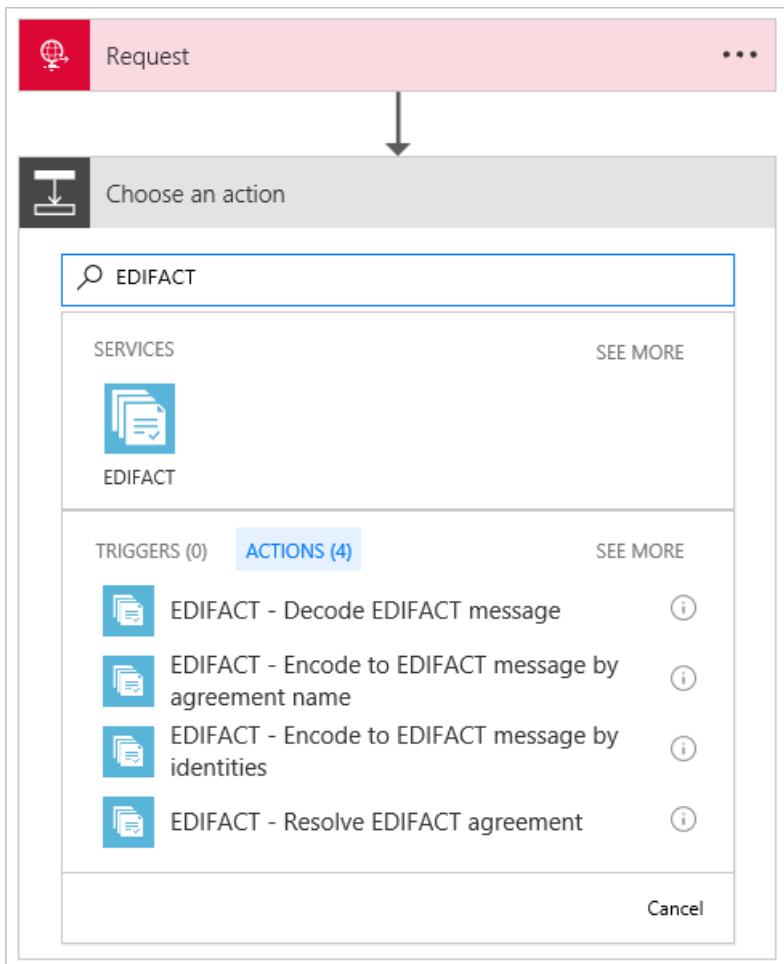
## Before you start

Here's the items you need:

- An Azure account; you can create a [free account](#)
- An [integration account](#) that's already defined and associated with your Azure subscription. You must have an integration account to use the Encode EDIFACT message connector.
- At least two [partners](#) that are already defined in your integration account
- An [EDIFACT agreement](#) that's already defined in your integration account

## Encode EDIFACT messages

1. [Create a logic app.](#)
2. The Encode EDIFACT message connector doesn't have triggers, so you must add a trigger for starting your logic app, like a Request trigger. In the Logic App Designer, add a trigger, and then add an action to your logic app.
3. In the search box, enter "EDIFACT" as your filter. Select either **Encode EDIFACT Message by agreement name** or **Encode to EDIFACT message by identities**.



4. If you didn't previously create any connections to your integration account, you're prompted to create that connection now. Name your connection, and select the integration account that you want to connect.

Properties with an asterisk are required.

PROPERTY	DETAILS
Connection Name *	Enter any name for your connection.
Integration Account *	Enter a name for your integration account. Make sure that your integration account and logic app are in the same Azure location.

5. When you're done, your connection details should look similar to this example. To finish creating your connection, choose **Create**.

 EDIFACT - Encode to EDIFACT message by agreement name ... ...

CONNECTION NAME\*

INTEGRATION ACCOUNT\*

Cancel Create

Your connection is now created.

 Encode to EDIFACT message by agreement name ...

NAME OF EDIFACT AGREEMENT\*

XML MESSAGE TO ENCODE\*

Connected to padint. [Change connection.](#)

#### Encode EDIFACT Message by agreement name

If you chose to encode EDIFACT messages by agreement name, open the **Name of X12 agreement** list, enter or select your EDIFACT agreement name. Enter the XML message to encode.

 Encode to EDIFACT message by agreement name ...

NAME OF EDIFACT AGREEMENT\*

XML MESSAGE TO ENCODE\*

Connected to padint. [Change connection.](#)

#### Encode EDIFACT Message by identities

If you choose to encode EDIFACT messages by identities, enter the sender identifier, sender qualifier, receiver identifier, and receiver qualifier as configured in your EDIFACT agreement. Select the XML message to encode.

 Encode to EDIFACT message by identities ...

SENDER IDENTIFIER\*

SENDER QUALIFIER\*

RECEIVER IDENTIFIER\*

RECEIVER QUALIFIER\*

XML MESSAGE TO ENCODE\*

Connected to padint. [Change connection.](#)

## EDIFACT Encode details

The Encode EDIFACT connector performs these tasks:

- Resolve the agreement by matching the sender qualifier & identifier and receiver qualifier and identifier
- Serializes the EDI interchange, converting XML-encoded messages into EDI transaction sets in the interchange.
- Applies transaction set header and trailer segments
- Generates an interchange control number, a group control number, and a transaction set control number for each outgoing interchange
- Replaces separators in the payload data
- Validates EDI and partner-specific properties
  - Schema validation of the transaction-set data elements against the message schema.
  - EDI validation performed on transaction-set data elements.
  - Extended validation performed on transaction-set data elements
- Generates an XML document for each transaction set.
- Requests a Technical and/or Functional acknowledgment (if configured).
  - As a technical acknowledgment, the CONTRL message indicates receipt of an interchange.
  - As a functional acknowledgment, the CONTRL message indicates acceptance or rejection of the received interchange, group, or message, with a list of errors or unsupported functionality

## Next steps

[Learn more about the Enterprise Integration Pack](#)

# Decode EDIFACT messages for Azure Logic Apps with the Enterprise Integration Pack

3/9/2017 • 3 min to read • [Edit Online](#)

With the Decode EDIFACT message connector, you can validate EDI and partner-specific properties, generate an XML document for each transaction set, and generate acknowledgment for processed transactions. To use this connector, you must add the connector to an existing trigger in your logic app.

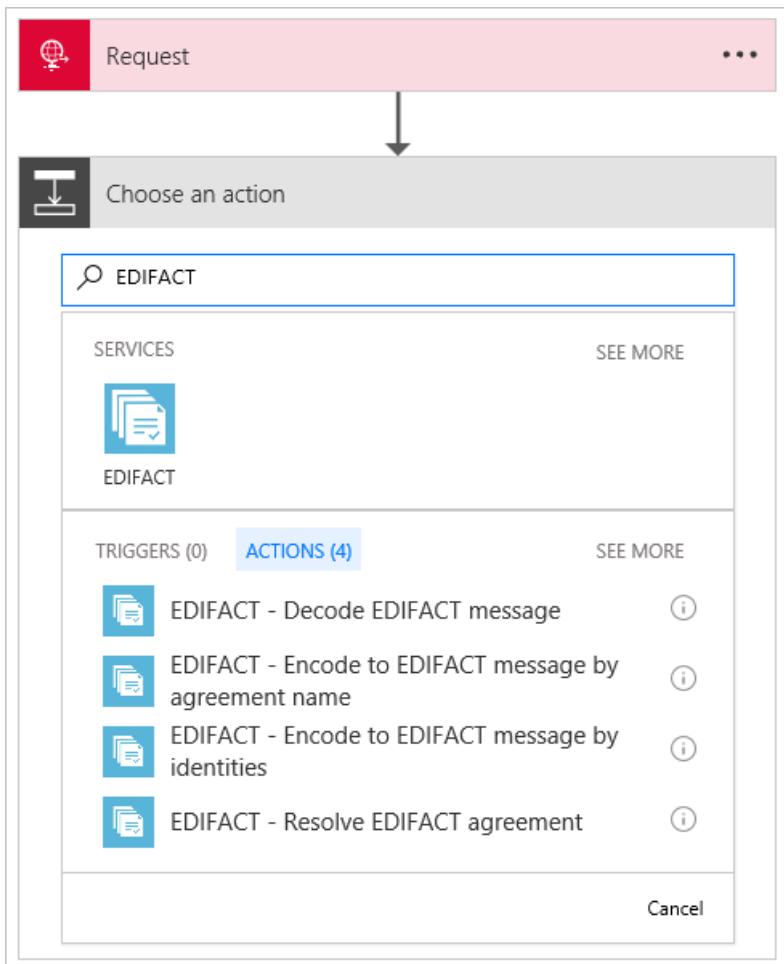
## Before you start

Here's the items you need:

- An Azure account; you can create a [free account](#)
- An [integration account](#) that's already defined and associated with your Azure subscription. You must have an integration account to use the Decode EDIFACT message connector.
- At least two [partners](#) that are already defined in your integration account
- An [EDIFACT agreement](#) that's already defined in your integration account

## Decode EDIFACT messages

1. [Create a logic app.](#)
2. The Decode EDIFACT message connector doesn't have triggers, so you must add a trigger for starting your logic app, like a Request trigger. In the Logic App Designer, add a trigger, and then add an action to your logic app.
3. In the search box, enter "EDIFACT" as your filter. Select **Decode EDIFACT Message**.



4. If you didn't previously create any connections to your integration account, you're prompted to create that connection now. Name your connection, and select the integration account that you want to connect.

Properties with an asterisk are required.

PROPERTY	DETAILS
Connection Name *	Enter any name for your connection.
Integration Account *	Enter a name for your integration account. Make sure that your integration account and logic app are in the same Azure location.

5. When you're done to finish creating your connection, choose **Create**. Your connection details should look similar to this example:

EDIFACT - Decode EDIFACT message

CONNECTION NAME\*

padmaconn

INTEGRATION ACCOUNT\*

EIPIntegration

Create

6. After your connection is created, as shown in this example, select the EDIFACT flat file message to decode.

Decode EDIFACT message

EDIFACT FLAT FILE MESSAGE TO DECODE\*

EDIFACT flat file message to decode

Connected to padmaconn. Change connection.

For example:

Decode EDIFACT message

EDIFACT FLAT FILE MESSAGE TO DECODE\*

Body

You can insert data from previous steps...

Outputs from Request

Body

Connected to padmaconn. Change connection.

## EDIFACT decoder details

The Decode EDIFACT connector performs these tasks:

- Resolve the agreement by matching the sender qualifier & identifier and receiver qualifier & identifier
- Splits multiple interchanges in a single message into separate.
- Validates the envelope against trading partner agreement
- Disassembles the interchange.
- Validates EDI and partner-specific properties includes
  - Validation of the structure of the interchange envelope.
  - Schema validation of the envelope against the control schema.
  - Schema validation of the transaction-set data elements against the message schema.
  - EDI validation performed on transaction-set data elements
- Verifies that the interchange, group, and transaction set control numbers are not duplicates (if configured)
  - Checks the interchange control number against previously received interchanges.
  - Checks the group control number against other group control numbers in the interchange.
  - Checks the transaction set control number against other transaction set control numbers in that group.
- Generates an XML document for each transaction set.

- Converts the entire interchange to XML
  - Split Interchange as transaction sets - suspend transaction sets on error: Parses each transaction set in an interchange into a separate XML document. If one or more transaction sets in the interchange fail validation, then EDIFACT Decode suspends only those transaction sets.
  - Split Interchange as transaction sets - suspend interchange on error: Parses each transaction set in an interchange into a separate XML document. If one or more transaction sets in the interchange fail validation, then EDIFACT Decode suspends the entire interchange.
  - Preserve Interchange - suspend transaction sets on error: Creates an XML document for the entire batched interchange. EDIFACT Decode suspends only those transaction sets that fail validation, while continuing to process all other transaction sets
  - Preserve Interchange - suspend interchange on error: Creates an XML document for the entire batched interchange. If one or more transaction sets in the interchange fail validation, then EDIFACT Decode suspends the entire interchange,
- Generates a Technical (control) and/or Functional acknowledgment (if configured).
  - A Technical Acknowledgment or the CONTRL ACK reports the results of a syntactical check of the complete received interchange.
  - A functional acknowledgment acknowledges accept or reject a received interchange or a group

## Next steps

[Learn more about the Enterprise Integration Pack](#)

# Exchange X12 messages for enterprise integration with logic apps

3/9/2017 • 12 min to read • [Edit Online](#)

Before you can exchange X12 messages for Azure Logic Apps, you must create an X12 agreement and store that agreement in your integration account. Here are the steps for how to create an X12 agreement.

## NOTE

This page covers the X12 features for Azure Logic Apps. For more information, see [EDIFACT](#).

## Before you start

Here's the items you need:

- An [integration account](#) that's already defined and associated with your Azure subscription
- At least two [partners](#) that are defined in your integration account and configured with the X12 identifier under **Business Identities**
- A required [schema](#) for uploading to your [integration account](#)

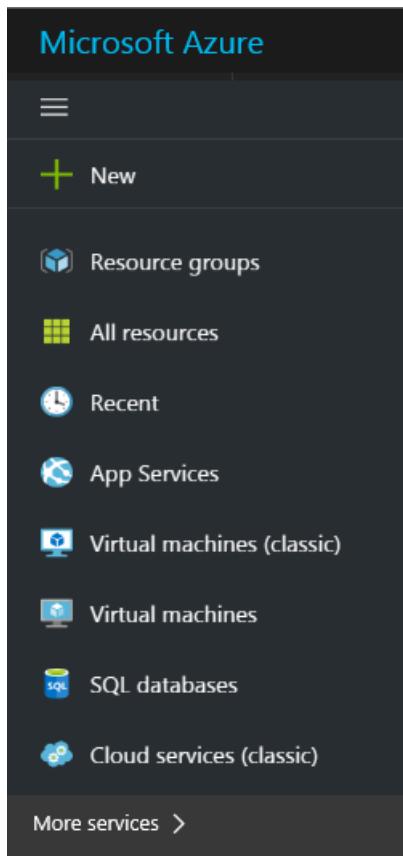
After you [create an integration account](#), [add partners](#), and have a [schema](#) that you want to use, you can create an X12 agreement by following these steps.

## Create an X12 agreement

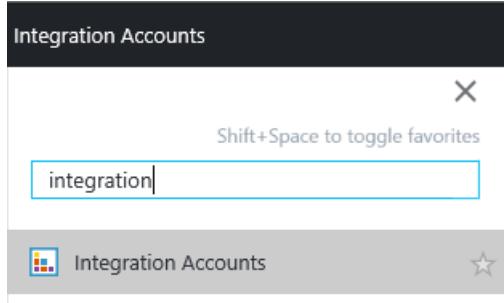
1. Sign in to the [Azure portal](#). From the left menu, select **More services**.

## TIP

If you don't see **More services**, you might have to expand the menu first. At the top of the collapsed menu, select **Show menu**.



2. In the search box, type "integration" as your filter. In the results list, select **Integration Accounts**.



3. In the **Integration Accounts** blade that opens, select the integration account where you want to add the agreement. If you don't see any integration accounts, [create one first](#).

A screenshot of the 'Integration Accounts' blade. The title bar says 'Integration Accounts' and 'Microsoft'. Below the title are buttons for '+ Add', 'Columns', and 'Refresh'. A message 'Subscriptions: All 3 selected – Don't see a subscription? Switch directories' is displayed. There are two input fields: 'Filter by name...' and 'All subscriptions'. Below these is a summary '1 items'. A table header row includes columns for 'NAME', 'TYPE', 'RESOURCE GROUP', and 'LOCATION'. The table contains one item: 'MyNewIntegrationAccount' (Integration Account), 'MyNewRG', and 'Brazil South'.

4. Select **Overview**, then select the **Agreements** tile. If you don't have an Agreements tile, add the tile first.

The screenshot shows the 'Components' section of the 'Agreements' blade. It displays five categories: Schemas (11), Maps (6), Certificates (1), Partners (16), and Agreements (0). The 'Agreements' category is highlighted with a red border.

5. In the Agreements blade that opens, choose **Add**.

The screenshot shows the 'Add' blade for creating a new agreement. The 'Name' field is highlighted with a red box. Other fields include 'Agreement type' (X12 selected), 'Host Partner', 'Host Identity', 'Guest Partner', and 'Guest Identity'. There are also 'Receive Settings' and 'Send Settings' sections.

6. Under **Add**, enter a **Name** for your agreement. For the agreement type, select **X12**. Select the **Host Partner**, **Host Identity**, **Guest Partner**, and **Guest Identity** for your agreement. For more property details, see the table in this step.

Add

\* Name  
AgreementWithNYTradingPartner 

\* Agreement type  
X12

\* Host Partner  
MyCompany

\* Host Identity  
ZZ : 12345678

\* Guest Partner  
NYTradingPartner

\* Guest Identity  
ZZ : 87654321

Receive Settings >

Send Settings >

PROPERTY	DESCRIPTION
Name	Name of the agreement
Agreement Type	Should be X12
Host Partner	An agreement needs both a host and guest partner. The host partner represents the organization that configures the agreement.
Host Identity	An identifier for the host partner
Guest Partner	An agreement needs both a host and guest partner. The guest partner represents the organization that's doing business with the host partner.
Guest Identity	An identifier for the guest partner
Receive Settings	These properties apply to all messages received by an agreement.
Send Settings	These properties apply to all messages sent by an agreement.

#### NOTE

Resolution of X12 agreement depends on matching the sender qualifier and identifier, and the receiver qualifier and identifier defined in the partner and incoming message. If these values change for your partner, update the agreement too.

## Configure how your agreement handles received messages

Now that you've set the agreement properties, you can configure how this agreement identifies and handles incoming messages received from your partner through this agreement.

- Under **Add**, select **Receive Settings**. Configure these properties based on your agreement with the partner that exchanges messages with you. For property descriptions, see the tables in this section.

**Receive Settings** is organized into these sections: Identifiers, Acknowledgment, Schemas, Envelopes, Control Numbers, Validations, and Internal Settings.

- After you're done, make sure to save your settings by choosing **OK**.

Now your agreement is ready to handle incoming messages that conform to your selected settings.

### Identifiers

#### Receive Settings

Identifiers

ISA1 (Authorization Qualifier)	00 - No Authorization Information... ▾	ISA3 (Security Qualifier)	00 - No Security Information Present ▾
ISA2	ISA2	ISA4	ISA4

PROPERTY	DESCRIPTION
ISA1 (Authorization Qualifier)	Select the Authorization qualifier value from the drop-down list.
ISA2	Optional. Enter Authorization information value. If the value you entered for ISA1 is other than 00, enter a minimum of one alphanumeric character and a maximum of 10.
ISA3 (Security Qualifier)	Select the Security qualifier value from the drop-down list.
ISA4	Optional. Enter the Security information value. If the value you entered for ISA3 is other than 00, enter a minimum of one alphanumeric character and a maximum of 10.

### Acknowledgment

#### Acknowledgement

<input type="checkbox"/> TA1 Expected	<input type="checkbox"/> FA Expected
FA Version	
997 (Version 4010) 997 (Version 5010) 999 (Version 5010)	
<input type="checkbox"/> Include AK2 / IK2 Loop	

PROPERTY	DESCRIPTION
TA1 expected	Returns a technical acknowledgment to the interchange sender
FA expected	Returns a functional acknowledgment to the interchange sender. Then select whether you want the 997 or 999 acknowledgments, based on the schema version
Include AK2/IK2 Loop	Enables generation of AK2 loops in functional acknowledgments for accepted transaction sets

## Schemas

Select a schema for each transaction type (ST1) and Sender Application (GS2). The receive pipeline disassembles the incoming message by matching the values for ST1 and GS2 in the incoming message with the values you set here, and the schema of the incoming message with the schema you set here.

Schemas			
VERSION	TRANSACTION TYPE (ST01)	SENDER APPLICATION (GS02)	SCHEMA
No results			
<input type="button" value="▼"/>	<input type="button" value="▼"/>	<input type="button" value="Select a schema"/>	<input type="button" value="..."/>

PROPERTY	DESCRIPTION
Version	Select the X12 version
Transaction Type (ST01)	Select the transaction type
Sender Application (GS02)	Select the sender application
Schema	Select the schema file you want to use. Schemas are added to your integration account.

### NOTE

Configure the required [Schema](#) that is uploaded to your [integration account](#).

## Envelopes

Envelopes
ISA11 Usage
<a href="#">Standard Identifier</a> <a href="#">Repetition Separator</a>

PROPERTY	DESCRIPTION

PROPERTY	DESCRIPTION
ISA11 Usage	<p>Specifies the separator to use in a transaction set:</p> <p>Select <b>Standard identifier</b> to use a period (.) for decimal notation, rather than the decimal notation of the incoming document in the EDI receive pipeline.</p> <p>Select <b>Repetition Separator</b> to specify the separator for repeated occurrences of a simple data element or a repeated data structure. For example, usually the carat (^) is used as the repetition separator. For HIPAA schemas, you can only use the carat.</p>

## Control Numbers

Control Numbers

Disallow Interchange control number duplicates

Check for duplicate ISA13 every (days)  
30

Disallow Group control number duplicates

Disallow Transaction set control number duplicates

PROPERTY	DESCRIPTION
Disallow Interchange Control Number duplicates	Block duplicate interchanges. Checks the interchange control number (ISA13) for the received interchange control number. If a match is detected, the receive pipeline doesn't process the interchange. You can specify the number of days for performing the check by giving a value for <i>Check for duplicate ISA13 every (days)</i> .
Disallow Group control number duplicates	Block interchanges with duplicate group control numbers.
Disallow Transaction set control number duplicates	Block interchanges with duplicate transaction set control numbers.

## Validations

Validations						
MESSAGE TYPE	EDI VALIDATION	EXTENDED VALIDATION	ALLOW LEADING/TRAILING SPACES	TRIM LEADING/TRAILING SPACES	.TRAILING SEPARATOR	...
Default	true	false	false	false	NotAllowed	...
	<input type="button" value="▼"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Mandatory <input type="button" value="▼"/> ...

When you complete each validation row, another is automatically added. If you don't specify any rules, then validation uses the "Default" row.

PROPERTY	DESCRIPTION
Message Type	Select the EDI message type.

PROPERTY	DESCRIPTION
EDI Validation	Perform EDI validation on data types as defined by the schema's EDI properties, length restrictions, empty data elements, and trailing separators.
Extended Validation	If the data type isn't EDI, validation is on the data element requirement and allowed repetition, enumerations, and data element length validation (min/max).
Allow Leading/Trailing Zeroes	Retain any additional leading or trailing zero and space characters. Don't remove these characters.
Trim Leading/Trailing Zeroes	Remove leading or trailing zero and space characters.
Trailing Separator Policy	<p>Generate trailing separators.</p> <p>Select <b>Not Allowed</b> to prohibit trailing delimiters and separators in the received interchange. If the interchange has trailing delimiters and separators, the interchange is declared not valid.</p> <p>Select <b>Optional</b> to accept interchanges with or without trailing delimiters and separators.</p> <p>Select <b>Mandatory</b> when the interchange must have trailing delimiters and separators.</p>

## Internal Settings

### Internal Settings

Convert implied decimal format Nn to base 10 numeric value

Create empty XML tags if trailing separators are allowed

### Inbound batch processing

- Split Interchange as transaction sets - suspend transaction sets on error
- Split Interchange as transaction sets - suspend interchange on error
- Preserve Interchange - suspend transaction sets on error
- Preserve Interchange - suspend interchange on error

PROPERTY	DESCRIPTION
Convert implied decimal format "Nn" to a base 10 numeric value	Converts an EDI number that is specified with the format "Nn" into a base-10 numeric value
Create empty XML tags if trailing separators are allowed	Select this check box to have the interchange sender include empty XML tags for trailing separators.
Split Interchange as transaction sets - suspend transaction sets on error	Parses each transaction set in an interchange into a separate XML document by applying the appropriate envelope to the transaction set. Suspends only the transactions where the validation fails.
Split Interchange as transaction sets - suspend interchange on error	Parses each transaction set in an interchange into a separate XML document by applying the appropriate envelope. Suspends entire interchange when one or more transaction sets in the interchange fail validation.

PROPERTY	DESCRIPTION
Preserve Interchange - suspend transaction sets on error	Leaves the interchange intact, creates an XML document for the entire batched interchange. Suspends only the transaction sets that fail validation, while continuing to process all other transaction sets.
Preserve Interchange - suspend interchange on error	Leaves the interchange intact, creates an XML document for the entire batched interchange. Suspends the entire interchange when one or more transaction sets in the interchange fail validation.

## Configure how your agreement sends messages

You can configure how this agreement identifies and handles outgoing messages that you send to your partner through this agreement.

- Under **Add**, select **Send Settings**. Configure these properties based on your agreement with your partner who exchanges messages with you. For property descriptions, see the tables in this section.

**Send Settings** is organized into these sections: Identifiers, Acknowledgment, Schemas, Envelopes, Character Sets and Separators, Control Numbers, and Validation.

- After you're done, make sure to save your settings by choosing **OK**.

Now your agreement is ready to handle outgoing messages that conform to your selected settings.

### Identifiers

PROPERTY	DESCRIPTION
ISA1 (Authorization Qualifier)	00 - No Authorization Information Present
ISA2	ISA2
ISA3 (Security Qualifier)	00 - No Security Information Present
ISA4	ISA4

PROPERTY	DESCRIPTION
Authorization qualifier (ISA1)	Select the Authorization qualifier value from the drop-down list.
ISA2	Enter Authorization information value. If this value is other than 00, then enter a minimum of one alphanumeric character and a maximum of 10.
Security qualifier (ISA3)	Select the Security qualifier value from the drop-down list.
ISA4	Enter the Security information value. If this value is other than 00, for the Value (ISA4) text box, then enter a minimum of one alphanumeric value and a maximum of 10.

### Acknowledgment

## Acknowledgement

TA1 Expected

FA Expected

### FA Version

997      999

PROPERTY	DESCRIPTION
TA1 expected	Return a technical acknowledgment (TA1) to the interchange sender. This setting specifies that the host partner who is sending the message requests an acknowledgment from the guest partner in the agreement. These acknowledgments are expected by the host partner based on the Receive Settings of the agreement.
FA expected	Return a functional acknowledgment (FA) to the interchange sender. Select whether you want the 997 or 999 acknowledgements, based on the schema versions you are working with. These acknowledgments are expected by the host partner based on the Receive Settings of the agreement.
FA Version	Select the FA version

## Schemas

### Acknowledgement

TA1 Expected

FA Expected

### FA Version

997      999

PROPERTY	DESCRIPTION
Version	Select the X12 version
Transaction Type (ST01)	Select the transaction type
SCHEMA	Select the schema to use. Schemas are located in your integration account. If you select schema first, it automatically configures version and transaction type

### NOTE

Configure the required [Schema](#) that is uploaded to your [integration account](#).

## Envelopes

### Envelopes

ISA11 Usage

Standard Identifier Repetition Separator

Repetition Separator

U

PROPERTY	DESCRIPTION
ISA11 Usage	<p>Specifies the separator to use in a transaction set:</p> <p>Select <b>Standard identifier</b> to use a period (.) for decimal notation, rather than the decimal notation of the incoming document in the EDI receive pipeline.</p> <p>Select <b>Repetition Separator</b> to specify the separator for repeated occurrences of a simple data element or a repeated data structure. For example, usually the carat (^) is used as the repetition separator. For HIPAA schemas, you can only use the carat.</p>

## Control Numbers

Control Version Number (ISA12):	00401 - Standards Approved for ...	Usage Indicator (ISA15):	Test				
SCHEMA	GS1	GS2	GS3	GS4	GS5	GS7	GS8
Default		BTS-SENDER	RECEIVE-APP	CCYYMMDD	HHMM	Transportation...	00401
Select a s... ▾	AA - Acc... ▾			CCYYMM... ▾	HHMM ▾	Transpor... ▾	...

### Control Numbers

Interchange Control Number (ISA13) : ⓘ	1	to ⓘ	999999999
Group Control Number (GS06) : ⓘ	1	to ⓘ	999999999
Transaction Set Control Number (ST02) : ⓘ	1	to ⓘ	999999999
Prefix :			
Suffix :			

PROPERTY	DESCRIPTION
Control Version Number (ISA12)	Select the version of the X12 standard
Usage Indicator (ISA15)	Select the context of an interchange. The values are information, production data, or test data
Schema	Generates the GS and ST segments for an X12-encoded interchange that it sends to the Send Pipeline
GS1	Optional, select a value for the functional code from the drop-down list
GS2	Optional, application sender
GS3	Optional, application receiver
GS4	Optional, select CCYYMMDD or YYMMDD
GS5	Optional, select HHMM, HHMMSS, or HHMMSSdd
GS7	Optional, select a value for the responsible agency from the drop-down list

PROPERTY	DESCRIPTION
GS8	Optional, version of the document
Interchange Control Number (ISA13)	Required, enter a range of values for the interchange control number. Enter a numeric value with a minimum of 1 and a maximum of 999999999
Group Control Number (GS06)	Required, enter a range of numbers for the group control number. Enter a numeric value with a minimum of 1 and a maximum of 999999999
Transaction Set Control Number (ST02)	Required, enter a range of numbers for the Transaction Set Control number. Enter a range of numeric values with a minimum of 1 and a maximum of 999999999
Prefix	Optional, designated for the range of transaction set control numbers used in acknowledgment. Enter a numeric value for the middle two fields, and an alphanumeric value (if desired) for the prefix and suffix fields. The middle fields are required and contain the minimum and maximum values for the control number
Suffix	Optional, designated for the range of transaction set control numbers used in an acknowledgment. Enter a numeric value for the middle two fields and an alphanumeric value (if desired) for the prefix and suffix fields. The middle fields are required and contain the minimum and maximum values for the control number

### Character Sets and Separators

Other than the character set, you can enter a different set of delimiters for each message type. If a character set isn't specified for a given message schema, then the default character set is used.

Character Sets and Separators						
* Character Set to be used :						
<input type="text" value="UTF8"/> <span style="float: right;">▼</span>						
SCHEMA	INPUT TYPE	COMPONENT SEPA...	DATA ELEMENT SE...	REPLACEMENT CH...	SEGMENT TERMIN...	SUFFIX
Default	:	*	\$	~	None	...
Select a sch... ▼	Char ▼	<input type="text"/> None ▼ ...				

PROPERTY	DESCRIPTION
Character Set to be used	To validate the properties, select the X12 character set. The options are Basic, Extended, and UTF8.
Schema	Select a schema from the drop-down list. After you complete each row, a new row is automatically added. For the selected schema, select the separators set that you want to use, based on the following separator descriptions.
Input Type	Select an input type from the drop-down list.

PROPERTY	DESCRIPTION
Component Separator	To separate composite data elements, enter a single character.
Data Element Separator	To separate simple data elements within composite data elements, enter a single character.
Replacement Character	Enter a replacement character used for replacing all separator characters in the payload data when generating the outbound X12 message.
Segment Terminator	To indicate the end of an EDI segment, enter a single character.
Suffix	Select the character that is used with the segment identifier. If you designate a suffix, then the segment terminator data element can be empty. If the segment terminator is left empty, then you must designate a suffix.

## Validation

Validation						
MESSAGE TYPE	EDI VALIDATION	EXTENDED VALIDATION	ALLOW LEADING/TRAI...	TRIM LEADING/TRAILI...	TRAILING SEPARATOR ...	
Default	true	false	false	false	NotAllowed	...
	<input type="button" value="▼"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="button" value="Not Allowed"/> <input type="button" value="▼"/> ...

When you complete each validation row, another is automatically added. If you don't specify any rules, then validation uses the "Default" row.

PROPERTY	DESCRIPTION
Message Type	Select the EDI message type.
EDI Validation	Perform EDI validation on data types as defined by the schema's EDI properties, length restrictions, empty data elements, and trailing separators.
Extended Validation	If the data type isn't EDI, validation is on the data element requirement and allowed repetition, enumerations, and data element length validation (min/max).
Allow Leading/Trailing Zeroes	Retain any additional leading or trailing zero and space characters. Don't remove these characters.
Trim Leading/Trailing Zeroes	Remove leading or trailing zero characters.

PROPERTY	DESCRIPTION
Trailing Separator Policy	<p>Generate trailing separators.</p> <p>Select <b>Not Allowed</b> to prohibit trailing delimiters and separators in the sent interchange. If the interchange has trailing delimiters and separators, the interchange is declared not valid.</p> <p>Select <b>Optional</b> to send interchanges with or without trailing delimiters and separators.</p> <p>Select <b>Mandatory</b> if the sent interchange must have trailing delimiters and separators.</p>

## Find your created agreement

- After you finish setting all your agreement properties, on the **Add** blade, choose **OK** to finish creating your agreement and return to your integration account blade.

Your newly added agreement now appears in your **Agreements** list.

- You can also view your agreements in your integration account overview. On your integration account blade, choose **Overview**, then select the **Agreements** tile.

The screenshot shows the 'MyNewIntegrationAccount' blade in the Azure portal. The left sidebar includes links for Overview, Access control (IAM), Tags, Settings (Callback URL, Schemas, Maps, Certificates, Partners, Agreements, Properties, Locks, Automation script), Monitoring (Diagnostics logs), and Help. The main area has tabs for Essentials and Components. Under Essentials, it shows a Resource group (MyNewRG), Location (Brazil South), and Subscription ID (Visual Studio Enterprise). Under Components, there are tiles for Schemas (0), Maps (0), Certificates (0), Partners (2), and Agreements (1). The 'Agreements' tile is highlighted with a blue border.

NAME	TYPE	HOST PARTNER	GUEST PARTNER
X12agreement	X12	partner1	partner2

## Learn more

- Learn more about the Enterprise Integration Pack

# Encode X12 messages for Azure Logic Apps with the Enterprise Integration Pack

3/9/2017 • 2 min to read • [Edit Online](#)

With the Encode X12 message connector, you can validate EDI and partner-specific properties, convert XML-encoded messages into EDI transaction sets in the interchange, and request a Technical Acknowledgement, Functional Acknowledgment, or both. To use this connector, you must add the connector to an existing trigger in your logic app.

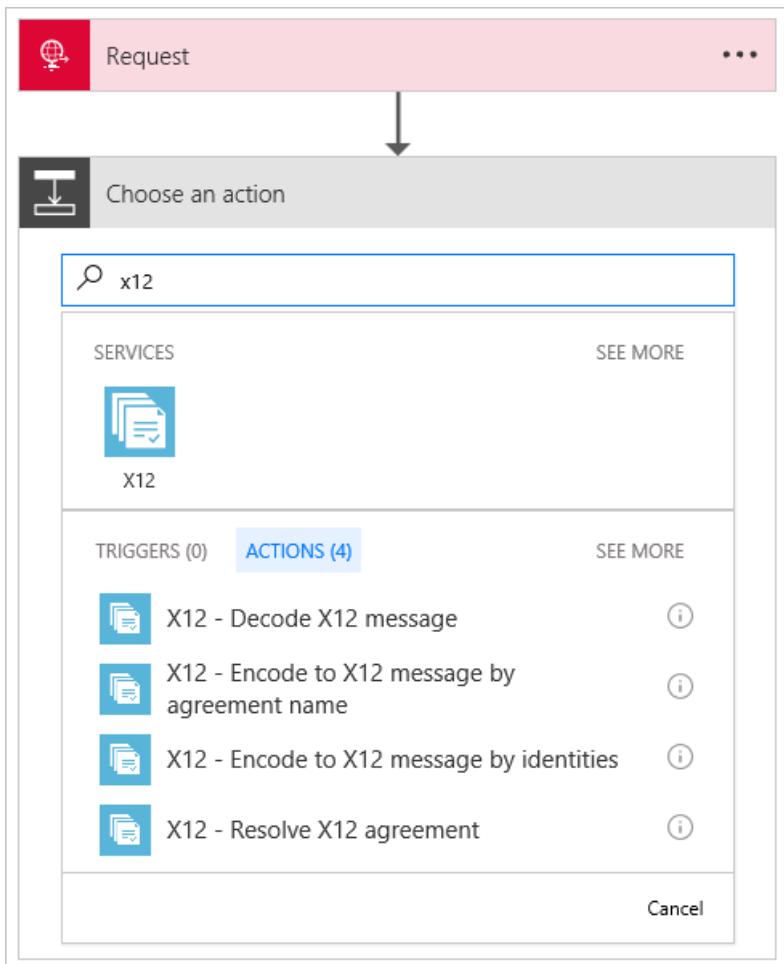
## Before you start

Here's the items you need:

- An Azure account; you can create a [free account](#)
- An [integration account](#) that's already defined and associated with your Azure subscription. You must have an integration account to use the Encode X12 message connector.
- At least two [partners](#) that are already defined in your integration account
- An [X12 agreement](#) that's already defined in your integration account

## Encode X12 messages

1. [Create a logic app](#).
2. The Encode X12 message connector doesn't have triggers, so you must add a trigger for starting your logic app, like a Request trigger. In the Logic App Designer, add a trigger, and then add an action to your logic app.
3. In the search box, enter "x12" for your filter. Select either **X12 - Encode to X12 message by agreement name** or **X12 - Encode to X12 message by identities**.



- If you didn't previously create any connections to your integration account, you're prompted to create that connection now. Name your connection, and select the integration account that you want to connect.

**X12 - Encode to X12 message by agreement name**

CONNECTION NAME*	Enter name for connection
INTEGRATION ACCOUNT*	EIPIntegration
<input type="button" value="Cancel"/> <input type="button" value="Create"/>	

Properties with an asterisk are required.

PROPERTY	DETAILS
Connection Name *	Enter any name for your connection.
Integration Account *	Enter a name for your integration account. Make sure that your integration account and logic app are in the same Azure location.

- When you're done, your connection details should look similar to this example. To finish creating your connection, choose **Create**.

X12 - Encode to X12 message by agreement name

CONNECTION NAME\*

padmaconn

INTEGRATION ACCOUNT\*

EIPIntegration

Cancel Create

Your connection is now created.

X12 - Encode to X12 message by agreement name

NAME OF X12 AGREEMENT\*

XML MESSAGE TO ENCODE\*

XML message to encode

Connected to padmaconn. Change connection.

#### Encode X12 messages by agreement name

If you chose to encode X12 messages by agreement name, open the **Name of X12 agreement** list, enter or select your existing X12 agreement. Enter the XML message to encode.

X12 - Encode to X12 message by agreement name

NAME OF X12 AGREEMENT\*

X12Agreement

XML MESSAGE TO ENCODE\*

Body

You can insert data from previous steps...

Outputs from Request

Body

Connected to padmaconn. Change connection.

#### Encode X12 messages by identities

If you choose to encode X12 messages by identities, enter the sender identifier, sender qualifier, receiver identifier, and receiver qualifier as configured in your X12 agreement. Select the XML message to encode.

X12 - Encode to X12 message by agreement name

...  
SENDER IDENTIFIER\*  
  
SENDER QUALIFIER\*  
  
RECEIVER IDENTIFIER\*  
  
RECEIVER QUALIFIER\*  
  
XML MESSAGE TO ENCODE\*  
  
Connected to padconn. [Change connection.](#)

## X12 Encode details

The X12 Encode connector performs these tasks:

- Agreement resolution by matching sender and receiver context properties.
- Serializes the EDI interchange, converting XML-encoded messages into EDI transaction sets in the interchange.
- Applies transaction set header and trailer segments
- Generates an interchange control number, a group control number, and a transaction set control number for each outgoing interchange
- Replaces separators in the payload data
- Validates EDI and partner-specific properties
  - Schema validation of the transaction-set data elements against the message Schema
  - EDI validation performed on transaction-set data elements.
  - Extended validation performed on transaction-set data elements
- Requests a Technical and/or Functional acknowledgment (if configured).
  - A Technical Acknowledgment generates as a result of header validation. The technical acknowledgment reports the status of the processing of an interchange header and trailer by the address receiver
  - A Functional Acknowledgment generates as a result of body validation. The functional acknowledgment reports each error encountered while processing the received document

## Next steps

[Learn more about the Enterprise Integration Pack](#)

# Decode X12 messages for Azure Logic Apps with the Enterprise Integration Pack

3/9/2017 • 3 min to read • [Edit Online](#)

With the Decode X12 message connector, you can validate EDI and partner-specific properties, generate an XML document for each transaction set, and generate acknowledgment for processed transactions. To use this connector, you must add the connector to an existing trigger in your logic app.

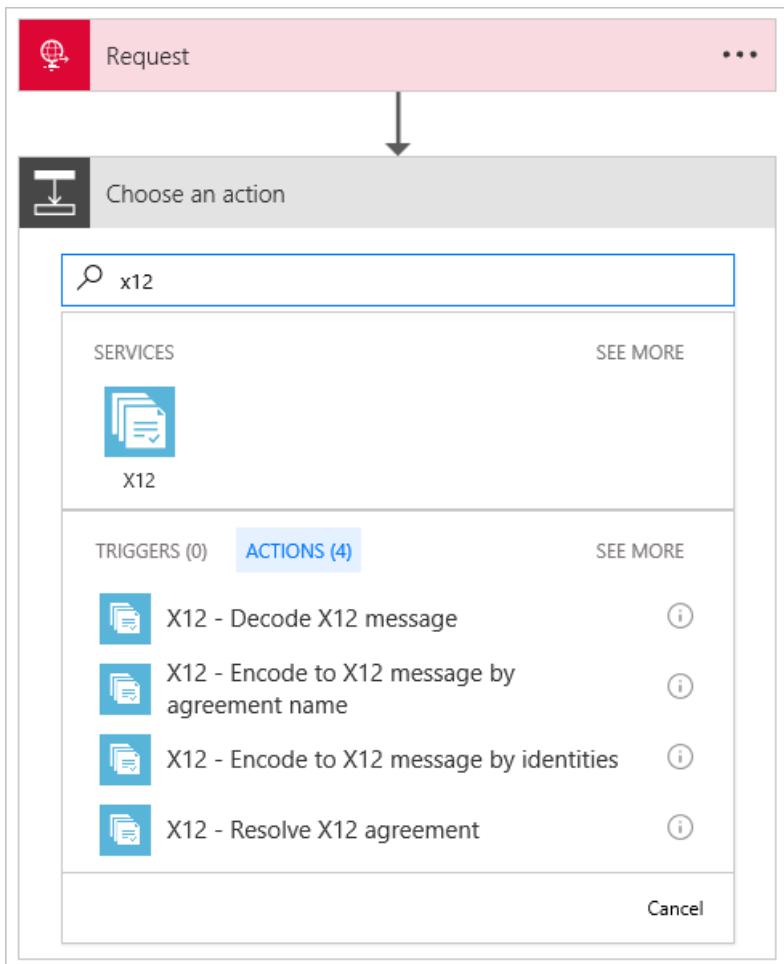
## Before you start

Here's the items you need:

- An Azure account; you can create a [free account](#)
- An [integration account](#) that's already defined and associated with your Azure subscription. You must have an integration account to use the Decode X12 message connector.
- At least two [partners](#) that are already defined in your integration account
- An [X12 agreement](#) that's already defined in your integration account

## Decode X12 messages

1. [Create a logic app.](#)
2. The Decode X12 message connector doesn't have triggers, so you must add a trigger for starting your logic app, like a Request trigger. In the Logic App Designer, add a trigger, and then add an action to your logic app.
3. In the search box, enter "x12" for your filter. Select **X12 - Decode X12 message**.



4. If you didn't previously create any connections to your integration account, you're prompted to create that connection now. Name your connection, and select the integration account that you want to connect.

CONNECTION NAME\*

INTEGRATION ACCOUNT\*

**Create**

Properties with an asterisk are required.

PROPERTY	DETAILS
Connection Name *	Enter any name for your connection.
Integration Account *	Enter a name for your integration account. Make sure that your integration account and logic app are in the same Azure location.

5. When you're done, your connection details should look similar to this example. To finish creating your connection, choose **Create**.

X12 - Decode X12 message

CONNECTION NAME\*

padmaconn

INTEGRATION ACCOUNT\*

EIPIntegration

Cancel Create

- After your connection is created, as shown in this example, select the X12 flat file message to decode.

Decode X12 message

X12 FLAT FILE MESSAGE TO DECODE\*

X12 flat file message to decode

Connected to padmaconn. Change connection.

For example:

Decode X12 message

X12 FLAT FILE MESSAGE TO DECODE\*

Body

You can insert data from previous steps...

Outputs from Request

Body

Connected to padmaconn. Change connection.

## X12 Decode details

The X12 Decode connector performs these tasks:

- Validates the envelope against trading partner agreement
- Generates an XML document for each transaction set.
- Validates EDI and partner-specific properties
  - EDI structural validation, and extended schema validation
  - Validation of the structure of the interchange envelope.
  - Schema validation of the envelope against the control schema.
  - Schema validation of the transaction-set data elements against the message schema.
  - EDI validation performed on transaction-set data elements
- Verifies that the interchange, group, and transaction set control numbers are not duplicates
  - Checks the interchange control number against previously received interchanges.
  - Checks the group control number against other group control numbers in the interchange.
  - Checks the transaction set control number against other transaction set control numbers in that group.
- Converts the entire interchange to XML
  - Split Interchange as transaction sets - suspend transaction sets on error: Parses each transaction set in an interchange into a separate XML document. If one or more transaction sets in the interchange fail validation, X12 Decode suspends only those transaction sets.
  - Split Interchange as transaction sets - suspend interchange on error: Parses each transaction set in an

- interchange into a separate XML document. If one or more transaction sets in the interchange fail validation, X12 Decode suspends the entire interchange.
- Preserve Interchange - suspend transaction sets on error: Creates an XML document for the entire batched interchange. X12 Decode suspends only those transaction sets that fail validation, while continuing to process all other transaction sets
  - Preserve Interchange - suspend interchange on error: Creates an XML document for the entire batched interchange. If one or more transaction sets in the interchange fail validation, X12 Decode suspends the entire interchange,
- Generates a Technical and/or Functional acknowledgment (if configured).
    - A Technical Acknowledgment generates as a result of header validation. The technical acknowledgment reports the status of the processing of an interchange header and trailer by the address receiver.
    - A Functional Acknowledgment generates as a result of body validation. The functional acknowledgment reports each error encountered while processing the received document

## Next steps

[Learn more about the Enterprise Integration Pack](#)

# Logic Apps B2B cross region disaster recovery

4/3/2017 • 4 min to read • [Edit Online](#)

B2B workloads involve money transactions like orders, invoices. For business, it is critical to quickly recover to meet business level SLAs as agreed with their partners during a disaster event. This topic demonstrates building a business continuity plan for B2B workloads. It covers

- Create an integration account in secondary region
- Establish a connection from primary region to secondary region
- Fail over to secondary region during a disaster event
- Fall back to primary region post-disaster event

## Create an integration account in secondary region

1. Select a secondary region and create an [integration account](#).
2. Add partners, schemas, and agreements for the required message flows where the run status needs to be replicated to secondary region integration account.

### TIP

Make sure consistency in integration account artifacts naming convention across regions

## Establish a connection from primary to secondary

To pull the run status from the primary region, create a Logic App in the secondary region. It should have a **trigger** and an **action**. The trigger should connect to primary region integration account and the action should connect to secondary region integration account. Based on the time interval, the trigger polls the primary region run status table pulls the new records if any and action updates them to secondary region integration account. This helps to get incremental runtime status from primary region to secondary region.

Business continuity in Logic Apps integration account is designed to support based on B2B protocols - X12, AS2, and EDIFACT. To find detailed steps, select respective links.

- X12
- AS2
- EDIFACT (coming soon)

## Fail over to secondary region during a disaster event

The recommendation is to deploy all primary region resources (for example SQL Azure or DocumentDB databases, or Service Bus / Event Hubs used for messaging, APIM, Logic Apps) in the secondary region as well. During a disaster event, when the primary region is not available for business continuity, direct traffic to the secondary region. Secondary region helps recover business functions quickly to meet recovery time/point objectives (RPO/RTO) as agreed with their partners. Also, minimizes efforts to fail over from one region to another region.

There is an expected latency while copying control numbers from primary to secondary region. To avoid sending duplicate generated control numbers to partners during a disaster event, it is recommended to bump up control numbers in the **secondary region agreements** using [PowerShell cmdlets](#).

## Fall back to primary region post-disaster event

When primary region is available, to fall back to primary region follow below steps

- Stop accepting messages from partners in the **secondary region**
- Bump up generated control numbers for all the **primary region agreements** using [PowerShell cmdlets](#)
- Direct traffic from secondary to primary region
- Check the Logic App created in the secondary region to pull run status from the primary is enabled

## X12

Business continuity for EDI X12 documents is designed based on control numbers

- Control numbers received (Inbound messages) from partners
- Control numbers generated (outbound messages) and send to partners

### TIP

You can also use [X12 quick start template](#) to create Logic Apps. Creating primary and secondary integration accounts are prerequisites to use the template. The template helps create 2 Logic Apps, one for received control number and another for generated control number. Respective triggers and actions are created in the Logic Apps, connects the trigger to primary integration account and action to secondary integration account.

### Control numbers received from the partners

1. Create a [Logic App](#) in the secondary region
2. Search **X12** and select **X12 - When a received control number is modified**

SERVICES SEE MORE

X12

TRIGGERS (2) ACTIONS (6) SEE MORE

X12 - When a generated control number is modified

X12 - When a received control number is modified

TELL US WHAT YOU NEED

Help us decide which services and triggers to add next with [UserVoice](#)

3. Selecting the trigger prompts to establish a connection to integration account. Trigger to be connected to primary region integration account. Give a connection name, select your **primary region integration account** from the list and click create.

When a received control number is modified ...

\*CONNECTION NAME  
padma primary region integration account

\*INTEGRATION ACCOUNT

primaryia	padmatestdr
secondaryia	padmatestdr
psrivasIntAcc	psrivasintAccount
rarayuduperfDecodeA1	rarayuduperftest
rarayuduperftestia1	rarayuduperftest
rarayuduperftestia10	rarayuduperftest
rarayuduperftestia2	rarayuduperftest
rarayuduperftestia3	rarayuduperftest
rarayuduperftestia4	rarayuduperftest

Create

Manually enter connection information

4. DateTime to start control number sync is optional. Frequency can be set to **Day, Hour, Minute, or Second** with an interval.

When a received control number is modified (i) ...

DateTime to start control number sync  
Initial date time to poll for changes.

How often do you want to check for items?

\*Frequency      \*Interval

Minute      3

Connected to padma primary region integration account. [Change connection.](#)

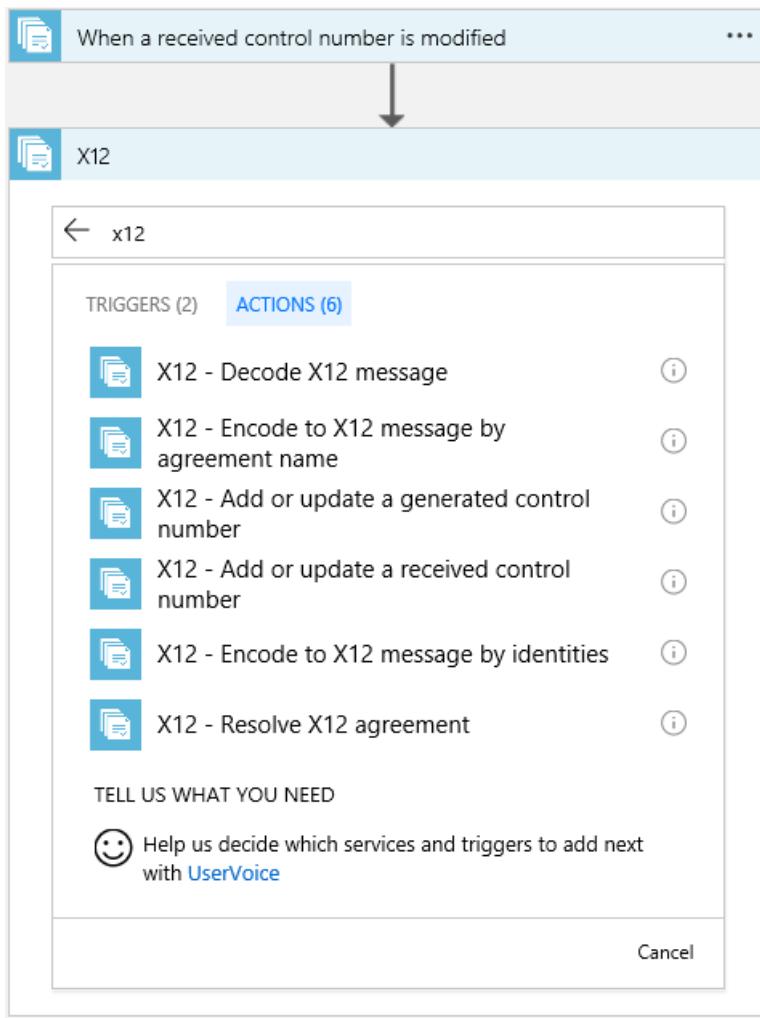
5. Click **New step** and **Add an action**

When a received control number is modified ...

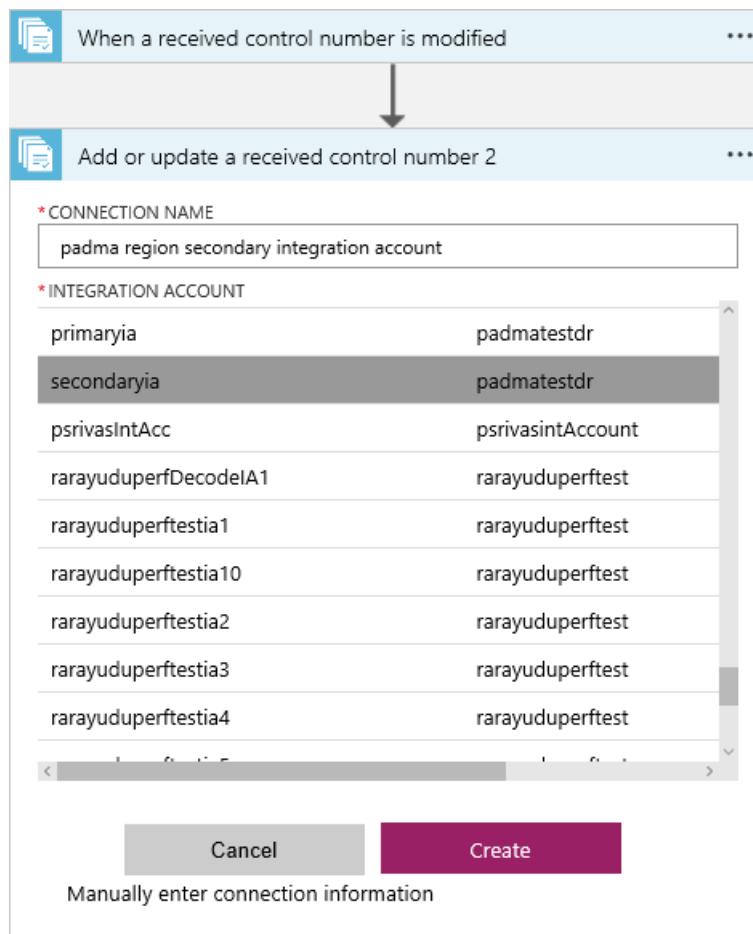
+ New step

Add an action
Add a condition
...
More

6. Search **X12** and select **X12 - Add or update a received control number**



7. Action to be connected to secondary integration account. Select **Change connection** and **Add new connection** lists the available integration accounts. Give a connection name, select your **secondary region integration account** from the list and click create.



#### 8. Select the dynamic content and save the logic app

The screenshot shows the logic app designer with the "Add or update a received control number" step selected. On the right, a sidebar titled "Add dynamic content from the apps and services used in this flow" lists several items: "AgreementName", "ControlNumberType", "ControlNumber", "ControlNumberChangedTime", "IsAcknowledgement", "IsMessageProcessingFailed", and "ControlNumberChangedTime". Below the sidebar, a note says "Connected to secondary region integration account. Change connection." At the bottom, there is a "+ New step" button.

#### 9. Based on the time interval, the trigger polls the primary region received control number table, pulls the new records and actions updates them to secondary region integration account. If they are no updates, the trigger status shows as skipped.

Runs history				Trigger History			
STATUS	START TIME	IDENTIFIER	DURAT...	STATUS	START TIME	FIRE...	
✓ Succeeded	2/22/2017 12:08 PM	08587138128008334894550198321	312 Mi...	● Skipped	2/22/2017 12:08 PM		
✓ Succeeded	2/22/2017 12:08 PM	08587138128008334895550198321	358 Mi...	✓ Succeeded	2/22/2017 12:08 PM	Fired	
✓ Succeeded	2/22/2017 12:08 PM	08587138128008334896550198321	317 Mi...	✓ Succeeded	2/22/2017 12:08 PM	Fired	
✓ Succeeded	2/22/2017 12:07 PM	08587138128312188402191250933	353 Mi...	✓ Succeeded	2/22/2017 12:08 PM	Fired	
✓ Succeeded	2/22/2017 12:07 PM	08587138128312188403191250933	364 Mi...	✓ Succeeded	2/22/2017 12:08 PM	Fired	
✓ Succeeded	2/22/2017 12:07 PM	08587138128312188404191250933	246 Mi...	● Skipped	2/22/2017 12:07 PM		

## Control numbers generated and from the partners

1. Create a [Logic App](#) in the secondary region
2. Search **X12** and select **X12 - When a generated control number is modified**

SERVICES SEE MORE

X12

TRIGGERS (2) ACTIONS (6) SEE MORE

- X12 - When a generated control number is modified (i)
- X12 - When a received control number is modified (i)

TELL US WHAT YOU NEED

Help us decide which services and triggers to add next with [UserVoice](#)

3. Selecting the trigger prompts to establish a connection to integration account. Trigger to be connected to primary region integration account. Give a connection name, select your **primary region integration account** from the list and click create.

When a generated control number is modified

\* CONNECTION NAME  
padma primary integration account

\* INTEGRATION ACCOUNT

primaryia	padmatestdr
secondaryia	padmatestdr
psrivasIntAcc	psrivasintAccount
rarayuduperfDecodeIA1	rarayuduperftest
rarayuduperftestia1	rarayuduperftest
rarayuduperftestia10	rarayuduperftest
rarayuduperftestia2	rarayuduperftest
rarayuduperftestia3	rarayuduperftest
rarayuduperftestia4	rarayuduperftest

Cancel      Create

Manually enter connection information

- DateTime to start control number sync is optional. Frequency can be set to **Day**, **Hour**, **Minute**, or **Second** with an interval.

When a generated control number is modified

DateTime to start control number sync  
Initial date time to poll for changes.

How often do you want to check for items?

\* Frequency      \* Interval

Minute      3

Connected to padma primary integration account. [Change connection](#).

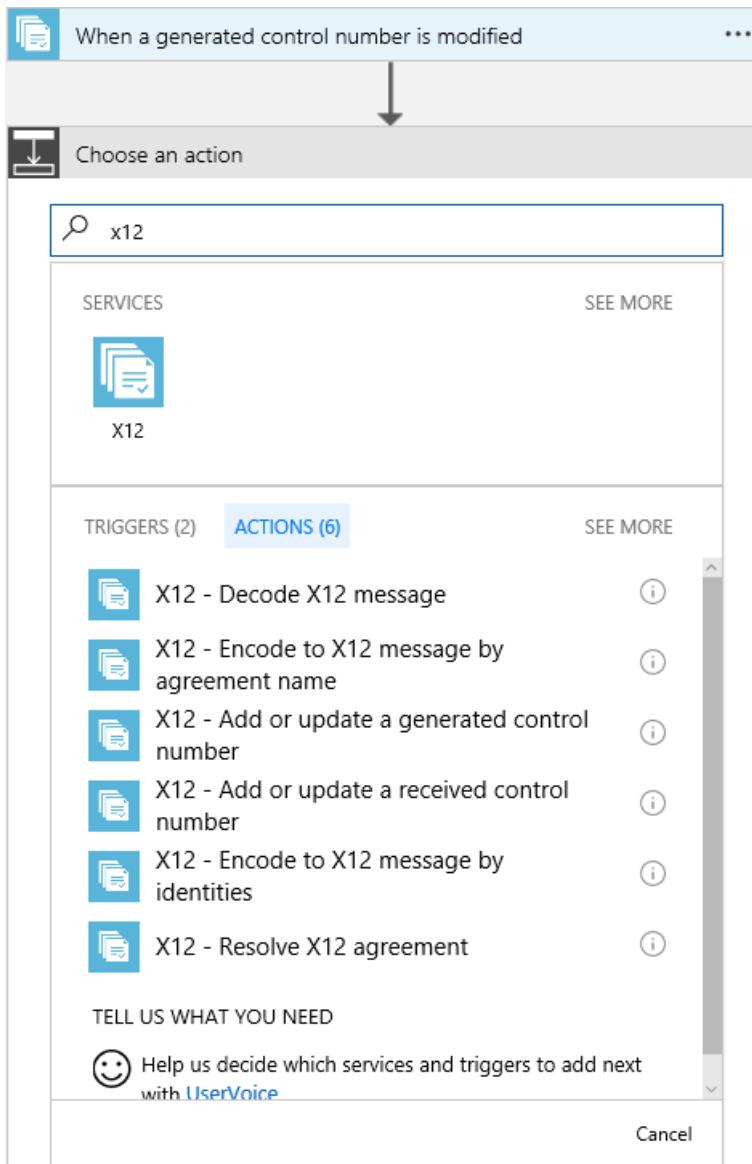
- Click **New step** and **Add an action**

When a generated control number is modified

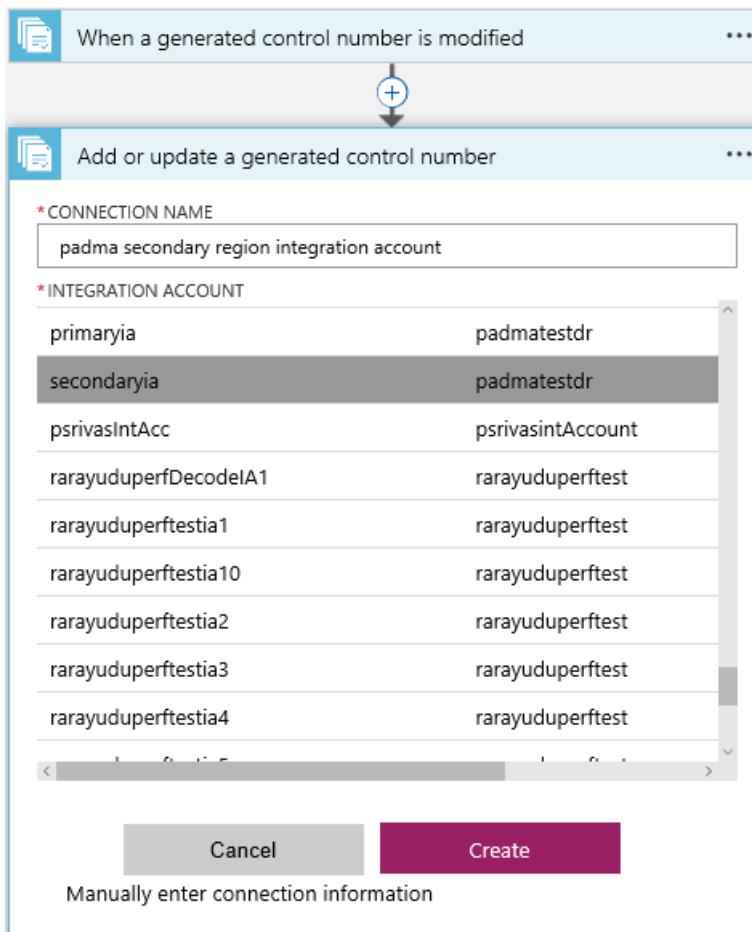
+ New step

Add an action      Add a condition      More

- Search **X12** and select **X12 - Add or update a generated control number**



7. Action to be connected to secondary integration account. Select **Change connection** and **Add new connection** lists the available integration accounts. Give a connection name, select your **secondary region integration account** from the list and click create.



#### 8. Select the dynamic content and save the logic app

The screenshot shows the Logic App designer with the 'Add or update a generated control number' step selected. The dynamic content fields are populated with values from the previous step:

- AgreementName: AgreementName
- ControlNumberType: ControlNumberType
- ControlNumber: ControlNumber
- ControlNumberChangedTime: ControlNumberChangedTime
- IsAcknowledgement: IsAcknowledgement
- IsMessageProcessingFailed: IsMessageProcessingFailed

A sidebar on the right lists the available dynamic content items:

- AgreementName
- ControlNumber
- ControlNumberChangedTime
- ControlNumberType
- IsAcknowledgement
- IsMessageProcessingFailed

#### 9. Based on the time interval, the trigger polls the primary region received control number table, pulls the new records and actions updates them to secondary region integration account. If they are no updates, the trigger status shows as skipped.

Runs history				Trigger History			
STATUS	START TIME	IDENTIFIER	DURAT...	STATUS	START TIME	FIRE...	
Succeeded	3/27/2017 2:59 PM	08587109549206869386937134533	21.06...	Skipped	3/30/2017 5:22 PM		
Succeeded	3/27/2017 2:59 PM	08587109549210577807016896146	569 Mi...	Skipped	3/30/2017 5:21 PM		
Succeeded	3/27/2017 2:29 PM	08587109567373628630015635565	789 Mi...	Skipped	3/30/2017 5:20 PM		
Succeeded	3/27/2017 2:29 PM	08587109567381667756246061857	21.39...	Skipped	3/30/2017 5:19 PM		
Succeeded	3/20/2017 4:30 PM	08587115542371590254255238613	1.42 S...	Skipped	3/30/2017 5:18 PM		

Based on the time interval, the incremental runtime status replicates from primary region to secondary region. During a disaster event, when the primary region is not available, direct traffic to the secondary region for business continuity.

## Next Steps

- Learn more about [monitoring B2B messages](#).

# Connect to on-premises data from logic apps

3/31/2017 • 2 min to read • [Edit Online](#)

To access on-premises data, you can set up a connection to an on-premises data gateway for supported Azure Logic Apps connectors. The following steps walk you through how to install and set up the on-premises data gateway to work with your logic apps. The on-premises data gateway supports these connections:

- BizTalk Server
- DB2
- File System
- Informix
- MQ
- Oracle Database
- SAP Application Server
- SAP Message Server
- SharePoint for HTTP only, not HTTPS
- SQL Server

For more information about these connections, see [Connectors for Azure Logic Apps](#).

## Requirements

- You must have a work or school email address in Azure to associate the on-premises data gateway with your account (Azure Active Directory based account).
- If you are using a Microsoft account, like @outlook.com, you can use your Azure account to [create a work or school email address](#).
- You must have already [installed the on-premises data gateway on a local machine](#).
- You can associate your installation to one gateway resource only. Your gateway can't be claimed by another Azure on-premises data gateway. Claim happens at ([creation during Step 2 in this topic](#)).

## Install and configure the connection

### 1. Install the on-premises data gateway

If you haven't already, follow these steps to [install the on-premises data gateway](#). Before you can continue with the other steps, make sure that you installed the data gateway on an on-premises machine.

### 2. Create an Azure on-premises data gateway resource

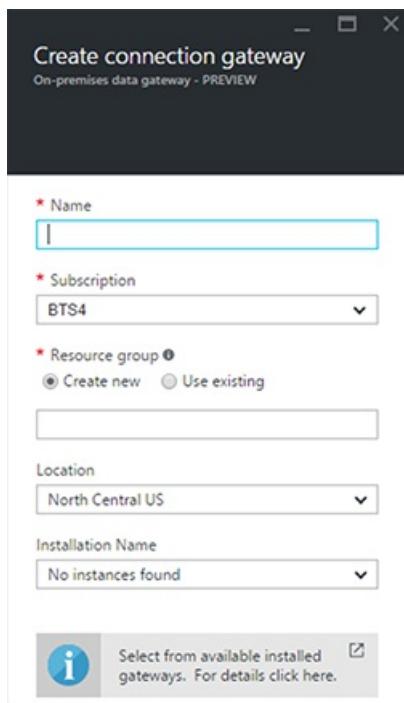
After you install the gateway, you must associate your Azure subscription with the gateway.

#### IMPORTANT

Ensure that the gateway resource is created in the same Azure region as your logic app. If you don't deploy the gateway resource to the same region, your logic app can't access the gateway.

1. Sign in to Azure using the same work or school email address that you used during gateway installation.
2. Choose **New**.

3. Find and select the **On-premises data gateway**.
4. To associate the gateway with your account, complete the information, including selecting the appropriate **Installation Name**.

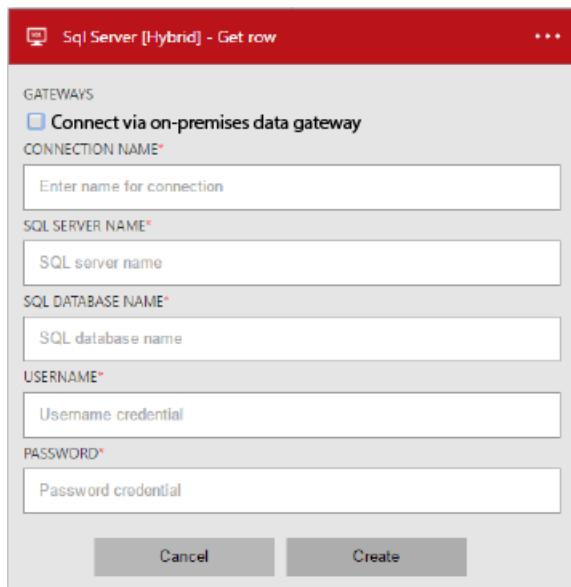


5. To create the resource, choose **Create**.

### 3. Create a logic app connection in Logic App Designer

Now that your Azure subscription is associated with an instance of the on-premises data gateway, you can create a connection to the gateway from your logic app.

1. Open a logic app and choose a connector that supports on-premises connectivity, like SQL Server.
2. Select **Connect via on-premises data gateway**.



3. Select the **Gateway** that you want to connect, and complete any other required connection information.
4. To create the connection, choose **Create**.

Your connection is now configured for your logic app to use.

## Edit your data gateway connection settings

After you add the data gateway connection to your logic app, you might have to make changes so you can adjust settings specific to that connection. You can find the connection in either of two places:

- On the logic app blade, under **Development Tools**, select **API Connections**. This list shows you all API Connections associated with your logic app, including your data gateway connection. To view and modify that connection's settings, select that connection.
- On the API Connections main blade, you can find all API Connections associated with your Azure subscription, including your data gateway connection. To view and modify the connection settings, select that connection.

## Next Steps

- [Common examples and scenarios for logic apps](#)
- [Enterprise integration features](#)

# Install an on-premises data gateway for Azure Logic Apps

3/31/2017 • 8 min to read • [Edit Online](#)

The on-premises data gateway supports these connections:

- BizTalk Server
- DB2
- File System
- Informix
- MQ
- Oracle Database
- SAP Application Server
- SAP Message Server
- SharePoint for HTTP only, not HTTPS
- SQL Server

For more information about these connections, see [Connectors for Azure Logic Apps](#).

## Installation and configuration

### Requirements

Minimum:

- .NET 4.5 Framework
- 64-bit version of Windows 7 or Windows Server 2008 R2 (or later)

Recommended:

- 8 Core CPU
- 8 GB Memory
- 64-bit version of Windows 2012 R2 (or later)

Related considerations:

- Only install the on-premises data gateway on a local machine. You can't install the gateway on a domain controller.
- Don't install the gateway on a computer that might turn off, go to sleep, or doesn't connect to the Internet because the gateway can't run under those circumstances. Also, gateway performance might suffer over a wireless network.
- You can only use a work or school email address in Azure, so that you can associate the on-premises data gateway with your Azure Active Directory-based account.

If you are using a Microsoft account, like @outlook.com, you can use your Azure account to [create a work or school email address](#).

### Install the gateway

1. [Download installer for the on-premises data gateway here](#).

2. Specify **On-premises data gateway** as the mode.
3. Sign in with your Azure work or school account.
4. Set up a new gateway, or you can migrate, restore, or take over an existing gateway.

To configure a gateway, provide a name for your gateway and a recovery key, then choose **Configure**.

Specify a recovery key that contains at least eight characters, and keep the key in a safe place. You need this key if you want to migrate, restore, or take over the gateway.

To migrate, restore, or take over an existing gateway, provide the recovery key that was specified when the gateway was created.

### Restart the gateway

The gateway runs as a Windows service, so like any other Windows service, you can start and stop the service in multiple ways. For example, you can open a command prompt with elevated permissions on the machine where the gateway is running, and run either these commands:

- To stop the service, run this command:

```
net stop PBIEgwService
```

- To start the service, run this command:

```
net start PBIEgwService
```

### Configure a firewall or proxy

To provide proxy information for your gateway, see [Configure proxy settings](#).

You can verify whether your firewall, or proxy, might block connections by running the following command from a PowerShell prompt. This command tests connectivity to the Azure Service Bus and only network connectivity, so the command doesn't have anything to do with the cloud server service or the gateway. This test helps determine whether your machine can actually connect to the internet.

```
Test-NetConnection -ComputerName watchdog.servicebus.windows.net -Port 9350
```

The results should look similar to this example. If **TcpTestSucceeded** is not true, you might be blocked by a firewall.

```
ComputerName      : watchdog.servicebus.windows.net
RemoteAddress    : 70.37.104.240
RemotePort       : 5672
InterfaceAlias   : vEthernet (Broadcom NetXtreme Gigabit Ethernet - Virtual Switch)
SourceAddress    : 10.120.60.105
PingSucceeded    : False
PingReplyDetails (RTT) : 0 ms
TcpTestSucceeded : True
```

If you want to be exhaustive, substitute the **ComputerName** and **Port** values with the values listed under [Configure ports](#) in this topic.

The firewall might also block connections that the Azure Service Bus makes to the Azure data centers. If so, approve (unblock) all the IP addresses for those data centers in your region. You can get a list of [Azure IP addresses here](#).

### Configure ports

The gateway creates an outbound connection to Azure Service Bus and communicates on outbound ports: TCP 443 (default), 5671, 5672, 9350 through 9354. The gateway doesn't require inbound ports.

Learn more about [hybrid solutions](#).

DOMAIN NAMES	OUTBOUND PORTS	DESCRIPTION
*.analysis.windows.net	443	HTTPS
*.login.windows.net	443	HTTPS
*.servicebus.windows.net	5671-5672	Advanced Message Queuing Protocol (AMQP)
*.servicebus.windows.net	443, 9350-9354	Listeners on Service Bus Relay over TCP (requires 443 for Access Control token acquisition)
*.frontend.clouddatahub.net	443	HTTPS
*.core.windows.net	443	HTTPS
login.microsoftonline.com	443	HTTPS
*.msftncsi.com	443	Used to test internet connectivity when the gateway is unreachable by the Power BI service.

If you have to approve IP addresses instead of the domains, you can download and use the [Microsoft Azure Datacenter IP ranges list](#). In some cases, the Azure Service Bus connections are made with IP Address rather than fully qualified domain names.

### Sign-in accounts

You can sign in with either a work or school account, which is your organization account. If you signed up for an Office 365 offering and didn't supply your actual work email, your sign-in address might look like jeff@contoso.onmicrosoft.com. Your account, within a cloud service, is stored within a tenant in Azure Active Directory (Azure AD). Usually, your Azure AD account's UPN matches the email address.

### Windows service account

For the Windows service logon credential, the on-premises data gateway is set up to use NT SERVICE\PBIEgwService. By default, the gateway has the right for "Log on as a service", within the context of the machine where you installing the gateway.

This service account isn't same account used for connecting to on-premises data sources, nor the work or school account that you use to sign in to cloud services.

## How the gateway works

When others interact with an element that's connected to an on-premises data source:

1. The cloud service creates a query, along with the encrypted credentials for the data source, and sends the query to the queue for the gateway to process.
2. The service analyzes the query and pushes the request to the Azure Service Bus.
3. The on-premises data gateway polls the Azure Service Bus for pending requests.
4. The gateway gets the query, decrypts the credentials, and connects to the data source with those credentials.
5. The gateway sends the query to the data source for execution.
6. The results are sent from the data source, back to the gateway, and then to the cloud service. The service then

uses the results.

## Frequently asked questions

### General

**Question:** Do I need a gateway for data sources in the cloud, such as SQL Azure?

**Answer:** No. A gateway connects to on-premises data sources only.

**Question:** What is the actual Windows service called?

**Answer:** In Services, the gateway is called Power BI Enterprise Gateway Service.

**Question:** Are there any inbound connections to the gateway from the cloud?

**Answer:** No. The gateway uses outbound connections to Azure Service Bus.

**Question:** What if I block outbound connections? What do I need to open?

**Answer:** See the ports and hosts that the gateway uses.

**Question:** Does the gateway have to be installed on the same machine as the data source?

**Answer:** No. The gateway connects to the data source using the connection information that was provided.

Consider the gateway as a client application in this sense. The gateway just needs the capability to connect to the server name that was provided.

**Question:** What is the latency for running queries to a data source from the gateway? What is the best architecture?

**Answer:** To reduce network latency, install the gateway as close to the data source as possible. If you can install the gateway on the actual data source, this proximity minimizes the latency introduced. Consider the data centers too. For example, if your service uses the West US datacenter, and you have SQL Server hosted in an Azure VM, your Azure VM should be in the West US too. This proximity minimizes latency and avoids egress charges on the Azure VM.

**Question:** Are there any requirements for network bandwidth?

**Answer:** We recommend that your network connection has good throughput. Every environment is different, and the amount of data being sent affects the results. Using ExpressRoute could help to guarantee a level of throughput between on-premises and the Azure data centers.

You can use the third-party tool Azure Speed Test app to help gauge your throughput.

**Question:** Can the gateway Windows service run with an Azure Active Directory account?

**Answer:** No. The Windows service must have a valid Windows account. By default, the service runs with the Service SID, NT SERVICE\PBIEgwService.

**Question:** How are results sent back to the cloud?

**Answer:** Results are sent through the Azure Service Bus.

**Question:** Where are my credentials stored?

**Answer:** The credentials that you enter for a data source are encrypted and stored in the gateway cloud service. The credentials are decrypted at the on-premises gateway.

### High availability/disaster recovery

**Question:** Are there any plans for enabling high availability scenarios with the gateway?

**Answer:** These scenarios are on the roadmap, but we don't have a timeline yet.

**Question:** What options are available for disaster recovery?

**Answer:** You can use the recovery key to restore or move a gateway. When you install the gateway, specify the recovery key.

**Question:** What is the benefit of the recovery key?

**Answer:** The recovery key provides a way to migrate or recover your gateway settings after a disaster.

## Troubleshooting

**Question:** Where are the gateway logs?

**Answer:** See Tools later in this topic.

**Question:** How can I see what queries are being sent to the on-premises data source?

**Answer:** You can enable query tracing, which includes the queries that are sent. Remember to change query tracing back to the original value when done troubleshooting. Leaving query tracing turned on creates larger logs.

You can also look at tools that your data source has for tracing queries. For example, you can use Extended Events or SQL Profiler for SQL Server and Analysis Services.

### Update to the latest version

Many issues can surface when the gateway version becomes outdated. As good general practice, make sure that you use the latest version. If you haven't updated the gateway for a month or longer, you might consider installing the latest version of the gateway, and see if you can reproduce the issue.

**Error: Failed to add user to group. (-2147463168 PBIEgwService Performance Log Users)**

You might get this error if you try to install the gateway on a domain controller, which isn't supported. Make sure that you deploy the gateway on a machine that isn't a domain controller.

## Tools

### Collect logs from the gateway configurer

You can collect several logs for the gateway. Always start with the logs!

#### Installer logs

```
%localappdata%\Temp\Power_BI_Gateway_-Enterprise.log
```

#### Configuration logs

```
%localappdata%\Microsoft\Power BI Enterprise Gateway\GatewayConfigurator.log
```

#### Enterprise gateway service logs

```
C:\Users\PBIEgwService\AppData\Local\Microsoft\Power BI Enterprise Gateway\EnterpriseGateway.log
```

#### Event logs

You can find the Data Management Gateway and PowerBIGateway logs under **Application and Services Logs**.

### Fiddler Trace

**Fiddler** is a free tool from Telerik that monitors HTTP traffic. You can see this traffic with the Power BI service from the client machine. This service might show errors and other related information.

## Next Steps

- [Connect to on-premises data from logic apps](#)
- [Enterprise integration features](#)
- [Connectors for Azure Logic Apps](#)

# Create templates for logic apps deployment and release management

3/3/2017 • 4 min to read • [Edit Online](#)

After a logic app has been created, you might want to create it as an Azure Resource Manager template. This way, you can easily deploy the logic app to any environment or resource group where you might need it. For more about Resource Manager templates, see [authoring Azure Resource Manager templates](#) and [deploying resources by using Azure Resource Manager templates](#).

## Logic app deployment template

A logic app has three basic components:

- **Logic app resource:** Contains information about things like pricing plan, location, and the workflow definition.
- **Workflow definition:** Describes your logic app's workflow steps and how the Logic Apps engine should execute the workflow. You can view this definition in your logic app's **Code View** window. In the logic app resource, you can find this definition in the `definition` property.
- **Connections:** Refers to separate resources that securely store metadata about any connector connections, such as a connection string and an access token. In the logic app resource, your logic app references these resources in the `parameters` section.

You can view all these pieces of existing logic apps by using a tool like [Azure Resource Explorer](#).

To make a template for a logic app to use with resource group deployments, you must define the resources and parameterize as needed. For example, if you're deploying to a development, test, and production environment, you likely want to use different connection strings to a SQL database in each environment. Or, you might want to deploy within different subscriptions or resource groups.

## Create a logic app deployment template

The easiest way to have a valid logic app deployment template is to use the [Visual Studio Tools for Logic Apps](#). The Visual Studio tools generate a valid deployment template that can be used across any subscription or location.

A few other tools can assist you as you create a logic app deployment template. You can author by hand, that is, by using the resources already discussed here to create parameters as needed. Another option is to use a [logic app template creator](#) PowerShell module. This open-source module first evaluates the logic app and any connections that it is using, and then generates template resources with the necessary parameters for deployment. For example, if you have a logic app that receives a message from an Azure Service Bus queue and adds data to an Azure SQL database, the tool preserves all the orchestration logic and parameterizes the SQL and Service Bus connection strings so that they can be set at deployment.

### NOTE

Connections must be within the same resource group as the logic app.

## Install the logic app template PowerShell module

The easiest way to install the module is via the [PowerShell Gallery](#), by using the command

```
Install-Module -Name LogicAppTemplate .
```

You also can install the PowerShell module manually:

1. Download the latest release of the [logic app template creator](#).
2. Extract the folder in your PowerShell module folder (usually  
%UserProfile%\Documents\WindowsPowerShell\Modules ).

For the module to work with any tenant and subscription access token, we recommend that you use it with the [ARMClient](#) command-line tool. This [blog post](#) discusses ARMClient in more detail.

### Generate a logic app template by using PowerShell

After PowerShell is installed, you can generate a template by using the following command:

```
armclient token $SubscriptionId | Get-LogicAppTemplate -LogicApp MyApp -ResourceGroup MyRG -SubscriptionId $SubscriptionId -Verbose | Out-File C:\template.json
```

\$SubscriptionId is the Azure subscription ID. This line first gets an access token via ARMClient, then pipes it through to the PowerShell script, and then creates the template in a JSON file.

## Add parameters to a logic app template

After you create your logic app template, you can continue to add or modify parameters that you might need. For example, if your definition includes a resource ID to an Azure function or nested workflow that you plan to deploy in a single deployment, you can add more resources to your template and parameterize IDs as needed. The same applies to any references to custom APIs or Swagger endpoints you expect to deploy with each resource group.

## Deploy a logic app template

You can deploy your template by using any tools like PowerShell, REST API, [Visual Studio Team Services Release Management](#), and template deployment through the Azure portal. Also, to store the values for parameters, we recommend that you create a [parameter file](#). Learn how to [deploy resources with Azure Resource Manager templates and PowerShell](#) or [deploy resources with Azure Resource Manager templates and the Azure portal](#).

### Authorize OAuth connections

After deployment, the logic app works end-to-end with valid parameters. However, you must still authorize OAuth connections to generate a valid access token. To authorize OAuth connections, open the logic app in the Logic Apps Designer, and authorize these connections. Or for automated deployment, you can use a script to consent to each OAuth connection. There's an example script on GitHub under the [LogicAppConnectionAuth](#) project.

## Visual Studio Team Services Release Management

A common scenario for deploying and managing an environment is to use a tool like Release Management in Visual Studio Team Services, with a logic app deployment template. Visual Studio Team Services includes a [Deploy Azure Resource Group](#) task that you can add to any build or release pipeline. You need to have a [service principal](#) for authorization to deploy, and then you can generate the release definition.

1. In Release Management, select **Empty** so that you create an empty definition.



## Create new release definition

### Select a template

#### Deployment



##### Azure Cloud Service Deployment

Deploy an Azure Cloud Service



##### Azure Website Deployment

Deploy and test your Azure website

#### Empty

Start with an empty definition

Next >

Cancel

2. Choose any resources you need for this, most likely including the logic app template that is generated manually or as part of the build process.
3. Add an **Azure Resource Group Deployment** task.
4. Configure with a [service principal](#), and reference the Template and Template Parameters files.
5. Continue to build out steps in the release process for any other environment, automated test, or approvers as needed.

# Design, build, and deploy Azure Logic Apps in Visual Studio

3/13/2017 • 5 min to read • [Edit Online](#)

Although the [Azure portal](#) offers a great way for you to create and manage Azure Logic Apps, you might want to use Visual Studio for designing, building, and deploying your logic apps. Visual Studio provides rich tools like the Logic App Designer for you to create logic apps, configure deployment and automation templates, and deploy to any environment.

To get started with Azure Logic Apps, learn [how to create your first logic app in the Azure portal](#).

## Installation steps

To install and configure Visual Studio tools for Azure Logic Apps, follow these steps.

### Prerequisites

- [Visual Studio 2015](#)
- [Latest Azure SDK](#) (2.9.1 or greater)
- [Azure PowerShell](#)
- Access to the web when using the embedded designer

### Install Visual Studio tools for Azure Logic Apps

After you install the prerequisites:

1. Open Visual Studio. On the **Tools** menu, select **Extensions and Updates**.
2. Expand the **Online** category so you can search online.
3. Browse or search for **Logic Apps** until you find **Azure Logic Apps Tools for Visual Studio**.
4. To download and install the extension, click **Download**.
5. Restart Visual Studio after installation.

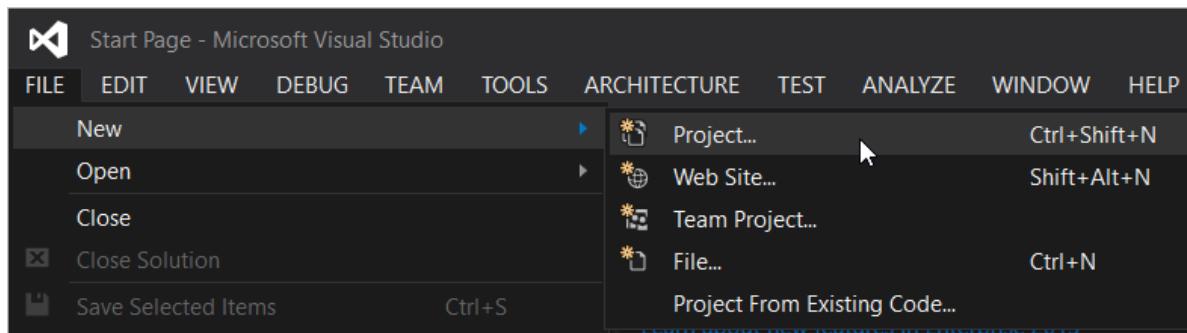
#### NOTE

You can also download Azure Logic Apps Tools for Visual Studio directly from the [Visual Studio Marketplace](#).

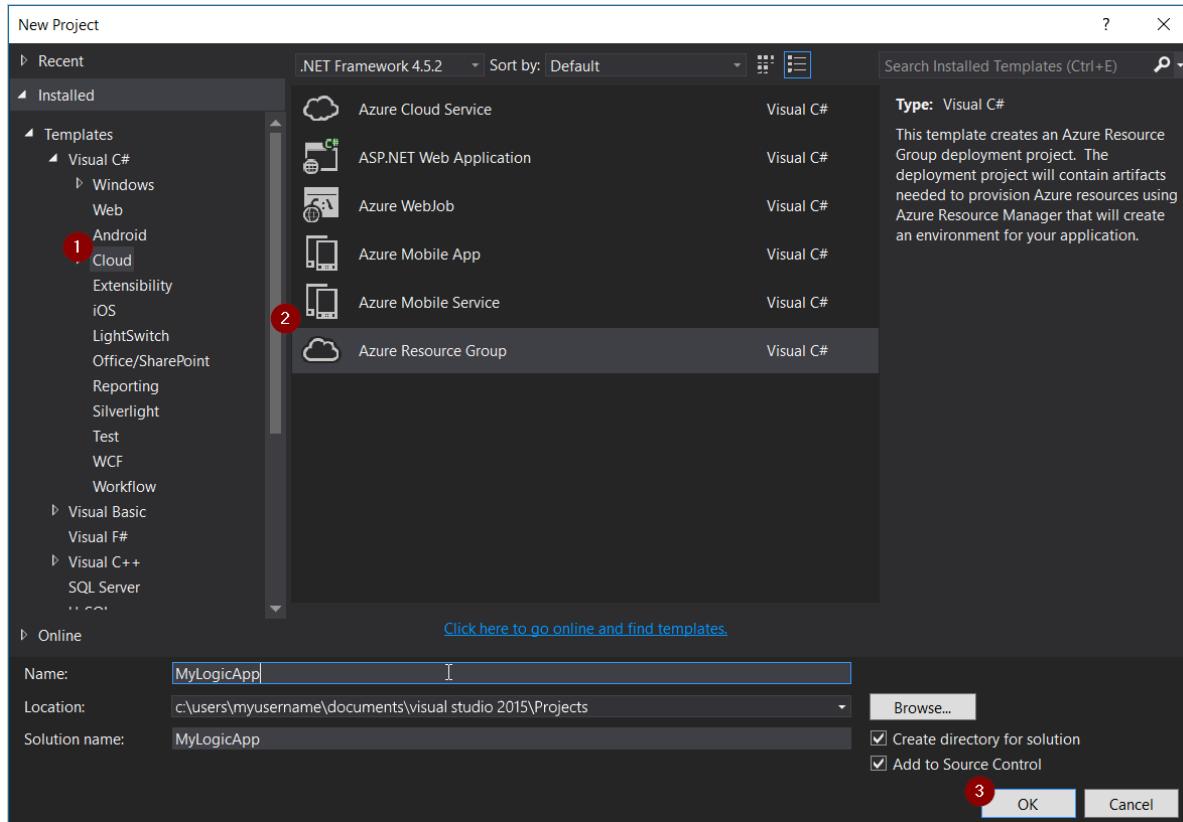
After you finish installation, you can use the Azure Resource Group project with Logic App Designer.

## Create your project

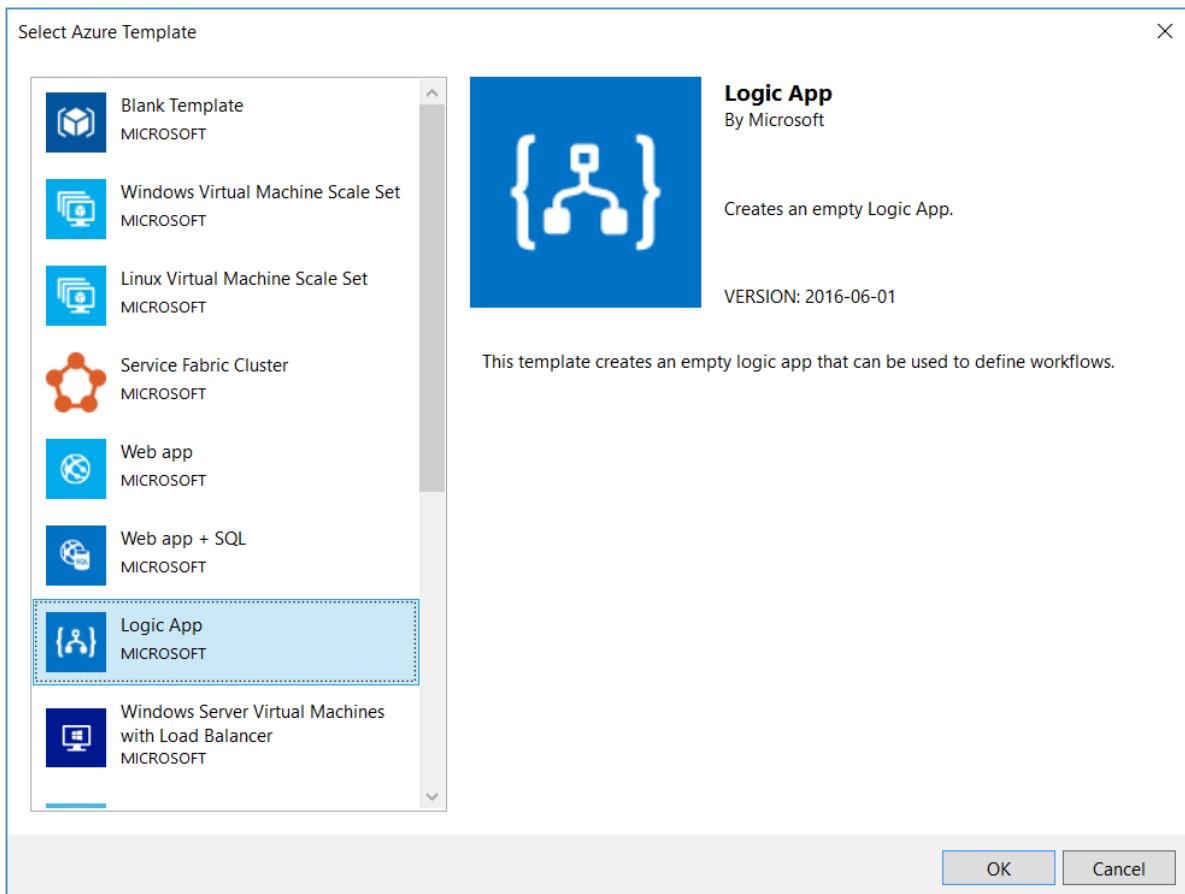
1. On the **File** menu, go to **New**, and select **Project**. Or to add your project to an existing solution, go to **Add**, and select **New Project**.



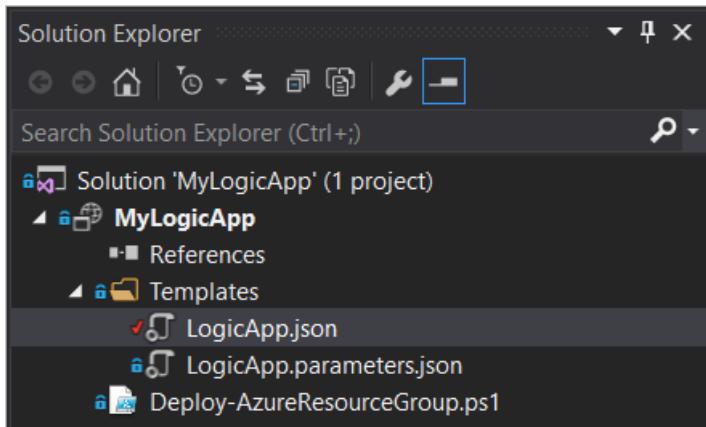
2. In the **New Project** window, find **Cloud**, and select **Azure Resource Group**. Name your project, and click **OK**.



3. Select the **Logic App** template, which creates a blank logic app deployment template for you to use. After you select your template, click **OK**.



You've now added your logic app project to your solution. In the Solution Explorer, your deployment file should appear.



## Create your logic app with Logic App Designer

When you have an Azure Resource Group project that contains a logic app, you can open the Logic App Designer in Visual Studio to create your workflow.

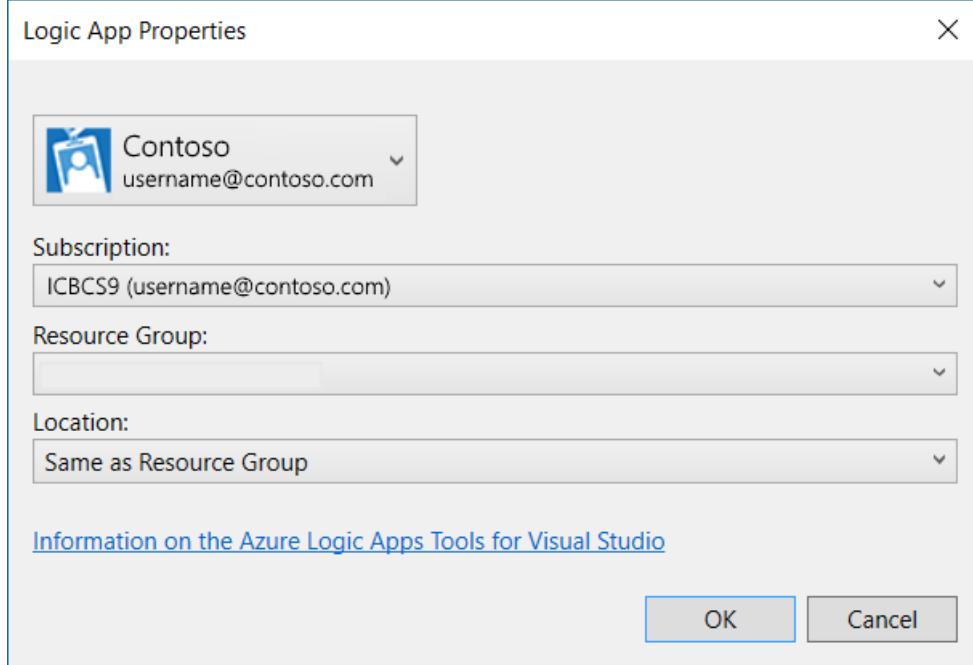
### NOTE

The designer requires an internet connection to query connectors for available properties and data. For example, if you use the Dynamics CRM Online connector, the designer queries your CRM instance to show available custom and default properties.

1. Right-click your `<template>.json` file, and select **Open with Logic App Designer**. (`Ctrl+L`)
2. Choose your Azure subscription, resource group, and location for your deployment template.

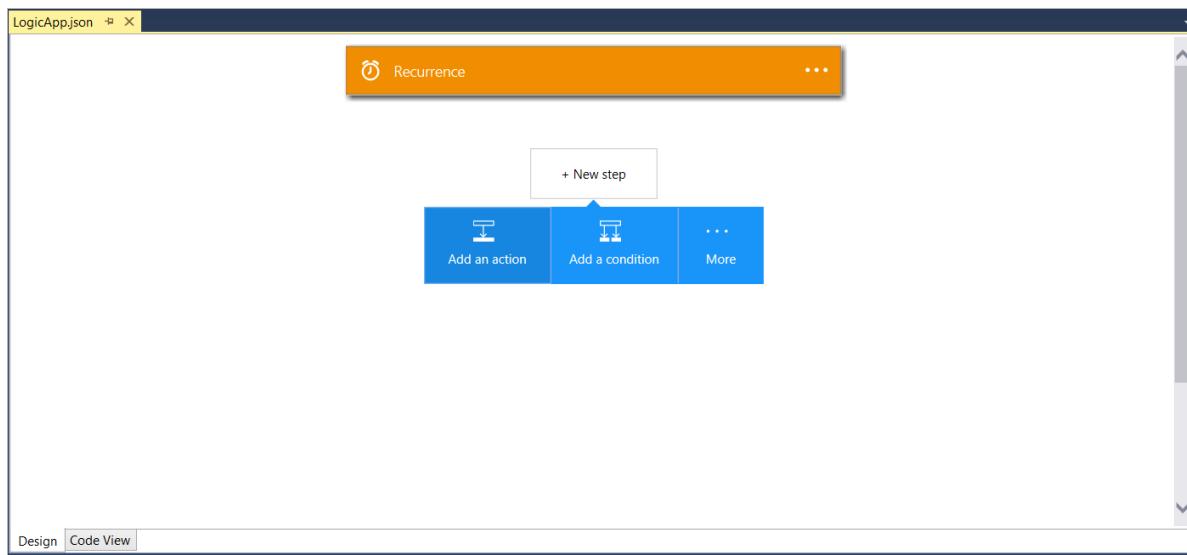
#### NOTE

Designing a logic app creates API Connection resources that query for properties during design. Visual Studio uses your selected resource group to create those connections during design time. To view or change any API Connections, go to the Azure portal, and browse for **API Connections**.



The designer uses the definition in the `<template>.json` file for rendering.

3. Create and design your logic app. Your deployment template is updated with your changes.



Visual Studio adds `Microsoft.Web/connections` resources to your resource file for any connections your logic app needs to function. These connection properties can be set when you deploy, and managed after you deploy in **API Connections** in the Azure portal.

#### Switch to JSON code view

To show the JSON representation for your logic app, select the **Code View** tab at the bottom of the designer.

To switch back to the full resource JSON, right-click the `<template>.json` file, and select **Open**.

#### Add references for dependent resources to Visual Studio deployment templates

When you want your logic app to reference dependent resources, you can use [Azure Resource Manager template](#)

[functions](#), like parameters, in your logic app deployment template. For example, you might want your logic app to reference an Azure Function or integration account that you want to deploy alongside your logic app. Follow these guidelines about how to use parameters in your deployment template so that the Logic App Designer renders correctly.

You can use logic app parameters in these kinds of triggers and actions:

- Child workflow
- Function app
- APIM call
- API connection runtime URL

And you can use these template functions: list below, includes parameters, variables, resourceId, concat, and so on. For example, here's how you can replace the Azure Function resource ID:

```
"parameters":{  
    "functionName": {  
        "type":"string",  
        "minLength":1,  
        "defaultValue":"<FunctionName>"  
    }  
},
```

And where you'd use parameters:

```
"MyFunction": {  
    "type": "Function",  
    "inputs": {  
        "body":{}  
    },  
    "function":{  
        "id": "[resourceId('Microsoft.Web/sites/functions', 'functionApp', parameters('functionName'))]"  
    }  
,  
    "runAfter":{}  
}
```

#### NOTE

For the Logic App Designer to work when you use parameters, you must provide default values, for example:

```
"parameters": {  
    "IntegrationAccount": {  
        "type":"string",  
        "minLength":1,  
  
        "defaultValue": "/subscriptions/<subscriptionID>/resourceGroups/<resourceGroupName>/providers/Microsoft.Logic/integrationAccounts/<integrationAccountName>"  
    }  
},
```

#### Save your logic app

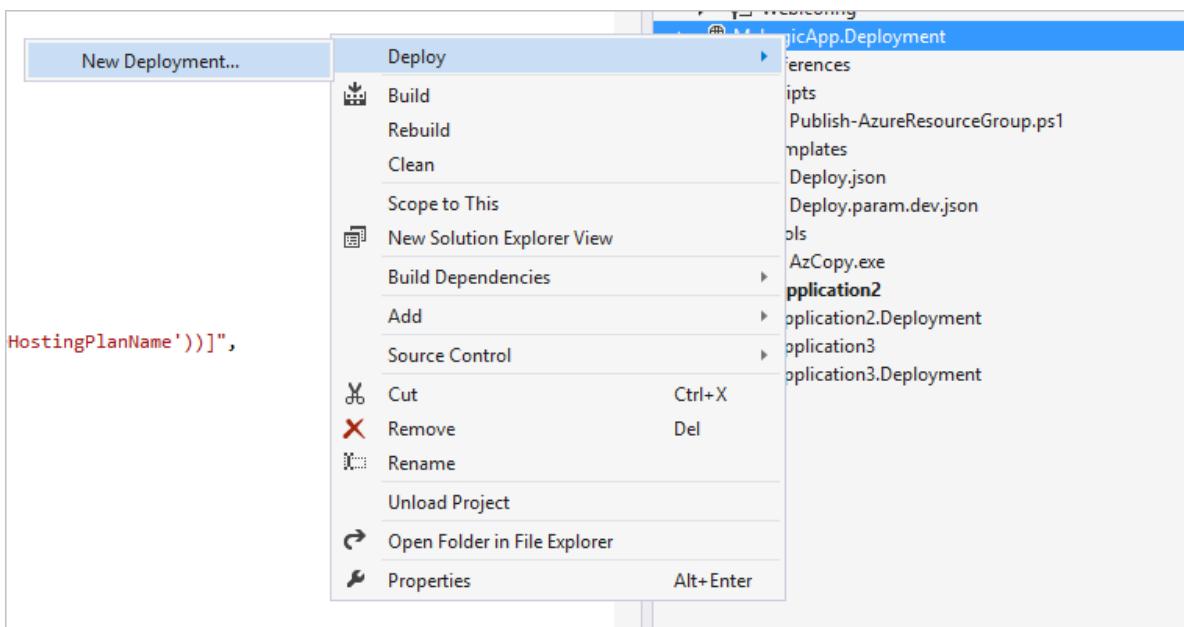
To save your logic app at anytime, go to **File > Save**. ([\*\*Ctrl+S\*\*](#))

If your logic app has any errors when you save your app, they appear in the Visual Studio **Outputs** window.

## Deploy your logic app from Visual Studio

After configuring your app, you can deploy directly from Visual Studio in just a couple steps.

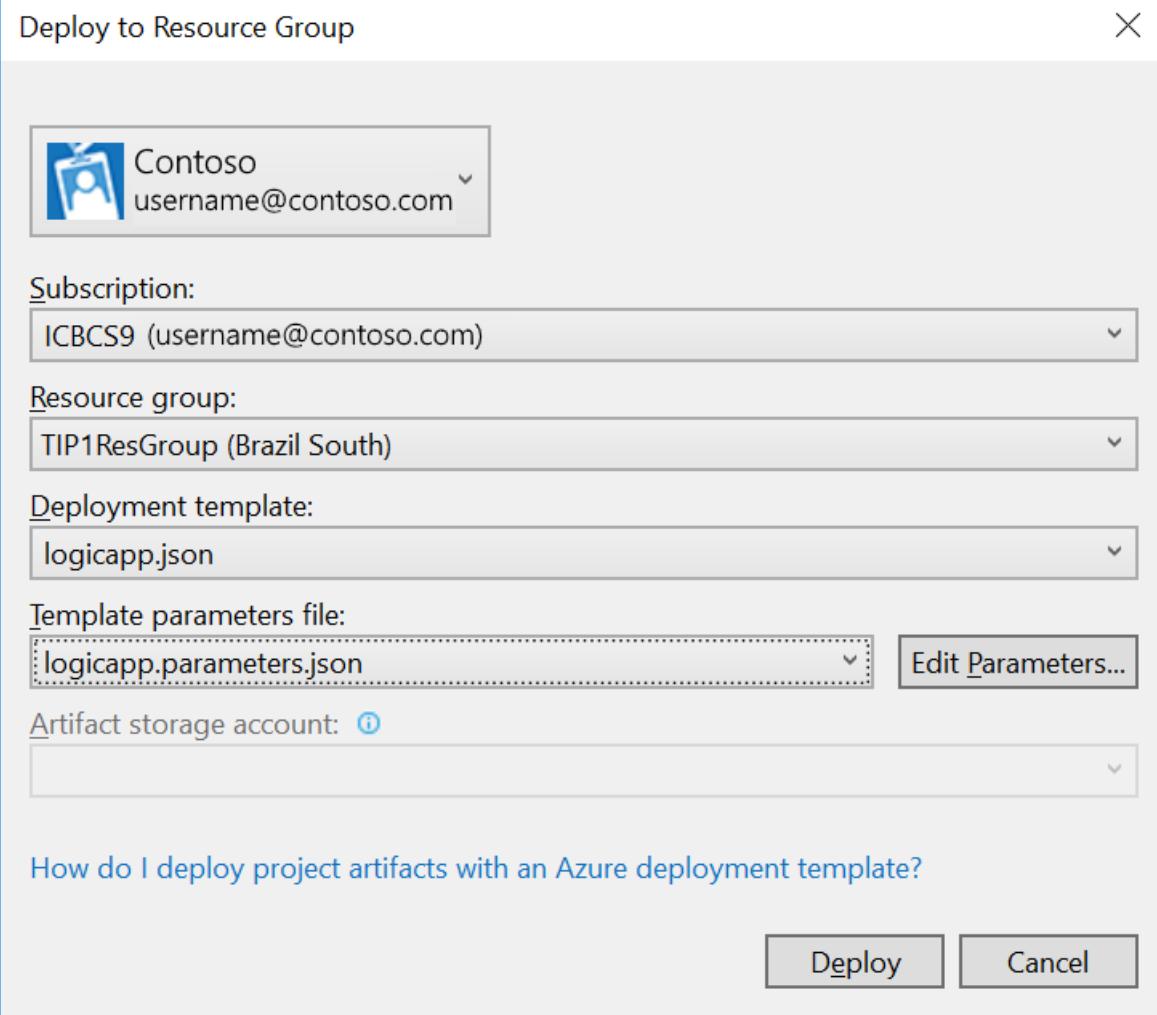
1. In Solution Explorer, right-click your project, and go to **Deploy > New Deployment...**



2. When you're prompted, sign in to your Azure subscription.
3. Now you must select the details for the resource group where you want to deploy your logic app. When you're done, select **Deploy**.

**NOTE**

Make sure that you select the correct template and parameters file for the resource group. For example, if you want to deploy to a production environment, choose the production parameters file.



The deployment status appears in the **Output** window. You might have to select **Azure Provisioning** in the **Show output from** list.

```
Output : Show output from: Azure Provisioning
05:43:57 - Transfer summary:
05:43:57 - -----
05:43:57 - Total files transferred: 3
05:43:57 - Transfer successfully: 3
05:43:57 - Transfer failed: 0
05:43:57 - Elapsed time: 00:00:00:02
05:43:57 - Done building target "UploadDrop" in project "MyLogicApp.Deployment.deployproj".
05:43:57 - Done building project "MyLogicApp.Deployment.deployproj".
05:43:57 - Build succeeded.
05:43:57 - The following parameter values will be used for this deployment:
05:43:57 -     droplocation: https://deploymentlogs.blob.core.windows.net/webapplication2
05:43:57 -     droplocationSasToken: <securestring>
05:43:57 -     webSitePackage: MyLogicApp/package.zip
05:43:57 -     webSiteName: blogordns
05:43:57 -     webSiteHostingPlanName: noentuhoneuth
05:43:57 -     webSiteLocation: westus
05:43:57 -     webSiteHostingPlanSKU: Free
05:43:57 -     webSiteHostingPlanWorkerSize: 0
05:43:57 - Starting deployment. This may take a while...
05:43:57 - Uploading template to Azure storage account 'deploymentlogs'.
05:43:58 - Creating resource group 'apiapps-c70beaef-6ec0-49a4-8215-7d55a08b7414-2015-04-11T10' in location 'westus'.
05:44:12 - Creating deployment 'VS_deploy' in resource group 'apiapps-c70beaef-6ec0-49a4-8215-7d55a08b7414-2015-04-11T10'.
```

In the future, you can edit your logic app in source control, and use Visual Studio to deploy new versions.

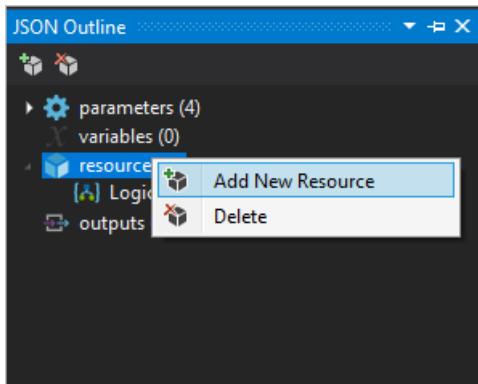
#### NOTE

If you change the definition in the Azure portal directly, those changes are overwritten when you deploy from Visual Studio next time.

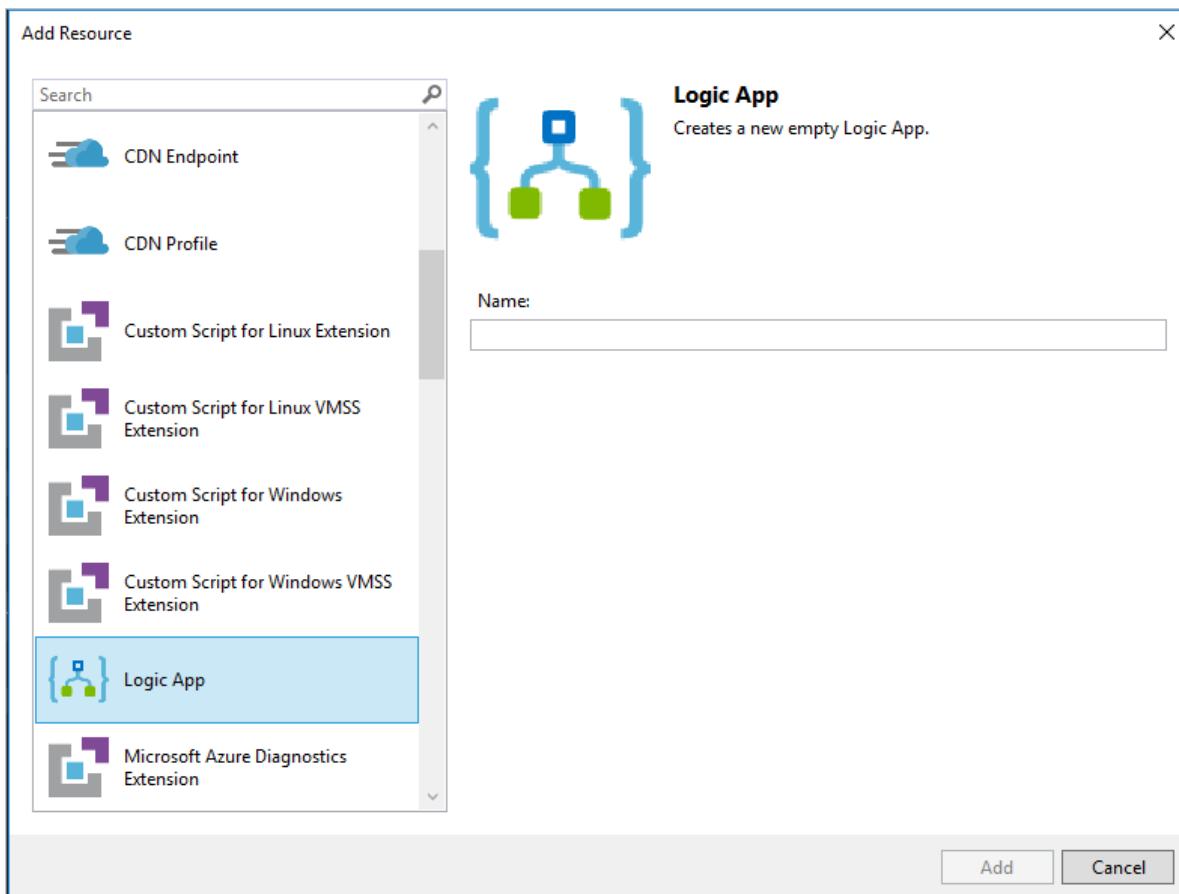
## Add your logic app to an existing Resource Group project

If you have an existing Resource Group project, you can add your logic app to that project in the JSON Outline window. You can also add another logic app alongside the app you previously created.

1. Open the <template>.json file.
2. To open the JSON Outline window, go to **View > Other Windows > JSON Outline**.
3. To add a resource to the template file, click **Add Resource** at the top of the JSON Outline window. Or in the JSON Outline window, right-click **resources**, and select **Add New Resource**.



4. In the **Add Resource** dialog box, find and select **Logic App**. Name your logic app, and choose **Add**.



## Next Steps

- [Manage logic apps with Visual Studio Cloud Explorer](#)
- [View common examples and scenarios](#)
- [Learn how to automate business processes with Azure Logic Apps](#)
- [Learn how to integrate your systems with Azure Logic Apps](#)

# Manage your logic apps with Visual Studio Cloud Explorer

3/1/2017 • 2 min to read • [Edit Online](#)

Although the [Azure portal](#) offers a great way for you to design and manage Azure Logic Apps, you can manage many Azure assets, including logic apps, in Visual Studio when you use Visual Studio Cloud Explorer. You can browse published logic apps, and perform tasks like enable and disable your logic apps or edit and view run histories.

## Installation steps

To install and configure Visual Studio tools for Azure Logic Apps, follow these steps.

### Prerequisites

- [Visual Studio 2015 or Visual Studio 2017](#)
- [Latest Azure SDK \(2.9.1 or greater\)](#)
- [Visual Studio Cloud Explorer](#)
- Access to the web when using the embedded designer

### Install Visual Studio tools for Logic Apps

After you install the prerequisites:

1. Open Visual Studio. On the **Tools** menu, select **Extensions and Updates**.
2. Expand the **Online** category so you can search online.
3. Browse or search for **Logic Apps** until you find **Azure Logic Apps Tools for Visual Studio**.
4. To download and install the extension, click **Download**.
5. Restart Visual Studio after installation.

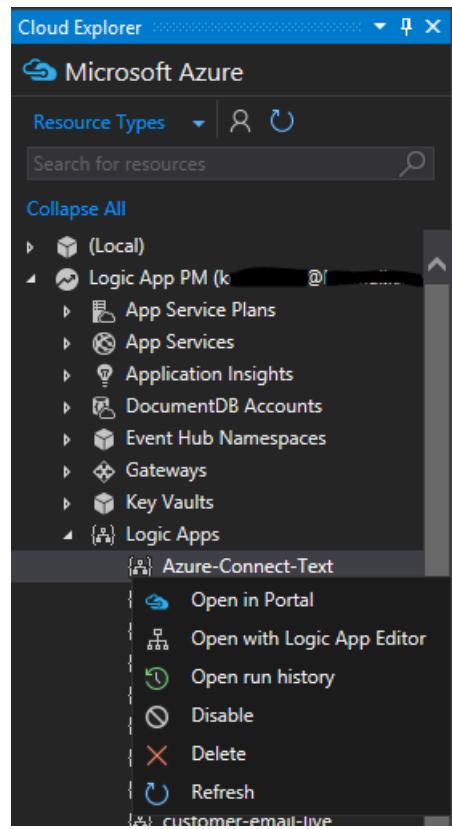
#### NOTE

You can also download Azure Logic Apps Tools for Visual Studio directly from the [Visual Studio Marketplace](#).

## Browse for logic apps in Cloud Explorer

1. To open Cloud Explorer, on the **View** menu, choose **Cloud Explorer**.
2. Browse for your logic app, either by resource group or by resource type.

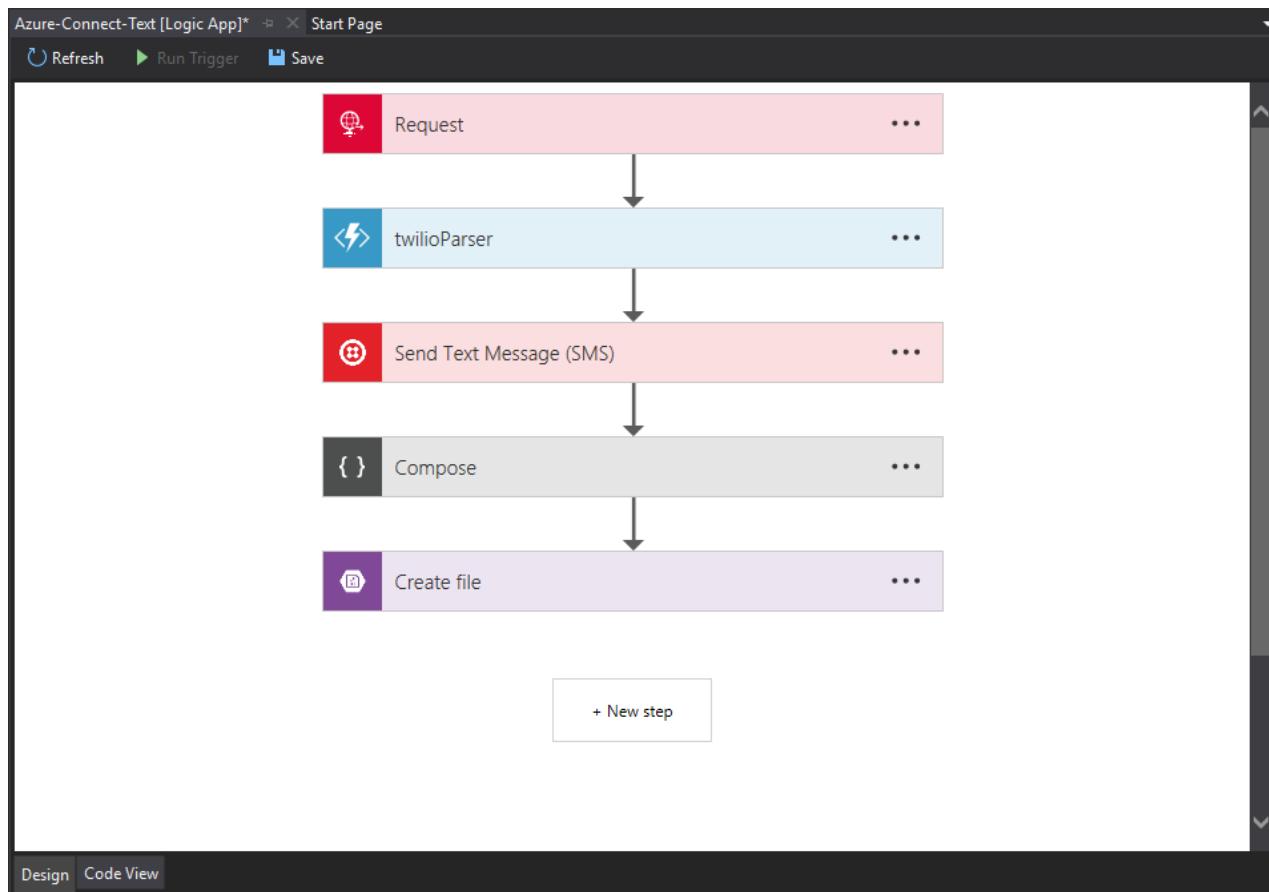
If you browse by resource type, select your Azure subscription, expand the Logic Apps section, then select a Logic App. You can either right-click a logic app, choose from the **Actions** menu at the bottom of Cloud Explorer.



## Edit your logic app with Logic App Designer

To open a currently deployed logic app in the same designer that you use in the Azure portal, right-click your logic app, and select **Open with Logic App Editor**.

In the designer, you can edit your logic app, save your updates to the cloud, and start a new run by choosing **Run Trigger**.



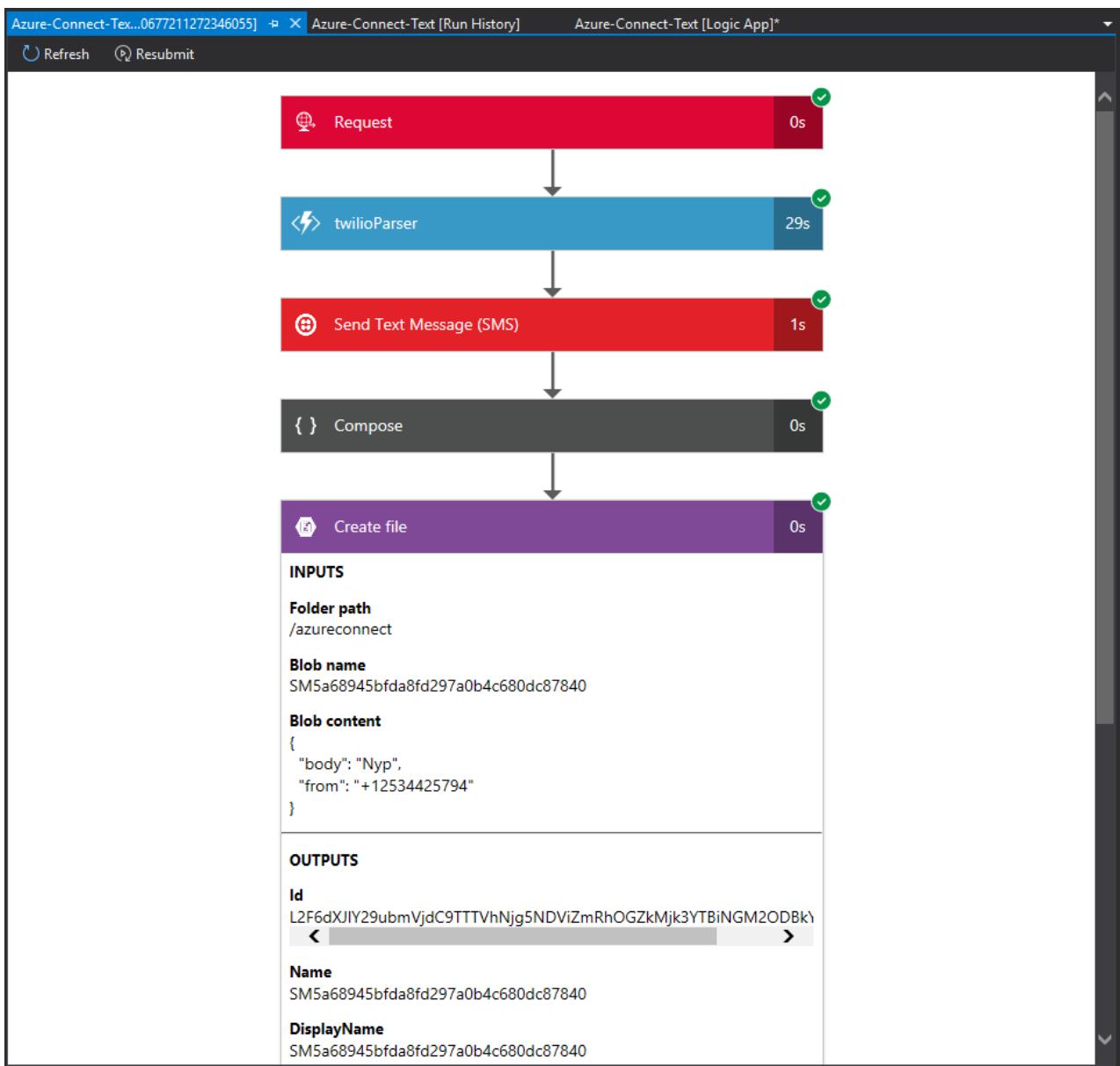
## Browse your logic app run history

To view the run history for your logic app, right-click your logic app, and select **Open run history**. To reorder your run history based on any of the properties shown, select the column header.

Azure-Connect-Text [Run History] <span style="float: right;">✖</span> Azure-Connect-Text [Logic App]*				
Status	Identifier	Start Time	End Time	Duration
<span style="color: green;">✓ Succeeded</span>	08587200335520677211272346055	12/12/2016 12:08:53 PM	12/12/2016 12:09:23 PM	30.22 Seconds
<span style="color: green;">✓ Succeeded</span>	08587209094198264575890638948	12/2/2016 8:51:05 AM	12/2/2016 8:51:32 AM	26.67 Seconds
<span style="color: green;">✓ Succeeded</span>	08587236421059776808982596931	10/31/2016 6:46:19 PM	10/31/2016 6:46:51 PM	32.3 Seconds
<span style="color: green;">✓ Succeeded</span>	08587267749235988305407694526	9/25/2016 12:32:41 PM	9/25/2016 12:33:20 PM	38.28 Seconds

4 Results

To show the run history for an instance so you can see the run results, including the inputs and outputs from each step, double-click one of the run instances.



## Next steps

- To get started with Azure Logic Apps, learn [how to create your first logic app in the Azure portal](#)
- [Design, build, and deploy logic apps in Visual Studio](#)
- [View common examples and scenarios](#)
- [Learn how to automate business processes with Azure Logic Apps](#)
- [Learn how to integrate your systems with Azure Logic Apps](#)

# Check the performance, and start diagnostic logging and alerts of your workflows in logic apps

2/28/2017 • 4 min to read • [Edit Online](#)

After you [create a logic app](#), you can see the full history of its execution in the Azure portal. You can also set up services like Azure Diagnostics and Azure Alerts to monitor events real-time, and alert you for events like "when more than 5 runs fail within an hour."

## Monitor in the Azure Portal

To view the history, select **Browse**, and select **Logic Apps**. A list of all logic apps in your subscription is displayed. Select the logic app you want to monitor. You will see a list of all actions and triggers that have occurred for this logic app.

The screenshot shows the Azure Logic App Overview blade. On the left, there's a sidebar with 'QUICK ACCESS' (Activity logs, Access control (IAM), Tags), 'SETTINGS' (Locks, Automation script), 'GENERAL' (Logic App Definition, Diagnostics, Properties, Quick Start Guides, Integration Account), and 'SUPPORT + TROUBLESHOOTING' (New support request). The main area has a 'Summary' tab selected, showing a table of 'All runs' with columns for STATUS, START TIME, and DURATION. Below it is a 'Trigger History' table with columns for STATUS, START TIME, and FIRED.

STATUS	START TIME	DURATION
Succeeded	7/22/2016, 7:48 PM	248 Milliseconds
Succeeded	7/22/2016, 7:48 PM	248 Milliseconds
Succeeded	7/22/2016, 7:48 PM	294 Milliseconds
Succeeded	7/22/2016, 7:48 PM	275 Milliseconds
Succeeded	7/22/2016, 7:48 PM	249 Milliseconds
Succeeded	7/22/2016, 7:48 PM	275 Milliseconds
Succeeded	7/22/2016, 7:48 PM	260 Milliseconds
Succeeded	7/22/2016, 7:48 PM	331 Milliseconds

STATUS	START TIME	FIRED
Succeeded	7/22/2016, 7:48 PM	Fired
Succeeded	7/22/2016, 7:48 PM	Fired
Succeeded	7/22/2016, 7:48 PM	
Succeeded	7/22/2016, 7:48 PM	Fired

There are a few sections on this blade that are helpful:

- **Summary** lists **All runs** and the **Trigger History**
  - **All runs** list the latest logic app runs. You can click any row for details on the run, or click on the tile to list more runs.
  - **Trigger History** lists all the trigger activity for this logic app. Trigger activity could be a "Skipped" check for new data (e.g. looking to see if a new file was added to FTP), "Succeeded" meaning data was returned to fire a logic app, or "Failed" corresponding an error in configuration.

- **Diagnostics** allows you to view runtime details and events, and subscribe to [Azure Alerts](#)

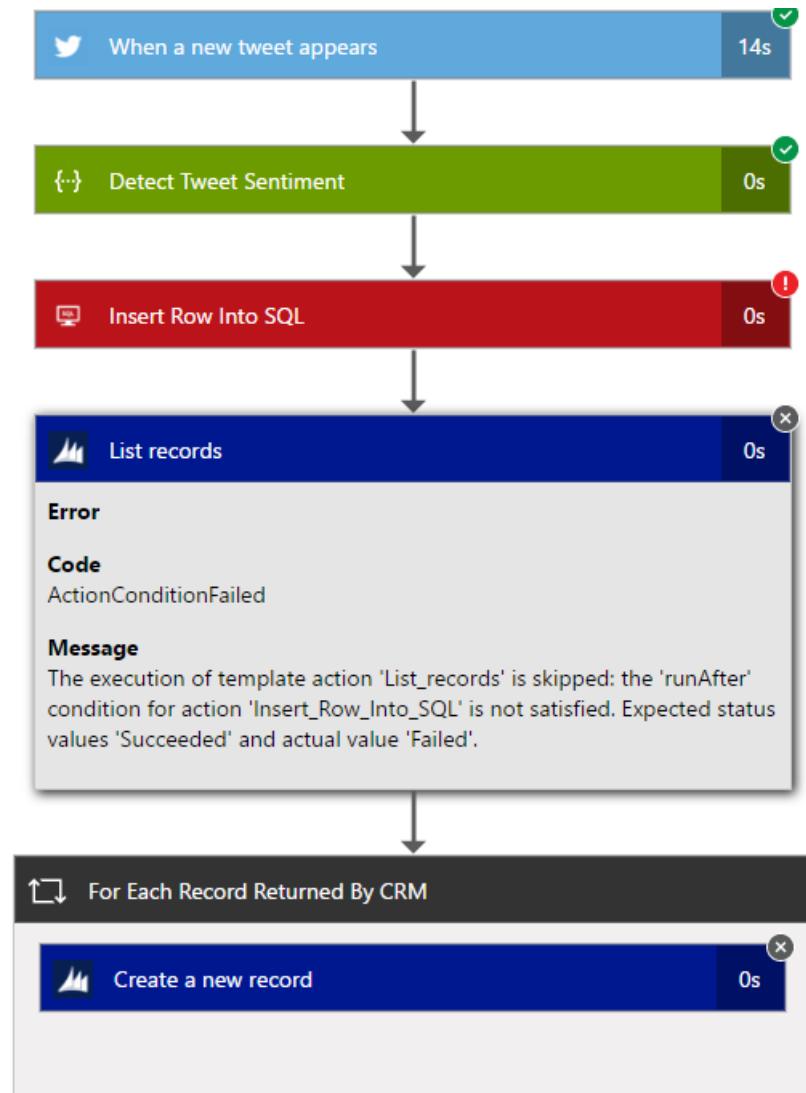
**NOTE**

All runtime details and events are encrypted at rest within the Logic App service. They are only decrypted upon a view request from a user. Access to these events can also be controlled by Azure Role-Based Access Control (RBAC).

## View the run details

This list of runs shows the **Status**, the **Start Time**, and the **Duration** of the particular run. Select any row to see details on that run.

The monitoring view shows you each step of the run, the inputs and outputs, and any error messages that may have occurred.



If you need any additional details like the run **Correlation ID** (that can be used for the REST API), you can click the **Run Details** button. This includes all steps, status, and inputs/outputs for the run.

## Azure Diagnostics and alerts

In addition to the details provided by the Azure Portal and REST API above, you can configure your logic app to use Azure Diagnostics for more rich details and debugging.

1. Click the **Diagnostics** section of the logic app blade
2. Click to configure the **Diagnostic Settings**
3. Configure an Event Hub or Storage Account to emit data to

**Diagnostics**

Save Discard

Status

Export to Event Hubs

---

Service Bus Namespace >

---

Export to Storage Account

---

Storage Account >

---

LOGS

WorkflowRuntime Retention (days) ⓘ  30

---

METRICS

1 minute Retention (days) ⓘ  5

 The storage account must be in the same region as the resource.

## Adding Azure Alerts

Once diagnostics are configured, you can add Azure Alerts to fire when certain thresholds are crossed. In the **Diagnostics** blade, select the **Alerts** tile and **Add alert**. This will walk you through configuring an alert based on a number of thresholds and metrics.

\* Name ⓘ

Description

\* Metric ⓘ

Action Latency

- Action Latency
- Action Success Latency
- Action Throttled Events
- Actions Completed
- Actions Failed
- Actions Skipped
- Actions Started
- Actions Succeeded
- Run Latency
- Run Success Latency
- Run Throttled Events
- Runs Cancelled
- Runs Completed
- Runs Failed**
- Runs Started
- Runs Succeeded
- Trigger Fire Latency
- Trigger Latency
- Trigger Success Latency
- Trigger Throttled Events

\* Period ⓘ

Over the last 5 minutes

You can configure the **Condition**, **Threshold**, and **Period** as desired. Finally, you can configure an email address to send a notification to, or configure a webhook. You can use the [request trigger](#) in a logic app to run on an alert as well (to do things like [post to Slack](#), [send a text](#), or [add a message to a queue](#)).

### Azure Diagnostics Settings

Each of these events contains details about the logic app and event like status. Here is an example of a *ActionCompleted* event:

```
{
    "time": "2016-07-09T17:09:54.4773148Z",
    "workflowId": "/SUBSCRIPTIONS/80D4FE69-ABCD-EFGH-A938-
9250F1C8AB03/RESOURCEGROUPS/MYRESOURCEGROUP/PROVIDERS/MICROSOFT.LOGIC/WORKFLOWS/MYLOGICAPP",
    "resourceId": "/SUBSCRIPTIONS/80D4FE69-ABCD-EFGH-A938-
9250F1C8AB03/RESOURCEGROUPS/MYRESOURCEGROUP/PROVIDERS/MICROSOFT.LOGIC/WORKFLOWS/MYLOGICAPP/RUNS/0858736114692
2712057/ACTIONS/HTTP",
    "category": "WorkflowRuntime",
    "level": "Information",
    "operationName": "Microsoft.Logic/workflows/workflowActionCompleted",
    "properties": {
        "$schema": "2016-06-01",
        "startTime": "2016-07-09T17:09:53.4336305Z",
        "endTime": "2016-07-09T17:09:53.5430281Z",
        "status": "Succeeded",
        "code": "OK",
        "resource": {
            "subscriptionId": "80d4fe69-ABCD-EFGH-a938-9250f1c8ab03",
            "resourceGroupName": "MyResourceGroup",
            "workflowId": "cff00d5458f944d5a766f2f9ad142553",
            "workflowName": "MyLogicApp",
            "runId": "08587361146922712057",
            "location": "eastus",
            "actionName": "Http"
        },
        "correlation": {
            "actionTrackingId": "e1931543-906d-4d1d-baed-dee72ddf1047",
            "clientTrackingId": "my-custom-tracking-id"
        },
        "trackedProperties": {
            "myProperty": "<value>"
        }
    }
}
```

The two properties that are especially useful for tracking and monitoring are *clientTrackingId* and *trackedProperties*.

#### **Client tracking ID**

The client tracking ID is a value that will correlate events across a logic app run, including any nested workflows called as a part of a logic app. This ID will be auto-generated if not provided, but you can manually specify the client tracking ID from a trigger by passing a `x-ms-client-tracking-id` header with the ID value in the trigger request (request trigger, HTTP trigger, or webhook trigger).

#### **Tracked properties**

Tracked properties can be added onto actions in the workflow definition to track inputs or outputs in diagnostics data. This can be useful if you wish to track data like an "order ID" in your telemetry. To add a tracked property, include the `trackedProperties` property on an action. Tracked properties can only track a single actions inputs and outputs, but you can use the `correlation` properties of the events to correlate across actions in a run.

```
{  
    "myAction": {  
        "type": "http",  
        "inputs": {  
            "uri": "http://uri",  
            "headers": {  
                "Content-Type": "application/json"  
            },  
            "body": "@triggerBody()"  
        },  
        "trackedProperties":{  
            "myActionHTTPStatusCode": "@action()['outputs']['statusCode']",  
            "myActionHTTPValue": "@action()['outputs']['body']['foo']",  
            "transactionId": "@action()['inputs']['body']['bar']"  
        }  
    }  
}
```

## Extending your solutions

You can leverage this telemetry from the Event Hub or Storage into other services like [Operations Management Suite](#), [Azure Stream Analytics](#), and [Power BI](#) to have real time monitoring of your integration workflows.

## Next Steps

- [Common examples and scenarios for logic apps](#)
- [Creating a Logic App Deployment Template](#)
- [Enterprise integration features](#)

4 min to read •

# Start or enable logging of AS2, X12, and EDIFACT messages to monitor success, errors, and message properties

2/28/2017 • 1 min to read • [Edit Online](#)

B2B communication involves message exchanges between two running business processes or applications. The relationship defines an agreement between business processes. After communication is established, you can set up message monitoring to check that communication is working as expected. For richer details and debugging, you can set up diagnostics for your integration account.

Message tracking is available for these B2B protocols: AS2, X12, and EDIFACT.

## Prerequisites

- An Azure account; you can create a [free account](#).
- An Integration Account; you can create an [Integration Account](#).
- A Logic App; you can create a [Logic App](#) and [enable logging](#).

## Enable logging for an integration account

You can enable logging for an integration account either with the **Azure portal** or with **Monitor**.

### Enable logging with Azure portal

1. Select your integration account, then select **Diagnostics logs**.

The screenshot shows the Azure portal interface for managing an Integration Account. The left pane displays a list of resources under 'All resources' for 'Microsoft'. A specific resource, 'IntegrationAccount', is highlighted and selected. The right pane provides detailed configuration options for this account, including monitoring and diagnostic settings.

2. Select your **Subscription** and **Resource Group**. From **Resource Type**, select **Integration Accounts**. From **Resource**, select your integration account. Click **Turn on Diagnostics** to enable diagnostics for your selected integration account.

This screenshot shows the 'Gain insights across Azure resources' blade. It focuses on the 'Turn on diagnostics' section, which lists the logs to be collected. The 'Resource type' is set to 'Integration Accounts' and the 'Resource' is set to 'IntegrationAccount'. The 'Subscription', 'Resource group', and 'Log search' fields are also present.

3. Select status **ON**.

This screenshot shows the 'Diagnostics settings' blade. It includes a warning message: 'Turning off diagnostics will disable monitoring charts and alerts for your resource.' The 'Status' section contains two buttons: 'On' (highlighted with a red box) and 'Off'.

4. Select **Send to Log Analytics** and configure Log Analytics to emit data.

Diagnostics settings

Save Discard

Status

On  Off

Archive to a storage account

Stream to an event hub

Send to Log Analytics

Log Analytics Configure

LOG

IntegrationAccountTrackingEvents

OMS Workspaces

- eastus
- East US
- [REDACTED]
- eastus
- [REDACTED]
- East US
- [REDACTED]
- southeastasia
- [REDACTED]
- australiasoutheast
- LogicAppTracking** East US
- [REDACTED]
- australiasoutheast

## Enable logging with Monitor

1. Select **Monitor, Diagnostics logs**.

☰

+ New

Resource groups

All resources

Recent

App Services

Virtual machines (classic)

Virtual machines

SQL databases

Cloud services (classic)

Subscriptions

Azure Active Directory

**Monitor**

Security Center

Billing

Help + support

Monitor - Diagnostics logs  
Microsoft - PREVIEW

Search (Ctrl+ /)

EXPLORE

Activity log

Metrics

**Diagnostics logs**

Log search

MANAGE

Alerts

ADVANCED

Application Insights

Management solutions

2. Select your **Subscription** and **Resource Group**. From **Resource Type**, select **Integration Accounts**. From **Resource**, select your integration account. Click **Turn on Diagnostics** to enable diagnostics for your selected integration account.

The screenshot shows the Azure Log Search interface. At the top, there are four dropdown menus: 'Subscription' (selected), 'Resource group' (EIPTemplates), 'Resource type' (Integration Accounts), and 'Resource' (IntegrationAccount). Below these, a breadcrumb navigation shows 'BTS4 > EIPTemplates > IntegrationAccount'. A red box highlights the 'Turn on diagnostic' link, which collects logs for 'IntegrationAccountTrackingEvents'.

3. Select status **ON**.

The screenshot shows the 'Diagnostics settings' dialog box. It has a warning message: 'Turning off diagnostics will disable monitoring charts and alerts for your resource.' Below it, a 'Status' section has two buttons: 'On' (highlighted with a red box) and 'Off'.

4. Select **Send to Log Analytics** and configure **Log Analytics** to emit data.

The screenshot shows the 'Diagnostics settings' dialog box with the 'Send to Log Analytics' checkbox checked (highlighted with a red box). The 'Log Analytics' section is expanded, showing a list of workspaces: eastus, [REDACTED] East US, [REDACTED] eastus, [REDACTED] East US, [REDACTED] southeastasia, [REDACTED] australiasoutheast, LogicAppTracking East US, and [REDACTED] australiasoutheast. Below this, the 'LOG' section shows 'IntegrationAccountTrackingEvents' selected (highlighted with a red box).

## Extend your solutions

In addition to the **Log Analytics**, you can configure your integration account and [Logic Apps](#) to an Event Hub or Storage Account.

Diagnostics

Save Discard

Status

Off On

Export to Event Hubs

Service Bus Namespace >

Export to Storage Account

Storage Account >

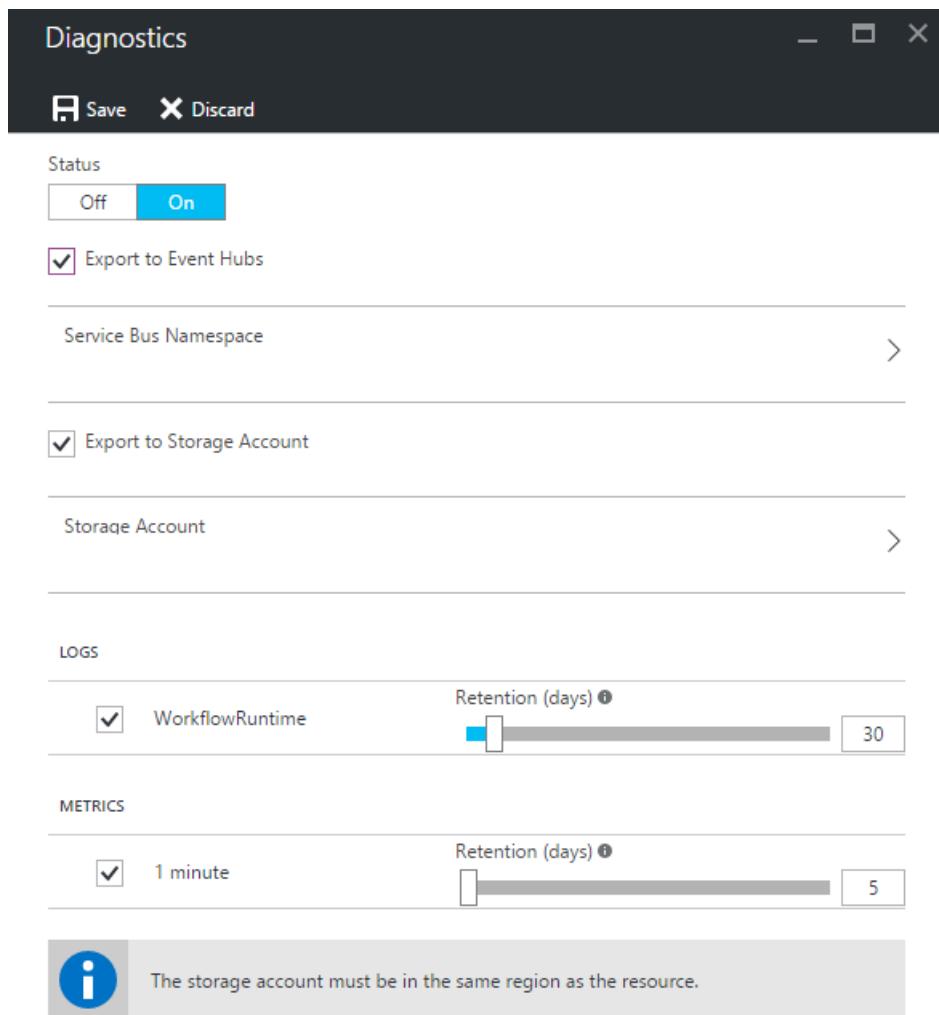
LOGS

WorkflowRuntime Retention (days) 30

METRICS

1 minute Retention (days) 5

 The storage account must be in the same region as the resource.



You can use this telemetry from the Event Hub or Storage into other services like [Azure Stream Analytics](#), and [Power BI](#) to have real-time monitoring of your integration workflows.

## Supported Tracking Schema

We support these tracking schema types, which all have fixed schemas except the Custom type.

- [Custom Tracking Schema](#)
- [AS2 Tracking Schema](#)
- [X12 Tracking Schema](#)

## Next steps

[Tracking B2B messages in OMS Portal](#)

[Learn more about the Enterprise Integration Pack](#)

# Start or enable tracking of AS2 messages and MDNs to monitor success, errors, and message properties

2/28/2017 • 2 min to read • [Edit Online](#)

You can use these AS2 tracking schemas in your Azure integration account to help you monitor business-to-business (B2B) transactions:

- AS2 message tracking schema
- AS2 MDN tracking schema

## AS2 message tracking schema

```
{  
    "agreementProperties": {  
        "senderPartnerName": "",  
        "receiverPartnerName": "",  
        "as2To": "",  
        "as2From": "",  
        "agreementName": ""  
    },  
    "messageProperties": {  
        "direction": "",  
        "messageId": "",  
        "dispositionType": "",  
        "fileName": "",  
        "isMessageFailed": "",  
        "isMessageSigned": "",  
        "isMessageEncrypted": "",  
        "isMessageCompressed": "",  
        "correlationMessageId": "",  
        "incomingHeaders": {  
        },  
        "outgoingHeaders": {  
        },  
        "isNrrEnabled": "",  
        "isMdnExpected": "",  
        "mdnType": ""  
    }  
}
```

PROPERTY	TYPE	DESCRIPTION
senderPartnerName	String	AS2 message sender's partner name. (Optional)
receiverPartnerName	String	AS2 message receiver's partner name. (Optional)
as2To	String	AS2 message receiver's name, from the headers of the AS2 message. (Mandatory)

PROPERTY	TYPE	DESCRIPTION
as2From	String	AS2 message sender's name, from the headers of the AS2 message. (Mandatory)
agreementName	String	Name of the AS2 agreement to which the messages are resolved. (Optional)
direction	String	Direction of the message flow, receive or send. (Mandatory)
messageId	String	AS2 message ID, from the headers of the AS2 message (Optional)
dispositionType	String	Message Disposition Notification (MDN) disposition type value. (Optional)
fileName	String	File name, from the header of the AS2 message. (Optional)
isMessageFailed	Boolean	Whether the AS2 message failed. (Mandatory)
isMessageSigned	Boolean	Whether the AS2 message was signed. (Mandatory)
isMessageEncrypted	Boolean	Whether the AS2 message was encrypted. (Mandatory)
isMessageCompressed	Boolean	Whether the AS2 message was compressed. (Mandatory)
correlationMessageId	String	AS2 message ID, to correlate messages with MDNs. (Optional)
incomingHeaders	Dictionary of JToken	Incoming AS2 message header details. (Optional)
outgoingHeaders	Dictionary of JToken	Outgoing AS2 message header details. (Optional)
isNrrEnabled	Boolean	Use default value if the value is not known. (Mandatory)
isMdnExpected	Boolean	Use default value if the value is not known. (Mandatory)
mdnType	Enum	Allowed values are <b>NotConfigured</b> , <b>Sync</b> , and <b>Async</b> . (Mandatory)

## AS2 MDN tracking schema

```

{
    "agreementProperties": {
        "senderPartnerName": "",
        "receiverPartnerName": "",
        "as2To": "",
        "as2From": "",
        "agreementName": "g"
    },
    "messageProperties": {
        "direction": "",
        "messageId": "",
        "originalMessageId": "",
        "dispositionType": "",
        "isMessageFailed": "",
        "isMessageSigned": "",
        "isNrrEnabled": "",
        "statusCode": "",
        "micVerificationStatus": "",
        "correlationMessageId": "",
        "incomingHeaders": {
        },
        "outgoingHeaders": {
        }
    }
}

```

PROPERTY	TYPE	DESCRIPTION
senderPartnerName	String	AS2 message sender's partner name. (Optional)
receiverPartnerName	String	AS2 message receiver's partner name. (Optional)
as2To	String	Partner name who receives the AS2 message. (Mandatory)
as2From	String	Partner name who sends the AS2 message. (Mandatory)
agreementName	String	Name of the AS2 agreement to which the messages are resolved. (Optional)
direction	String	Direction of the message flow, receive or send. (Mandatory)
messageld	String	AS2 message ID. (Optional)
originalMessageId	String	AS2 original message ID. (Optional)
dispositionType	String	MDN disposition type value. (Optional)
isMessageFailed	Boolean	Whether the AS2 message failed. (Mandatory)
isMessageSigned	Boolean	Whether the AS2 message was signed. (Mandatory)

PROPERTY	TYPE	DESCRIPTION
isNrrEnabled	Boolean	Use default value if the value is not known. (Mandatory)
statusCode	Enum	Allowed values are <b>Accepted</b> , <b>Rejected</b> , and <b>AcceptedWithErrors</b> . (Mandatory)
micVerificationStatus	Enum	Allowed values are <b>NotApplicable</b> , <b>Succeeded</b> , and <b>Failed</b> . (Mandatory)
correlationMessageId	String	Correlation ID. The original message ID (the message ID of the message for which MDN is configured). (Optional)
incomingHeaders	Dictionary of JToken	Indicates incoming message header details. (Optional)
outgoingHeaders	Dictionary of JToken	Indicates outgoing message header details. (Optional)

## Next steps

- Learn more about the [Enterprise Integration Pack](#).
- Learn more about [monitoring B2B messages](#).
- Learn more about [B2B custom tracking schemas](#).
- Learn more about [X12 tracking schemas](#).
- Learn about [tracking B2B messages in the Operations Management Suite portal](#).

# Start or enable tracking of X12 messages to monitor success, errors, and message properties

2/28/2017 • 7 min to read • [Edit Online](#)

You can use these X12 tracking schemas in your Azure integration account to help you monitor business-to-business (B2B) transactions:

- X12 transaction set tracking schema
- X12 transaction set acknowledgement tracking schema
- X12 interchange tracking schema
- X12 interchange acknowledgement tracking schema
- X12 functional group tracking schema
- X12 functional group acknowledgement tracking schema

## X12 transaction set tracking schema

```
{  
    "agreementProperties": {  
        "senderPartnerName": "",  
        "receiverPartnerName": "",  
        "senderQualifier": "",  
        "senderIdentifier": "",  
        "receiverQualifier": "",  
        "receiverIdentifier": "",  
        "agreementName": ""  
    },  
    "messageProperties": {  
        "direction": "",  
        "interchangeControlNumber": "",  
        "functionalGroupControlNumber": "",  
        "transactionSetControlNumber": "",  
        "CorrelationMessageId": "",  
        "messageType": "",  
        "isMessageFailed": "",  
        "isTechnicalAcknowledgmentExpected": "",  
        "isFunctionalAcknowledgmentExpected": "",  
        "needAk2LoopForValidMessages": "",  
        "segmentsCount": ""  
    }  
}
```

PROPERTY	TYPE	DESCRIPTION
senderPartnerName	String	X12 message sender's partner name. (Optional)
receiverPartnerName	String	X12 message receiver's partner name. (Optional)
senderQualifier	String	Send partner qualifier. (Mandatory)

PROPERTY	TYPE	DESCRIPTION
senderIdentifier	String	Send partner identifier. (Mandatory)
receiverQualifier	String	Receive partner qualifier. (Mandatory)
receiverIdentifier	String	Receive partner identifier. (Mandatory)
agreementName	String	Name of the X12 agreement to which the messages are resolved. (Optional)
direction	Enum	Direction of the message flow, receive or send. (Mandatory)
interchangeControlNumber	String	Interchange control number. (Optional)
functionalGroupControlNumber	String	Functional control number. (Optional)
transactionSetControlNumber	String	Transaction set control number. (Optional)
CorrelationMessageId	String	Correlation message ID. A combination of {AgreementName} {GroupControlNumber} {TransactionSetControlNumber}. (Optional)
messageType	String	Transaction set or document type. (Optional)
isMessageFailed	Boolean	Whether the X12 message failed. (Mandatory)
isTechnicalAcknowledgmentExpected	Boolean	Whether the technical acknowledgement is configured in the X12 agreement. (Mandatory)
isFunctionalAcknowledgmentExpected	Boolean	Whether the functional acknowledgement is configured in the X12 agreement. (Mandatory)
needAk2LoopForValidMessages	Boolean	Whether the AK2 loop is required for a valid message. (Mandatory)
segmentsCount	Integer	Number of segments in the X12 transaction set. (Optional)

## X12 transaction set acknowledgement tracking schema

```

{
    "agreementProperties": {
        "senderPartnerName": "",
        "receiverPartnerName": "",
        "senderQualifier": "",
        "senderIdentifier": "",
        "receiverQualifier": "",
        "receiverIdentifier": "",
        "agreementName": ""
    },
    "messageProperties": {
        "direction": "",
        "interchangeControlNumber": "",
        "functionalGroupControlNumber": "",
        "isaSegment": "",
        "gsSegment": "",
        "respondingFunctionalGroupControlNumber": "",
        "respondingFunctionalGroupId": "",
        "respondingTransactionSetControlNumber": "",
        "respondingTransactionSetId": "",
        "statusCode": "",
        "processingStatus": "",
        "CorrelationMessageId": "",
        "isMessageFailed": "",
        "ak2Segment": "",
        "ak3Segment": "",
        "ak5Segment": ""
    }
}

```

PROPERTY	TYPE	DESCRIPTION
senderPartnerName	String	X12 message sender's partner name. (Optional)
receiverPartnerName	String	X12 message receiver's partner name. (Optional)
senderQualifier	String	Send partner qualifier. (Mandatory)
senderIdentifier	String	Send partner identifier. (Mandatory)
receiverQualifier	String	Receive partner qualifier. (Mandatory)
receiverIdentifier	String	Receive partner identifier. (Mandatory)
agreementName	String	Name of the X12 agreement to which the messages are resolved. (Optional)
direction	Enum	Direction of the message flow, receive or send. (Mandatory)
interchangeControlNumber	String	Interchange control number of the functional acknowledgement. Value populates only for the send side where functional acknowledgement is received for the messages sent to partner. (Optional)

PROPERTY	TYPE	DESCRIPTION
functionalGroupControlNumber	String	Functional group control number of the functional acknowledgement. Value populates only for the send side where functional acknowledgement is received for the messages sent to partner. (Optional)
isaSegment	String	ISA segment of the message. Value populates only for the send side where functional acknowledgement is received for the messages sent to partner. (Optional)
gsSegment	String	GS segment of the message. Value populates only for the send side where functional acknowledgement is received for the messages sent to partner. (Optional)
respondingfunctionalGroupControlNumber	String	Responding interchange control number. (Optional)
respondingFunctionalGroupId	String	Responding functional group ID, which maps to AK101 in the acknowledgement. (Optional)
respondingtransactionSetControlNumber	String	Responding transaction set control number. (Optional)
respondingTransactionSetId	String	Responding transaction set ID, which maps to AK201 in the acknowledgement. (Optional)
statusCode	Boolean	Transaction set acknowledgement status code. (Mandatory)
segmentsCount	Enum	Acknowledgement status code. Allowed values are <b>Accepted</b> , <b>Rejected</b> , and <b>AcceptedWithErrors</b> . (Mandatory)
processingStatus	Enum	Processing status of the acknowledgement. Allowed values are <b>Received</b> , <b>Generated</b> , and <b>Sent</b> . (Mandatory)
CorrelationMessageId	String	Correlation message ID. A combination of {AgreementName} {GroupControlNumber} {TransactionSetControlNumber}. (Optional)
isMessageFailed	Boolean	Whether the X12 message failed. (Mandatory)

PROPERTY	TYPE	DESCRIPTION
ak2Segment	String	Acknowledgement for a transaction set within the received functional group. (Optional)
ak3Segment	String	Reports errors in a data segment. (Optional)
ak5Segment	String	Reports whether the transaction set identified in the AK2 segment is accepted or rejected, and why. (Optional)

## X12 interchange tracking schema

```
{
  "agreementProperties": {
    "senderPartnerName": "",
    "receiverPartnerName": "",
    "senderQualifier": "",
    "senderIdentifier": "",
    "receiverQualifier": "",
    "receiverIdentifier": "",
    "agreementName": ""
  },
  "messageProperties": {
    "direction": "",
    "interchangeControlNumber": "",
    "isaSegment": "",
    "isTechnicalAcknowledgmentExpected": "",
    "isMessageFailed": "",
    "isa09": "",
    "isa10": "",
    "isa11": "",
    "isa12": "",
    "isa14": "",
    "isa15": "",
    "isa16": ""
  }
}
```

PROPERTY	TYPE	DESCRIPTION
senderPartnerName	String	X12 message sender's partner name. (Optional)
receiverPartnerName	String	X12 message receiver's partner name. (Optional)
senderQualifier	String	Send partner qualifier. (Mandatory)
senderIdentifier	String	Send partner identifier. (Mandatory)
receiverQualifier	String	Receive partner qualifier. (Mandatory)
receiverIdentifier	String	Receive partner identifier. (Mandatory)

PROPERTY	TYPE	DESCRIPTION
agreementName	String	Name of the X12 agreement to which the messages are resolved. (Optional)
direction	Enum	Direction of the message flow, receive or send. (Mandatory)
interchangeControlNumber	String	Interchange control number. (Optional)
isaSegment	String	Message ISA segment. (Optional)
isTechnicalAcknowledgmentExpected	Boolean	Whether the technical acknowledgement is configured in the X12 agreement. (Mandatory)
isMessageFailed	Boolean	Whether the X12 message failed. (Mandatory)
isa09	String	X12 document interchange date. (Optional)
isa10	String	X12 document interchange time. (Optional)
isa11	String	X12 interchange control standards identifier. (Optional)
isa12	String	X12 interchange control version number. (Optional)
isa14	String	X12 acknowledgement is requested. (Optional)
isa15	String	Indicator for test or production. (Optional)
isa16	String	Element separator. (Optional)

## X12 interchange acknowledgement tracking schema

```

{
  "agreementProperties": {
    "senderPartnerName": "",
    "receiverPartnerName": "",
    "senderQualifier": "",
    "senderIdentifier": "",
    "receiverQualifier": "",
    "receiverIdentifier": "",
    "agreementName": ""
  },
  "messageProperties": {
    "direction": "",
    "interchangeControlNumber": "",
    "isaSegment": "",
    "respondingInterchangeControlNumber": "",
    "isMessageFailed": "",
    "statusCode": "",
    "processingStatus": "",
    "ta102": "",
    "ta103": "",
    "ta105": ""
  }
}

```

PROPERTY	TYPE	DESCRIPTION
senderPartnerName	String	X12 message sender's partner name. (Optional)
receiverPartnerName	String	X12 message receiver's partner name. (Optional)
senderQualifier	String	Send partner qualifier. (Mandatory)
senderIdentifier	String	Send partner identifier. (Mandatory)
receiverQualifier	String	Receive partner qualifier. (Mandatory)
receiverIdentifier	String	Receive partner identifier. (Mandatory)
agreementName	String	Name of the X12 agreement to which the messages are resolved. (Optional)
direction	Enum	Direction of the message flow, receive or send. (Mandatory)
interchangeControlNumber	String	Interchange control number of the technical acknowledgement that's received from partners. (Optional)
isaSegment	String	ISA segment for the technical acknowledgement that's received from partners. (Optional)
respondingInterchangeControlNumber	String	Interchange control number for the technical acknowledgement that's received from partners. (Optional)

PROPERTY	TYPE	DESCRIPTION
isMessageFailed	Boolean	Whether the X12 message failed. (Mandatory)
statusCode	Enum	Interchange acknowledgement status code. Allowed values are <b>Accepted</b> , <b>Rejected</b> , and <b>AcceptedWithErrors</b> . (Mandatory)
processingStatus	Enum	Acknowledgement status. Allowed values are <b>Received</b> , <b>Generated</b> , and <b>Sent</b> . (Mandatory)
ta102	String	Interchange date. (Optional)
ta103	String	Interchange time. (Optional)
ta105	String	Interchange note code. (Optional)

## X12 functional group tracking schema

```
{
  "agreementProperties": {
    "senderPartnerName": "",
    "receiverPartnerName": "",
    "senderQualifier": "",
    "senderIdentifier": "",
    "receiverQualifier": "",
    "receiverIdentifier": "",
    "agreementName": ""
  },
  "messageProperties": {
    "direction": "",
    "interchangeControlNumber": "",
    "functionalGroupControlNumber": "",
    "gsSegment": "",
    "isTechnicalAcknowledgmentExpected": "",
    "isFunctionalAcknowledgmentExpected": "",
    "isMessageFailed": "",
    "gs01": "",
    "gs02": "",
    "gs03": "",
    "gs04": "",
    "gs05": "",
    "gs07": "",
    "gs08": ""
  }
}
```

PROPERTY	TYPE	DESCRIPTION
senderPartnerName	String	X12 message sender's partner name. (Optional)
receiverPartnerName	String	X12 message receiver's partner name. (Optional)

PROPERTY	TYPE	DESCRIPTION
senderQualifier	String	Send partner qualifier. (Mandatory)
senderIdentifier	String	Send partner identifier. (Mandatory)
receiverQualifier	String	Receive partner qualifier. (Mandatory)
receiverIdentifier	String	Receive partner identifier. (Mandatory)
agreementName	String	Name of the X12 agreement to which the messages are resolved. (Optional)
direction	Enum	Direction of the message flow, receive or send. (Mandatory)
interchangeControlNumber	String	Interchange control number. (Optional)
functionalGroupControlNumber	String	Functional control number. (Optional)
gsSegment	String	Message GS segment. (Optional)
isTechnicalAcknowledgmentExpected	Boolean	Whether the technical acknowledgement is configured in the X12 agreement. (Mandatory)
isFunctionalAcknowledgmentExpected	Boolean	Whether the functional acknowledgement is configured in the X12 agreement. (Mandatory)
isMessageFailed	Boolean	Whether the X12 message failed. (Mandatory)
gs01	String	Functional identifier code. (Optional)
gs02	String	Application sender's code. (Optional)
gs03	String	Application receiver's code. (Optional)
gs04	String	Functional group date. (Optional)
gs05	String	Functional group time. (Optional)
gs07	String	Responsible agency code. (Optional)
gs08	String	Version/release/industry identifier code. (Optional)

## X12 functional group acknowledgement tracking schema

```

{
  "agreementProperties": {
    "senderPartnerName": "",
    "receiverPartnerName": "",
    "senderQualifier": "",
    "senderIdentifier": "",
    "receiverQualifier": "",
    "receiverIdentifier": "",
    "agreementName": ""
  },
  "messageProperties": {
    "direction": "",
    "interchangeControlNumber": "",
    "functionalGroupControlNumber": "",
    "isaSegment": "",
    "gsSegment": "",
    "respondingFunctionalGroupControlNumber": "",
    "respondingFunctionalGroupId": "",
    "isMessageFailed": "",
    "statusCode": "",
    "processingStatus": "",
    "ak903": "",
    "ak904": "",
    "ak9Segment": ""
  }
}

```

PROPERTY	TYPE	DESCRIPTION
senderPartnerName	String	X12 message sender's partner name. (Optional)
receiverPartnerName	String	X12 message receiver's partner name. (Optional)
senderQualifier	String	Send partner qualifier. (Mandatory)
senderIdentifier	String	Send partner identifier. (Mandatory)
receiverQualifier	String	Receive partner qualifier. (Mandatory)
receiverIdentifier	String	Receive partner identifier. (Mandatory)
agreementName	String	Name of the X12 agreement to which the messages are resolved. (Optional)
direction	Enum	Direction of the message flow, receive or send. (Mandatory)
interchangeControlNumber	String	Interchange control number, which populates for the send side when a technical acknowledgement is received from partners. (Optional)
functionalGroupControlNumber	String	Functional group control number of the technical acknowledgement, which populates for the send side when a technical acknowledgement is received from partners. (Optional)

PROPERTY	TYPE	DESCRIPTION
isaSegment	String	Same as interchange control number, but populated only in specific cases. (Optional)
gsSegment	String	Same as functional group control number, but populated only in specific cases. (Optional)
respondingfunctionalGroupControlNumber	String	Control number of the original functional group. (Optional)
respondingFunctionalGroupId	String	Maps to AK101 in the acknowledgement functional group ID. (Optional)
isMessageFailed	Boolean	Whether the X12 message failed. (Mandatory)
statusCode	Enum	Acknowledgement status code. Allowed values are <b>Accepted</b> , <b>Rejected</b> , and <b>AcceptedWithErrors</b> . (Mandatory)
processingStatus	Enum	Processing status of the acknowledgement. Allowed values are <b>Received</b> , <b>Generated</b> , and <b>Sent</b> . (Mandatory)
ak903	String	Number of transaction sets received. (Optional)
ak904	String	Number of transaction sets accepted in the identified functional group. (Optional)
ak9Segment	String	Whether the functional group identified in the AK1 segment is accepted or rejected, and why. (Optional)

## Next steps

- Learn more about [monitoring B2B messages](#).
- Learn more about [AS2 tracking schemas](#).
- Learn more about [B2B custom tracking schemas](#).
- Learn about [tracking B2B messages in the Operations Management Suite portal](#).
- Learn more about the [Enterprise Integration Pack](#).

# Enable tracking to monitor your complete workflow, end-to-end

2/28/2017 • 1 min to read • [Edit Online](#)

There is built-in tracking that you can enable for different parts of your business-to-business workflow, such as tracking AS2 or X12 messages. When you create workflows that includes a logic app, BizTalk Server, SQL Server, or any other layer, then you can enable custom tracking that logs events from the beginning to the end of your workflow.

This topic provides custom code that you can use in the layers outside of your logic app.

## Custom tracking schema

```
{
    "sourceType": "",
    "source": {

        "workflow": {
            "systemId": ""
        },
        "runInstance": {
            "runId": ""
        },
        "operation": {
            "operationName": "",
            "repeatItemScopeName": "",
            "repeatItemIndex": "",
            "trackingId": "",
            "correlationId": "",
            "clientRequestId": ""
        }
    },
    "events": [
        {
            "eventLevel": "",
            "eventTime": "",
            "recordType": "",
            "record": {
            }
        }
    ]
}
```

PROPERTY	TYPE	DESCRIPTION
sourceType		Type of the run source. Allowed values are <b>Microsoft.Logic/workflows</b> and <b>custom</b> . (Mandatory)
Source		If the source type is <b>Microsoft.Logic/workflows</b> , the source information needs to follow this schema. If the source type is <b>custom</b> , the schema is a JToken. (Mandatory)

PROPERTY	TYPE	DESCRIPTION
systemId	String	Logic app system ID. (Mandatory)
runId	String	Logic app run ID. (Mandatory)
operationName	String	Name of the operation (for example, action or trigger). (Mandatory)
repeatItemScopeName	String	Repeat item name if the action is inside a <code>foreach</code> / <code>until</code> loop. (Mandatory)
repeatItemIndex	Integer	Whether the action is inside a <code>foreach</code> / <code>until</code> loop. Indicates the repeated item index. (Mandatory)
trackingId	String	Tracking ID, to correlate the messages. (Optional)
correlationId	String	Correlation ID, to correlate the messages. (Optional)
clientRequestId	String	Client can populate it to correlate messages. (Optional)
eventLevel		Level of the event. (Mandatory)
eventTime		Time of the event, in UTC format YYYY-MM-DDTHH:MM:SS.00000Z. (Mandatory)
recordType		Type of the track record. Allowed value is <b>custom</b> . (Mandatory)
record		Custom record type. Allowed format is JToken. (Mandatory)

## B2B protocol tracking schemas

For information about B2B protocol tracking schemas, see:

- [AS2 tracking schemas](#)
- [X12 tracking schemas](#)

## Next steps

- Learn more about [monitoring B2B messages](#).
- Learn about [tracking B2B messages in the Operations Management Suite portal](#).
- Learn more about the [Enterprise Integration Pack](#).

# Track B2B messages in the Operations Management Suite portal

2/27/2017 • 4 min to read • [Edit Online](#)

B2B communication involves the exchange of messages between two running business processes or applications. Use the following web-based tracking features in the Operations Management Suite portal to confirm whether messages are processed correctly:

- Message count and status
- Acknowledgments status
- Correlate messages with acknowledgments
- Detailed error descriptions for failures
- Search capabilities

## Prerequisites

- An Azure account. You can create a [free account](#).
- An integration account. You can create an [integration account](#) and set up logging. To set up logging, see [Monitor B2B messages](#).
- A logic app. You can create a [logic app](#) and set up logging. To set up logging, see [Azure Diagnostics and alerts](#).

## Add Logic Apps B2B solution to the Operations Management Suite portal

1. In the Azure portal, select **More Services**, search for log analytics and then select **Log Analytics**.

Preview Microsoft Azure Log Analytics

New

Resource groups

App Services

SQL databases

SQL data warehouses

NoSQL (DocumentDB)

Virtual machines

Storage accounts

Load balancers

Virtual networks

All resources

Azure Active Directory

Monitor

Azure Advisor

Security Center

Billing

Help + support

More services >

Shift+Space to toggle favorites

log analytics

Log Analytics

Monitor Keywords: logs PREVIEW ★

2. Select your **Log Analytics**.

Log Analytics			
Subscriptions: 1 of 5 selected			
NAME	RESOURCE GROUP	LOCATION	SUBSCRIPTION
padma	BTS4		
2 items			
padmaomsportal	EITemplates	East US	BTS4
dfpadoms	padmarg	East US	BTS4

3. Select **OMS Portal**. The Operations Management Suite portal home page appears.

padmaomsportal  
Log Analytics

Search (Ctrl+ /)

OMS Portal Delete

Essentials ▾

Resource group (change)  
eiptemplates

Status  
Active

Location  
East US

Subscription name (change)  
[redacted]

Subscription ID  
[redacted]

Workspace Name  
padmaomsportal

Workspace Id  
[redacted]

Pricing tier  
Standalone

Management services  
Operations logs

Management

Overview Log Search OMS Portal

Pricing tier

Standalone  
PADMAOMSPORTAL

No daily limit  
Data retention 31 days

4. Select **Solutions Gallery**.

Microsoft Operations Management Suite

Find...

Settings

Get started ➔

1 Data sources connected

Log Search

My Dashboard

2 NEW Solutions Gallery

Usage

Settings

5. Select **Logic Apps B2B**.

The screenshot shows the Microsoft Operations Management Suite Solutions Gallery. On the left, there's a sidebar with icons for Home, Add, Log Search, My Dashboard, and Solutions Gallery. The main area is titled "Solutions Gallery" and contains several cards:

- Insight & Analytics** (Microsoft): Monitor and Troubleshoot Application and Infrastructure issues using Log Analytics. Includes 2 solutions.
- Automation & Control** (Microsoft): Increase control with automation and configuration management. Includes 2 solutions.
- All solutions** (highlighted with a red border): Logic Apps B2B NEW (Available). Track and monitor the status of your B2B Logic Apps processing.
- AD Replication Status** (Available): Identify Active Directory replication issues in your environment.

6. Select **Add** to add **Logic Apps B2B** messages to the home page.

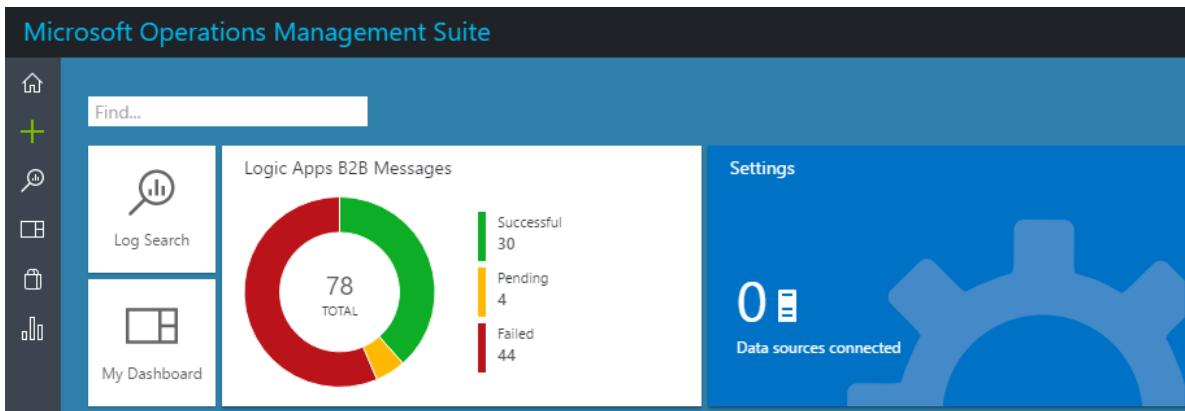
The screenshot shows the Microsoft Operations Management Suite interface. On the left, the sidebar has an "Add" button highlighted with a red border. The main area shows the "Logic Apps B2B NEW" solution details, which are marked as "Available". The "Add" button is also highlighted with a red border. To the right, the "Logic Apps B2B" dashboard is displayed, showing various metrics and charts. One chart shows message counts by status: AS2 (Successful: 3.7k, Pending: 4.9k, Failed: 1.7k) and X12 (Successful: 289, Pending: 22, Failed: 41).

7. **Logic Apps B2B Messages** appears on the home page.

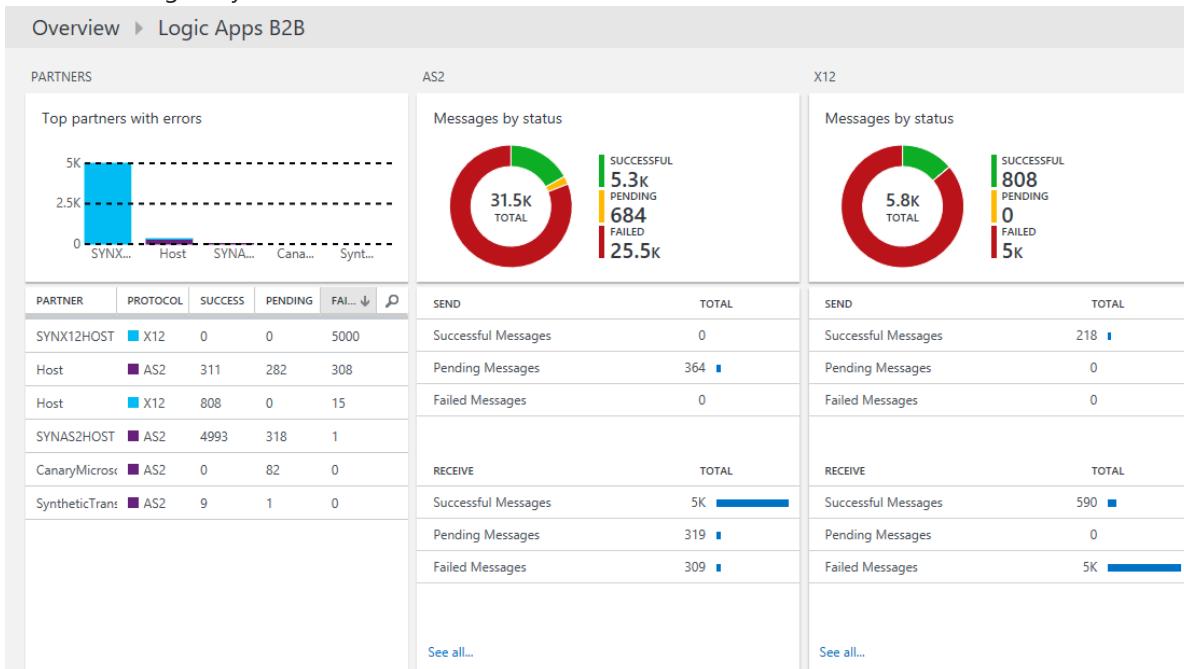
The screenshot shows the Microsoft Operations Management Suite home page. On the left, the sidebar includes "Log Search", "My Dashboard" (with 1 NEW), and "Solutions Gallery". The main area features a "Get started" section with a gear icon and "1 Data sources connected". To the right, there is a large card titled "Logic Apps B2B Messages" with a donut chart showing 0 total messages across three categories: Successful (0), Pending (0), and Failed (0). This card is also highlighted with a red border.

## Track data in the Operations Management Suite portal

- After the messages are processed, the updated message count appears.



2. Select **Logic Apps B2B Messages** on the home page to view the AS2 and X12 message status. The data is based on a single day.



3. Select an AS2 or X12 message by status to go to the message list. The next screenshot shows the AS2 message list. The AS2 and X12 message status property descriptions appear later in "Message list property descriptions."

The AS2 Message List table displays 46 messages. The columns include SENDER, RECEIVER, LOGIC APP, STATUS, ACK, DIRECTION, CORRELATION ID, MESSAGE ID, and TIMESTAMP.

SENDER	RECEIVER	LOGIC APP	STATUS	ACK	DIRECTION	CORRELATION ID	MESSAGE ID	TIMESTAMP
Fabrikam	Contoso	testbatchtracking	Accepted	Received	d2179dbd-c8fe-4187-951a-67c022fa-a0d1-4083-9b45-			1/25/2017, 10:05:25 AM
Contoso	Fabrikam	AS2Encode	Pending	Send	7f498521-92a2-49a5-8a96-<RD0003FFABAA51_762F3e			1/25/2017, 12:00:05 PM
		AS2Encode	Not Requir	Send	17d584a1-ff80-41c4-b3fa-t			1/25/2017, 12:04:52 PM
		AS2Encode	Not Requir	Send	ab310c3e-6b17-4ea6-bf08-			1/25/2017, 12:06:51 PM
Fabrikam	Contoso	testbatchtracking	Accepted	Received	d88614a6-2556-4a74-b7bc	67c022fa-a0d1-4083-9b45-		1/25/2017, 1:08:04 PM
Fabrikam	Contoso	testbatchtracking	Accepted	Received	77af33fe-e1f9-4714-904c-1	67c022fa-a0d1-4083-9b45-		1/25/2017, 1:08:12 PM
Fabrikam	Contoso	testbatchtracking	Accepted	Received	0953d170-5041-450c-b023	67c022fa-a0d1-4083-9b45-		1/25/2017, 2:48:05 PM

4. Select a row in the AS2 or X12 message list to go to the log search. Log search lists all of the actions that have the same run ID.

## Message list property descriptions

### AS2 message list property descriptions

PROPERTY	DESCRIPTION
Sender	The guest partner that is configured in the receive settings, or the host partner that is configured in the send settings for an AS2 agreement.
Receiver	The host partner that is configured in the receive settings, or the guest partner that is configured in the send settings for an AS2 agreement.
Logic App	Logic app where the AS2 actions are configured.
Status	AS2 message status Success = Received or sent a good AS2 message, no MDN is configured Success = Received or sent a good AS2 message, MDN is configured and received or MDN is sent Failed = Received a bad AS2 message, no MDN is configured Pending = Received or sent a good AS2 message, MDN is configured and a functional ack is expected
Ack	MDN message status Accepted = Received or sent a positive MDN Pending = Waiting to receive or send an MDN Rejected = Received or sent a negative MDN Not Required = MDN is not configured in the agreement
Direction	AS2 message direction.
Correlation ID	ID to correlate all the triggers and actions within a Logic app.
Message ID	AS2 message ID, from the headers of the AS2 message.
Timestamp	Time at which the AS2 action processes the message.

### X12 message list property descriptions

PROPERTY	DESCRIPTION
Sender	The guest partner that is configured in the receive settings, or the host partner that is configured in the send settings for an AS2 agreement.

PROPERTY	DESCRIPTION
Receiver	The host partner that is configured in the receive settings, or the guest partner that is configured in the send settings for an AS2 agreement.
Logic App	Logic app where the AS2 actions are configured.
Status	X12 message status Success = Received or sent a good X12 message, no functional ack is configured Success = Received or sent a good X12 message, functional ack is configured and received or a functional ack is sent Failed = Received or sent a bad X12 message Pending = Received or sent a good X12 message, functional ack is configured and a functional ack is expected.
Ack	Functional Ack (997) status Accepted = Received or sent a positive functional ack Rejected = Received or sent a negative functional ack Pending = Expecting a functional ack but didn't receive it Pending = Generated a functional ack but couldn't send it to partner Not Required = Functional Ack is not configured
Direction	X12 message direction.
Correlation ID	ID to correlate all of the triggers and actions within a Logic app.
Msg type	EDI X12 message type.
ICN	Interchange Control Number of the X12 message.
TSCN	Transactional Set Control Number of the X12 message.
Timestamp	Time at which X12 action processes the message.

## Queries in the Operations Management Suite portal

On the search page, you can create a query. When you search, you can filter the results by using facet controls.

### Create a query

1. In the log search, write a query and select **Save**. **Save Search** appears. To write a query, see [Track B2B messages in the Operations Management Suite portal by using a query](#).

Microsoft Operations Management Suite

Log Search

Export Alert Save Favorites History

Type="AzureDiagnostics" source\_runInstanceId\_s=08587222870686547921761230681 event\_record\_messageProperties\_interchangeControlNumber\_s=000000027

3 Results List Table

11/16/2016 10:10:40.983 AM | AzureDiagnostics  
... TimeGenerated : 11/16/2016 10:10:40.983 AM  
... event\_record\_agreementProperties\_senderQualifier\_s : ZZ  
... event\_record\_agreementProperties\_senderIdentifier\_s : 87654321  
... event\_record\_agreementProperties\_receiverQualifier\_s : ZZ  
... event\_record\_agreementProperties\_receiverIdentifier\_s : 12345678  
... event\_record\_messageProperties\_interchangeControlNumber\_s : 000000027

2. In **Save Search** add a **name** and **category**, and then select **Save**.

Save Search

X

Name

Search by control number

Category

Logic Apps B2B (Preview)

Save this query as a computer group:

Yes No

Save Cancel

3. To view the query, select **favorites**.

Log Search

Export Alert Save Favorites History

# Saved Searches

Find...

SHOW ALL...

## Logic Apps B2B (Preview)

Failed AS2 Messages by Receive Partner

Failed AS2 Messages by Send Partner

Failed AS2 Messages by Workflow

Failed X12 Messages by Receive Partner

Failed X12 Messages by Send Partner

Failed X12 Messages by Workflow

Search by control number

Show Less...

## Use a saved query

- In the log search, select **favorites** to view saved queries. To view query results, select a query.

The screenshot displays two windows side-by-side. On the left is the 'Log Search' window, which includes a toolbar with 'Export', 'Alert', 'Save', 'Favorites' (which is highlighted with a red box), and 'History'. Below the toolbar is a search bar containing the query: 'Type="AzureDiagnostics" source\_runInstance\_runId\_s=08587222870686547921761230681 event\_r'. The search results pane shows 3 results in 'List' view, with one result expanded to show detailed log entries. On the right is the 'Saved Searches' window, which has a 'Find...' search bar at the top. Below it is a list of saved search items under the heading 'Logic Apps B2B (Preview)', including 'Failed AS2 Messages by Receive Partner', 'Failed AS2 Messages by Send Partner', 'Failed AS2 Messages by Workflow', 'Failed X12 Messages by Receive Partner', 'Failed X12 Messages by Send Partner', and 'Failed X12 Messages by Workflow'. A 'Search by control number' input field is also present in this window, highlighted with a red box.

## Next steps

[Custom Tracking Schema](#)

[AS2 Tracking Schema](#)

[X12 Tracking Schema](#)

[Learn more about the Enterprise Integration Pack](#)

# Track B2B messages in the Operations Management Suite portal by using a query

1/31/2017 • 1 min to read • [Edit Online](#)

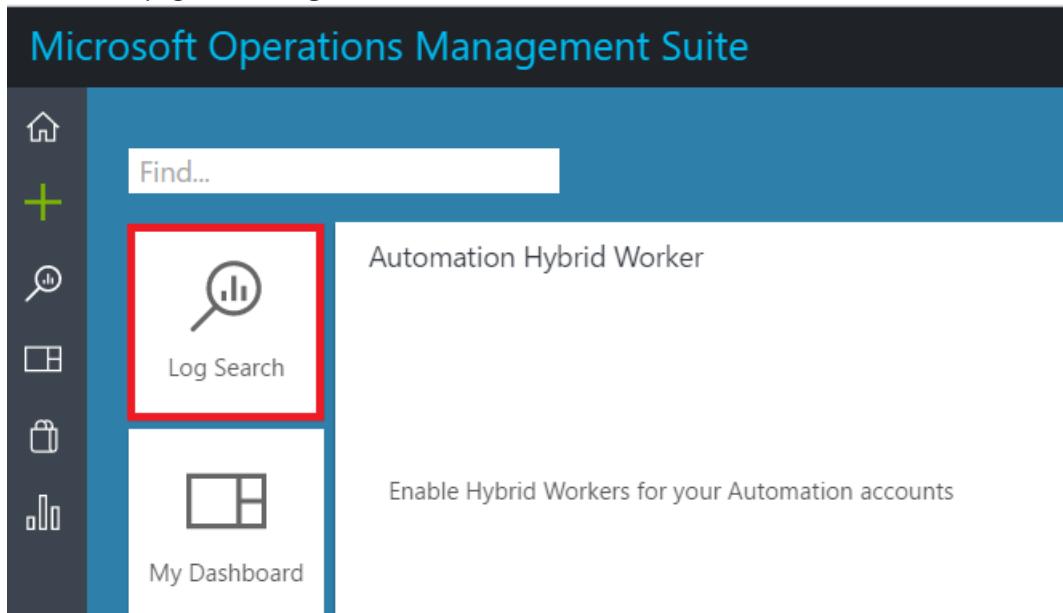
To track business-to-business (B2B) messages in the Operations Management Suite portal, you can create a query that filters data for a specific interchange control number.

## Prereqs

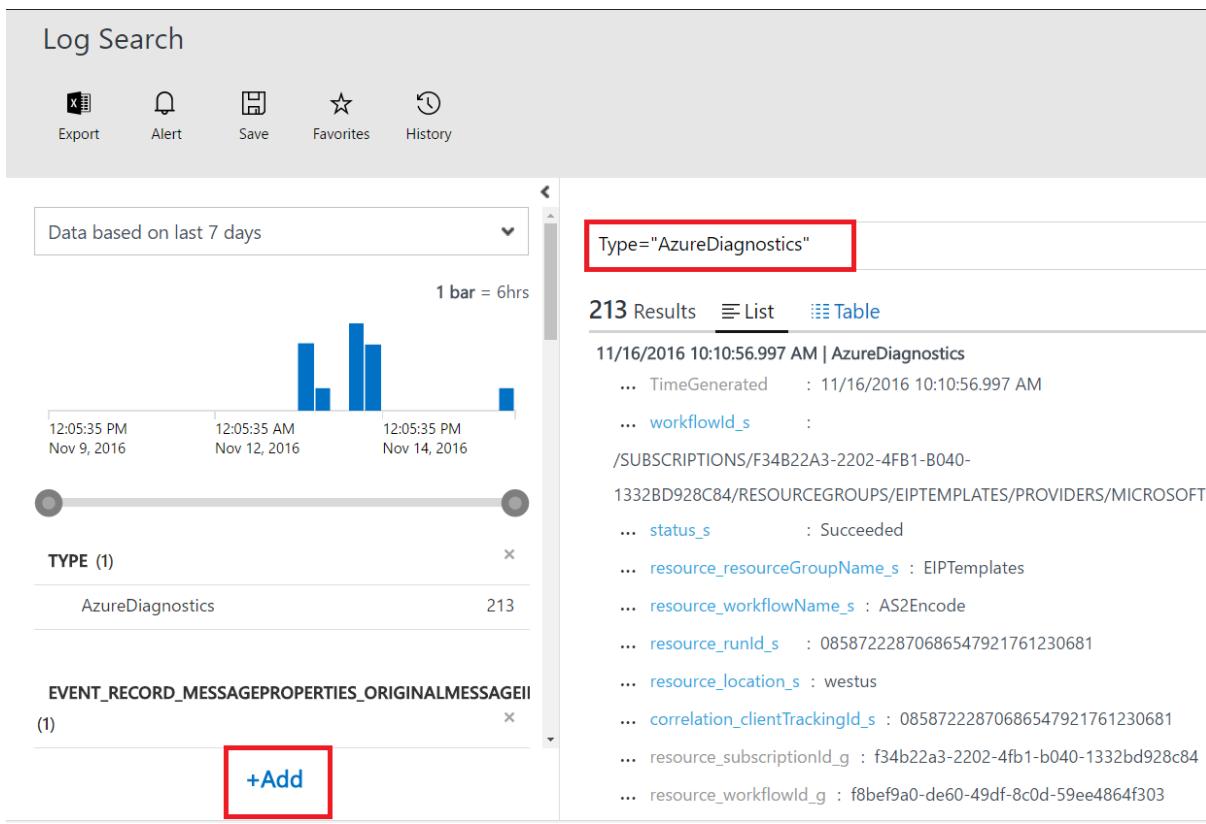
For debugging and for more detailed diagnostics information, turn on diagnostics in your [integration account](#) for your [logic apps](#) that have X12 connectors. Then, do the steps to [publish diagnostic data](#) to the Operations Management Suite portal.

## Search for an interchange control number

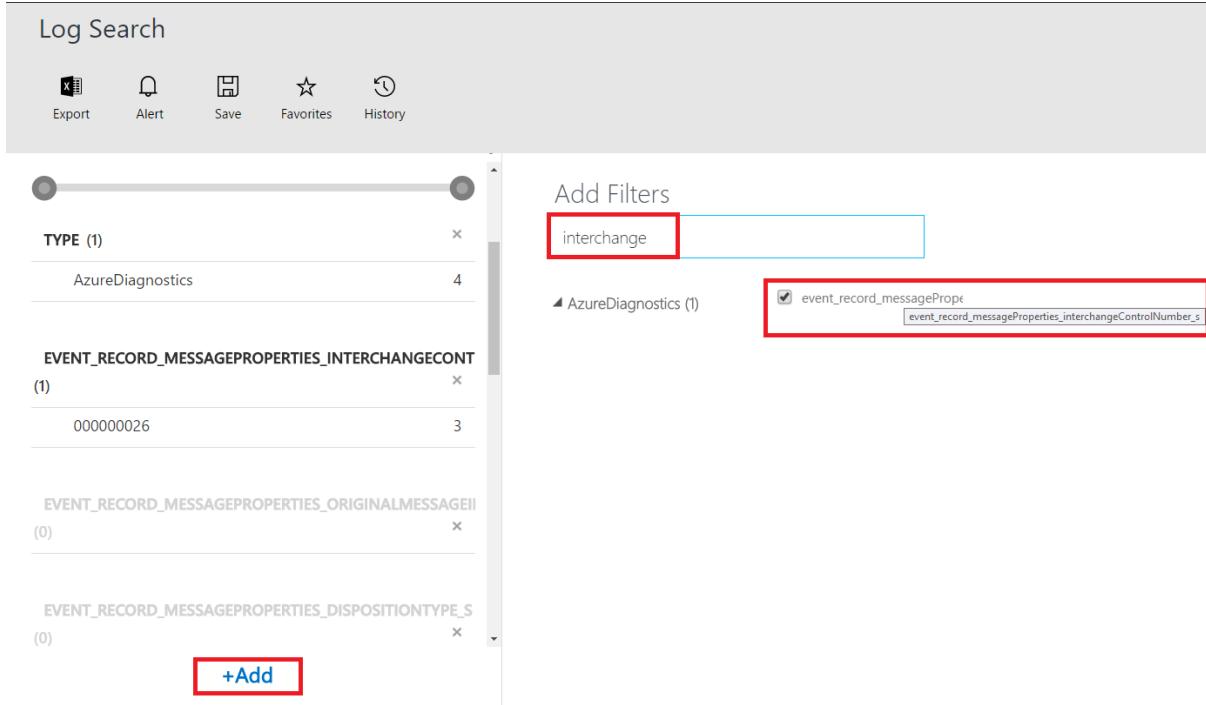
1. On the start page, select **Log Search**.



2. In the search box, type **Type="AzureDiagnostics"**. To filter data, select **Add**.



3. Type **interchange**, select **event\_record\_messageProperties\_interchangeControlNumber\_s**, and then select **Add**.



4. Select the control number that you want to filter data for, and then select **Apply**.

Log Search

Export Alert Save Favorites History

TYPE (1)

AzureDiagnostics 213

EVENT\_RECORD\_MESSAGEPROPERTIES\_INTERCHANGECONT  
(7) x

3

000000022 3

000000023 000000023 3

000000024 3

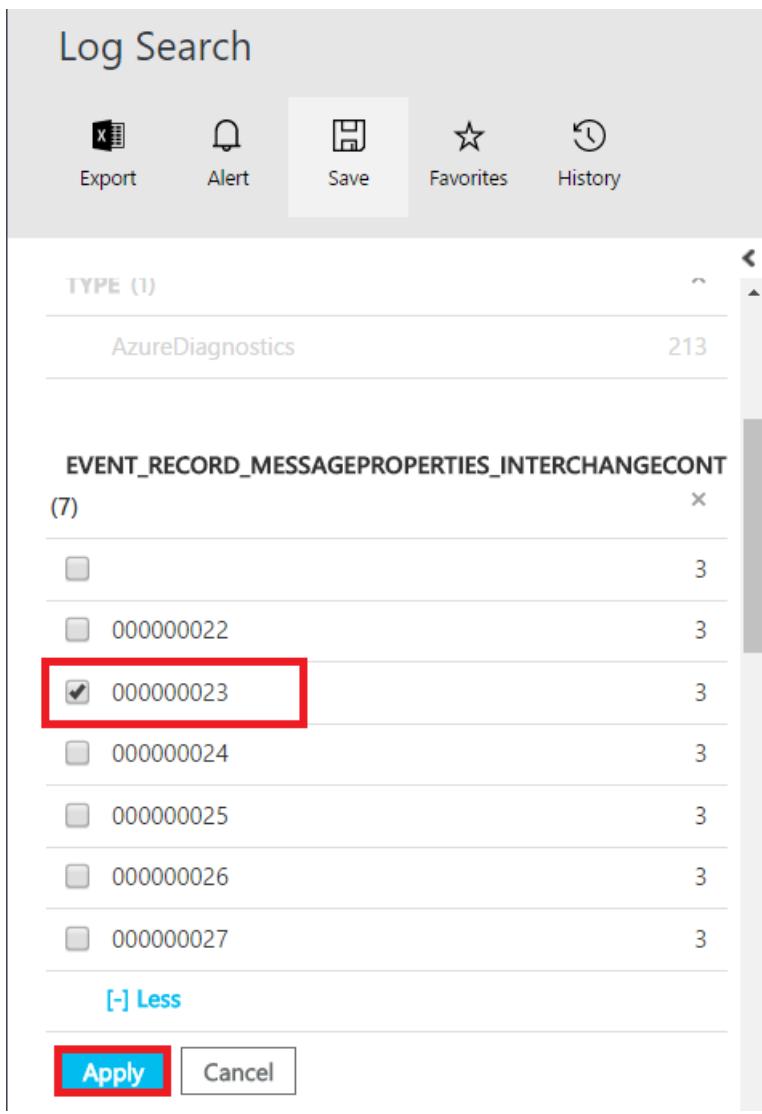
000000025 3

000000026 3

000000027 3

[\[-\] Less](#)

**Apply** **Cancel**



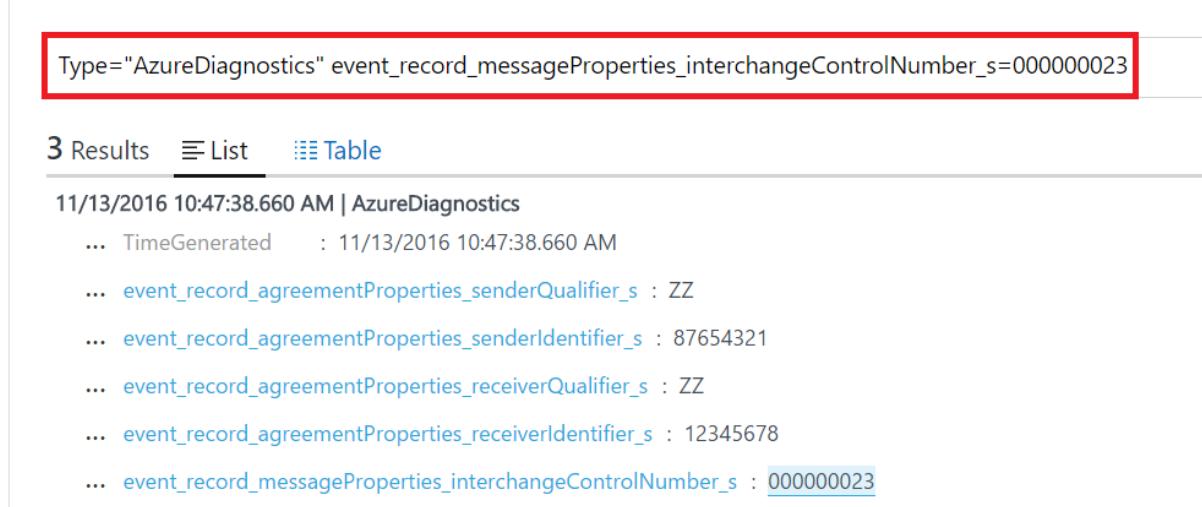
5. Find the query that you created to filter data for the selected control number.

Type="AzureDiagnostics" event\_record\_messageProperties\_interchangeControlNumber\_s=000000023

**3 Results** [List](#) [Table](#)

11/13/2016 10:47:38.660 AM | AzureDiagnostics

... TimeGenerated : 11/13/2016 10:47:38.660 AM  
... event\_record\_agreementProperties\_senderQualifier\_s : ZZ  
... event\_record\_agreementProperties\_senderIdentifier\_s : 87654321  
... event\_record\_agreementProperties\_receiverQualifier\_s : ZZ  
... event\_record\_agreementProperties\_receiverIdentifier\_s : 12345678  
... event\_record\_messageProperties\_interchangeControlNumber\_s : 000000023



6. Type a name for the query, and then save it.

7. To view the query, and to use it in future searches, select **Favorites**.

8. You can update the query to search for a new interchange control number.

## Next steps

- Learn more about [custom tracking schemas](#).
- Learn more about [AS2 tracking schemas](#).

- Learn more about [X12 tracking schemas](#).
- Learn more about the [Enterprise Integration Pack](#).

# Examples and common scenarios for Azure Logic Apps

3/29/2017 • 3 min to read • [Edit Online](#)

To help you learn more about the many patterns and capabilities in Azure Logic Apps, here are common examples and scenarios.

## Key scenarios for logic apps

Azure Logic Apps provides resilient orchestration and integration for different services. The Logic Apps service is "serverless", so you don't have to worry about scale or instances - all you have to do is define the workflow (trigger and actions). The underlying platform handles scale, availability, and performance. Any scenario where you need to coordinate multiple actions, especially across multiple systems, is a great use case for Azure Logic Apps. Here are some patterns and examples.

## Respond to triggers and extend actions

Every logic app begins with a trigger. For example, your workflow can start with a schedule event, a manual invocation, or an event from an external system, such as the "when a file is added to an FTP server" trigger. Azure Logic Apps currently supports over 100 ready-to-use connectors, ranging from on-premises SAP to Azure Cognitive Services. For systems and services that might not have published connectors, you can also extend logic apps.

- [Tutorial: Build an AI-powered social dashboard in minutes with Logic Apps and Power BI](#)
- [Create custom triggers or actions](#)
- [Set up long-running actions for workflow runs](#)
- [Respond to external events and actions with webhooks](#)
- [Call, trigger, or nest workflows with synchronous responses to HTTP requests](#)
- [Tutorial: Respond to Twilio SMS webhooks and send a text response](#)

## Error handling, logging, and control flow capabilities

Logic apps include rich capabilities for advanced control flow, like conditions, switches, loops, and scopes. To ensure resilient solutions, you can also implement error and exception handling in your workflows. For notification and diagnostic logs for workflow run status, Azure Logic Apps also provides monitoring and alerts.

- [Perform different actions with switch statements](#)
- [Process items in arrays and collections with loops and batches in logic apps](#)
- [Author error and exception handling in a workflow](#)
- [Configure Azure Alerts and diagnostics](#)
- [Use case: How a healthcare company uses logic app exception handling for HL7 FHIR workflows](#)

## Deploy and manage logic apps

You can fully develop and deploy logic apps with Visual Studio, Visual Studio Team Services, or any other source control and automated build tools. To support deployment for workflows and dependent connections in a resource template, logic apps use Azure resource deployment templates. Visual Studio tools automatically generate these templates, which you can check in to source control for versioning.

- [Create an automated deployment template](#)
- [Build and deploy logic apps from Visual Studio](#)
- [Monitor the health of your logic apps](#)

## Content types, conversions, and transformations within a run

You can access, convert, and transform multiple content types by using the many functions in the Azure Logic Apps [workflow definition language](#). For example, you can convert between a string, JSON, and XML with the `@json()` and `@xml()` workflow expressions. The Logic Apps engine preserves content types to support content transfer in a lossless manner between services.

- [Handle non-JSON content types, like `application/xml`, `application/octet-stream`, and `multipart/formdata`](#)
- [How workflow expressions work in logic apps](#)
- [Reference: Azure Logic Apps workflow definition language](#)

## Other integrations and capabilities

Logic apps also offer integration with many services, like Azure Functions, Azure API Management, Azure App Services, and custom HTTP endpoints, for example, REST and SOAP.

- [Create a real-time social dashboard with Azure Serverless](#)
- [Call Azure Functions from logic apps](#)
- [Scenario: Trigger logic apps with Azure Functions](#)
- [Blog: Call SOAP endpoints from logic apps](#)

## Next Steps

- [Handle errors and exceptions in logic apps](#)
- [Author workflow definitions with the workflow definition language](#)
- [Submit your comments, questions, feedback, or suggestions for how we can improve Azure Logic Apps](#)

# Create a real-time customer insights dashboard with Azure Logic Apps and Azure Functions

3/30/2017 • 4 min to read • [Edit Online](#)

Azure Serverless tools provide powerful capabilities to quickly build and host applications in the cloud, without having to think about infrastructure. In this scenario, we will create a dashboard to trigger on customer feedback, analyze feedback with machine learning, and publish insights a source like Power BI or Azure Data Lake.

## Overview of the scenario and tools used

In order to implement this solution, we will leverage the two key components of serverless apps in Azure: [Azure Functions](#) and [Azure Logic Apps](#).

Logic Apps is a serverless workflow engine in the cloud. It provides orchestration across serverless components, and also connects to over 100 services and APIs. For this scenario, we will create a logic app to trigger on feedback from customers. Some of the connectors that can assist in reacting to customer feedback include Outlook.com, Office 365, Survey Monkey, Twitter, and an HTTP Request [from a web form](#). For the workflow below, we will monitor a hashtag on Twitter.

Functions provide serverless compute in the cloud. In this scenario, we will use Azure Functions to flag tweets from customers based on a series of pre-defined key words.

The entire solution can be [build in Visual Studio](#) and [deployed as part of a resource template](#). There is also video walkthrough of the scenario [on Channel 9](#).

## Building the logic app to trigger on customer data

After [creating a logic app](#) in Visual Studio or the Azure portal:

1. Add a trigger for **On New Tweets** from Twitter
2. Configure the trigger to listen to tweets on a keyword or hashtag.

### NOTE

The recurrence property on the trigger will determine how frequently the logic app checks for new items on polling-based triggers

The screenshot shows the Microsoft Logic App builder interface. At the top, there's a header with a Twitter icon, the text "When a new tweet is posted", and a three-dot menu icon. Below this, there's a search bar labeled "\* Search text" containing the value "#Build". A section titled "How often do you want to check for items?" contains two dropdowns: "Frequency" set to "Minute" and "Interval" set to "3". At the bottom of the step, it says "Connected to jeffhollan. [Change connection](#)".

This app will now fire on all new tweets. We can then take that tweet data and understand more of the sentiment expressed. For this we use the [Azure Cognitive Service](#) to detect sentiment of text.

1. Click **New Step**
2. Select or search for the **Text Analytics** connector
3. Select the **Detect Sentiment** operation
4. If prompted, provide a valid Cognitive Services key for the Text Analytics service
5. Add the **Tweet Text** as the text to analyze.

Now that we have the tweet data, and insights on the tweet, a number of other connectors may be relevant:

- Power BI - Add Rows to Streaming Dataset: View tweets real time on a Power BI dashboard.
- Azure Data Lake - Append file: Add customer data to an Azure Data Lake dataset to include in analytics jobs.
- SQL - Add rows: Store data in a database for later retrieval.
- Slack - Send message: Alert a slack channel on negative feedback that requires actions.

An Azure Function can also be used to do more custom compute on top of the data.

## Enriching the data with an Azure Function

Before we can create a function, we need to have a function app in our Azure subscription. Details on creating an Azure Function in the portal can [be found here](#)

For a function to be called directly from a logic app, it needs to have an HTTP trigger binding. We recommend using the **HttpTrigger** template.

In this scenario, the request body of the Azure Function would be the tweet text. In the function code, simply define logic on if the tweet text contains a key word or phrase. The function itself could be kept as simple or complex as needed for the scenario.

At the end of the function, simply return a response to the logic app with some data. This could be a simple boolean value (e.g. `containsKeyword`), or a complex object.

The screenshot shows a logic app step titled "Tag if tweet contains specific words". The "Request Body" section contains a "Tweet text" input field with a Twitter icon. Below the input field is a "Show advanced options" button.

#### TIP

When accessing a complex response from a function in a logic app, use the Parse JSON action.

Once the function is saved, it can be added into the logic app created above. In the logic app:

1. Click to add a **New Step**
2. Select the **Azure Functions** connector
3. Select to choose an existing function, and browse to the function created
4. Send in the **Tweet Text** for the **Request Body**

## Running and monitoring the solution

One of the benefits of authoring serverless orchestrations in Logic Apps is the rich debug and monitoring capabilities. Any run (current or historic) can be viewed from within Visual Studio, the Azure portal, or via the REST API and SDKs.

One of the easiest ways to test a logic app is using the **Run** button in the designer. Clicking **Run** will continue to poll the trigger every 5 seconds until an event is detected, and give a live view as the run progresses.

Previous run histories can be viewed on the Overview blade in the Azure portal, or using the Visual Studio Cloud Explorer.

## Creating a deployment template for automated deployments

Once a solution has been developed, it can be captured and deployed via an Azure deployment template to any Azure region in the world. This is useful for both modifying parameters for different versions of this workflow, but also for integrating this solution in a build and release pipeline. Details on creating a deployment template can be found [in this article](#).

Azure Functions can also be incorporated in the deployment template - so the entire solution with all dependencies can be managed as a single template. An example of a function deployment template can be found in the [Azure quickstart template repository](#).

## Next steps

- [See other examples and scenarios for Azure Logic Apps](#)
- [Watch a video walkthrough on creating this solution end-to-end](#)
- [Learn how to handle and catch exceptions within a logic app](#)

# Scenario: Trigger a logic app with Azure Functions and Azure Service Bus

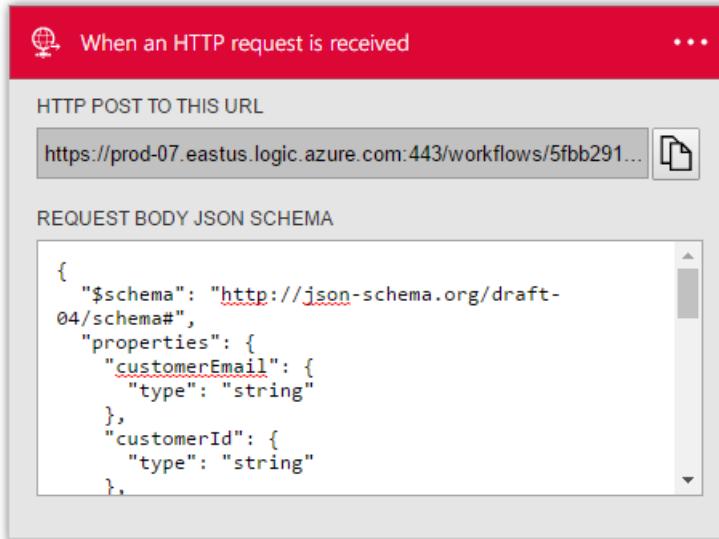
2/10/2017 • 1 min to read • [Edit Online](#)

You can use Azure Functions to create a trigger for a logic app when you need to deploy a long-running listener or task. For example, you can create a function that listens in on a queue and then immediately fire a logic app as a push trigger.

## Build the logic app

In this example, you have a function running for each logic app that needs to be triggered. First, create a logic app that has an HTTP request trigger. The function calls that endpoint whenever a queue message is received.

1. Create a logic app.
2. Select the **Manual - When an HTTP Request is Received** trigger. Optionally, you can specify a JSON schema to use with the queue message by using a tool like [jsonschema.net](#). Paste the schema in the trigger. Schemas help the designer understand the shape of the data and flow properties more easily through the workflow.
3. Add any additional steps that you want to occur after a queue message is received. For example, send an email via Office 365.
4. Save the logic app to generate the callback URL for the trigger to this logic app. The URL appears on the trigger card.



## Build the function

Next, you must create a function that acts as the trigger and listens to the queue.

1. In the [Azure Functions portal](#), select **New Function**, and then select the **ServiceBusQueueTrigger - C#** template.

The screenshot shows the Azure Functions portal interface. On the left, there's a sidebar with a list of available function templates. In the center, a grid displays six specific templates with their descriptions:

- GitHub WebHook - C#**: A C# function that will be run whenever it receives a GitHub webhook request.
- GitHub Webhook - Node**: A Node.js function that will be run whenever it receives a GitHub webhook request.
- HttpTrigger - C#**: A C# function that will be run whenever it receives an HTTP request.
- ServiceBusQueueTrigger - C#**: A C# function that will be run whenever a message is added to a specified Service Bus queue.
- ServiceBusQueueTrigger - Node**: A Node.js function that will be run whenever a message is added to a specified Service Bus queue.
- ServiceBusTopicTrigger - C#**: A C# function that will be run whenever a message is added to the specified Service Bus topic.

2. Configure the connection to the Service Bus queue, which uses the Azure Service Bus SDK `OnMessageReceive()` listener.
3. Write a basic function to call the logic app endpoint (created earlier) by using the queue message as a trigger. Here's a full example of a function. The example uses an `application/json` message content type, but you can change this type as necessary.

```
using System;
using System.Threading.Tasks;
using System.Net.Http;
using System.Text;

private static string logicAppUri = @"https://prod-05.westus.logic.azure.com:443/.....";

public static void Run(string myQueueItem, TraceWriter log)
{
    log.Info($"C# ServiceBus queue trigger function processed message: {myQueueItem}");
    using (var client = new HttpClient())
    {
        var response = client.PostAsync(logicAppUri, new StringContent(myQueueItem, Encoding.UTF8,
"application/json")).Result;
    }
}
```

To test, add a queue message via a tool like [Service Bus Explorer](#). See the logic app fire immediately after the function receives the message.

# Scenario: Exception handling and error logging for logic apps

3/8/2017 • 7 min to read • [Edit Online](#)

This scenario describes how you can extend a logic app to better support exception handling. We've used a real-life use case to answer the question: "Does Azure Logic Apps support exception and error handling?"

## NOTE

The current Azure Logic Apps schema provides a standard template for action responses. This template includes both internal validation and error responses returned from an API app.

## Scenario and use case overview

Here's the story as the use case for this scenario:

A well-known healthcare organization engaged us to develop an Azure solution that would create a patient portal by using Microsoft Dynamics CRM Online. They needed to send appointment records between the Dynamics CRM Online patient portal and Salesforce. We were asked to use the [HL7 FHIR](#) standard for all patient records.

The project had two major requirements:

- A method to log records sent from the Dynamics CRM Online portal
- A way to view any errors that occurred within the workflow

## TIP

For a high-level video about this project, see [Integration User Group](#).

## How we solved the problem

We chose [Azure DocumentDB](#) as a repository for the log and error records (DocumentDB refers to records as documents). Because Azure Logic Apps has a standard template for all responses, we would not have to create a custom schema. We could create an API app to **Insert** and **Query** for both error and log records. We could also define a schema for each within the API app.

Another requirement was to purge records after a certain date. DocumentDB has a property called [Time to Live](#) (TTL), which allowed us to set a **Time to Live** value for each record or collection. This capability eliminated the need to manually delete records in DocumentDB.

## IMPORTANT

To complete this tutorial, you need to create a DocumentDB database and two collections (Logging and Errors).

## Create the logic app

The first step is to create the logic app and open the app in Logic App Designer. In this example, we are using parent-child logic apps. Let's assume that we have already created the parent and are going to create one child

logic app.

Because we are going to log the record coming out of Dynamics CRM Online, let's start at the top. We must use a **Request** trigger because the parent logic app triggers this child.

### Logic app trigger

We are using a **Request** trigger as shown in the following example:

```
"triggers": {  
    "request": {  
        "type": "request",  
        "kind": "http",  
        "inputs": {  
            "schema": {  
                "properties": {  
                    "CRMID": {  
                        "type": "string"  
                    },  
                    "recordType": {  
                        "type": "string"  
                    },  
                    "salesforceID": {  
                        "type": "string"  
                    },  
                    "update": {  
                        "type": "boolean"  
                    }  
                },  
                "required": [  
                    "CRMID",  
                    "recordType",  
                    "salesforceID",  
                    "update"  
                ],  
                "type": "object"  
            }  
        }  
    }  
},
```

## Steps

We must log the source (request) of the patient record from the Dynamics CRM Online portal.

1. We must get a new appointment record from Dynamics CRM Online.

The trigger coming from CRM provides us with the **CRM PatientId**, **record type**, **New or Updated Record** (new or update Boolean value), and **SalesforceId**. The **SalesforceId** can be null because it's only used for an update. We get the CRM record by using the CRM **PatientID** and the **Record Type**.

2. Next, we need to add our DocumentDB API app **InsertLogEntry** operation as shown here.

### Insert log entry designer view

InsertLogEntry

PRESCRIBERID  
CRMID

OPERATION  
New\_Patient

You can insert data from previous steps...

Outputs from manual

Body CRMID recordType salesforceID  
update

SOURCE  
Headers

SALESFORCEID  
salesforceID

DATE  
Date

...

Insert error entry designer view

**CreateErrorRecord**

Enter a valid integer

STATUSCODE

```
actions('Create_Appointment')['outputs']['statusCode']
```

MESSAGE

```
actions('Create_Appointment')['outputs']['body']['message']
```

SOURCE

```
@{concat(triggerBody()['description'],
  ' START: ', triggerBody()['start'],
  ' END: ', triggerBody()['end'],
  ' COMMENT: ', triggerBody()['comment']) }
```

ACTION

```
Create_Appointment
```

ERRORS

RESOLVED

0

NOTES

ISERROR

true

PATIENTID

```
@{replace(triggerBody()['participant'][0]['actor']['display'], ' ', '')}
```

SEVERITY

4

...

### Check for create record failure

**Condition**

CONDITION

```
@equals(actions('CreateErrorRecord')['status'], 'Failed')
```

If yes

Add an action

If no, do nothing

Add an action

**CreateErrorRecord**

...

# Logic app source code

## NOTE

The following examples are samples only. Because this tutorial is based on an implementation now in production, the value of a **Source Node** might not display properties that are related to scheduling an appointment.>

## Logging

The following logic app code sample shows how to handle logging.

### Log entry

Here is the logic app source code for inserting a log entry.

```
"InsertLogEntry": {
    "metadata": {
        "apiDefinitionUrl": "https://.../swagger/docs/v1",
        "swaggerSource": "website"
    },
    "type": "Http",
    "inputs": {
        "body": {
            "date": "@{outputs('Gets_NewPatientRecord')['headers']['Date']}",
            "operation": "New Patient",
            "patientId": "@{triggerBody()['CRMID']}",
            "providerId": "@{triggerBody()['providerID']}",
            "source": "@{outputs('Gets_NewPatientRecord')['headers']}"
        },
        "method": "post",
        "uri": "https://.../api/Log"
    },
    "runAfter": {
        "Gets_NewPatientRecord": ["Succeeded"]
    }
}
```

### Log request

Here is the log request message posted to the API app.

```
{
    "uri": "https://.../api/Log",
    "method": "post",
    "body": {
        "date": "Fri, 10 Jun 2016 22:31:56 GMT",
        "operation": "New Patient",
        "patientId": "6b115f6d-a7ee-e511-80f5-3863bb2eb2d0",
        "providerId": "",
        "source": "{\"Pragma\":\"no-cache\", \"x-ms-request-id\":\"e750c9a9-bd48-44c4-bbba-1688b6f8a132\", \"OData-Version\":\"4.0\", \"Cache-Control\":\"no-cache\", \"Date\":\"Fri, 10 Jun 2016 22:31:56 GMT\", \"Set-Cookie\":\"ARRAffinity=785f4334b5e64d2db0b84edcc1b84f1bf37319679aefce206b51510e56fd9770; Path=/; Domain=127.0.0.1\", \"Server\":\"Microsoft-IIS/8.0, Microsoft-HTTPAPI/2.0\", \"X-AspNet-Version\":\"4.0.30319\", \"X-Powered-By\":\"ASP.NET\", \"Content-Length\":1935, \"Content-Type\":\"application/json; odata.metadata=minimal; odata.streaming=true\", \"Expires\":\"-1\"}"
    }
}
```

### Log response

Here is the log response message from the API app.

```
{
  "statusCode": 200,
  "headers": {
    "Pragma": "no-cache",
    "Cache-Control": "no-cache",
    "Date": "Fri, 10 Jun 2016 22:32:17 GMT",
    "Server": "Microsoft-IIS/8.0",
    "X-AspNet-Version": "4.0.30319",
    "X-Powered-By": "ASP.NET",
    "Content-Length": "964",
    "Content-Type": "application/json; charset=utf-8",
    "Expires": "-1"
  },
  "body": {
    "ttl": 2592000,
    "id": "6b115f6d-a7ee-e511-80f5-3863bb2eb2d0_1465597937",
    "_rid": "XngRAOT6IQEHAAAAAAA==",
    "_self": "dbs/XngRAA==/colls/XngRAOT6IQE=/docs/XngRAOT6IQEHAAAAAAA==/",
    "_ts": 1465597936,
    "_etag": "\"0400fc2f-0000-0000-0000-575b3ff00000\"",
    "patientID": "6b115f6d-a7ee-e511-80f5-3863bb2eb2d0",
    "timestamp": "2016-06-10T22:31:56Z",
    "source": "{\"Pragma\":\"no-cache\",\"x-ms-request-id\":\"e750c9a9-bd48-44c4-bbba-1688b6f8a132\",\"OData-Version\":\"4.0\",\"Cache-Control\":\"no-cache\",\"Date\":\"Fri, 10 Jun 2016 22:31:56 GMT\",\"Set-Cookie\":\"ARRAffinity=785f4334b5e64d2db0b84edcc1b84f1bf37319679aefce206b51510e56fd9770;Path=/;Domain=127.0.0.1\",\"Server\":\"Microsoft-IIS/8.0,Microsoft-HTTPAPI/2.0\",\"X-AspNet-Version\":\"4.0.30319\",\"X-Powered-By\":\"ASP.NET\",\"Content-Length\":\"1935\",\"Content-Type\":\"application/json; odata.metadata=minimal;odata.streaming=true\",\"Expires\":\"-1\"}",
    "operation": "New Patient",
    "salesforceId": "",
    "expired": false
  }
}
```

Now let's look at the error handling steps.

## Error handling

The following logic app code sample shows how you can implement error handling.

### Create error record

Here is the logic app source code for creating an error record.

```

"actions": {
    "CreateErrorRecord": {
        "metadata": {
            "apiDefinitionUrl": "https://.../swagger/docs/v1",
            "swaggerSource": "website"
        },
        "type": "Http",
        "inputs": {
            "body": {
                "action": "New_Patient",
                "isError": true,
                "crmId": "@{triggerBody()['CRMID']}",
                "patientID": "@{triggerBody()['CRMID']}",
                "message": "@{body('Create_NewPatientRecord')['message']}",
                "providerId": "@{triggerBody()['providerId']}",
                "severity": 4,
                "source": "@{actions('Create_NewPatientRecord')['inputs']['body']}",
                "statusCode": "@{int(outputs('Create_NewPatientRecord')['statusCode'])}",
                "salesforceId": "",
                "update": false
            },
            "method": "post",
            "uri": "https://.../api/CrMtoSfError"
        },
        "runAfter": {
            {
                "Create_NewPatientRecord": ["Failed"]
            }
        }
    }
}

```

#### Insert error into DocumentDB--request

```
{
    "uri": "https://.../api/CrMtoSfError",
    "method": "post",
    "body": {
        "action": "New_Patient",
        "isError": true,
        "crmId": "6b115f6d-a7ee-e511-80f5-3863bb2eb2d0",
        "patientId": "6b115f6d-a7ee-e511-80f5-3863bb2eb2d0",
        "message": "Salesforce failed to complete task: Message: duplicate value found: Account_ID_MED__c duplicates value on record with id: 001U0000001c83gK",
        "providerId": "",
        "severity": 4,
        "salesforceId": "",
        "update": false,
        "source": ""

        {"/Account_Class_vod_c/": "/PRAC/", "/Account_Status_MED_c/": "/I/", "/CRM_HUB_ID_c/": "/6b115f6d-a7ee-e511-80f5-3863bb2eb2d0/", "/Credentials_vod_c/", "/DTC_ID_MED_c/": "", "/Fax/": "", "/FirstName/": "/A/", "/Gender_vod_c/": "", "/IMS_ID_c/": "", "/LastName/": "/BAILEY/", "/MasterID_mp_c/": "", "/C_ID_MED_c/": "/851588/", "/Middle_vod_c/": "", "/NPI_vod_c/": "", "/PDRP_MED_c/": false, "/PersonDoNotCall/": false, "/PersonEmail/": "", "/PersonHasOptedOutOfEmail/": false, "/PersonHasOptedOutOfFax/": false, "/PersonMobilePhone/": "", "/Phone/": "", "/Practicing_Specialty_c/": "/FM - FAMILY MEDICINE/", "/Primary_City_c/": "", "/Primary_State_c/": "", "/Primary_Street_Line2_c/": "", "/Primary_Street_c/": "", "/Primary_Zip_c/": "", "/RecordTypeId/": "/012U0000000JaPWIA0/", "/Request_Date_c/": "/2016-06-10T22:31:55.9647467Z/", "/ONY_ID_c/": "", "/Specialty_1_vod_c/": "", "/Suffix_vod_c/": "", "/Website/": ""},
        "statusCode": "400"
    }
}
```

#### Insert error into DocumentDB--response

```
{
  "statusCode": 200,
  "headers": {
    "Pragma": "no-cache",
    "Cache-Control": "no-cache",
    "Date": "Fri, 10 Jun 2016 22:31:57 GMT",
    "Server": "Microsoft-IIS/8.0",
    "X-AspNet-Version": "4.0.30319",
    "X-Powered-By": "ASP.NET",
    "Content-Length": "1561",
    "Content-Type": "application/json; charset=utf-8",
    "Expires": "-1"
  },
  "body": {
    "id": "6b115f6d-a7ee-e511-80f5-3863bb2eb2d0-1465597917",
    "_rid": "sQx2APhVzAA8AAAAAAA==",
    "_self": "dbs/sQx2AA=/colls/sQx2APhVzAA=/docs/sQx2APhVzAA8AAAAAAA==/",
    "_ts": 1465597912,
    "_etag": "\"0c00eaac-0000-0000-0000-575b3fdc0000\"",
    "prescriberId": "6b115f6d-a7ee-e511-80f5-3863bb2eb2d0",
    "timestamp": "2016-06-10T22:31:57.3651027Z",
    "action": "New_Patient",
    "salesforceId": "",
    "update": false,
    "body": "CRM failed to complete task: Message: duplicate value found: CRM_HUB_ID__c duplicates value on record with id: 001U000001c83gK",
    "source": "
{/\"Account_Class_vod_c/":"/PRAC/", /\"Account_Status_MED_c/":"/I/", /\"CRM_HUB_ID_c/":"/6b115f6d-a7ee-e511-80f5-3863bb2eb2d0/", /\"Credentials_vod_c/":"/DO - Degree level is DO", /\"DTC_ID_MED_c/":"/", /\"Fax/":"/", /\"FirstName/":"/A/", /\"Gender_vod_c/":"/", /\"IMS_ID_c/":"/", /\"LastName/":"/BAILEY", /\"MterID_mp_c/":"/", /\"Medicis_ID_MED_c/":"/851588", /\"Middle_vod_c/":"/", /\"NPI_vod_c/": "/", /\"PDRP_MED_c/":false, /\"PersonDoNotCall/":false, /\"PersonEmail/":"/", /\"PersonHasOptedOutOfEmail/":false, /\"PersonHasOptedOutOffax/":false, /\"PersonMobilePhone/":"/", /\"Phone/":"/", /\"Practicing_Specialty_c/":"/FM - FAMILY MEDICINE", /\"Primary_City_c/":"/", /\"Primary_State_c/":"/", /\"Primary_Street_Line2_c/":"/", /\"Primary_Street_c/":"/", /\"Primary_Zip_c/":"/", /\"RecordTypeId/":"/012U0000000JaPWIA0", /\"Request_Date_c/":"/2016-06-10T22:31:55.9647467Z", /\"XXXXXX/":"/", /\"Specialty_1_vod_c/":"/", /\"Suffix_vod_c/":"/", /\"Website/":"/"}",
    "code": 400,
    "errors": null,
    "isError": true,
    "severity": 4,
    "notes": null,
    "resolved": 0
  }
}
```

#### Salesforce error response

```
{
    "statusCode": 400,
    "headers": {
        "Pragma": "no-cache",
        "x-ms-request-id": "3e8e4884-288e-4633-972c-8271b2cc912c",
        "X-Content-Type-Options": "nosniff",
        "Cache-Control": "no-cache",
        "Date": "Fri, 10 Jun 2016 22:31:56 GMT",
        "Set-Cookie":
"ARRAffinity=785f4334b5e64d2db0b84edcc1b84f1bf37319679aefce206b51510e56fd9770;Path=/;Domain=127.0.0.1",
        "Server": "Microsoft-IIS/8.0,Microsoft-HTTPAPI/2.0",
        "X-AspNet-Version": "4.0.30319",
        "X-Powered-By": "ASP.NET",
        "Content-Length": "205",
        "Content-Type": "application/json; charset=utf-8",
        "Expires": "-1"
    },
    "body": {
        "status": 400,
        "message": "Salesforce failed to complete task: Message: duplicate value found: Account_ID_MED__C duplicates value on record with id: 001U0000001c83gK",
        "source": "Salesforce.Common",
        "errors": []
    }
}
```

## Return the response back to parent logic app

After you get the response, you can pass the response back to the parent logic app.

### Return success response to parent logic app

```
"SuccessResponse": {
    "runAfter": {
        "UpdateNew_CRMPatientResponse": ["Succeeded"]
    },
    "inputs": {
        "body": {
            "status": "Success"
        },
        "headers": {
            "Content-type": "application/json",
            "x-ms-date": "@utcnow()"
        },
        "statusCode": 200
    },
    "type": "Response"
}
```

### Return error response to parent logic app

```

"ErrorResponse": {
    "runAfter": {
        "Create_NewPatientRecord": [
            "Failed"
        ],
        "inputs": {
            "body": {
                "status": "BadRequest"
            },
            "headers": {
                "Content-type": "application/json",
                "x-ms-date": "@utcnow()"
            },
            "statusCode": 400
        },
        "type": "Response"
    }
}

```

## DocumentDB repository and portal

Our solution added capabilities with [DocumentDB](#).

### Error management portal

To view the errors, you can create an MVC web app to display the error records from DocumentDB. The **List**, **Details**, **Edit**, and **Delete** operations are included in the current version.

#### NOTE

Edit operation: DocumentDB replaces the entire document. The records shown in the **List** and **Detail** views are samples only. They are not actual patient appointment records.

Here are examples of our MVC app details created with the previously described approach.

#### Error management list

CRM to SF Error Management

#### List of Errors

PrescriberId	Action	TimeStamp	Body	Resolved	
ce1820b0-ee2c-e611-80e7-5065f38a5ba1	Create_SFaddress	6/8/2016 4:16:50 PM	Salesforce failed to complete task: Message: duplicate value found: CRM_Hub_Id_c duplicates value on record with id: a01m0000005H3hu	No	<a href="#">Details</a>
ce1820b0-ee2c-e611-80e7-5065f38a5ba1	Create_SFaddress	6/8/2016 4:16:58 PM	Salesforce failed to complete task: Message: duplicate value found: CRM_Hub_Id_c duplicates value on record with id: a01m0000005H3hu	No	<a href="#">Details</a>
4433c660-ac2d-e611-80e7-c4346bdc0271	Create_SFcallPlan	6/8/2016 7:23:00 PM	Salesforce failed to complete task: Message: duplicate value found: External_ID_MED__c duplicates value on record with id: a2pm0000000TFWg	No	<a href="#">Details</a>
12c3f133-b02d-e611-80e7-c4346bdc0271	Update Decile	6/8/2016 7:36:48 PM	Salesforce failed to complete task: Message: Unable to create/update fields: Account_MED__c. Please check the security settings of this field and verify that it is read/write for your profile or permission set.	No	<a href="#">Details</a>
8ea10937-fe2e-e611-80e9-5065f38a3ba1	Update Prescriber	6/10/2016 11:32:30 AM	Salesforce failed to complete task: Message: Provided external ID field does not exist or is not accessible: 001m000000X8lmoAAF123	No	<a href="#">Details</a>
8ea10937-fe2e-e611-80e9-5065f38a3ba1	Update_ATL	6/10/2016 11:32:32 AM	Salesforce failed to complete task: Message: Account: id value of incorrect type: 001m000000X8lmoAAF123	No	<a href="#">Details</a>

#### Error management detail view

## View

### Error Response

TimeStamp 6/8/2016 4:16:50 PM  
Code 400  
Body Salesforce failed to complete task: Message: duplicate value found: CRM\_Hub\_Id\_c duplicates value on record with id: a01m0000005H3hu  
Source {"Account\_vod\_c":"001m000000X74NAAZ","Address\_Type\_MED\_c":"Mailing","Address\_line\_2\_vod\_c":"test str 2","CRM\_Hub\_Id\_c":"ce1820b0-ee2c-e611-80e7-5065f38a5ba1","City\_vod\_c":"edison","Country\_vod\_c":"United States","Fax\_vod\_c":"","License\_Expiration\_Date\_vod\_c":"2016-06-30","License\_State\_MED\_c":"NJ","License\_Status\_vod\_c":"Valid\_vod","License\_vod\_c":"342198","Name":"test str 1","Phone\_vod\_c":"","Primary\_vod\_c":"false","SLX\_Address\_Line\_3\_c":"","State\_vod\_c":"NJ","Zip\_vod\_c":"435465"}  
Action Create\_SFaddress  
Notes  
IsError   
PrescriberId ce1820b0-ee2c-e611-80e7-5065f38a5ba1

[Back to List](#)

© 2016 - VNBConsulting, Inc

## Log management portal

To view the logs, we also created an MVC web app. Here are examples of our MVC app details created with the previously described approach.

### Sample log detail view

PeterJamesChalmers\_1466191912

Document

Save Discard Delete Properties

```

1 [
2   "id": "PeterJamesChalmers_1466191912",
3   "patientId": "PeterJamesChalmers",
4   "timestamp": "2016-06-17T19:31:51.8360138Z",
5   "source": "Discussion on the results of your recent MRI START: 2013-12-10T09:00:00Z END: 2013-12-10T11:00:00Z COMMENT: Further expand on the results of the MRI and determine the next actions that may be appropriate.",
6   "operation": "General Discussion",
7   "Provider": "DrAdamCareful",
8   "ttl": 2592000,
9   "notDeleted": true
10 ]

```

Properties	
_RID	Uw4fAJrEEwEqAAAAAAA==
_TS	Fri, 17 Jun 2016 19:31:50 GMT
_SELF	dbs/Uw4fAA==/colls/Uw4fAJrEEwE=/docs/Uw4fAJrEEwE=/_self
_ETAG	"0000dc00-0000-0000-0000-576450290C
_ATTACHMENTS	attachments/

## API app details

### Logic Apps exception management API

Our open-source Azure Logic Apps exception management API app provides functionality as described here - there are two controllers:

- **ErrorController** inserts an error record (document) in a DocumentDB collection.
- **LogController** Inserts a log record (document) in a DocumentDB collection.

#### TIP

Both controllers use `async Task<dynamic>` operations, allowing operations to resolve at runtime, so we can create the DocumentDB schema in the body of the operation.

Every document in DocumentDB must have a unique ID. We are using `PatientId` and adding a timestamp that is converted to a Unix timestamp value (double). We truncate the value to remove the fractional value.

You can view the source code of our error controller API [from GitHub](#).

We call the API from a logic app by using the following syntax:

```
"actions": {
    "CreateErrorRecord": {
        "metadata": {
            "apiDefinitionUrl": "https://.../swagger/docs/v1",
            "swaggerSource": "website"
        },
        "type": "Http",
        "inputs": {
            "body": {
                "action": "New_Patient",
                "isError": true,
                "crmId": "@{triggerBody()['CRMID']}",
                "prescriberId": "@{triggerBody()['CRMID']}",
                "message": "@{body('Create_NewPatientRecord')['message']}",
                "salesforceId": "@{triggerBody()['salesforceID']}",
                "severity": 4,
                "source": "@{actions('Create_NewPatientRecord')['inputs']['body']}",
                "statusCode": "@{int(outputs('Create_NewPatientRecord')['statusCode'])}",
                "update": false
            },
            "method": "post",
            "uri": "https://.../api/CrMtoSfError"
        },
        "runAfter": {
            "Create_NewPatientRecord": ["Failed"]
        }
    }
}
```

The expression in the preceding code sample checks for the *Create\_NewPatientRecord* status of **Failed**.

## Summary

- You can easily implement logging and error handling in a logic app.
- You can use DocumentDB as the repository for log and error records (documents).
- You can use MVC to create a portal to display log and error records.

### Source code

The source code for the Logic Apps exception management API application is available in this [GitHub repository](#).

## Next steps

- [View more logic app examples and scenarios](#)
- [Learn about monitoring logic apps](#)
- [Create automated deployment templates for logic apps](#)

# Receive data in logic apps with the B2B features in the Enterprise Integration Pack

2/7/2017 • 2 min to read • [Edit Online](#)

After you create an integration account that has partners and agreements, you are ready to create a business to business (B2B) workflow for your logic app with the [Enterprise Integration Pack](#).

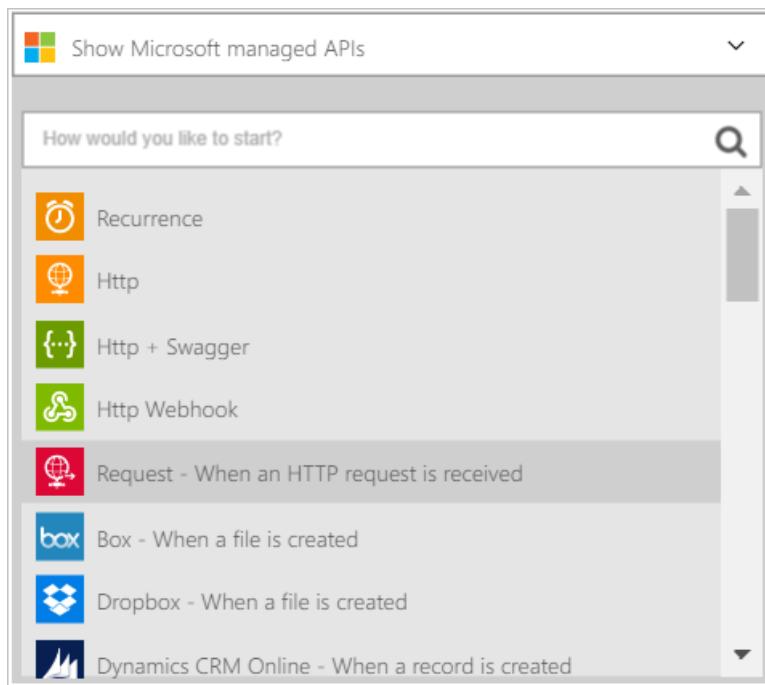
## Prerequisites

To use the AS2 and X12 actions, you must have an Enterprise Integration Account. Learn [how to create an Enterprise Integration Account](#).

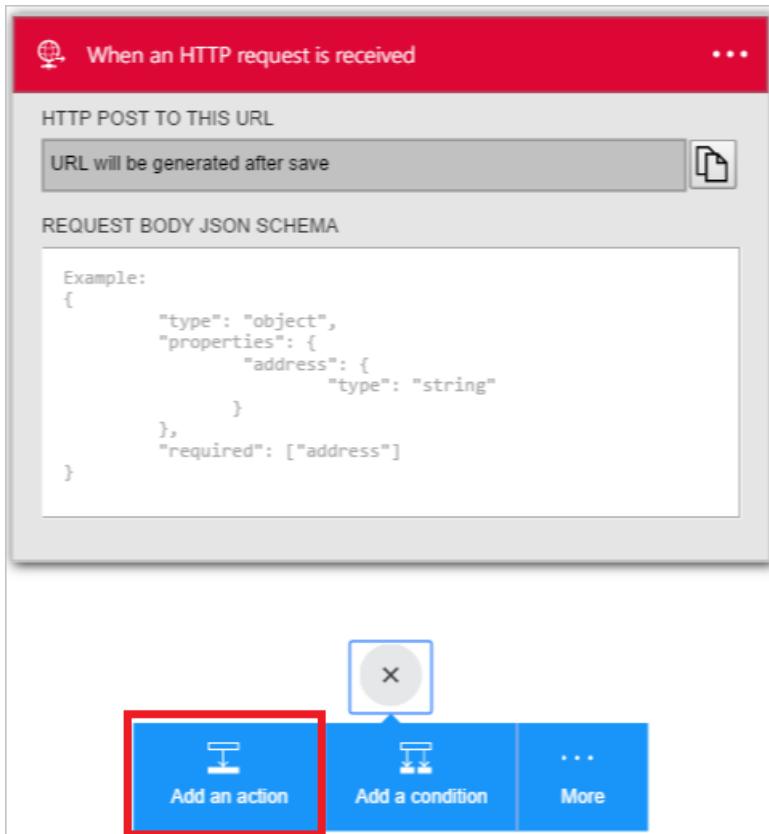
## Create a logic app with B2B connectors

Follow these steps to create a B2B logic app that uses the AS2 and X12 actions to receive data from a trading partner:

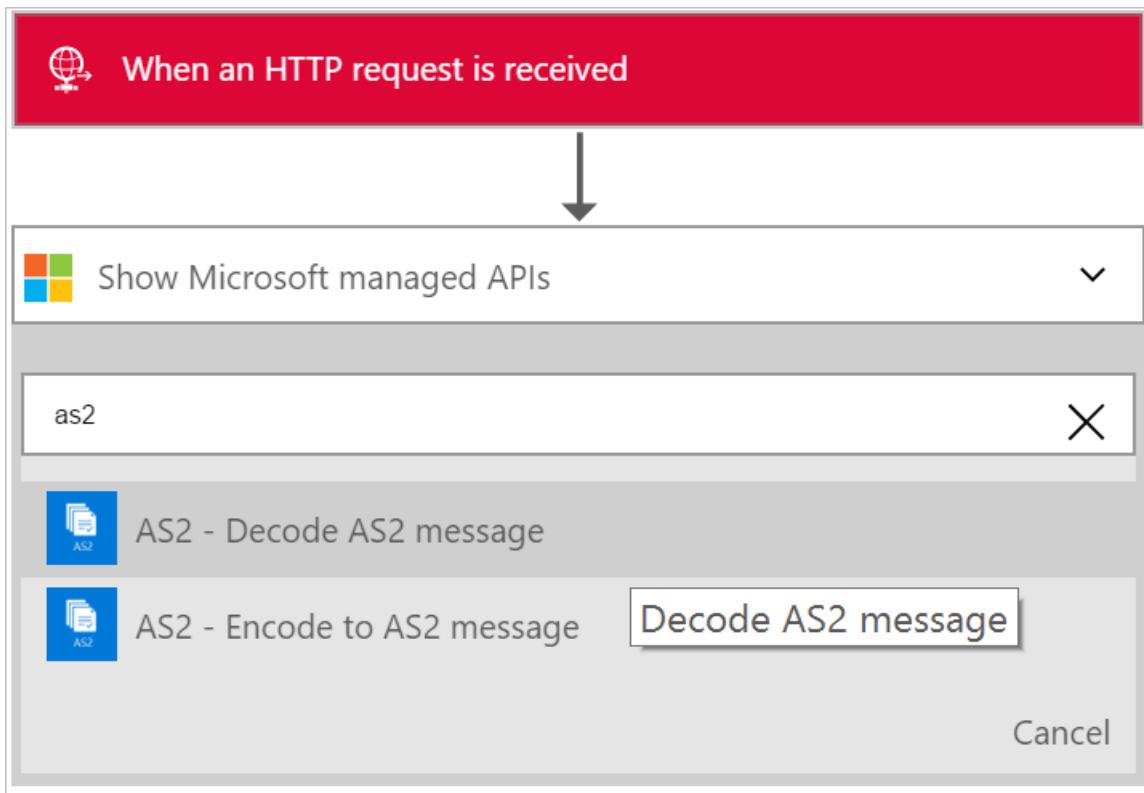
1. Create a logic app, then [link your app to your integration account](#).
2. Add a **Request - When an HTTP request is received** trigger to your logic app.



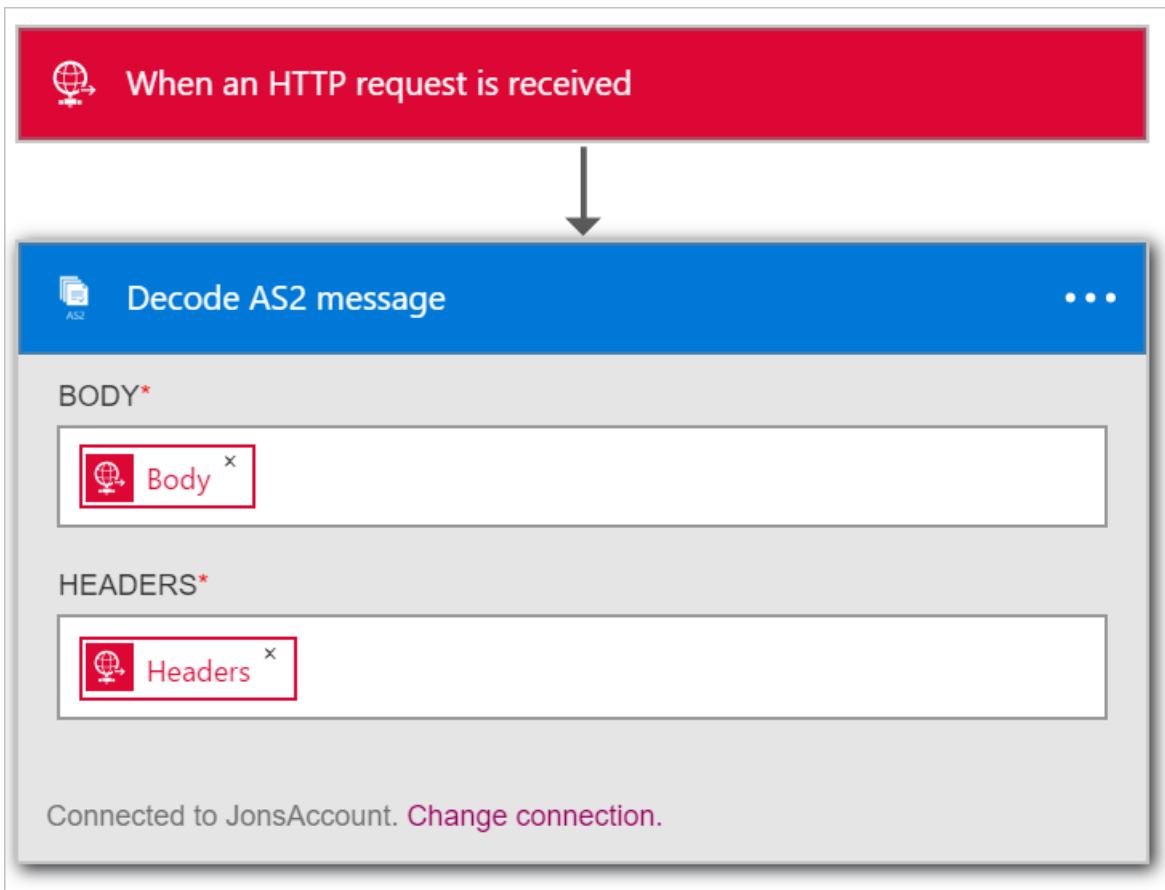
3. To add the **Decode AS2** action, select **Add an action**.



4. To filter all actions to the one that you want, enter the word **as2** in the search box.



5. Select the **AS2 - Decode AS2 message** action.



6. Add the **Body** that you want to use as input. In this example, select the body of the HTTP request that triggers the logic app. Or enter an expression that inputs the headers in the **HEADERS** field:

```
@triggerOutputs()['headers']
```

7. Add the required **Headers** for AS2, which you can find in the HTTP request headers. In this example, select the headers of the HTTP request that trigger the logic app.
8. Now add the Decode X12 message action. Select **Add an action**.



When an HTTP request is received



Decode AS2 message

...

BODY\*



HEADERS\*



Connected to JonsAccount. [Change connection.](#)



Add an action

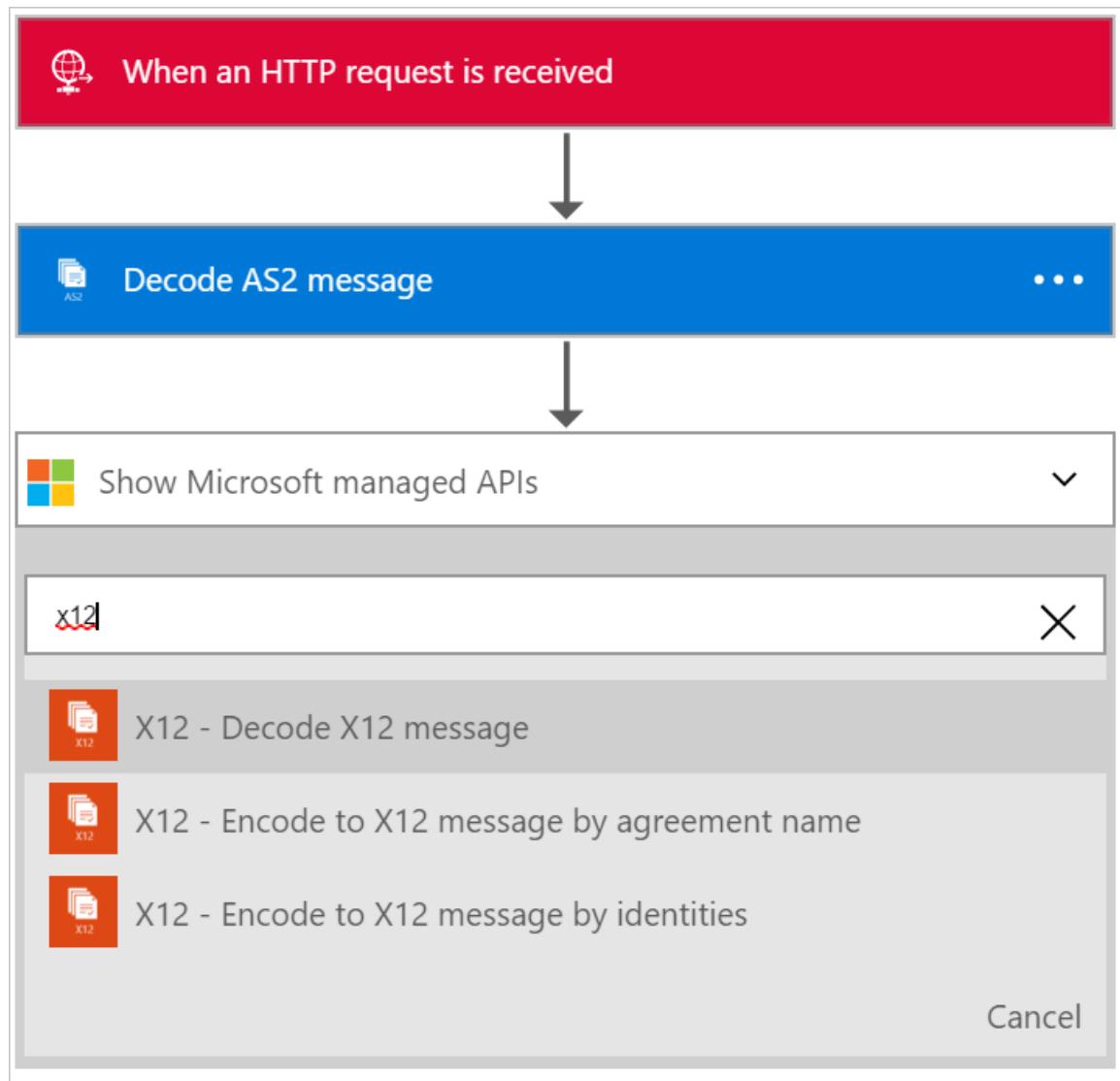


Add a condition

...

More

9. To filter all actions to the one that you want, enter the word **x12** in the search box.



10. Select the **X12 - Decode X12 message** action.

The screenshot shows the "Decode X12 message (Preview)" action selected in the Power Automate interface. Above it, the "Request" and "Decode AS2 message (Preview)" actions are also visible. The "Decode X12 message" action has a preview window below it containing the following dynamic content:

```
@base64ToBinary(body('Decode_AS2_message'))?['AS2Message']?['Content']
```

Below the preview window, there is a "Connected to EIPIntegration. [Change connection](#)." link.

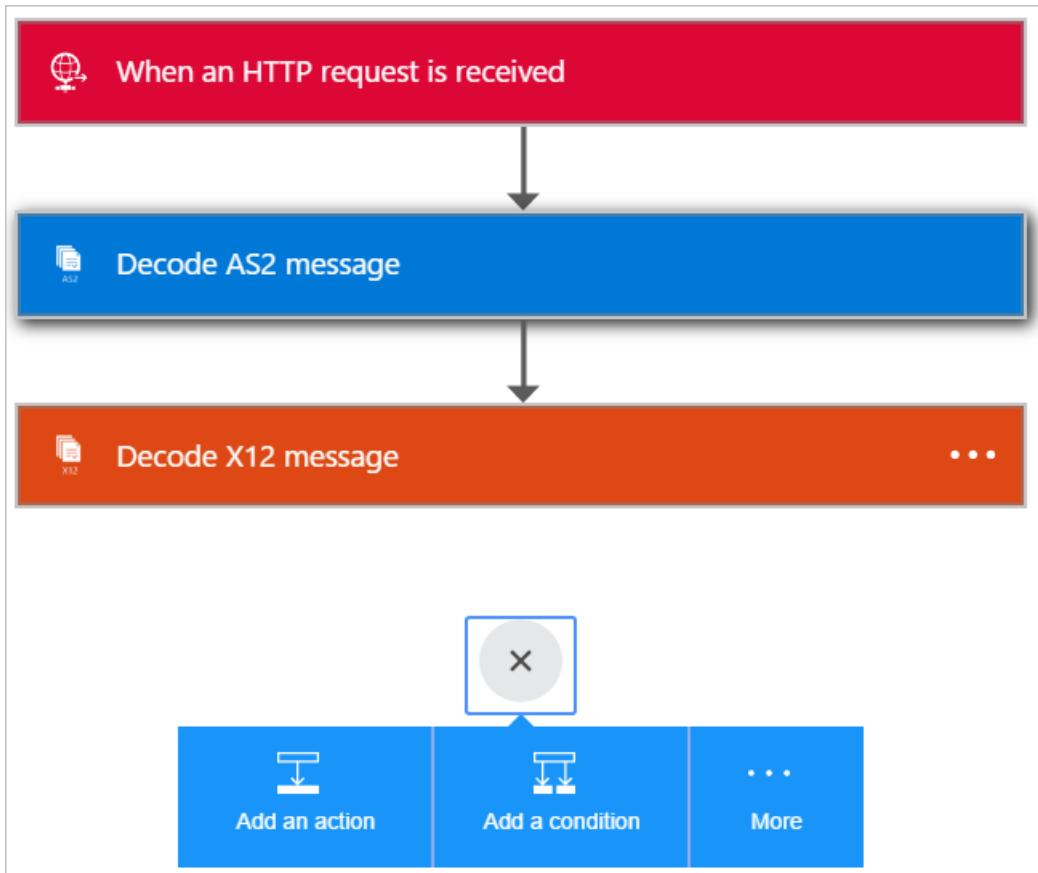
11. Now you must specify the input to this action. This input is the output from the previous AS2 action.

The actual message content is in a JSON object and is base64 encoded, so you must specify an expression as the input. Enter the following expression in the **X12 FLAT FILE MESSAGE TO DECODE** input field:

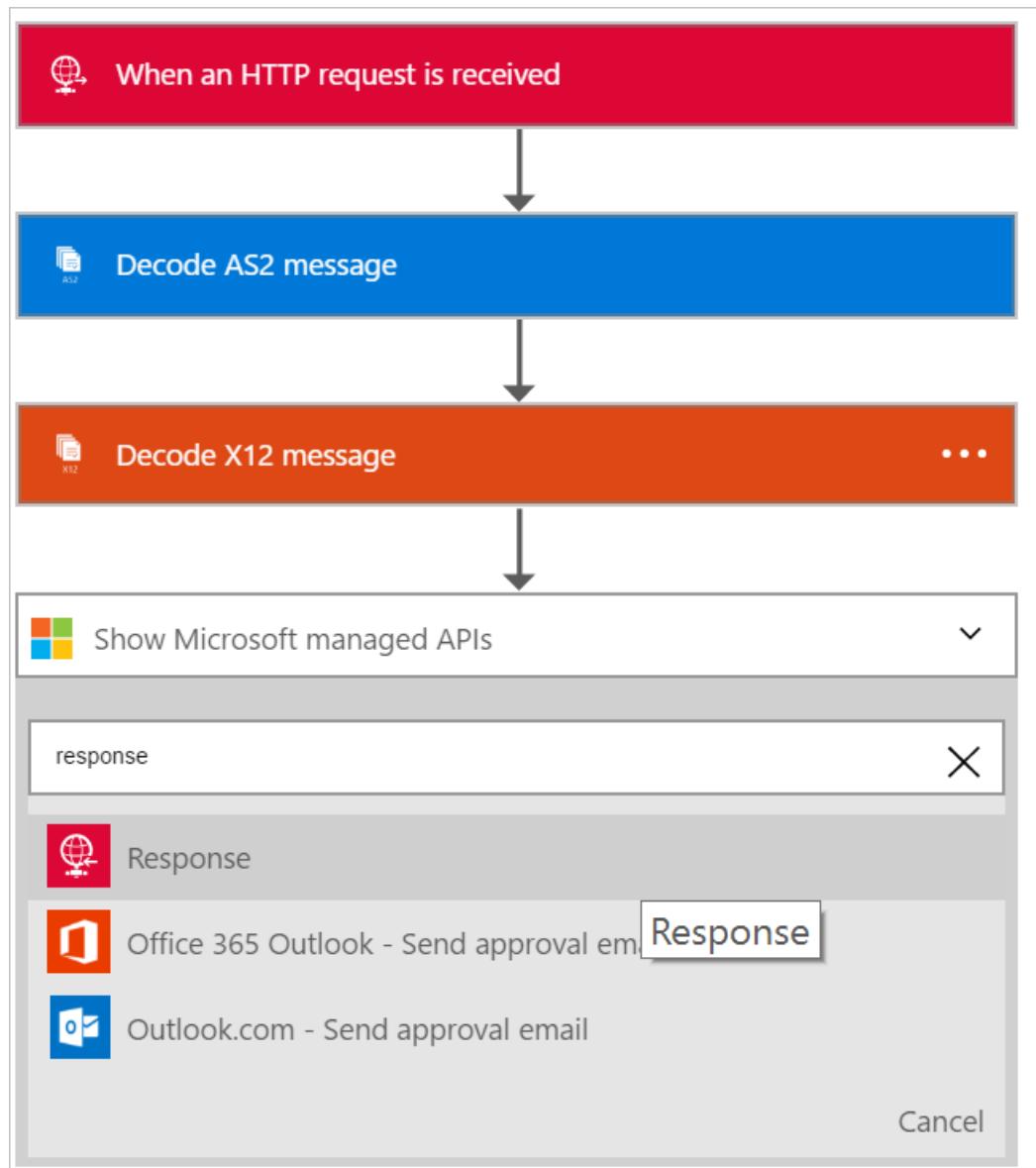
```
@base64ToString(body('Decode_AS2_message')?['AS2Message']?['Content'])
```

Now add steps to decode the X12 data received from the trading partner and output items in a JSON object. To notify the partner that the data was received, you can send back a response containing the AS2 Message Disposition Notification (MDN) in an HTTP Response Action.

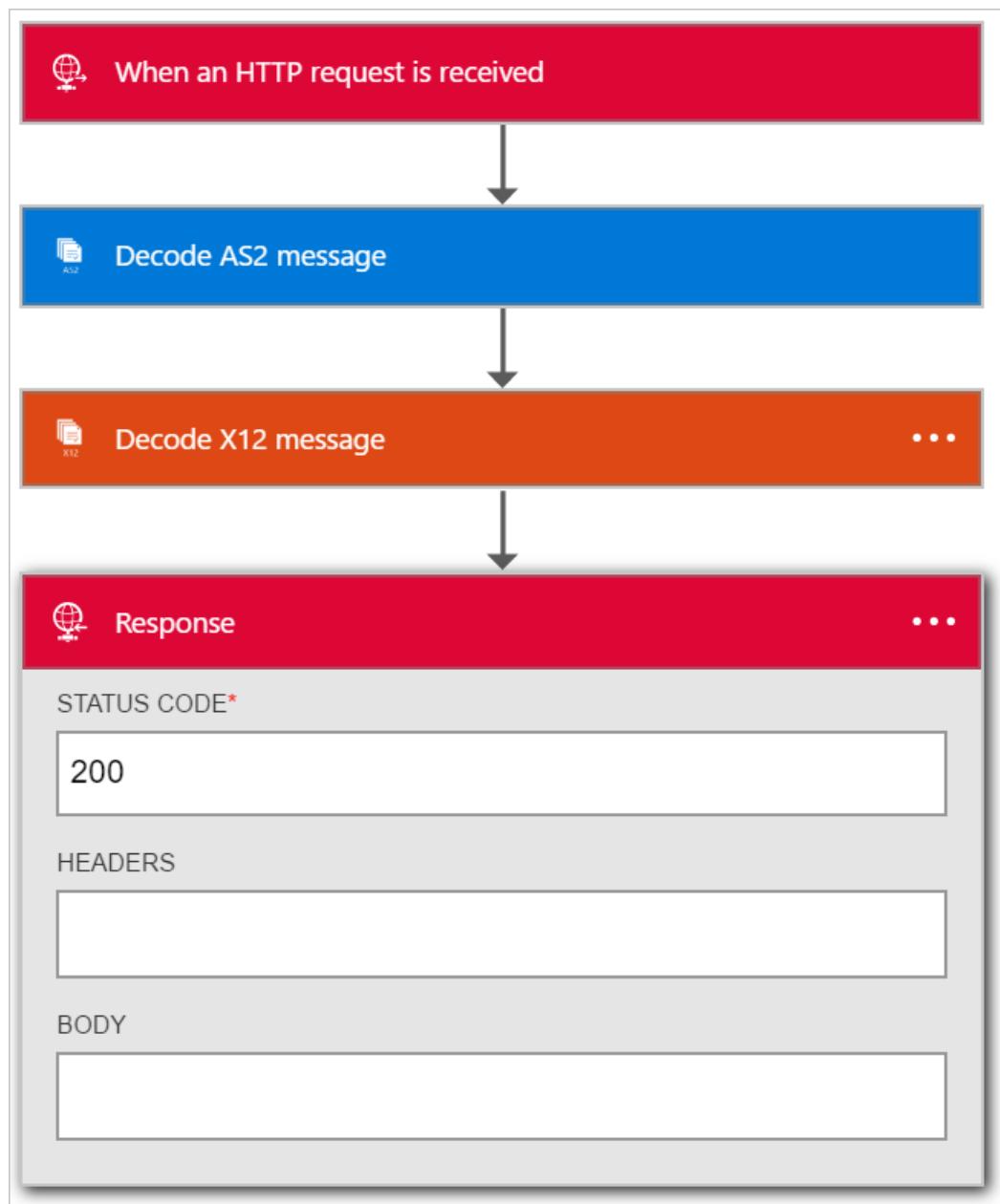
12. To add the **Response** action, choose **Add an action**.



13. To filter all actions to the one that you want, enter the word **response** in the search box.

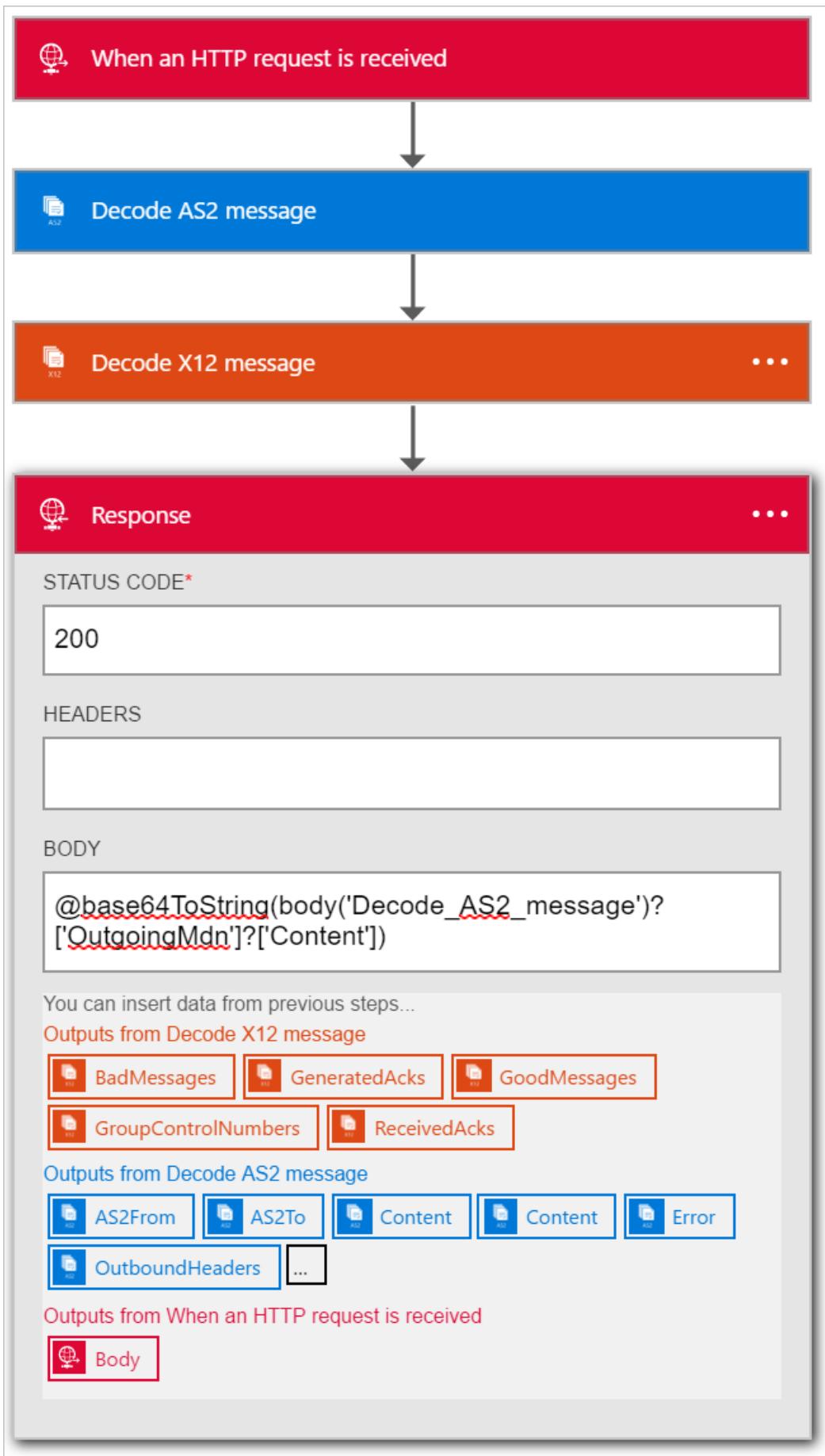


14. Select the **Response** action.

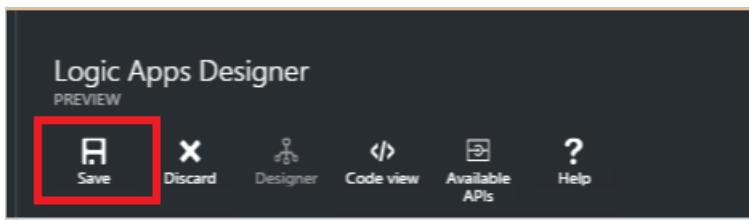


15. To access the MDN from the output of the **Decode X12 message** action, set the response **BODY** field with this expression:

```
@base64ToString(body('Decode_X12_message')?['OutgoingMdn']?['Content'])
```



16. Save your work.



You are now done setting up your B2B logic app. In a real world application, you might want to store the decoded X12 data in a line-of-business (LOB) app or data store. To connect your own LOB apps and use these APIs in your logic app, you can add further actions or write custom APIs.

## Features and use cases

- The AS2 and X12 decode and encode actions let you exchange data between trading partners by using industry standard protocols in logic apps.
- To exchange data with trading partners, you can use AS2 and X12 with or without each other.
- The B2B actions help you create partners and agreements easily in your integration account and consume them in a logic app.
- When you extend your logic app with other actions, you can send and receive data between other apps and services like SalesForce.

## Learn more

[Learn more about the Enterprise Integration Pack](#)

# Logic Apps pricing model

3/16/2017 • 3 min to read • [Edit Online](#)

Azure Logic Apps allows you to scale and execute integration workflows in the cloud. Following are details on the billing and pricing plans for Logic Apps.

## Consumption pricing

Newly created Logic Apps use a consumption plan. With the Logic Apps consumption pricing model, you only pay for what you use. Logic Apps are not throttled when using a consumption plan. All actions executed in a run of a logic app instance are metered.

### What are action executions?

Every step in a logic app definition is an action, which includes triggers, control flow steps like conditions, scopes, for each loops, do until loops, calls to connectors and calls to native actions. Triggers are special actions that are designed to instantiate a new instance of a logic app when a particular event occurs. There are several different behaviors for triggers, which could affect how the logic app is metered.

- **Polling trigger** – this trigger continually polls an endpoint until it receives a message that satisfies the criteria for creating an instance of a logic app. The polling interval can be configured in the trigger in the Logic App Designer. Each polling request, even if it doesn't create an instance of a logic app, counts as an action execution.
- **Webhook trigger** – this trigger waits for a client to send it a request on a particular endpoint. Each request sent to the webhook endpoint counts as an action execution. The Request and the HTTP Webhook trigger are both webhook triggers.
- **Recurrence trigger** – this trigger creates an instance of the logic app based on the recurrence interval configured in the trigger. For example, a recurrence trigger can be configured to run every three days or even every minute.

Trigger executions can be seen in the Logic Apps resource blade in the Trigger History part.

All actions that were executed, whether they were successful or failed are metered as an action execution. Actions that were skipped due to a condition not being met or actions that didn't execute because the logic app terminated before completion are not counted as action executions.

Actions executed within loops are counted per iteration of the loop. For example, a single action in a for each loop iterating through a list of 10 items are counted as the number of items in the list (10) multiplied by the number of actions in the loop (1) plus one for the initiation of the loop, which, in this example, would be  $(10 * 1) + 1 = 11$  action executions. Disabled Logic Apps cannot have new instances instantiated and therefore, while disabled, are not charged. Be mindful that after disabling a logic app it may take a little time for the instances to quiesce before being completely disabled.

### Integration Account Usage

Included in consumption-based usage is an [integration account](#) for exploration, development, and testing purposes allowing you to use the [B2B/EDI](#) and [XML processing](#) features of Logic Apps at no additional cost. You are able to create a maximum of one account per region and store up to 10 Agreements and 25 maps. Schemas, certificates, and partners have no limits and you can upload as many as you need.

In addition to the inclusion of integration accounts with consumption, you can also create standard integration accounts without these limits and with our standard Logic Apps SLA. For more information, see [Azure pricing](#).

## App Service plans

Logic apps previously created referencing an App Service Plan continues to behave as before. Depending on the plan chosen, are throttled after the prescribed daily executions are exceeded but are billed using the action execution meter. EA customers that have an App Service Plan in their subscription, which does not have to be explicitly associated with the Logic App, get the included quantities benefit. For example, if you have a Standard App Service Plan in your EA subscription and a Logic App in the same subscription then you aren't charged for 10,000 action executions per day (see following table).

App Service Plans and their daily allowed action executions:

	FREE/SHARED/BASIC	STANDARD	PREMIUM
Action executions per day	200	10,000	50,000

### Convert from App Service Plan pricing to Consumption

To change a Logic App that has an App Service Plan associated with it to a consumption model, remove the reference to the App Service Plan in the Logic App definition. This change can be done with a call to a PowerShell cmdlet: `Set-AzureRmLogicApp -ResourceGroupName 'rgname' -Name 'wfname' -UseConsumptionModel -Force`

## Pricing

For pricing details, see [Logic Apps Pricing](#).

## Next steps

- [An overview of Logic Apps](#)
- [Create your first logic app](#)

# Workflow Definition Language schema for Azure Logic Apps

3/27/2017 • 31 min to read • [Edit Online](#)

A workflow definition contains the actual logic that executes as a part of your logic app. This definition includes one or more triggers that start the logic app, and one or more actions for the logic app to take.

## Basic workflow definition structure

Here is the basic structure of a workflow definition:

```
{  
  "$schema": "<schema-of-the-definition>",  
  "contentVersion": "<version-number-of-definition>",  
  "parameters": { <parameter-definitions-of-definition> },  
  "triggers": [ { <definition-of-flow-triggers> } ],  
  "actions": [ { <definition-of-flow-actions> } ],  
  "outputs": { <output-of-definition> }  
}
```

### NOTE

The [Workflow Management REST API](#) documentation has information on how to create and manage logic app workflows.

ELEMENT NAME	REQUIRED	DESCRIPTION
\$schema	No	Specifies the location for the JSON schema file that describes the version of the definition language. This location is required when you reference a definition externally. For this document, the location is: <a href="https://schema.management.azure.com/providers/Microsoft.Logic/2018-01-preview/workflowdefinition.json#">https://schema.management.azure.com/providers/Microsoft.Logic/2018-01-preview/workflowdefinition.json#</a>
contentVersion	No	Specifies the definition version. When you deploy a workflow using the definition, you can use this value to make sure that the right definition is used.
parameters	No	Specifies parameters used to input data into the definition. A maximum of 50 parameters can be defined.
triggers	No	Specifies information for the triggers that initiate the workflow. A maximum of 250 triggers can be defined.
actions	No	Specifies actions that are taken as the flow executes. A maximum of 250 actions can be defined.
outputs	No	Specifies information about the deployed resource. A maximum of 10 outputs can be defined.

## Parameters

This section specifies all the parameters that are used in the workflow definition at deployment time. All parameters must be declared in this section before they can be used in other sections of the definition.

The following example shows the structure of a parameter definition:

```
"parameters": {  
  "<parameter-name>": {  
    "type": "<type-of-parameter-value>",  
    "defaultValue": <default-value-of-parameter>,  
    "allowedValues": [ <array-of-allowed-values> ],  
    "metadata": { "key": { "name": "value" } }  
  }  
}
```

ELEMENT NAME	REQUIRED	DESCRIPTION
type	Yes	<p><b>Type:</b> string</p> <p><b>Declaration:</b></p> <div style="border: 1px solid black; padding: 5px;">"parameters": {"parameter1": {"type": "string"}}</div> <p><b>Specification:</b></p> <div style="border: 1px solid black; padding: 5px;">"parameters": {"parameter1": {"value": "myparamvalue1"}}</div> <p><b>Type:</b> securestring</p> <p><b>Declaration:</b></p> <div style="border: 1px solid black; padding: 5px;">"parameters": {"parameter1": {"type": "securestring"}}</div> <p><b>Specification:</b></p> <div style="border: 1px solid black; padding: 5px;">"parameters": {"parameter1": {"value": "myparamvalue1"}}</div> <p><b>Type:</b> int</p> <p><b>Declaration:</b></p> <div style="border: 1px solid black; padding: 5px;">"parameters": {"parameter1": {"type": "int"}}</div> <p><b>Specification:</b></p> <div style="border: 1px solid black; padding: 5px;">"parameters": {"parameter1": {"value": 5}}</div> <p><b>Type:</b> bool</p> <p><b>Declaration:</b></p> <div style="border: 1px solid black; padding: 5px;">"parameters": {"parameter1": {"type": "array"}}</div> <p><b>Specification:</b></p> <div style="border: 1px solid black; padding: 5px;">"parameters": {"parameter1": { "value": true }}</div> <p><b>Type:</b> array</p> <p><b>Declaration:</b></p> <div style="border: 1px solid black; padding: 5px;">"parameters": {"parameter1": {"type": "array"}}</div> <p><b>Specification:</b></p> <div style="border: 1px solid black; padding: 5px;">"parameters": {"parameter1": { "value": [ array-of-values ]}}</div> <p><b>Type:</b> object</p> <p><b>Declaration:</b></p> <div style="border: 1px solid black; padding: 5px;">"parameters": {"parameter1": {"type": "object"}}</div> <p><b>Specification:</b></p> <div style="border: 1px solid black; padding: 5px;">"parameters": {"parameter1": { "value": { JSON-object } }}</div> <p><b>Type:</b> secureobject</p> <p><b>Declaration:</b></p> <div style="border: 1px solid black; padding: 5px;">"parameters": {"parameter1": {"type": "object"}}</div> <p><b>Specification:</b></p> <div style="border: 1px solid black; padding: 5px;">"parameters": {"parameter1": { "value": { JSON-object } }}</div> <p><b>Note:</b> The <code>securestring</code> and <code>secureobject</code> types are not returned in <code>GET</code> operations. All passwords, keys, and secrets should use this type.</p>
defaultValue	No	Specifies the default value for the parameter when no value is specified at the time the resource is created.
allowedValues	No	Specifies an array of allowed values for the parameter.
metadata	No	Specifies additional information about the parameter, such as a readable description or design-time data used by Visual Studio or other tools.

This example shows how you can use a parameter in the body section of an action:

```

"body" :
{
  "property1": "@parameters('parameter1')"
}

```

Parameters can also be used in outputs.

## Triggers and actions

Triggers and actions specify the calls that can participate in workflow execution. For details about this section, see [Workflow Actions and Triggers](#).

## Outputs

Outputs specify information that can be returned from a workflow run. For example, if you have a specific status or value that you want to track for each run, you can include that data in the run outputs. The data appears in the Management REST API for that run, and in the management UI for that run in the Azure portal. You can also flow these outputs to other external systems like PowerBI for creating dashboards. Outputs are *not* used to respond to incoming requests on the Service REST API. To respond to an incoming request using the `response` action type, here's an example:

```

"outputs": {
  "key1": {
    "value": "value1",
    "type" : "<type-of-value>"
  }
}

```

ELEMENT NAME	REQUIRED	DESCRIPTION
key1	Yes	Specifies the key identifier for the output. Replace <b>key1</b> with a name that you want to use to identify the output.
value	Yes	Specifies the value of the output.
type	Yes	Specifies the type for the value that was specified. Possible types of values are: <ul style="list-style-type: none"> <li>- <code>string</code></li> <li>- <code>securestring</code></li> <li>- <code>int</code></li> <li>- <code>bool</code></li> <li>- <code>array</code></li> <li>- <code>object</code></li> </ul>

## Expressions

JSON values in the definition can be literal, or they can be expressions that are evaluated when the definition is used. For example:

```
"name": "value"
```

or

```
"name": "@parameters('password') "
```

### NOTE

Some expressions get their values from runtime actions that might not exist at the beginning of the execution. You can use **functions** to help retrieve some of these values.

Expressions can appear anywhere in a JSON string value and always result in another JSON value. When a JSON value has been determined to be an expression, the body of the expression is extracted by removing the at-sign (@). If a literal string is needed that starts with @, that string must be escaped by using @@. The following examples show how expressions are evaluated.

JSON VALUE	RESULT
"parameters"	The characters 'parameters' are returned.
"parameters[1]"	The characters 'parameters[1]' are returned.

JSON VALUE	RESULT
"@@"	A 1 character string that contains '@' is returned.
" @"	A 2 character string that contains ' @' is returned.

With *string interpolation*, expressions can also appear inside strings where expressions are wrapped in `@{ ... }`. For example:

```
"name" : "First Name: @{parameters('firstName')} Last Name: @{parameters('lastName')}
```

The result is always a string, which makes this feature similar to the `concat` function. Suppose you defined `myNumber` as `42` and `myString` as `sampleString`:

JSON VALUE	RESULT
"@parameters('myString')"	Returns <code>sampleString</code> as a string.
"@{parameters('myString')}"	Returns <code>sampleString</code> as a string.
"@parameters('myNumber')"	Returns <code>42</code> as a <i>number</i> .
"@{parameters('myNumber')}"	Returns <code>42</code> as a <i>string</i> .
"Answer is: @{parameters('myNumber')}"	Returns the string <code>Answer is: 42</code> .
"@concat('Answer is: ', string(parameters('myNumber')))"	Returns the string <code>Answer is: 42</code>
"Answer is: @@{parameters('myNumber')}"	Returns the string <code>Answer is: @{parameters('myNumber')}</code> .

## Operators

Operators are the characters that you can use inside expressions or functions.

OPERATOR	DESCRIPTION
.	The dot operator allows you to reference properties of an object
?	The question mark operator lets you reference null properties of an object without a runtime error. For example, you can use this expression to handle null trigger outputs: <code>@coalesce(trigger().outputs?.body?.property1, 'my default value')</code>
'	The single quotation mark is the only way to wrap string literals. You cannot use double-quotes inside expressions because this punctuation conflicts with the JSON quote that wraps the whole expression.
[]	The square brackets can be used to get a value from an array with a specific index. For example, if you have an action that passes <code>range(0,10)</code> in to the <code>forEach</code> function, you can use this function to get items out of arrays: <code>myArray[item()]</code>

## Functions

You can also call functions within expressions. The following table shows the functions that can be used in an expression.

EXPRESSION	EVALUATION
"@function('Hello')"	Calls the function member of the definition with the literal string Hello as the first parameter.
"@function('It's Cool!')"	Calls the function member of the definition with the literal string 'It's Cool!' as the first parameter
"@function().prop1"	Returns the value of the prop1 property of the <code>myFunction</code> member of the definition.

EXPRESSION	EVALUATION
"@function('Hello').prop1"	Calls the function member of the definition with the literal string 'Hello' as the first parameter and returns the prop1 property of the object.
"@function(parameters('Hello'))"	Evaluates the Hello parameter and passes the value to function

## Referencing functions

You can use these functions to reference outputs from other actions in the logic app or values passed in when the logic app was created. For example, you can reference the data from one step to use it in another.

FUNCTION NAME	DESCRIPTION
parameters	<p>Returns a parameter value that is defined in the definition.</p> <pre>parameters('password')</pre> <p><b>Parameter number:</b> 1</p> <p><b>Name:</b> Parameter</p> <p><b>Description:</b> Required. The name of the parameter whose values you want.</p>
action	<p>Enables an expression to derive its value from other JSON name and value pairs or the output of the current runtime action. The property represented by propertyPath in the following example is optional. If propertyPath is not specified, the reference is to the whole action object. This function can only be used inside do-until conditions of an action.</p> <pre>action().outputs.body.propertyPath</pre>
actions	<p>Enables an expression to derive its value from other JSON name and value pairs or the output of the runtime action. These expressions explicitly declare that one action depends on another action. The property represented by propertyPath in the following example is optional. If propertyPath is not specified, the reference is to the whole action object. You can use either this element or the conditions element to specify dependencies, but you do not need to use both for the same dependent resource.</p> <pre>actions('myAction').outputs.body.propertyPath</pre> <p><b>Parameter number:</b> 1</p> <p><b>Name:</b> Action name</p> <p><b>Description:</b> Required. The name of the action whose values you want.</p>
trigger	<p>Enables an expression to derive its value from other JSON name and value pairs or the output of the runtime trigger. The property represented by propertyPath in the following example is optional. If propertyPath is not specified, the reference is to the whole trigger object.</p> <pre>trigger().outputs.body.propertyPath</pre> <p>When used inside a trigger's inputs, the function returns the outputs of the previous execution. However, when used inside a trigger's condition, the <code>trigger</code> function returns the outputs of the current execution.</p>
actionOutputs	<p>This function is shorthand for <code>actions('actionName').outputs</code></p> <p><b>Parameter number:</b> 1</p> <p><b>Name:</b> Action name</p> <p><b>Description:</b> Required. The name of the action whose values you want.</p>
actionBody and body	<p>These functions are shorthand for <code>actions('actionName').outputs.body</code></p> <p><b>Parameter number:</b> 1</p> <p><b>Name:</b> Action name</p> <p><b>Description:</b> Required. The name of the action whose values you want.</p>
triggerOutputs	<p>This function is shorthand for <code>trigger().outputs</code></p>
triggerBody	<p>This function is shorthand for <code>trigger().outputs.body</code></p>

FUNCTION NAME	DESCRIPTION
item	<p>When used inside a repeating action, this function returns the item that is in the array for this iteration of the action. For example, if you have an action that runs for each item an array of messages, you can use this syntax:</p> <pre>"input1" : "@item().subject"</pre>

## Collection functions

These functions operate over collections and generally apply to Arrays, Strings, and sometimes Dictionaries.

FUNCTION NAME	DESCRIPTION
contains	<p>Returns true if dictionary contains a key, list contains value, or string contains substring. For example, this function returns <code>true</code>:</p> <pre>contains('abacaba','aca')</pre> <p><b>Parameter number:</b> 1</p> <p><b>Name:</b> Within collection</p> <p><b>Description:</b> Required. The collection to search within.</p> <p><b>Parameter number:</b> 2</p> <p><b>Name:</b> Find object</p> <p><b>Description:</b> Required. The object to find inside the <b>Within collection</b>.</p>
length	<p>Returns the number of elements in an array or string. For example, this function returns <code>3</code>:</p> <pre>length('abc')</pre> <p><b>Parameter number:</b> 1</p> <p><b>Name:</b> Collection</p> <p><b>Description:</b> Required. The collection for which to get the length.</p>
empty	<p>Returns true if object, array, or string is empty. For example, this function returns <code>true</code>:</p> <pre>empty('')</pre> <p><b>Parameter number:</b> 1</p> <p><b>Name:</b> Collection</p> <p><b>Description:</b> Required. The collection to check if it is empty.</p>
intersection	<p>Returns a single array or object that has common elements between arrays or objects passed in. For example, this function returns <code>[1, 2]</code>:</p> <pre>intersection([1, 2, 3], [101, 2, 1, 10],[6, 8, 1, 2])</pre> <p>The parameters for the function can either be a set of objects or a set of arrays (not a mixture of both). If there are two objects with the same name, the last object with that name appears in the final object.</p> <p><b>Parameter number:</b> 1 ... n</p> <p><b>Name:</b> Collection n</p> <p><b>Description:</b> Required. The collections to evaluate. An object must be in all collections passed in to appear in the result.</p>
union	<p>Returns a single array or object with all the elements that are in either array or object passed to this function. For example, this function returns <code>[1, 2, 3, 10, 101]</code>:</p> <pre>union([1, 2, 3], [101, 2, 1, 10])</pre> <p>The parameters for the function can either be a set of objects or a set of arrays (not a mixture thereof). If there are two objects with the same name in the final output, the last object with that name appears in the final object.</p> <p><b>Parameter number:</b> 1 ... n</p> <p><b>Name:</b> Collection n</p> <p><b>Description:</b> Required. The collections to evaluate. An object that appears in any of the collections also appears in the result.</p>

FUNCTION NAME	DESCRIPTION
first	<p>Returns the first element in the array or string passed in. For example, this function returns <code>0</code>:</p> <pre>first([0,2,3])</pre> <p><b>Parameter number:</b> 1</p> <p><b>Name:</b> Collection</p> <p><b>Description:</b> Required. The collection to take the first object from.</p>
last	<p>Returns the last element in the array or string passed in. For example, this function returns <code>3</code>:</p> <pre>last('0123')</pre> <p><b>Parameter number:</b> 1</p> <p><b>Name:</b> Collection</p> <p><b>Description:</b> Required. The collection to take the last object from.</p>
take	<p>Returns the first <b>Count</b> elements from the array or string passed in. For example, this function returns <code>[1, 2]</code>:</p> <pre>take([1, 2, 3, 4], 2)</pre> <p><b>Parameter number:</b> 1</p> <p><b>Name:</b> Collection</p> <p><b>Description:</b> Required. The collection from where to take the first <b>Count</b> objects.</p> <p><b>Parameter number:</b> 2</p> <p><b>Name:</b> Count</p> <p><b>Description:</b> Required. The number of objects to take from the <b>Collection</b>. Must be a positive integer.</p>
skip	<p>Returns the elements in the array starting at index <b>Count</b>. For example, this function returns <code>[3, 4]</code>:</p> <pre>skip([1, 2, 3, 4], 2)</pre> <p><b>Parameter number:</b> 1</p> <p><b>Name:</b> Collection</p> <p><b>Description:</b> Required. The collection to skip the first <b>Count</b> objects from.</p> <p><b>Parameter number:</b> 2</p> <p><b>Name:</b> Count</p> <p><b>Description:</b> Required. The number of objects to remove from the front of <b>Collection</b>. Must be a positive integer.</p>
join	<p>Returns a string with each item of an array joined by a delimiter, for example this returns <code>"1,2,3,4"</code>:</p> <pre>join([1, 2, 3, 4], ',')</pre> <p><b>Parameter number:</b> 1</p> <p><b>Name:</b> Collection</p> <p><b>Description:</b> Required. The collection to join items from.</p> <p><b>Parameter number:</b> 2</p> <p><b>Name:</b> Delimiter</p> <p><b>Description:</b> Required. The string to delimit items with.</p>

## String functions

The following functions only apply to strings. You can also use some collection functions on strings.

FUNCTION NAME	DESCRIPTION
---------------	-------------

FUNCTION NAME	DESCRIPTION
concat	<p>Combines any number of strings together. For example, if parameter 1 is <code>p1</code>, this function returns <code>somevalue-p1-somevalue</code>:</p> <pre>concat('somevalue',parameters('parameter1'),'-somevalue')</pre> <p><b>Parameter number:</b> 1 ... n</p> <p><b>Name:</b> String n</p> <p><b>Description:</b> Required. The strings to combine into a single string.</p>
substring	<p>Returns a subset of characters from a string. For example, this function returns <code>abc</code>:</p> <pre>substring('somevalue-abc-somevalue',10,3)</pre> <p><b>Parameter number:</b> 1</p> <p><b>Name:</b> String</p> <p><b>Description:</b> Required. The string from which the substring is taken.</p> <p><b>Parameter number:</b> 2</p> <p><b>Name:</b> Start index</p> <p><b>Description:</b> Required. The index of where the substring begins in parameter 1.</p> <p><b>Parameter number:</b> 3</p> <p><b>Name:</b> Length</p> <p><b>Description:</b> Required. The length of the substring.</p>
replace	<p>Replaces a string with a given string. For example, this function returns <code>the new string</code>:</p> <pre>replace('the old string', 'old', 'new')</pre> <p><b>Parameter number:</b> 1</p> <p><b>Name:</b> string</p> <p><b>Description:</b> Required. The string that is searched for parameter 2 and updated with parameter 3, when parameter 2 is found in parameter 1.</p> <p><b>Parameter number:</b> 2</p> <p><b>Name:</b> Old string</p> <p><b>Description:</b> Required. The string to replace with parameter 3, when a match is found in parameter 1</p> <p><b>Parameter number:</b> 3</p> <p><b>Name:</b> New string</p> <p><b>Description:</b> Required. The string that is used to replace the string in parameter 2 when a match is found in parameter 1.</p>
guid	<p>This function generates a globally unique string (GUID). For example, this function can generate this GUID: <code>c2ecc88d-88c8-4096-912c-d6f2e2b138ce</code></p> <pre>guid()</pre> <p><b>Parameter number:</b> 1</p> <p><b>Name:</b> Format</p> <p><b>Description:</b> Optional. A single format specifier that indicates <a href="#">how to format the value of this Guid</a>. The format parameter can be "N", "D", "B", "P", or "X". If format is not provided, "D" is used.</p>
toLowerCase	<p>Converts a string to lowercase. For example, this function returns <code>two by two is four</code>:</p> <pre>toLowerCase('Two by Two is Four')</pre> <p><b>Parameter number:</b> 1</p> <p><b>Name:</b> String</p> <p><b>Description:</b> Required. The string to convert to lower casing. If a character in the string does not have a lowercase equivalent, the character is included unchanged in the returned string.</p>

FUNCTION NAME	DESCRIPTION
toUpperCase	<p>Converts a string to uppercase. For example, this function returns <code>TWO BY TWO IS FOUR</code> :</p> <pre>toUpperCase('Two by Two is Four')</pre> <p><b>Parameter number:</b> 1</p> <p><b>Name:</b> String</p> <p><b>Description:</b> Required. The string to convert to upper casing. If a character in the string does not have an uppercase equivalent, the character is included unchanged in the returned string.</p>
indexof	<p>Find the index of a value within a string case insensitively. For example, this function returns <code>7</code> :</p> <pre>indexof('hello, world.', 'world')</pre> <p><b>Parameter number:</b> 1</p> <p><b>Name:</b> String</p> <p><b>Description:</b> Required. The string that may contain the value.</p> <p><b>Parameter number:</b> 2</p> <p><b>Name:</b> String</p> <p><b>Description:</b> Required. The value to search the index of.</p>
lastindexof	<p>Find the last index of a value within a string case insensitively. For example, this function returns <code>3</code> :</p> <pre>lastindexof('foofoo', 'foo')</pre> <p><b>Parameter number:</b> 1</p> <p><b>Name:</b> String</p> <p><b>Description:</b> Required. The string that may contain the value.</p> <p><b>Parameter number:</b> 2</p> <p><b>Name:</b> String</p> <p><b>Description:</b> Required. The value to search the index of.</p>
startswith	<p>Checks if the string starts with a value case insensitively. For example, this function returns <code>true</code> :</p> <pre>lastindexof('hello, world', 'hello')</pre> <p><b>Parameter number:</b> 1</p> <p><b>Name:</b> String</p> <p><b>Description:</b> Required. The string that may contain the value.</p> <p><b>Parameter number:</b> 2</p> <p><b>Name:</b> String</p> <p><b>Description:</b> Required. The value the string may start with.</p>
endswith	<p>Checks if the string ends with a value case insensitively. For example, this function returns <code>true</code> :</p> <pre>lastindexof('hello, world', 'world')</pre> <p><b>Parameter number:</b> 1</p> <p><b>Name:</b> String</p> <p><b>Description:</b> Required. The string that may contain the value.</p> <p><b>Parameter number:</b> 2</p> <p><b>Name:</b> String</p> <p><b>Description:</b> Required. The value the string may end with.</p>

FUNCTION NAME	DESCRIPTION
split	<p>Splits the string using a separator. For example, this function returns <code>["a", "b", "c"]</code>:</p> <pre>split('a;b;c',';')</pre> <p><b>Parameter number:</b> 1</p> <p><b>Name:</b> String</p> <p><b>Description:</b> Required. The string that is split.</p> <p><b>Parameter number:</b> 2</p> <p><b>Name:</b> String</p> <p><b>Description:</b> Required. The separator.</p>

## Logical functions

These functions are useful inside conditions and can be used to evaluate any type of logic.

FUNCTION NAME	DESCRIPTION
equals	<p>Returns true if two values are equal. For example, if parameter1 is someValue, this function returns <code>true</code>:</p> <pre>equals(parameters('parameter1'), 'someValue')</pre> <p><b>Parameter number:</b> 1</p> <p><b>Name:</b> Object 1</p> <p><b>Description:</b> Required. The object to compare to <b>Object 2</b>.</p> <p><b>Parameter number:</b> 2</p> <p><b>Name:</b> Object 2</p> <p><b>Description:</b> Required. The object to compare to <b>Object 1</b>.</p>
less	<p>Returns true if the first argument is less than the second. Note, values can only be of type integer, float, or string. For example, this function returns <code>true</code>:</p> <pre>less(10,100)</pre> <p><b>Parameter number:</b> 1</p> <p><b>Name:</b> Object 1</p> <p><b>Description:</b> Required. The object to check if it is less than <b>Object 2</b>.</p> <p><b>Parameter number:</b> 2</p> <p><b>Name:</b> Object 2</p> <p><b>Description:</b> Required. The object to check if it is greater than <b>Object 1</b>.</p>
lessOrEquals	<p>Returns true if the first argument is less than or equal to the second. Note, values can only be of type integer, float, or string. For example, this function returns <code>true</code>:</p> <pre>lessOrEquals(10,10)</pre> <p><b>Parameter number:</b> 1</p> <p><b>Name:</b> Object 1</p> <p><b>Description:</b> Required. The object to check if it is less or equal to <b>Object 2</b>.</p> <p><b>Parameter number:</b> 2</p> <p><b>Name:</b> Object 2</p> <p><b>Description:</b> Required. The object to check if it is greater than or equal to <b>Object 1</b>.</p>

FUNCTION NAME	DESCRIPTION
greater	<p>Returns true if the first argument is greater than the second. Note, values can only be of type integer, float, or string. For example, this function returns <code>false</code>:</p> <pre>greater(10,10)</pre> <p><b>Parameter number:</b> 1</p> <p><b>Name:</b> Object 1</p> <p><b>Description:</b> Required. The object to check if it is greater than <b>Object 2</b>.</p> <p><b>Parameter number:</b> 2</p> <p><b>Name:</b> Object 2</p> <p><b>Description:</b> Required. The object to check if it is less than <b>Object 1</b>.</p>
greaterOrEquals	<p>Returns true if the first argument is greater than or equal to the second. Note, values can only be of type integer, float, or string. For example, this function returns <code>false</code>:</p> <pre>greaterOrEquals(10,100)</pre> <p><b>Parameter number:</b> 1</p> <p><b>Name:</b> Object 1</p> <p><b>Description:</b> Required. The object to check if it is greater than or equal to <b>Object 2</b>.</p> <p><b>Parameter number:</b> 2</p> <p><b>Name:</b> Object 2</p> <p><b>Description:</b> Required. The object to check if it is less than or equal to <b>Object 1</b>.</p>
and	<p>Returns true if both parameters are true. Both arguments need to be Booleans. For example, this function returns <code>false</code>:</p> <pre>and(greater(1,10),equals(0,0))</pre> <p><b>Parameter number:</b> 1</p> <p><b>Name:</b> Boolean 1</p> <p><b>Description:</b> Required. The first argument that must be <code>true</code>.</p> <p><b>Parameter number:</b> 2</p> <p><b>Name:</b> Boolean 2</p> <p><b>Description:</b> Required. The second argument must be <code>true</code>.</p>
or	<p>Returns true if either parameter is true. Both arguments need to be Booleans. For example, this function returns <code>true</code>:</p> <pre>or(greater(1,10),equals(0,0))</pre> <p><b>Parameter number:</b> 1</p> <p><b>Name:</b> Boolean 1</p> <p><b>Description:</b> Required. The first argument that may be <code>true</code>.</p> <p><b>Parameter number:</b> 2</p> <p><b>Name:</b> Boolean 2</p> <p><b>Description:</b> Required. The second argument may be <code>true</code>.</p>
not	<p>Returns true if the parameters are <code>false</code>. Both arguments need to be Booleans. For example, this function returns <code>true</code>:</p> <pre>not(contains('200 Success','Fail'))</pre> <p><b>Parameter number:</b> 1</p> <p><b>Name:</b> Boolean</p> <p><b>Description:</b> Returns true if the parameters are <code>false</code>. Both arguments need to be Booleans. This function returns <code>true</code>:</p> <pre>not(contains('200 Success','Fail'))</pre>

FUNCTION NAME	DESCRIPTION
if	<p>Returns a specified value based on whether the expression resulted in <code>true</code> or <code>false</code>. For example, this function returns <code>"yes"</code>:</p> <pre>if&gt;equals(1, 1), 'yes', 'no')</pre> <p><b>Parameter number:</b> 1</p> <p><b>Name:</b> Expression</p> <p><b>Description:</b> Required. A boolean value that determines which value the expression should return.</p> <p><b>Parameter number:</b> 2</p> <p><b>Name:</b> True</p> <p><b>Description:</b> Required. The value to return if the expression is <code>true</code>.</p> <p><b>Parameter number:</b> 3</p> <p><b>Name:</b> False</p> <p><b>Description:</b> Required. The value to return if the expression is <code>false</code>.</p>

## Conversion functions

These functions are used to convert between each of the native types in the language:

- string
- integer
- float
- boolean
- arrays
- dictionaries
- forms

FUNCTION NAME	DESCRIPTION
int	<p>Convert the parameter to an integer. For example, this function returns 100 as a number, rather than a string:</p> <pre>int('100')</pre> <p><b>Parameter number:</b> 1</p> <p><b>Name:</b> Value</p> <p><b>Description:</b> Required. The value that is converted to an integer.</p>
string	<p>Convert the parameter to a string. For example, this function returns <code>'10'</code>:</p> <pre>string(10)</pre> <p>You can also convert an object to a string. For example, if the <code>myPar</code> parameter is an object with one property <code>abc : xyz</code>, then this function returns <code>{"abc" : "xyz"}</code>:</p> <pre>string(parameters('myPar'))</pre> <p><b>Parameter number:</b> 1</p> <p><b>Name:</b> Value</p> <p><b>Description:</b> Required. The value that is converted to a string.</p>

FUNCTION NAME	DESCRIPTION
json	<p>Convert the parameter to a JSON type value and is the opposite of <code>string()</code>. For example, this function returns <code>[1,2,3]</code> as an array, rather than a string:</p> <pre>parse('[1,2,3]')</pre> <p>Likewise, you can convert a string to an object. For example, this function returns <code>{ "abc" : "xyz" }</code>:</p> <pre>json('{"abc" : "xyz"}')</pre> <p><b>Parameter number:</b> 1</p> <p><b>Name:</b> String</p> <p><b>Description:</b> Required. The string that is converted to a native type value.</p> <p>The <code>json()</code> function supports XML input too. For example, the parameter value of:</p> <pre>&lt;?xml version="1.0"?&gt; &lt;root&gt; &lt;person id='1'&gt; &lt;name&gt;Alan&lt;/name&gt; &lt;occupation&gt;Engineer&lt;/occupation&gt; &lt;/person&gt; &lt;/root&gt;</pre> <p>is converted to this JSON:</p> <pre>{ "?xml": { "@version": "1.0" }, "root": { "person": [ { "@id": "1", "name": "Alan", "occupation": "Engineer" } ] } }</pre>
float	<p>Convert the parameter argument to a floating-point number. For example, this function returns <code>10.333</code>:</p> <pre>float('10.333')</pre> <p><b>Parameter number:</b> 1</p> <p><b>Name:</b> Value</p> <p><b>Description:</b> Required. The value that is converted to a floating-point number.</p>
bool	<p>Convert the parameter to a Boolean. For example, this function returns <code>false</code>:</p> <pre>bool(0)</pre> <p><b>Parameter number:</b> 1</p> <p><b>Name:</b> Value</p> <p><b>Description:</b> Required. The value that is converted to a boolean.</p>
coalesce	<p>Returns the first non-null object in the arguments passed in. <b>Note:</b> An empty string is not null. For example, if parameters 1 and 2 are not defined, this function returns <code>fallback</code>:</p> <pre>coalesce(parameters('parameter1'), parameters('parameter2'), 'fallback')</pre> <p><b>Parameter number:</b> 1 ... n</p> <p><b>Name:</b> Objectn</p> <p><b>Description:</b> Required. The objects to check for null.</p>
base64	<p>Returns the base64 representation of the input string. For example, this function returns <code>c29tZSBzdHJpbmc=</code>:</p> <pre>base64('some string')</pre> <p><b>Parameter number:</b> 1</p> <p><b>Name:</b> String 1</p> <p><b>Description:</b> Required. The string to encode into base64 representation.</p>
base64ToBinary	<p>Returns a binary representation of a base64 encoded string. For example, this function returns the binary representation of <code>some string</code>:</p> <pre>base64ToBinary('c29tZSBzdHJpbmc=')</pre> <p><b>Parameter number:</b> 1</p> <p><b>Name:</b> String</p> <p><b>Description:</b> Required. The base64 encoded string.</p>

FUNCTION NAME	DESCRIPTION
base64ToString	<p>Returns a string representation of a based64 encoded string. For example, this function returns <code>some string</code>:</p> <pre>base64ToString('c29tZSBzdHJpbmc=')</pre> <p><b>Parameter number:</b> 1</p> <p><b>Name:</b> String</p> <p><b>Description:</b> Required. The base64 encoded string.</p>
Binary	<p>Returns a binary representation of a value. For example, this function returns a binary representation of <code>some string</code>:</p> <pre>binary('some string')</pre> <p><b>Parameter number:</b> 1</p> <p><b>Name:</b> Value</p> <p><b>Description:</b> Required. The value that is converted to binary.</p>
dataUriToBinary	<p>Returns a binary representation of a data URI. For example, this function returns the binary representation of <code>some string</code>:</p> <pre>dataUriToBinary('data:;base64,c29tZSBzdHJpbmc=')</pre> <p><b>Parameter number:</b> 1</p> <p><b>Name:</b> String</p> <p><b>Description:</b> Required. The data URI to convert to binary representation.</p>
dataUriToString	<p>Returns a string representation of a data URI. For example, this function returns <code>some string</code>:</p> <pre>dataUriToString('data:;base64,c29tZSBzdHJpbmc=')</pre> <p><b>Parameter number:</b> 1</p> <p><b>Name:</b> String</p> <p><b>Description:</b> Required. The data URI to convert to String representation.</p>
dataUri	<p>Returns a data URI of a value. For example, this function returns this data URI <code>text/plain;charset=utf8;base64,c29tZSBzdHJpbmc=</code>:</p> <pre>dataUri('some string')</pre> <p><b>Parameter number:</b> 1</p> <p><b>Name:</b> Value</p> <p><b>Description:</b> Required. The value to convert to data URI.</p>
decodeBase64	<p>Returns a string representation of an input based64 string. For example, this function returns <code>some string</code>:</p> <pre>decodeBase64('c29tZSBzdHJpbmc=')</pre> <p><b>Parameter number:</b> 1</p> <p><b>Name:</b> String</p> <p><b>Description:</b> Returns a string representation of an input based64 string.</p>
encodeUriComponent	<p>URL-escapes the string that's passed in. For example, this function returns <code>You+Are%3ACool%2FAwesome</code>:</p> <pre>encodeUriComponent('You Are:Cool/Awesome')</pre> <p><b>Parameter number:</b> 1</p> <p><b>Name:</b> String</p> <p><b>Description:</b> Required. The string to escape URL-unsafe characters from.</p>

FUNCTION NAME	DESCRIPTION
decodeURIComponent	<p>Un-URL-escapes the string that's passed in. For example, this function returns <code>You Are:Cool/Awesome</code>:</p> <pre>encodeURIComponent('You+Are%3ACool%2FAwesome')</pre> <p><b>Parameter number:</b> 1</p> <p><b>Name:</b> String</p> <p><b>Description:</b> Required. The string to decode the URL-unsafe characters from.</p>
decodeDataUri	<p>Returns a binary representation of an input data URI string. For example, this function returns the binary representation of <code>some string</code>:</p> <pre>decodeDataUri('data:;base64,c29tZSBzdHJpbmc=')</pre> <p><b>Parameter number:</b> 1</p> <p><b>Name:</b> String</p> <p><b>Description:</b> Required. The dataURI to decode into a binary representation.</p>
encodeURIComponent	<p>Returns a URI encoded representation of a value. For example, this function returns <code>You+Are%3ACool%2FAwesome</code>:</p> <pre>encodeURIComponent('You Are:Cool/Awesome')</pre> <p><b>Parameter number:</b> 1</p> <p><b>Name:</b> String</p> <p><b>Description:</b> Required. The string to be URI encoded.</p>
encodeURIComponentToBinary	<p>Returns a binary representation of a URI encoded string. For example, this function returns a binary representation of <code>You Are:Cool/Awesome</code>:</p> <pre>encodeURIComponentToBinary('You+Are%3ACool%2FAwesome')</pre> <p><b>Parameter number:</b> 1</p> <p><b>Name:</b> String</p> <p><b>Description:</b> Required. The URI encoded string.</p>
encodeURIComponentToString	<p>Returns a string representation of a URI encoded string. For example, this function returns <code>You Are:Cool/Awesome</code>:</p> <pre>encodeURIComponentToString('You+Are%3ACool%2FAwesome')</pre> <p><b>Parameter number:</b> 1</p> <p><b>Name:</b> String</p> <p><b>Description:</b> Required. The URI encoded string.</p>
xml	<p>Return an XML representation of the value. For example, this function returns XML content represented by <code>'&lt;name&gt;Alan&lt;/name&gt;'</code>:</p> <pre>xml('&lt;name&gt;Alan&lt;/name&gt;')</pre> <p>The <code>xml()</code> function supports JSON object input too. For example, the parameter <code>{ "abc": "xyz" }</code> is converted to XML content:</p> <pre>\&lt;abc&gt;xyz\&lt;/abc&gt;</pre> <p><b>Parameter number:</b> 1</p> <p><b>Name:</b> Value</p> <p><b>Description:</b> Required. The value to convert to XML.</p>

FUNCTION NAME	DESCRIPTION
xpath	<p>Return an array of XML nodes matching the xpath expression of a value that the xpath expression evaluates to.</p> <p><b>Example 1</b></p> <p>Assume the value of parameter <code>p1</code> is a string representation of this XML:</p> <pre>&lt;?xml version="1.0"?&gt; &lt;lab&gt; &lt;robot&gt; &lt;parts&gt;5&lt;/parts&gt; &lt;name&gt;R1&lt;/name&gt; &lt;/robot&gt; &lt;robot&gt; &lt;parts&gt;8&lt;/parts&gt; &lt;name&gt;R2&lt;/name&gt; &lt;/robot&gt; &lt;/lab&gt;</pre> <p>This code: <code>xpath(xml(parameters('p1')), '/lab/robot/name')</code></p> <p>returns</p> <pre>[ &lt;name&gt;R1&lt;/name&gt;, &lt;name&gt;R2&lt;/name&gt; ]</pre> <p>while this code:</p> <pre>xpath(xml(parameters('p1')), ' sum(/lab/robot/parts)')</pre> <p>returns</p> <pre>13</pre> <p><b>Example 2</b></p> <p>Given the following XML content:</p> <pre>&lt;?xml version="1.0"?&gt; &lt;File xmlns="http://foo.com"&gt; &lt;Location&gt;bar&lt;/Location&gt; &lt;/File&gt;</pre> <p>This code:</p> <pre>@xpath(xml(body('Http')), '/*[name()=\"File\"]/*[name()=\"Location\"]')</pre> <p>or this code:</p> <pre>@xpath(xml(body('Http')), '/*[local-name()=\"File\" and namespace-uri()=\"http://foo.com\"]/*[local-name()=\"Location\" and namespace-uri()=\"\"]')</pre> <p>returns</p> <pre>&lt;Location xmlns="http://abc.com"&gt;xyz&lt;/Location&gt;</pre> <p>And this code:</p> <pre>@xpath(xml(body('Http')), 'string(*[name()=\"File\"]/*[name()=\"Location\"]'))</pre> <p>returns</p> <pre>xyz</pre> <p><b>Parameter number:</b> 1</p> <p><b>Name:</b> Xml</p> <p><b>Description:</b> Required. The XML on which to evaluate the XPath expression.</p> <p><b>Parameter number:</b> 2</p> <p><b>Name:</b> XPath</p> <p><b>Description:</b> Required. The XPath expression to evaluate.</p>
array	<p>Convert the parameter to an array. For example, this function returns <code>["abc"]</code>:</p> <pre>array('abc')</pre> <p><b>Parameter number:</b> 1</p> <p><b>Name:</b> Value</p> <p><b>Description:</b> Required. The value that is converted to an array.</p>
createArray	<p>Creates an array from the parameters. For example, this function returns <code>["a", "c"]</code>:</p> <pre>createArray('a', 'c')</pre> <p><b>Parameter number:</b> 1 ... n</p> <p><b>Name:</b> Any n</p> <p><b>Description:</b> Required. The values to combine into an array.</p>

FUNCTION NAME	DESCRIPTION
triggerFormDataValue	<p>Returns a single value matching the key name from form-data or form-encoded trigger output. If there are multiple matches it will error. For example, the following will return <code>bar : triggerFormDataValue('foo')</code></p> <p><b>Parameter number:</b> 1</p> <p><b>Name:</b> Key Name</p> <p><b>Description:</b> Required. The key name of the form data value to return.</p>
triggerFormDataMultiValues	<p>Returns an array of values matching the key name from form-data or form-encoded trigger output. For example, the following will return <code>["bar"] : triggerFormDataValue('foo')</code></p> <p><b>Parameter number:</b> 1</p> <p><b>Name:</b> Key Name</p> <p><b>Description:</b> Required. The key name of the form data values to return.</p>
triggerMultipartBody	<p>Returns the body for a part in a multipart output of the trigger.</p> <p><b>Parameter number:</b> 1</p> <p><b>Name:</b> Index</p> <p><b>Description:</b> Required. The index of the part to retrieve.</p>
formDataValue	<p>Returns a single value matching the key name from form-data or form-encoded action output. If there are multiple matches it will error. For example, the following will return <code>bar : formDataValue('someAction', 'foo')</code></p> <p><b>Parameter number:</b> 1</p> <p><b>Name:</b> Action Name</p> <p><b>Description:</b> Required. The name of the action with a form-data or form-encoded response.</p> <p><b>Parameter number:</b> 2</p> <p><b>Name:</b> Key Name</p> <p><b>Description:</b> Required. The key name of the form data value to return.</p>
formDataMultiValues	<p>Returns an array of values matching the key name from form-data or form-encoded action output. For example, the following will return <code>["bar"] : formDataMultiValues('someAction', 'foo')</code></p> <p><b>Parameter number:</b> 1</p> <p><b>Name:</b> Action Name</p> <p><b>Description:</b> Required. The name of the action with a form-data or form-encoded response.</p> <p><b>Parameter number:</b> 2</p> <p><b>Name:</b> Key Name</p> <p><b>Description:</b> Required. The key name of the form data values to return.</p>
multipartBody	<p>Returns the body for a part in a multipart output of an action.</p> <p><b>Parameter number:</b> 1</p> <p><b>Name:</b> Action Name</p> <p><b>Description:</b> Required. The name of the action with a multipart response.</p> <p><b>Parameter number:</b> 2</p> <p><b>Name:</b> Index</p> <p><b>Description:</b> Required. The index of the part to retrieve.</p>

## Math functions

These functions can be used for either types of numbers: **integers** and **floats**.

FUNCTION NAME	DESCRIPTION
add	<p>Returns the result from adding the two numbers. For example, this function returns <code>20.333</code> :</p> <pre>add(10,10.333)</pre> <p><b>Parameter number:</b> 1</p> <p><b>Name:</b> Summand 1</p> <p><b>Description:</b> Required. The number to add to <b>Summand 2</b>.</p> <p><b>Parameter number:</b> 2</p> <p><b>Name:</b> Summand 2</p> <p><b>Description:</b> Required. The number to add to <b>Summand 1</b>.</p>
sub	<p>Returns the result from subtracting two numbers. For example, this function returns <code>-0.333</code> :</p> <pre>sub(10,10.333)</pre> <p><b>Parameter number:</b> 1</p> <p><b>Name:</b> Minuend</p> <p><b>Description:</b> Required. The number that <b>Subtrahend</b> is removed from.</p> <p><b>Parameter number:</b> 2</p> <p><b>Name:</b> Subtrahend</p> <p><b>Description:</b> Required. The number to remove from the <b>Minuend</b>.</p>
mul	<p>Returns the result from multiplying the two numbers. For example, this function returns <code>103.33</code> :</p> <pre>mul(10,10.333)</pre> <p><b>Parameter number:</b> 1</p> <p><b>Name:</b> Multiplicand 1</p> <p><b>Description:</b> Required. The number to multiply <b>Multiplicand 2</b> with.</p> <p><b>Parameter number:</b> 2</p> <p><b>Name:</b> Multiplicand 2</p> <p><b>Description:</b> Required. The number to multiply <b>Multiplicand 1</b> with.</p>
div	<p>Returns the result from dividing the two numbers. For example, this function returns <code>1.0333</code> :</p> <pre>div(10.333,10)</pre> <p><b>Parameter number:</b> 1</p> <p><b>Name:</b> Dividend</p> <p><b>Description:</b> Required. The number to divide by the <b>Divisor</b>.</p> <p><b>Parameter number:</b> 2</p> <p><b>Name:</b> Divisor</p> <p><b>Description:</b> Required. The number to divide the <b>Dividend</b> by.</p>
mod	<p>Returns the remainder after dividing the two numbers (modulo). For example, this function returns <code>2</code> :</p> <pre>mod(10,4)</pre> <p><b>Parameter number:</b> 1</p> <p><b>Name:</b> Dividend</p> <p><b>Description:</b> Required. The number to divide by the <b>Divisor</b>.</p> <p><b>Parameter number:</b> 2</p> <p><b>Name:</b> Divisor</p> <p><b>Description:</b> Required. The number to divide the <b>Dividend</b> by. After the division, the remainder is taken.</p>

FUNCTION NAME	DESCRIPTION
min	<p>There are two different patterns for calling this function.</p> <p>Here <code>min</code> takes an array, and the function returns <code>0</code>:</p> <pre>min([0,1,2])</pre> <p>Alternatively, this function can take a comma-separated list of values and also returns <code>0</code>:</p> <pre>min(0,1,2)</pre> <p><b>Note:</b> All values must be numbers, so if the parameter is an array, the array has to only have numbers.</p> <p><b>Parameter number:</b> 1</p> <p><b>Name:</b> Collection or Value</p> <p><b>Description:</b> Required. Either an array of values to find the minimum value, or the first value of a set.</p> <p><b>Parameter number:</b> 2 ... <i>n</i></p> <p><b>Name:</b> Value <i>n</i></p> <p><b>Description:</b> Optional. If the first parameter is a Value, then you can pass additional values and the minimum of all passed values is returned.</p>
max	<p>There are two different patterns for calling this function.</p> <p>Here <code>max</code> takes an array, and the function returns <code>2</code>:</p> <pre>max([0,1,2])</pre> <p>Alternatively, this function can take a comma-separated list of values and also returns <code>2</code>:</p> <pre>max(0,1,2)</pre> <p><b>Note:</b> All values must be numbers, so if the parameter is an array, the array has to only have numbers.</p> <p><b>Parameter number:</b> 1</p> <p><b>Name:</b> Collection or Value</p> <p><b>Description:</b> Required. Either an array of values to find the maximum value, or the first value of a set.</p> <p><b>Parameter number:</b> 2 ... <i>n</i></p> <p><b>Name:</b> Value <i>n</i></p> <p><b>Description:</b> Optional. If the first parameter is a Value, then you can pass additional values and the maximum of all passed values is returned.</p>
range	<p>Generates an array of integers starting from a certain number. You define the length of the returned array.</p> <p>For example, this function returns <code>[3,4,5,6]</code>:</p> <pre>range(3,4)</pre> <p><b>Parameter number:</b> 1</p> <p><b>Name:</b> Start index</p> <p><b>Description:</b> Required. The first integer in the array.</p> <p><b>Parameter number:</b> 2</p> <p><b>Name:</b> Count</p> <p><b>Description:</b> Required. This value is the number of integers that is in the array.</p>

FUNCTION NAME	DESCRIPTION
rand	<p>Generates a random integer within the specified range (inclusive on both ends). For example, this function might return 42 :</p> <pre>rand(-1000,1000)</pre> <p><b>Parameter number:</b> 1</p> <p><b>Name:</b> Minimum</p> <p><b>Description:</b> Required. The lowest integer that can be returned.</p> <p><b>Parameter number:</b> 2</p> <p><b>Name:</b> Maximum</p> <p><b>Description:</b> Required. This value is the highest integer that could be returned.</p>

## Date functions

FUNCTION NAME	DESCRIPTION
utcnow	<p>Returns the current timestamp as a string, for example: 2017-03-15T13:27:36Z :</p> <pre>utcnow()</pre> <p><b>Parameter number:</b> 1</p> <p><b>Name:</b> Format</p> <p><b>Description:</b> Optional. Either a <a href="#">single format specifier character</a> or a <a href="#">custom format pattern</a> that indicates how to format the value of this timestamp. If format is not provided, the ISO 8601 format ("o") is used.</p>
addseconds	<p>Adds an integer number of seconds to a string timestamp passed in. The number of seconds can be positive or negative. By default, the result is a string in ISO 8601 format ("o"), unless a format specifier is provided. For example: 2015-03-15T13:27:00Z :</p> <pre>addseconds('2015-03-15T13:27:36Z', -36)</pre> <p><b>Parameter number:</b> 1</p> <p><b>Name:</b> Timestamp</p> <p><b>Description:</b> Required. A string that contains the time.</p> <p><b>Parameter number:</b> 2</p> <p><b>Name:</b> Seconds</p> <p><b>Description:</b> Required. The number of seconds to add. Can be negative to subtract seconds.</p> <p><b>Parameter number:</b> 3</p> <p><b>Name:</b> Format</p> <p><b>Description:</b> Optional. Either a <a href="#">single format specifier character</a> or a <a href="#">custom format pattern</a> that indicates how to format the value of this timestamp. If format is not provided, the ISO 8601 format ("o") is used.</p>

FUNCTION NAME	DESCRIPTION
addminutes	<p>Adds an integer number of minutes to a string timestamp passed in. The number of minutes can be positive or negative. By default, the result is a string in ISO 8601 format ("o"), unless a format specifier is provided. For example:</p> <pre>2015-03-15T14:00:36Z :</pre> <pre>addminutes('2015-03-15T13:27:36Z', 33)</pre> <p><b>Parameter number:</b> 1</p> <p><b>Name:</b> Timestamp</p> <p><b>Description:</b> Required. A string that contains the time.</p> <p><b>Parameter number:</b> 2</p> <p><b>Name:</b> Minutes</p> <p><b>Description:</b> Required. The number of minutes to add. Can be negative to subtract minutes.</p> <p><b>Parameter number:</b> 3</p> <p><b>Name:</b> Format</p> <p><b>Description:</b> Optional. Either a <a href="#">single format specifier character</a> or a <a href="#">custom format pattern</a> that indicates how to format the value of this timestamp. If format is not provided, the ISO 8601 format ("o") is used.</p>
addhours	<p>Adds an integer number of hours to a string timestamp passed in. The number of hours can be positive or negative. By default, the result is a string in ISO 8601 format ("o"), unless a format specifier is provided. For example:</p> <pre>2015-03-16T01:27:36Z :</pre> <pre>addhours('2015-03-15T13:27:36Z', 12)</pre> <p><b>Parameter number:</b> 1</p> <p><b>Name:</b> Timestamp</p> <p><b>Description:</b> Required. A string that contains the time.</p> <p><b>Parameter number:</b> 2</p> <p><b>Name:</b> Hours</p> <p><b>Description:</b> Required. The number of hours to add. Can be negative to subtract hours.</p> <p><b>Parameter number:</b> 3</p> <p><b>Name:</b> Format</p> <p><b>Description:</b> Optional. Either a <a href="#">single format specifier character</a> or a <a href="#">custom format pattern</a> that indicates how to format the value of this timestamp. If format is not provided, the ISO 8601 format ("o") is used.</p>
adddays	<p>Adds an integer number of days to a string timestamp passed in. The number of days can be positive or negative. By default, the result is a string in ISO 8601 format ("o"), unless a format specifier is provided. For example:</p> <pre>2015-02-23T13:27:36Z :</pre> <pre>addseconds('2015-03-15T13:27:36Z', -20)</pre> <p><b>Parameter number:</b> 1</p> <p><b>Name:</b> Timestamp</p> <p><b>Description:</b> Required. A string that contains the time.</p> <p><b>Parameter number:</b> 2</p> <p><b>Name:</b> Days</p> <p><b>Description:</b> Required. The number of days to add. Can be negative to subtract days.</p> <p><b>Parameter number:</b> 3</p> <p><b>Name:</b> Format</p> <p><b>Description:</b> Optional. Either a <a href="#">single format specifier character</a> or a <a href="#">custom format pattern</a> that indicates how to format the value of this timestamp. If format is not provided, the ISO 8601 format ("o") is used.</p>

FUNCTION NAME	DESCRIPTION
formatDateTime	<p>Returns a string in date format. By default, the result is a string in ISO 8601 format ("o"), unless a format specifier is provided. For example:</p> <pre>2015-02-23T13:27:36Z :</pre> <pre>formatDateTime('2015-03-15T13:27:36Z', 'o')</pre> <p><b>Parameter number:</b> 1</p> <p><b>Name:</b> Date</p> <p><b>Description:</b> Required. A string that contains the date.</p> <p><b>Parameter number:</b> 2</p> <p><b>Name:</b> Format</p> <p><b>Description:</b> Optional. Either a <a href="#">single format specifier character</a> or a <a href="#">custom format pattern</a> that indicates how to format the value of this timestamp. If format is not provided, the ISO 8601 format ("o") is used.</p>
startOfDay	<p>Returns the start of the day to a string timestamp passed in. For example</p> <pre>2017-03-15T00:00:00Z :</pre> <pre>startOfDay('2017-03-15T13:27:36Z')</pre> <p><b>Parameter number:</b> 1</p> <p><b>Name:</b> Timestamp</p> <p><b>Description:</b> Required. This is a string that contains the time.</p> <p><b>Parameter number:</b> 2</p> <p><b>Name:</b> Format</p> <p><b>Description:</b> Optional. Either a <a href="#">single format specifier character</a> or a <a href="#">custom format pattern</a> that indicates how to format the value of this timestamp. If format is not provided, the ISO 8601 format ("o") is used.</p>
startOfMonth	<p>Returns the start of the month to a string timestamp passed in. For example</p> <pre>2017-03-01T00:00:00Z :</pre> <pre>startOfMonth('2017-03-15T13:27:36Z')</pre> <p><b>Parameter number:</b> 1</p> <p><b>Name:</b> Timestamp</p> <p><b>Description:</b> Required. This is a string that contains the time.</p> <p><b>Parameter number:</b> 2</p> <p><b>Name:</b> Format</p> <p><b>Description:</b> Optional. Either a <a href="#">single format specifier character</a> or a <a href="#">custom format pattern</a> that indicates how to format the value of this timestamp. If format is not provided, the ISO 8601 format ("o") is used.</p>
startOfHour	<p>Returns the start of the hour to a string timestamp passed in. For example</p> <pre>2017-03-15T13:00:00Z :</pre> <pre>startOfHour('2017-03-15T13:27:36Z')</pre> <p><b>Parameter number:</b> 1</p> <p><b>Name:</b> Timestamp</p> <p><b>Description:</b> Required. This is a string that contains the time.</p> <p><b>Parameter number:</b> 2</p> <p><b>Name:</b> Format</p> <p><b>Description:</b> Optional. Either a <a href="#">single format specifier character</a> or a <a href="#">custom format pattern</a> that indicates how to format the value of this timestamp. If format is not provided, the ISO 8601 format ("o") is used.</p>

FUNCTION NAME	DESCRIPTION
dayOfWeek	<p>Returns the day of week component of a string timestamp. Sunday is 0, Monday is 1, and so on. For example <code>3</code> :</p> <pre>dayOfWeek('2017-03-15T13:27:36Z')</pre> <p><b>Parameter number:</b> 1</p> <p><b>Name:</b> Timestamp</p> <p><b>Description:</b> Required. This is a string that contains the time.</p>
dayOfMonth	<p>Returns the day of month component of a string timestamp. For example <code>15</code> :</p> <pre>dayOfMonth('2017-03-15T13:27:36Z')</pre> <p><b>Parameter number:</b> 1</p> <p><b>Name:</b> Timestamp</p> <p><b>Description:</b> Required. This is a string that contains the time.</p>
dayOfYear	<p>Returns the day of year component of a string timestamp. For example <code>74</code> :</p> <pre>dayOfYear('2017-03-15T13:27:36Z')</pre> <p><b>Parameter number:</b> 1</p> <p><b>Name:</b> Timestamp</p> <p><b>Description:</b> Required. This is a string that contains the time.</p>
ticks	<p>Returns the ticks property of a string timestamp. For example <code>1489603019</code> :</p> <pre>ticks('2017-03-15T18:36:59Z')</pre> <p><b>Parameter number:</b> 1</p> <p><b>Name:</b> Timestamp</p> <p><b>Description:</b> Required. This is a string that contains the time.</p>

## Workflow functions

These functions help you get information about the workflow itself at run time.

FUNCTION NAME	DESCRIPTION
listCallbackUrl	<p>Returns a string to call to invoke the trigger or action.</p> <p><b>Note:</b> This function can only be used in an <b>httpWebhook</b> and <b>apiConnectionWebhook</b>, not in a <b>manual</b>, <b>recurrence</b>, <b>http</b>, or <b>apiConnection</b>.</p> <p>For example, the <code>listCallbackUrl()</code> function returns:</p> <pre>https://prod-01.westus.logic.azure.com:443/workflows/1235...ABCD/triggers/manual/run?api-version=2015-08-01-preview&amp;sp=%2Ftriggers%2Fmanual%2Frun&amp;sv=1.0&amp;sig=xxx...xxx</pre>
workflow	<p>This function provides you all the details for the workflow itself at the runtime. You can get everything that's available in the Management API, such as name, location, and resourceId. See the <a href="#">Rest API</a> for details on those properties. For example, this function returns a location like <code>westus</code> :</p> <pre>workflow().location</pre> <p><b>Note:</b> <code>@workflow</code> is currently supported within triggers.</p>

## Next steps

[Workflow actions and triggers](#)

# Workflow actions and triggers for Azure Logic Apps

3/21/2017 • 22 min to read • [Edit Online](#)

Logic apps consist of triggers and actions. There are six types of triggers. Each type has different interface and different behavior. You can also learn about other details by looking at the details of the [Workflow Definition Language](#).

Read on to learn more about triggers and actions and how you might use them to build logic apps to improve your business processes and workflows.

## Triggers

A trigger specifies the calls that can initiate a run of your logic app workflow. Here are the two different ways to initiate a run of your workflow:

- A polling trigger
- A push trigger - by calling the [Workflow Service REST API](#)

All triggers contain these top-level elements:

```
"<name-of-the-trigger>" : {  
    "type": "<type-of-trigger>",  
    "inputs": { <settings-for-the-call> },  
    "recurrence": {  
        "frequency": "Second|Minute|Hour|Week|Month|Year",  
        "interval": "<recurrence interval in units of frequency>"  
    },  
    "conditions": [ <array-of-required-conditions > ],  
    "splitOn" : "<property to create runs for>",  
    "operationOptions": "<operation options on the trigger>"  
}
```

## Trigger types and their inputs

You can use these types of triggers:

- **Request** - Makes the logic app an endpoint for you to call
- **Recurrence** - Fires based on a defined schedule
- **HTTP** - Polls an HTTP web endpoint. The HTTP endpoint must conform to a specific triggering contract - either by using a 202-async pattern, or by returning an array
- **ApiConnection** - Polls like the HTTP trigger, however, it takes advantage of the [Microsoft-managed APIs](#)
- **HTTPWebhook** - Opens an endpoint, similar to the Manual trigger, however, it also calls out to a specified URL to register and unregister
- **ApiConnectionWebhook** - Operates like the HTTPWebhook trigger by taking advantage of the Microsoft-managed APIs

Each trigger type has a different set of **inputs** that defines its behavior.

## Request trigger

This trigger serves as an endpoint that you call via an HTTP Request to invoke your logic app. A request trigger looks like this example:

```

"<name-of-the-trigger>" : {
    "type" : "request",
    "kind": "http",
    "inputs" : {
        "schema" : {
            "properties" : {
                "myInputProperty1" : { "type" : "string" },
                "myInputProperty2" : { "type" : "number" }
            },
            "required" : [ "myInputProperty1" ],
            "type" : "object"
        }
    }
}

```

There is also an optional property called **schema**:

ELEMENT NAME	REQUIRED	DESCRIPTION
schema	No	A JSON schema that validates the incoming request. Useful for helping subsequent workflow steps know which properties to reference.

To invoke this endpoint, you need to call the *listCallbackUrl* API. See [Workflow Service REST API](#).

## Recurrence trigger

A Recurrence trigger is one that runs based on a defined schedule. Such a trigger might look like this example:

```

"dailyReport" : {
    "type": "recurrence",
    "recurrence": {
        "frequency": "Day",
        "interval": "1"
    }
}

```

As you can see, it is a simple way to run a workflow.

ELEMENT NAME	REQUIRED	DESCRIPTION
frequency	Yes	How often the trigger executes. Use only one of these possible values: second, minute, hour, day, week, month, or year
interval	Yes	Interval of the given frequency for the recurrence
startTime	No	If a startTime is provided without a UTC offset, this timeZone is used.
timeZone	no	If a startTime is provided without a UTC offset, this timeZone is used.

You can also schedule a trigger to start executing at some point in the future. For example, if you want to start a weekly report every Monday you can schedule the logic app to start every Monday by creating the following

trigger:

```
"dailyReport" : {  
    "type": "recurrence",  
    "recurrence": {  
        "frequency": "Week",  
        "interval": "1",  
        "startTime" : "2015-06-22T00:00:00Z"  
    }  
}
```

## HTTP trigger

HTTP triggers poll a specified endpoint and check the response to determine whether the workflow should be executed. The inputs object takes the set of parameters required to construct an HTTP call:

ELEMENT NAME	REQUIRED	DESCRIPTION	TYPE
method	yes	Can be one of the following HTTP methods: GET, POST, PUT, DELETE, PATCH, or HEAD	String
uri	yes	The http or https endpoint that is called. Maximum of 2 kilobytes.	String
queries	No	An object representing the query parameters to add to the URL. For example, <div style="border: 1px solid black; padding: 5px;"><code>"queries" : { "api-version": "2015-02-01" }</code></div> adds <div style="border: 1px solid black; padding: 5px;"><code>?api-version=2015-02-01</code></div> to the URL.	Object
headers	No	An object representing each of the headers that is sent to the request. For example, to set the language and type on a request: <div style="border: 1px solid black; padding: 5px;"><code>"headers" : { "Accept-Language": "en-us", "Content-Type": "application/json" }</code></div>	Object
body	No	An object representing the payload that is sent to the endpoint.	Object
retryPolicy	No	An object that lets you customize the retry behavior for 4xx or 5xx errors.	Object

Element Name	Required	Description	Type
authentication	No	<p>Represents the method that the request should be authenticated. For details on this object, see <a href="#">Scheduler Outbound Authentication</a>.</p> <p>Beyond scheduler, there is one more supported property: <code>authority</code>. By default, this value is <code>https://login.windows.net</code> when not specified, but you can use a different audience like <code>https://login.windows\-ppe.net</code></p>	Object

The HTTP trigger requires the HTTP API to conform with a specific pattern to work well with your logic app. It requires the following fields:

Response	Description
Status code	Status code 200 (OK) to cause a run. Any other status code doesn't cause a run.
Retry-after header	Number of seconds until the logic app polls the endpoint again.
Location header	The URL to call on the next polling interval. If not specified, the original URL is used.

Here are some examples of different behaviors for different types of requests:

Response Code	Retry-After	Behavior
200	(none)	Not a valid trigger, Retry-After is required, or else the engine never polls for the next request.
202	60	Do not trigger the workflow. The next attempt happens in one minute.
200	10	Run the workflow, and check again for more content in 10 seconds.
400	(none)	Bad request, do not run the workflow. If there is no <b>Retry Policy</b> defined, then the default policy is used. After the number of retries has been reached, the trigger is no longer valid.
500	(none)	Server error, do not run the workflow. If there is no <b>Retry Policy</b> defined, then the default policy is used. After the number of retries has been reached, the trigger is no longer valid.

The outputs of an HTTP trigger look like this example:

ELEMENT NAME	DESCRIPTION	TYPE
headers	The headers of the http response.	Object
body	The body of the http response.	Object

## API Connection trigger

The API connection trigger is similar to the HTTP trigger in its basic functionality. However, the parameters for identifying the action are different. Here is an example:

```
"dailyReport" : {
    "type": "ApiConnection",
    "inputs": {
        "host": {
            "api": {
                "runtimeUrl": "https://myarticles.example.com/"
            },
        }
    "connection": {
        "name": "@parameters('$connections')['myconnection'].name"
    }
},
"method": "POST",
"body": {
    "category": "awesomest"
}
}
```

ELEMENT NAME	REQUIRED	TYPE	DESCRIPTION
host	Yes		The ApiApp hosted gateway and id.
method	Yes	String	Can be one of the following HTTP methods: <b>GET</b> , <b>POST</b> , <b>PUT</b> , <b>DELETE</b> , <b>PATCH</b> , or <b>HEAD</b>
queries	No	Object	Represents the query parameters to be added to the URL. For example, <div style="border: 1px solid black; padding: 5px; display: inline-block;"> <pre>"queries" : { "api-version": "2015-02-01" }</pre> </div> adds <div style="border: 1px solid black; padding: 5px; display: inline-block;"> <pre>?api-version=2015-02-01</pre> </div> to the URL.

ELEMENT NAME	REQUIRED	TYPE	DESCRIPTION
headers	No	Object	Represents each of the headers that is sent to the request. For example, to set the language and type on a request: <pre>"headers" : { "Accept-Language": "en-us", "Content-Type": "application/json" }</pre>
body	No	Object	Represents the payload that is sent to the endpoint.
retryPolicy	No	Object	Allows you to customize the retry behavior for 4xx or 5xx errors.
authentication	No	Object	Represents the method that the request should be authenticated. For details on this object, see <a href="#">Scheduler Outbound Authentication</a>

The properties for host are:

ELEMENT NAME	REQUIRED	DESCRIPTION
api runtimeUrl	Yes	The endpoint of the managed API.
connection name		Must be a reference to a parameter called <code>\$connection</code> and is the name of the managed API connection that the workflow uses.

The outputs of an API connection trigger are:

ELEMENT NAME	TYPE	DESCRIPTION
headers	Object	The headers of the http response.
body	Object	The body of the http response.

## HTTPWebhook trigger

The HTTPWebhook trigger opens an endpoint, similar to the manual trigger, but the HTTPWebhook trigger also calls out to a specified URL to register and unregister. Here's an example of what an HTTPWebhook trigger might look like:

```

"myappspottrigger": {
    "type": "httpWebhook",
    "inputs": {
        "subscribe": {
            "method": "POST",
            "uri": "https://pubsubhubbub.appspot.com/subscribe",
            "headers": { },
            "body": {
                "hub.callback": "@{listCallbackUrl()}",
                "hub.mode": "subscribe",
                "hub.topic": "https://pubsubhubbub.appspot.com/articleCategories/technology"
            },
            "authentication": { },
            "retryPolicy": { }
        },
        "unsubscribe": {
            "url": "https://pubsubhubbub.appspot.com/subscribe",
            "body": {
                "hub.callback": "@{workflow().endpoint}@{listCallbackUrl()}",
                "hub.mode": "unsubscribe",
                "hub.topic": "https://pubsubhubbub.appspot.com/articleCategories/technology"
            },
            "method": "POST",
            "authentication": { }
        }
    },
    "conditions": [ ]
}

```

Many of these sections are optional, and the behavior of the Webhook depends on which sections are provided or omitted.

The properties of a Webhook are as follows:

ELEMENT NAME	REQUIRED	DESCRIPTION
subscribe	No	The outgoing request that is called when the trigger is created and performs the initial registration.
unsubscribe	No	The outgoing request when the trigger is deleted.

- **Subscribe** is the outgoing call that's made to start listening to events. This call starts with the same set of parameters that the normal HTTP actions do. This outgoing call is made any time the workflow changes in any way, for example, whenever the credentials are rolled, or the trigger's input parameters change.

To support this call, there is a new function: `@listCallbackUrl()`. This function returns a unique URL for this specific trigger in this workflow. It represents the unique identifier for the endpoints that use the Service REST.

- **Unsubscribe** is called when an operation renders this trigger invalid, including:

- Deleting or disabling the trigger
- Deleting or disabling the workflow
- Deleting or disabling the subscription

The logic app automatically calls the unsubscribe action. The parameters to this function are the same as the HTTP trigger.

The outputs of the HTTPWebhook trigger are the contents of the incoming request:

ELEMENT NAME	TYPE	DESCRIPTION
headers	Object	The headers of the http request.
body	Object	The body of the http request.

Limits on a webhook action can be specified in the same manner as [HTTP Asynchronous Limits](#).

## Conditions

For any trigger, you can use one or more conditions to determine whether the workflow should run or not. For example:

```
"dailyReport" : {  
    "type": "recurrence",  
    "conditions": [ {  
        "expression": "@parameters('sendReports')"  
    } ],  
    "recurrence": {  
        "frequency": "Day",  
        "interval": "1"  
    }  
}
```

In this case, the report only triggers while the workflow's `sendReports` parameter is set to true. Finally, conditions may reference the status code of the trigger. For example, you could kick off a workflow only when your website returns a status code 500, as follows:

```
"conditions": [  
    {  
        "expression": "@equals(triggers().code, 'InternalServerError')"  
    }  
]
```

### NOTE

When any expression references the status code of the trigger (in any way), the default behavior (trigger only on 200 (OK)) is replaced. For example, if you want to trigger on both status code 200 and status code 201, you have to include:

```
@or>equals(triggers().code, 200), equals(triggers().code,201))
```

 as your condition.

## Start multiple runs for a request

To kick off multiple runs for a single request, `splitOn` is useful, for example, when you want to poll an endpoint that can have multiple new items between polling intervals.

With `splitOn`, you specify the property inside the response payload that contains the array of items, each of which you want to use to start a run of the trigger. For example, imagine you have an API that returns the following response:

```
{
    "Status" : "success",
    "Rows" : [
        {
            "id" : 938109380,
            "name" : "mycoolrow"
        },
        {
            "id" : 938109381,
            "name" : "another row"
        }
    ]
}
```

Your logic app only needs the Rows content, so you can construct your trigger like this example:

```
"mysplitter" : {
    "type" : "http",
    "recurrence": {
        "frequency": "Minute",
        "interval": "1"
    },
    "inputs" : {
        "uri" : "https://mydomain.com/myAPI",
        "method" : "GET"
    },
    "splitOn" : "@triggerBody()?.Rows"
}
```

Then, in the workflow definition, `@triggerBody().name` returns `mycoolrow` for the first run, and `another row` for the second run. The trigger outputs look like this example:

```
{
    "body" : {
        "id" : 938109381,
        "name" : "another row"
    }
}
```

So if you use `SplitOn`, you can't get the properties that are outside the array, in this case, the `Status` field.

#### **NOTE**

In this example, we use the `?` operator to be able to avoid a failure if the `Rows` property is not present.

## Single run instance

You can configure triggers that have a recurrence property to only fire if all active runs have completed. If a scheduled recurrence occurs while there is an in-progress run, the trigger skips and waits until the next scheduled recurrence interval to check again.

You can configure this setting through the operation options:

```

"triggers": {
    "mytrigger": {
        "type": "http",
        "inputs": { ... },
        "recurrence": { ... },
        "operationOptions": "singleInstance"
    }
}

```

## Types and inputs

There are many types of actions, each with unique behavior. Collection actions may contain many other actions within itself.

### Standard actions

- **HTTP** This action calls an HTTP web endpoint.
- **ApiConnection** - This action behaves like the HTTP action, but uses the Microsoft-managed APIs.
- **ApiConnectionWebhook** - Like HTTPWebhook, but uses the Microsoft-managed APIs.
- **Response** - This action defines a response for an incoming call.
- **Wait** - This simple action waits a fixed amount of time or until a specific time.
- **Workflow** - This action represents a nested workflow.

### Collection actions

- **Scope** - This action is a logical grouping of other actions.
- **Condition** - This action evaluates an expression and executes the corresponding result branch.
- **ForEach** - This looping action iterates through an array and performs inner actions for each item.
- **Until** - This looping action executes inner actions until a condition results to true.

Each type of action has a different set of **inputs** that define an action's behavior.

## HTTP action

HTTP actions call a specified endpoint and check the response to determine whether the workflow should run. The **inputs** object takes the set of parameters required to construct the HTTP call:

ELEMENT NAME	REQUIRED	TYPE	DESCRIPTION
method	Yes	String	Can be one of the following HTTP methods: <b>GET</b> , <b>POST</b> , <b>PUT</b> , <b>DELETE</b> , <b>PATCH</b> , or <b>HEAD</b>
uri	Yes	String	The http or https endpoint that is called. Maximum length is 2 kilobytes.

Element Name	Required	Type	Description
queries	No	Object	<p>Represents the query parameters to add to the URL. For example,</p> <pre>"queries" : { "api-version": "2015-02-01" }</pre> <p>adds <code>?api-version=2015-02-01</code> to the URL.</p>
headers	No	Object	<p>Represents each of the headers that is sent to the request. For example, to set the language and type on a request:</p> <pre>"headers" : { "Accept-Language": "en-us", "Content-Type": "application/json" }</pre>
body	No	Object	Represents the payload that is sent to the endpoint.
retryPolicy	No	Object	Lets you customize the retry behavior for 4xx or 5xx errors.
operationsOptions	No	String	Defines the set of special behaviors to override.
authentication	No	Object	<p>Represents the method that the request should be authenticated. For details on this object, see <a href="#">Scheduler Outbound Authentication</a>. Beyond scheduler, there is one more supported property: <code>authority</code>. By default, this is <code>https://login.windows.net</code> when not specified, but you can use a different audience like <code>https://login.windows-ppe.net</code></p>

HTTP actions (and API Connection) actions support retry policies. A retry policy applies to intermittent failures, characterized as HTTP status codes 408, 429, and 5xx, in addition to any connectivity exceptions. This policy is described using the `retryPolicy` object defined as shown here:

```
"retryPolicy" : {
    "type": "<type-of-retry-policy>",
    "interval": <retry-interval>,
    "count": <number-of-retry-attempts>
}
```

The retry interval is specified in the ISO 8601 format. This interval has a default and minimum value of 20 seconds, while the maximum value is one hour. The default and maximum retry count is four hours. If the retry policy

definition is not specified, a `fixed` strategy is used with default retry count and interval values. To disable the retry policy, set its type to `None`.

For example, the following action retries fetching the latest news two times, if there are intermittent failures, for a total of three executions, with a 30-second delay between each attempt:

```
"latestNews" : {
    "type": "http",
    "inputs": {
        "method": "GET",
        "uri": "uri": "https://mynews.example.com/latest",
        "retryPolicy" : {
            "type": "fixed",
            "interval": "PT30S",
            "count": 2
        }
    }
}
```

## Asynchronous patterns

By default, all HTTP-based actions support the standard asynchronous operation pattern. So if the remote server indicates that the request is accepted for processing with a 202 (Accepted) response, the Logic Apps engine keeps polling the URL specified in the response's location header until reaching a terminal state (a non-202 response).

To disable the asynchronous behavior previously described, set a `DisableAsyncPattern` option in the action inputs. In this case, the output of the action is based on the initial 202 response from the server.

```
"invokeLongRunningOperation" : {
    "type": "http",
    "inputs": {
        "method": "POST",
        "uri": "https://host.example.com/resources"
    },
    "operationOptions": "DisableAsyncPattern"
}
```

## Asynchronous Limits

An asynchronous pattern can be limited in its duration to a specific time interval. If the time interval elapses without reaching a terminal state, the status of the action will be marked `Cancelled` with a code of `ActionTimedOut`. The limit timeout is specified in ISO 8601 format. Limits can be specified with the following syntax:

```
"<action-name>": {
    "type": "workflow|webhook|http|apiconnectionwebhook|apiconnection",
    "inputs": { },
    "limit": {
        "timeout": "PT10S"
    }
}
```

## API Connection

API Connection is an action that references a Microsoft-managed connector. This action requires a reference to a valid connection, and information on the API and parameters required.

ELEMENT NAME	REQUIRED	TYPE	DESCRIPTION
host	Yes	Object	Represents the connector information such as the runtimeUrl and reference to the connection object
method	Yes	String	Can be one of the following HTTP methods: <b>GET</b> , <b>POST</b> , <b>PUT</b> , <b>DELETE</b> , <b>PATCH</b> , or <b>HEAD</b>
path	Yes	String	The path of the API operation.
queries	No	Object	<p>Represents the query parameters to add to the URL. For example,</p> <pre>"queries" : { "api-version": "2015-02-01" }</pre> <p>adds</p> <pre>?api-version=2015-02-01</pre> <p>to the URL.</p>
headers	No	Object	Represents each of the headers that is sent to the request. For example, to set the language and type on a request:
			<pre>"headers" : { "Accept-Language": "en-us", "Content-Type": "application/json" }</pre>
body	No	Object	Represents the payload that is sent to the endpoint.
retryPolicy	No	Object	Lets you customize the retry behavior for 4xx or 5xx errors.
operationsOptions	No	String	Defines the set of special behaviors to override.

```

"Send_Email": {
    "type": "apiconnection",
    "inputs": {
        "host": {
            "api": {
                "runtimeUrl": "https://logic-apis-df.azure-apim.net/apim/office365"
            },
            "connection": {
                "name": "@parameters('$connections')['office365']['connectionId']"
            }
        },
        "method": "post",
        "body": {
            "Subject": "New Tweet from @{triggerBody()['TweetedBy']}",
            "Body": "@{triggerBody()['TweetText']}",
            "To": "me@example.com"
        },
        "path": "/Mail"
    },
    "runAfter": {}
}

```

## API Connection webhook action

```

"Send_approval_email": {
    "type": "apiconnectionwebhook",
    "inputs": {
        "host": {
            "api": {
                "runtimeUrl": "https://logic-apis-df.azure-apim.net/apim/office365"
            },
            "connection": {
                "name": "@parameters('$connections')['office365']['connectionId']"
            }
        },
        "body": {
            "Message": {
                "Subject": "Approval Request",
                "Options": "Approve, Reject",
                "Importance": "Normal",
                "To": "me@email.com"
            }
        },
        "path": "/approvalmail",
        "authentication": "@parameters('$authentication')"
    },
    "runAfter": {}
}

```

Limits on a webhook action can be specified in the same manner as [HTTP Asynchronous Limits](#).

## Response action

This action type contains the entire response payload from an HTTP request and includes a statusCode, body, and headers:

```

"myresponse" : {
    "type" : "response",
    "inputs" : {
        "statusCode" : 200,
        "body" : {
            "contentFieldOne" : "value100",
            "anotherField" : 10.001
        },
        "headers" : {
            "x-ms-date" : "@utcnow()", 
            "Content-type" : "application/json"
        }
    },
    "runAfter": {}
}

```

The response action has special restrictions that don't apply to other actions. Specifically:

- Response actions cannot be parallel in a definition because a deterministic response to the incoming request is required.
- If a response action is reached after the incoming request has received a response, the action is considered failed (conflict), and as a result, the run is **Failed**.
- A workflow with Response actions cannot have **splitOn** in its trigger because one call causes many runs. As a result, this should be validated when the flow is PUT and cause a Bad Request.

## Wait action

The **wait** action suspends workflow execution for the specified interval. For example, to wait 15 minutes, you can use this snippet:

```

"waitForFifteenMinutes" : {
    "type": "wait",
    "inputs": {
        "interval": {
            "unit" : "minute",
            "count" : 15
        }
    }
}

```

Alternatively, to wait until a specific moment in time, you can use this example:

```

"waitForOctober" : {
    "type": "wait",
    "inputs": {
        "until": {
            "timestamp" : "2016-10-01T00:00:00Z"
        }
    }
}

```

### NOTE

The wait duration can be either specified using the **interval** object or the **until** object, but not both.

NAME	REQUIRED	TYPE	DESCRIPTION
interval	No	Object	The wait duration based on amount of time.
interval unit	Yes	String	One of these intervals: second, minute, hour, day, week, month, year.
interval count	Yes	String	Duration based on the given internal unit.
until	No	Object	The wait duration based on a point in time.
until timestamp	Yes	String	String The point in time in UTC when the wait expires.

## Query action

The `query` action lets you filter an array based on a condition. For example, to select numbers greater than 2, you can use:

```
"FilterNumbers" : {
  "type": "query",
  "inputs": {
    "from": [ 1, 3, 0, 5, 4, 2 ],
    "where": "@greater(item(), 2)"
  }
}
```

The output from the `query` action is an array that has elements from the input array that satisfy the condition.

### NOTE

If no values satisfy the `where` condition, the result is an empty array.

NAME	REQUIRED	TYPE	DESCRIPTION
from	Yes	Array	The source array.
where	Yes	String	The condition to apply to each element of the source array.

## Terminate action

The Terminate action stops execution of the workflow run, aborting any in-flight actions, and skipping any remaining actions. For example, to terminate a run with status **Failed**, you can use the following snippet:

```

"HandleUnexpectedResponse" : {
    "type": "terminate",
    "inputs": {
        "runStatus" : "failed",
        "runError": {
            "code": "UnexpectedResponse",
            "message": "Received an unexpected response."
        }
    }
}

```

#### NOTE

Actions already completed are not affected by the terminate action.

NAME	REQUIRED	TYPE	DESCRIPTION
runStatus	Yes	String	The target run status. Either <b>Failed</b> or <b>Cancelled</b> .
runError	No	Object	The error details. Only supported when <b>runStatus</b> is set to <b>Failed</b> .
runError code	No	String	The run error code.
runError message	No	String	The run error message.

## Compose action

The Compose action lets you construct an arbitrary object. The output of the compose action is the result of evaluating its inputs. For example, you can use the compose action to merge outputs of multiple actions:

```

"composeUserRecord" : {
    "type": "compose",
    "inputs": {
        "firstName": "@actions('getUser').firstName",
        "alias": "@actions('getUser').alias",
        "thumbnailLink": "@actions('lookupThumbnail').url"
    }
}

```

#### NOTE

The **Compose** action can be used to construct any output, including objects, arrays, and any other type natively supported by logic apps like XML and binary.

## Workflow action

NAME	REQUIRED	TYPE	DESCRIPTION

NAME	REQUIRED	TYPE	DESCRIPTION
host id	Yes	String	The resource ID of the workflow that you want to call.
host triggerName	Yes	String	The name of the trigger that you want to invoke.
queries	No	Object	Represents the query parameters to add to the URL. For example, <div style="border: 1px solid black; padding: 5px; margin-left: 20px;">"queries" : { "api-version": "2015-02-01" }</div> adds <div style="border: 1px solid black; padding: 5px; margin-left: 20px;">?api-version=2015-02-01</div> to the URL.
headers	No	Object	Represents each of the headers that is sent to the request. For example, to set the language and type on a request: <div style="border: 1px solid black; padding: 5px; margin-left: 20px;">"headers" : { "Accept-Language": "en-us", "Content-Type": "application/json" }</div>
body	No	Object	Represents the payload sent to the endpoint.

```
"mynestedwf" : {
    "type" : "workflow",
    "inputs" : {
        "host" : {
            "id" : "/subscriptions/xxxxxxxxzzz/resourceGroups/rg001/providers/Microsoft.Logic/mywf001",
            "triggerName" : "mytrigger001"
        },
        "queries" : {
            "extrafield" : "specialValue"
        },
        "headers" : {
            "x-ms-date" : "@utcnow()", "Content-type" : "application/json"
        },
        "body" : {
            "contentFieldOne" : "value100",
            "anotherField" : 10.001
        }
    },
    "runAfter": {}
}
```

An access check is made on the workflow (more specifically, the trigger), meaning you need access to the workflow.

The outputs from the `workflow` action are based on what you defined in the `response` action in the child workflow. If you have not defined any `response` action, then the outputs are empty.

## Collection actions (scopes and loops)

Some action types can contain actions within themselves. Reference actions within a collection can be referenced directly outside of the collection. If you defined `http` in a scope, `@body('http')` is still valid anywhere in a workflow. Actions within a collection can `runAfter` only other actions within the same collection.

### Scope action

The `scope` action lets you logically group actions in a workflow.

NAME	REQUIRED	TYPE	DESCRIPTION
actions	Yes	Object	Inner actions to execute within the scope

```
{
  "myScope": {
    "type": "scope",
    "actions": {
      "call_bing": {
        "type": "http",
        "inputs": {
          "url": "http://www.bing.com"
        }
      }
    }
  }
}
```

### ForEach action

This looping action iterates through an array and performs inner actions for each item. By default, the foreach loop executes in parallel (20 executions in parallel at a time). You can set execution rules using the `operationOptions` parameter.

NAME	REQUIRED	TYPE	DESCRIPTION
actions	Yes	Object	Inner actions to execute within the loop
foreach	Yes	string	The array to iterate over
operationOptions	no	string	Any operation options for behavior. Currently only supports <code>sequential</code> to execute iterations sequentially (default behavior is parallel)

```

"forEach_email": {
    "type": "foreach",
    "foreach": "@body('email_filter')",
    "actions": {
        "send_email": {
            "type": "ApiConnection",
            "inputs": {
                "body": {
                    "to": "@item()", 
                    "from": "me@contoso.com",
                    "message": "Hello, thank you for ordering"
                }
            },
            "host": {
                "connection": {
                    "id": "@parameters('$connections')['office365']['connection']['id']"
                }
            }
        }
    },
    "runAfter": {
        "email_filter": [ "Succeeded" ]
    }
}

```

## Until action

This looping action executes inner actions until a condition results to true.

NAME	REQUIRED	TYPE	DESCRIPTION
actions	Yes	Object	Inner actions to execute within the loop
expression	Yes	string	The expression to evaluate after each iteration
limit	yes	Object	The limits for the loop - at least one limit must be defined
count	no	int	The limit to the number of iterations that can be performed
timeout	no	string	The timeout for how long it should loop. ISO 8601 format

```

"Until_succeeded": {
    "actions": {
        "Http": {
            "inputs": {
                "method": "GET",
                "uri": "http://myurl"
            },
            "runAfter": {},
            "type": "Http"
        }
    },
    "expression": "@equals(outputs('Http')['statusCode', 200]",
    "limit": {
        "count": 1000,
        "timeout": "PT1H"
    },
    "runAfter": {},
    "type": "Until"
}

```

## Conditions - If Action

The `If` action lets you evaluate a condition and execute a branch based on whether the expression evaluates to `true`.

NAME	REQUIRED	TYPE	DESCRIPTION
actions	Yes	Object	Inner actions to execute when expression evaluates to <code>true</code>
expression	Yes	string	The expression to evaluate
else	no	Object	Inner actions to execute when expression evaluates to <code>false</code>

```

"My_condition": {
    "actions": {
        "If_true": {
            "inputs": {
                "method": "GET",
                "uri": "http://myurl"
            },
            "runAfter": {},
            "type": "Http"
        }
    },
    "else": {
        "actions": {
            "if_false": {
                "inputs": {
                    "method": "GET",
                    "uri": "http://myurl"
                },
                "runAfter": {},
                "type": "Http"
            }
        }
    },
    "expression": "@equals(triggerBody(), json(true))",
    "runAfter": {},
    "type": "If"
}

```

The following table shows examples of how conditions can use expressions in an action:

JSON VALUE	RESULT
<pre>"expression": "@parameters('hasSpecialAction')"</pre>	Any value that would evaluate to true causes this condition to pass. Only Boolean expressions are supported. To convert other types to Boolean, use functions <code>empty</code> , <code>equals</code> .
<pre>"expression": "@greater(actions('act1').output.value, parameters('threshold'))"</pre>	Comparison functions are supported. For the example here, the action only executes when the output of act1 is greater than the threshold.
<pre>"expression": "@or(greater(actions('act1').output.value, parameters('threshold')), less(actions('act1').output.value, 100))"</pre>	Logic functions are also supported to create nested Boolean expressions. In this case, the action executes when the output of act1 is above the threshold or below 100.
<pre>"expression": "@equals(length(actions('act1').outputs.errors), 0)"</pre>	You can use array functions to check if an array has any items. In this case, the action executes when the errors array is empty.
<pre>"expression": "parameters('hasSpecialAction')"</pre>	Error - not a valid condition because @ is required for conditions.

If a condition evaluates successfully, the condition is marked as `Succeeded`. Actions within either the `actions` or `else` objects evaluate to `Succeeded` when executed and succeeded, `Failed` when executed and failed, or `Skipped` when that branch is not executed.

## Next steps

[Workflow Service REST API](#)

# Schema updates for Azure Logic Apps - June 1, 2016

2/28/2017 • 4 min to read • [Edit Online](#)

This new schema and API version for Azure Logic Apps includes key improvements that make logic apps more reliable and easier to use:

- [Scopes](#) let you group or nest actions as a collection of actions.
- [Conditions and loops](#) are now first-class actions.
- More precise ordering for running actions with the `runAfter` property, replacing `dependsOn`

To upgrade your logic apps from the August 1, 2015 preview schema to the June 1, 2016 schema, [check out the upgrade section](#).

## Scopes

This schema includes scopes, which let you group actions together, or nest actions inside each other. For example, a condition can contain another condition. Learn more about [scope syntax](#), or review this basic scope example:

```
{
  "actions": {
    "My_Scope": {
      "type": "scope",
      "actions": {
        "Http": {
          "inputs": {
            "method": "GET",
            "uri": "http://www.bing.com"
          },
          "runAfter": {},
          "type": "Http"
        }
      }
    }
  }
}
```

## Conditions and loops changes

In previous schema versions, conditions and loops were parameters associated with a single action. This schema lifts this limitation, so conditions and loops now appear as action types. Learn more about [loops and scopes](#), or review this basic example for a condition action:

```
{
    "If_trigger_is_some-trigger": {
        "type": "If",
        "expression": "@equals(triggerBody(), 'some-trigger')",
        "runAfter": { },
        "actions": {
            "Http_2": {
                "inputs": {
                    "method": "GET",
                    "uri": "http://www.bing.com"
                },
                "runAfter": {},
                "type": "Http"
            }
        },
        "else": {
            {
                "if_trigger_is_another-trigger": "..."
            }
        }
    }
}
```

## 'runAfter' property

The `runAfter` property replaces `dependsOn`, providing more precision when you specify the run order for actions based on the status of previous actions.

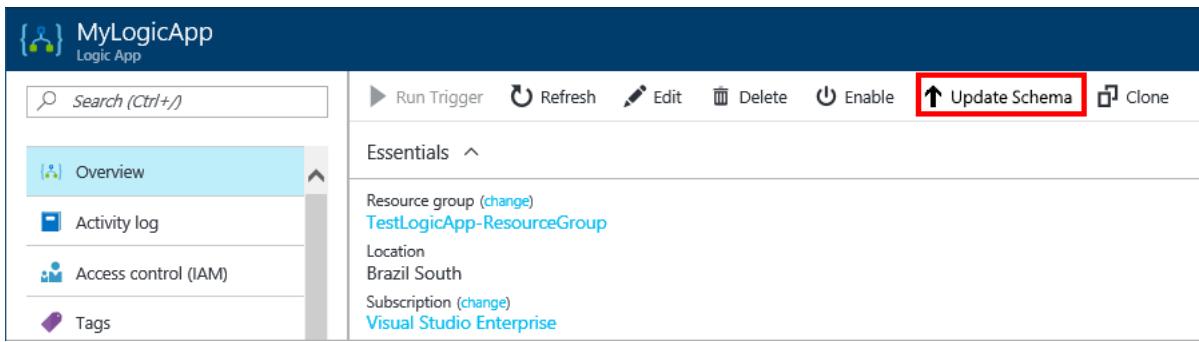
The `dependsOn` property was synonymous with "the action ran and was successful", no matter how many times you wanted to execute an action, based on whether the previous action was successful, failed, or skipped. The `runAfter` property provides that flexibility as an object that specifies all the action names after which the object runs. This property also defines an array of statuses that are acceptable as triggers. For example, if you wanted to run after step A succeeds and also after step B succeeds or fails, you construct this `runAfter` property:

```
{
    "...",
    "runAfter": {
        "A": ["Succeeded"],
        "B": ["Succeeded", "Failed"]
    }
}
```

## Upgrade your schema

Upgrading to the new schema only takes a few steps. The upgrade process includes running the upgrade script, saving as a new logic app, and if you want, possibly overwriting the previous logic app.

1. In the Azure portal, open your logic app.
2. Go to **Overview**. On the logic app toolbar, choose **Update Schema**.



The upgraded definition is returned, which you can copy and paste into a resource definition if necessary. However, we **strongly recommend** you choose **Save As** to make sure that all connection references are valid in the upgraded logic app.

3. In the upgrade blade toolbar, choose **Save As**.
4. Enter the logic name and status. To deploy your upgraded logic app, choose **Create**.
5. Confirm that your upgraded logic app works as expected.

#### NOTE

If you are using a manual or request trigger, the callback URL changes in your new logic app. Test the new URL to make sure the end-to-end experience works. To preserve previous URLs, you can clone over your existing logic app.

6. *Optional* To overwrite your previous logic app with the new schema version, on the toolbar, choose **Clone**, next to **Update Schema**. This step is necessary only if you want to keep the same resource ID or request trigger URL of your logic app.

## Upgrade tool notes

### Mapping conditions

In the upgraded definition, the tool makes a best effort at grouping true and false branch actions together as a scope. Specifically, the designer pattern of `@equals(actions('a').status, 'Skipped')` should appear as an `else` action. However, if the tool detects unrecognizable patterns, the tool might create separate conditions for both the true and the false branch. You can remap actions after upgrading, if necessary.

### 'foreach' loop with condition

In the new schema, you can use the filter action to replicate the pattern of a `foreach` loop with a condition per item, but this change should automatically happen when you upgrade. The condition becomes a filter action before the foreach loop for returning only an array of items that match the condition, and that array is passed into the foreach action. For an example, see [Loops and scopes](#).

### Resource tags

After you upgrade, resource tags are removed, so you must reset them for the upgraded workflow.

## Other changes

### Renamed 'manual' trigger to 'request' trigger

The `manual` trigger type was deprecated and renamed to `request` with type `http`. This change creates more consistency for the kind of pattern that the trigger is used to build.

### New 'filter' action

To filter a large array down to a smaller set of items, the new `filter` type accepts an array and a condition, evaluates the condition for each item, and returns an array with items meeting the condition.

### Restrictions for 'foreach' and 'until' actions

The `foreach` and `until` loop are restricted to a single action.

### New 'trackedProperties' for actions

Actions can now have an additional property called `trackedProperties`, which is sibling to the `runAfter` and `type` properties. This object specifies certain action inputs or outputs that you want to include in the Azure Diagnostic telemetry, emitted as part of a workflow. For example:

```
{  
    "Http": {  
        "inputs": {  
            "method": "GET",  
            "uri": "http://www.bing.com"  
        },  
        "runAfter": {},  
        "type": "Http",  
        "trackedProperties": {  
            "responseCode": "@action().outputs.statusCode",  
            "uri": "@action().inputs.uri"  
        }  
    }  
}
```

## Next Steps

- [Create workflow definitions for logic apps](#)
- [Create deployment templates for logic apps](#)

# Schema updates for Azure Logic Apps - August 1, 2015 preview

2/28/2017 • 8 min to read • [Edit Online](#)

This new schema and API version for Azure Logic Apps includes key improvements that make logic apps more reliable and easier to use:

- The **APIApp** action type is updated to a new **APIConnection** action type.
- **Repeat** is renamed to **Foreach**.
- The **HTTP Listener API App** is no longer required.
- Calling child workflows uses a [new schema](#).

## Move to API connections

The biggest change is that you no longer have to deploy API Apps into your Azure subscription so you can use APIs. Here are the ways that you can use APIs:

- Managed APIs
- Your custom Web APIs

Each way is handled slightly differently because their management and hosting models are different. One advantage of this model is you're no longer constrained to resources that are deployed in your Azure resource group.

### Managed APIs

Microsoft manages some APIs on your behalf, such as Office 365, Salesforce, Twitter, and FTP. You can use some managed APIs as-is, such as Bing Translate, while others require configuration. This configuration is called a *connection*.

For example, when you use Office 365, you must create a connection that contains your Office 365 sign-in token. This token is securely stored and refreshed so that your logic app can always call the Office 365 API. Alternatively, if you want to connect to your SQL or FTP server, you must create a connection that has the connection string.

In this definition, these actions are called **APIConnection**. Here is an example of a connection that calls Office 365 to send an email:

```
{
  "actions": {
    "Send_Email": {
      "type": "ApiConnection",
      "inputs": {
        "host": {
          "api": {
            "runtimeUrl": "https://msmanaged-na.azure-apim.net/apim/office365"
          },
          "connection": {
            "name": "@parameters('$connections')['shared_office365']['connectionId']"
          }
        },
        "method": "post",
        "body": {
          "Subject": "Reminder",
          "Body": "Don't forget!",
          "To": "me@contoso.com"
        },
        "path": "/Mail"
      }
    }
  }
}
```

The `host` object is portion of inputs that is unique to API connections, and contains tow parts: `api` and `connection`.

The `api` has the runtime URL of where that managed API is hosted. You can see all the available managed APIs by calling

```
GET https://management.azure.com/subscriptions/{subid}/providers/Microsoft.Web/managedApis/?api-version=2015-08-01-preview
```

When you use an API, the API might or might not have any *connection parameters* defined. If the API doesn't, no *connection* is required. If the API does, you must create a connection. The created connection has the name that you choose. You then reference the name in the `connection` object inside the `host` object. To create a connection in a resource group, call:

```
PUT
https://management.azure.com/subscriptions/{subid}/resourceGroups/{rgname}/providers/Microsoft.Web/connections/{name}?api-version=2015-08-01-preview
```

With the following body:

```
{
  "properties": {
    "api": {
      "id": "/subscriptions/{subid}/providers/Microsoft.Web/managedApis/azureblob"
    },
    "parameterValues": {
      "accountName": "{The name of the storage account -- the set of parameters is different for each API}"
    }
  },
  "location": "{Logic app's location}"
}
```

## Deploy managed APIs in an Azure Resource Manager template

You can create a full application in an Azure Resource Manager template as long as interactive sign-in isn't required. If sign-in is required, you can set up everything with the Azure Resource Manager template, but you still

have to visit the portal to authorize the connections.

```
"resources": [{"  
    "apiVersion": "2015-08-01-preview",  
    "name": "azureblob",  
    "type": "Microsoft.Web/connections",  
    "location": "[resourceGroup().location]",  
    "properties": {  
        "api": {  
            "id": "  
[concat(subscription().id, '/providers/Microsoft.Web/locations/westus/managedApis/azureblob')]"  
        },  
        "parameterValues": {  
            "accountName": "[parameters('storageAccountName')]",  
            "accessKey": "[parameters('storageAccountKey')]"  
        }  
    }, {  
        "type": "Microsoft.Logic/workflows",  
        "apiVersion": "2015-08-01-preview",  
        "name": "[parameters('logicAppName')]",  
        "location": "[resourceGroup().location]",  
        "dependsOn": "[[resourceId('Microsoft.Web/connections', 'azureblob')]]"  
    ],  
    "properties": {  
        "sku": {  
            "name": "[parameters('sku')]",  
            "plan": {  
                "id": "[concat(resourceGroup().id,  
'/providers/Microsoft.Web/serverfarms/', parameters('svcPlanName'))]"  
            }  
        },  
        "definition": {  
            "$schema": "https://schema.management.azure.com/providers/Microsoft.Logic/schemas/2015-08-01-  
preview/workflowdefinition.json#",  
            "actions": {  
                "Create_file": {  
                    "type": "apiconnection",  
                    "inputs": {  
                        "host": {  
                            "api": {  
                                "runtimeUrl": "https://logic-apis-westus.azure-apim.net/apim/azureblob"  
                            },  
                            "connection": {  
                                "name": "@parameters('$connections')['azureblob']['connectionId']"  
                            }  
                        },  
                        "method": "post",  
                        "queries": {  
                            "folderPath": "[concat('/', parameters('containerName'))]",  
                            "name": "helloworld.txt"  
                        },  
                        "body": "@decodeDataUri('data:, Hello+world!')",  
                        "path": "/datasets/default/files"  
                    },  
                    "conditions": []  
                }  
            },  
            "contentVersion": "1.0.0.0",  
            "outputs": {},  
            "parameters": {  
                "$connections": {  
                    "defaultValue": {},  
                    "type": "Object"  
                }  
            },  
            "triggers": {  
                "recurrence": {  

```

```

        "recurrence": {
            "frequency": "Day",
            "interval": 1
        }
    }
},
"parameters": {
    "$connections": {
        "value": {
            "azureblob": {
                "connectionId": [
                    concat(resourceGroup().id,'/providers/Microsoft.Web/connections/azureblob')
                ],
                "connectionName": "azureblob",
                "id": [
                    concat(subscription().id,'/providers/Microsoft.Web/locations/westus/managedApis/azureblob')
                ]
            }
        }
    }
}
]

```

You can see in this example that the connections are just resources that live in your resource group. They reference the managed APIs available to you in your subscription.

## Your custom Web APIs

If you use your own APIs, not Microsoft-managed ones, use the built-in **HTTP** action to call them. For an ideal experience, you should expose a Swagger endpoint for your API. This endpoint enables the Logic App Designer to render the inputs and outputs for your API. Without Swagger, the designer can only show the inputs and outputs as opaque JSON objects.

Here is an example showing the new `metadata.apiDefinitionUrl` property:

```
{
    "actions": {
        "mycustomAPI": {
            "type": "http",
            "metadata": {
                "apiDefinitionUrl": "https://mysite.azurewebsites.net/api/apidef/"
            },
            "inputs": {
                "uri": "https://mysite.azurewebsites.net/api/getsomeodata",
                "method": "GET"
            }
        }
    }
}
```

If you host your Web API on Azure App Service, your Web API automatically appears in the list of actions available in the designer. If not, you have to paste in the URL directly. The Swagger endpoint must be unauthenticated to be usable in the Logic App Designer, although you can secure the API itself with whatever methods that Swagger supports.

## Call deployed API apps with 2015-08-01-preview

If you previously deployed an API App, you can call the app with the **HTTP** action.

For example, if you use Dropbox to list files, your **2014-12-01-preview** schema version definition might have something like:

```
{
    "$schema": "https://schema.management.azure.com/providers/Microsoft.Logic/schemas/2014-12-01-preview/workflowdefinition.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "/subscriptions/423db32d-...-",
        "b59f14c962f1/resourcegroups/avdemo/providers/Microsoft.AppService/apiapps/dropboxconnector/token": {
            "defaultValue": "eyJ0eX...wCn90",
            "type": "String",
            "metadata": {
                "token": {
                    "name": "/subscriptions/423db32d-...-"
                }
            }
        },
        "b59f14c962f1/resourcegroups/avdemo/providers/Microsoft.AppService/apiapps/dropboxconnector/token"
    },
    "actions": {
        "dropboxconnector": {
            "type": "ApiApp",
            "inputs": {
                "apiVersion": "2015-01-14",
                "host": {
                    "id": "/subscriptions/423db32d-...-",
                    "b59f14c962f1/resourcegroups/avdemo/providers/Microsoft.AppService/apiapps/dropboxconnector",
                    "gateway": "https://avdemo.azurewebsites.net"
                },
                "operation": "ListFiles",
                "parameters": {
                    "FolderPath": "/myfolder"
                },
                "authentication": {
                    "type": "Raw",
                    "scheme": "Zumo",
                    "parameter": "@parameters('/subscriptions/423db32d-...-",
                    "b59f14c962f1/resourcegroups/avdemo/providers/Microsoft.AppService/apiapps/dropboxconnector/token')"
                }
            }
        }
    }
}
```

You can construct the equivalent HTTP action like this example, while the parameters section of the Logic app definition remains unchanged:

```
{
    "actions": {
        "dropboxconnector": {
            "type": "Http",
            "metadata": {
                "apiDefinitionUrl": "https://avdemo.azurewebsites.net/api/service/apidef/dropboxconnector/?api-version=2015-01-14&format=swagger-2.0-standard"
            },
            "inputs": {
                "uri": "https://avdemo.azurewebsites.net/api/service/invoke/dropboxconnector/ListFiles?api-version=2015-01-14",
                "method": "POST",
                "body": {
                    "FolderPath": "/myfolder"
                },
                "authentication": {
                    "type": "Raw",
                    "scheme": "Zumo",
                    "parameter": "@parameters('/subscriptions/423db32d-...-b59f14c962f1/resourcegroups/avdemo/providers/Microsoft.AppService/apiapps/dropboxconnector/token')"
                }
            }
        }
    }
}
```

Walking through these properties one-by-one:

ACTION PROPERTY	DESCRIPTION
<code>type</code>	<code>Http</code> instead of <code>APIapp</code>
<code>metadata.apiDefinitionUrl</code>	To use this action in the Logic App Designer, include the metadata endpoint, which is constructed from: <code>{api app host.gateway}/api/service/apidef/{last segment of the api app host.id}/?api-version=2015-01-14&amp;format=swagger-2.0-standard</code>
<code>inputs.uri</code>	Constructed from: <code>{api app host.gateway}/api/service/invoke/{last segment of the api app host.id}/{api app operation}?api-version=2015-01-14</code>
<code>inputs.method</code>	Always <code>POST</code>
<code>inputs.body</code>	Identical to the API App parameters
<code>inputs.authentication</code>	Identical to the API App authentication

This approach should work for all API App actions. However, remember that these previous API Apps are no longer supported. So you should move to one of the two other previous options, a managed API or hosting your custom Web API.

## Renamed 'repeat' to 'foreach'

For the previous schema version, we received much customer feedback that **Repeat** was confusing and didn't properly capture that **Repeat** was really a for-each loop. As a result, we have renamed `repeat` to `foreach`. For example, previously you would write:

```
{
  "actions": {
    "pingBing": {
      "type": "Http",
      "repeat": "@range(0,2)",
      "inputs": {
        "method": "GET",
        "uri": "https://www.bing.com/search?q=@{repeatItem()}"
      }
    }
  }
}
```

Now you would write:

```
{
  "actions": {
    "pingBing": {
      "type": "Http",
      "foreach": "@range(0,2)",
      "inputs": {
        "method": "GET",
        "uri": "https://www.bing.com/search?q=@{item()}"
      }
    }
  }
}
```

The function `@repeatItem()` was previously used to reference the current item being iterated over. This function is now simplified to `@item()`.

### Reference outputs from 'foreach'

For simplification, the outputs from `foreach` actions are not wrapped in an object called `repeatItems`. While the outputs from the previous `repeat` example were:

```
{
  "repeatItems": [
    {
      "name": "pingBing",
      "inputs": {
        "uri": "https://www.bing.com/search?q=0",
        "method": "GET"
      },
      "outputs": {
        "headers": { },
        "body": "<!DOCTYPE html><html lang=\"en\" xml:lang=\"en\"  

xmlns=\"http://www.w3.org/1999/xhtml\" xmlns:Web=\"http://schemas.live.com/Web/\">...</html>"
      }
      "status": "Succeeded"
    }
  ]
}
```

Now these outputs are:

```
[
  {
    "name": "pingBing",
    "inputs": {
      "uri": "https://www.bing.com/search?q=0",
      "method": "GET"
    },
    "outputs": {
      "headers": { },
      "body": "<!DOCTYPE html><html lang=\"en\" xml:lang=\"en\" xmlns=\"http://www.w3.org/1999/xhtml\" xmlns:Web=\"http://schemas.live.com/Web/\">...</html>",
      "status": "Succeeded"
    }
  }
]
```

Previously, to get to the body of the action when referencing these outputs:

```
{
  "actions": {
    "secondAction": {
      "type": "Http",
      "repeat": "@outputs('pingBing').repeatItems",
      "inputs": {
        "method": "POST",
        "uri": "http://www.example.com",
        "body": "@repeatItem().outputs.body"
      }
    }
  }
}
```

Now you can do instead:

```
{
  "actions": {
    "secondAction": {
      "type": "Http",
      "foreach": "@outputs('pingBing')",
      "inputs": {
        "method": "POST",
        "uri": "http://www.example.com",
        "body": "@item().outputs.body"
      }
    }
  }
}
```

With these changes, the functions `@repeatItem()`, `@repeatBody()`, and `@repeatOutputs()` are removed.

## Native HTTP listener

The HTTP Listener capabilities are now built in. So you no longer need to deploy an HTTP Listener API App. See [the full details for how to make your Logic app endpoint callable here](#).

With these changes, we removed the `@accessKeys()` function, which we replaced with the `@listCallbackURL()` function for getting the endpoint when necessary. Also, you must now define at least one trigger in your logic app. If you want to `/run` the workflow, you must have one of these triggers: `manual`, `apiConnectionWebhook`, or `httpWebhook`.

## Call child workflows

Previously, calling child workflows required going to the workflow, getting the access token, and pasting the token in the logic app definition where you want to call that child workflow. With the new schema, the Logic Apps engine automatically generates a SAS at runtime for the child workflow so you don't have to paste any secrets into the definition. Here is an example:

```
"mynestedwf": {
    "type": "workflow",
    "inputs": {
        "host": {
            "id": "/subscriptions/xxxxxxxxzzz/resourceGroups/rg001/providers/Microsoft.Logic/mywf001",
            "triggerName": "myendpointtrigger"
        },
        "queries": {
            "extrafield": "specialValue"
        },
        "headers": {
            "x-ms-date": "@utcnow()",
            "Content-type": "application/json"
        },
        "body": {
            "contentFieldOne": "value100",
            "anotherField": 10.001
        }
    },
    "conditions": []
}
```

A second improvement is we are giving the child workflows full access to the incoming request. That means that you can pass parameters in the *queries* section and in the *headers* object and that you can fully define the entire body.

Finally, there are required changes to the child workflow. While you could previously call a child workflow directly, now you must define a trigger endpoint in the workflow for the parent to call. Generally, you would add a trigger that has `manual` type, and then use that trigger in the parent definition. Note the `host` property specifically has a `triggerName` because you must always specify which trigger you are invoking.

## Other changes

### New 'queries' property

All action types now support a new input called `queries`. This input can be a structured object, rather than you having to assemble the string by hand.

### Renamed 'parse()' function to 'json()'

We are adding more content types soon, so we renamed the `parse()` function to `json()`.

## Coming soon: Enterprise Integration APIs

We don't have managed versions yet of the Enterprise Integration APIs, like AS2. Meanwhile, you can use your existing deployed BizTalk APIs through the HTTP action. For details, see "Using your already deployed API apps" in the [integration roadmap](#).