

Introduction to Microsoft Flow



Agenda

- Flow Fundamentals
- Writing Flow Expressions
- Converting Between Types
- Working with Arrays
- Advanced Techniques

Thanks for coming up with the idea for this session



Stephen Siciliano, Principal Group PM Manager



Deep Dive into PowerApps and Flow

- Two action-packed days of building PowerApps and Flows
 1. Getting Started with PowerApps Studio
 2. Designing PowerApps using Advanced Techniques
 3. Building PowerApps for SharePoint Online
 4. Introduction to Microsoft Flow
 5. Designing Flows to Automate an Approval Process
 6. Building PowerApps and Flows for Power BI
 7. Working with the Common Data Service for Apps
 8. Managing Application Lifecycle with PowerApps and Flow
- More info
 - <https://www.criticalpathtraining.com>
 - info@criticalpathtraining.com



What is Flow?

- Service for automating workflows across other services
 - Designed by Microsoft for business users more than developers
- What can you do with Flow?
 - Get notifications
 - Create, copy, update and delete files
 - Collect and clean data
 - Automate approvals



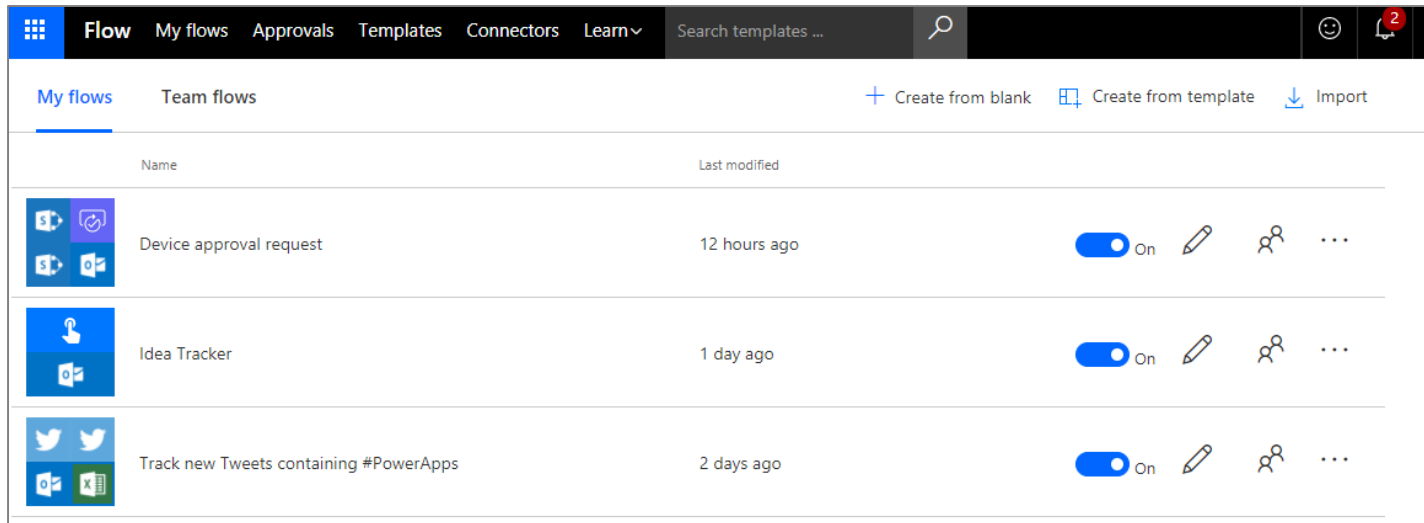
Building Blocks of Flow

- Flows whether created from a template or from scratch:
 - Will contain **building blocks** that work together in certain ways (much like a flowchart)
- Building blocks of Flow:
 - **Services** - sources and destinations of data in a flow
 - **Triggers** - events that start a flow
 - **Actions** - tasks accomplished by the flow
 - **Conditions** - allow for branching if/then logic in a flow
 - **Loops** - for iterating over actions more than once



Creating and Managing Flows

- Each user has a list of flows
 - Flows can be shared with or transferred to other users

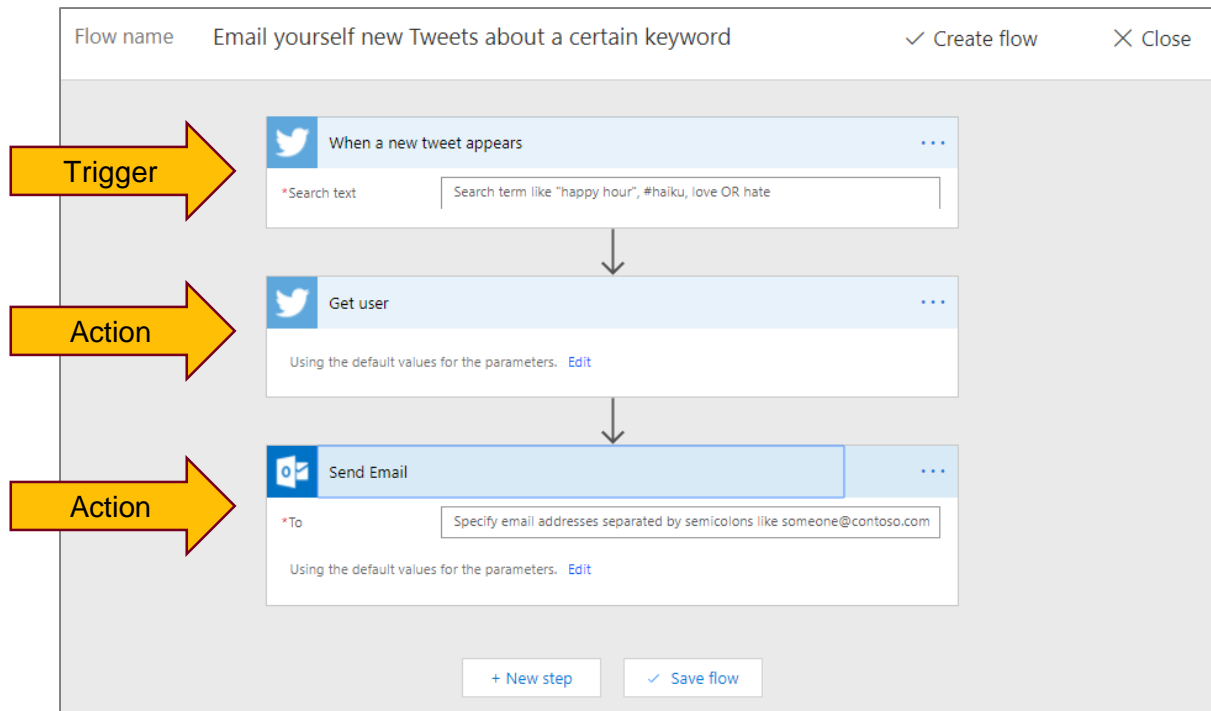


- Two ways to create a new flow
 - Create from template
 - Create from blank



Working with the Flow Designer

- Flow Designer provides UI for building flows
 - You build flows by adding and configuring steps
 - There are 3 types of steps: triggers, actions or conditions
 - Rename steps before referencing them in later steps

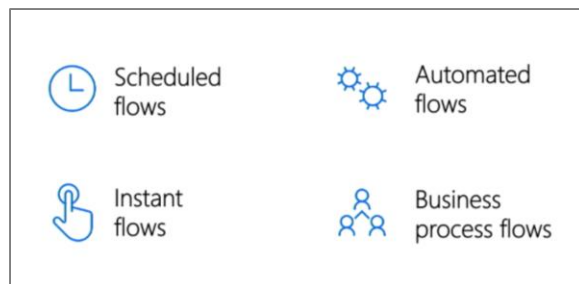


Example Award Winning Flow



Types of Flows

- Automated Flows
 - Runs when something happens (e.g. new list item)
- Scheduled Flows
 - Runs periodically based on an interval
- Instant Flows
 - Runs when a user clicks a button
- Business Process Flows
 - Integrated with CDS for Apps and Dynamics 365





DEMO

Changing the Flow Trigger Type

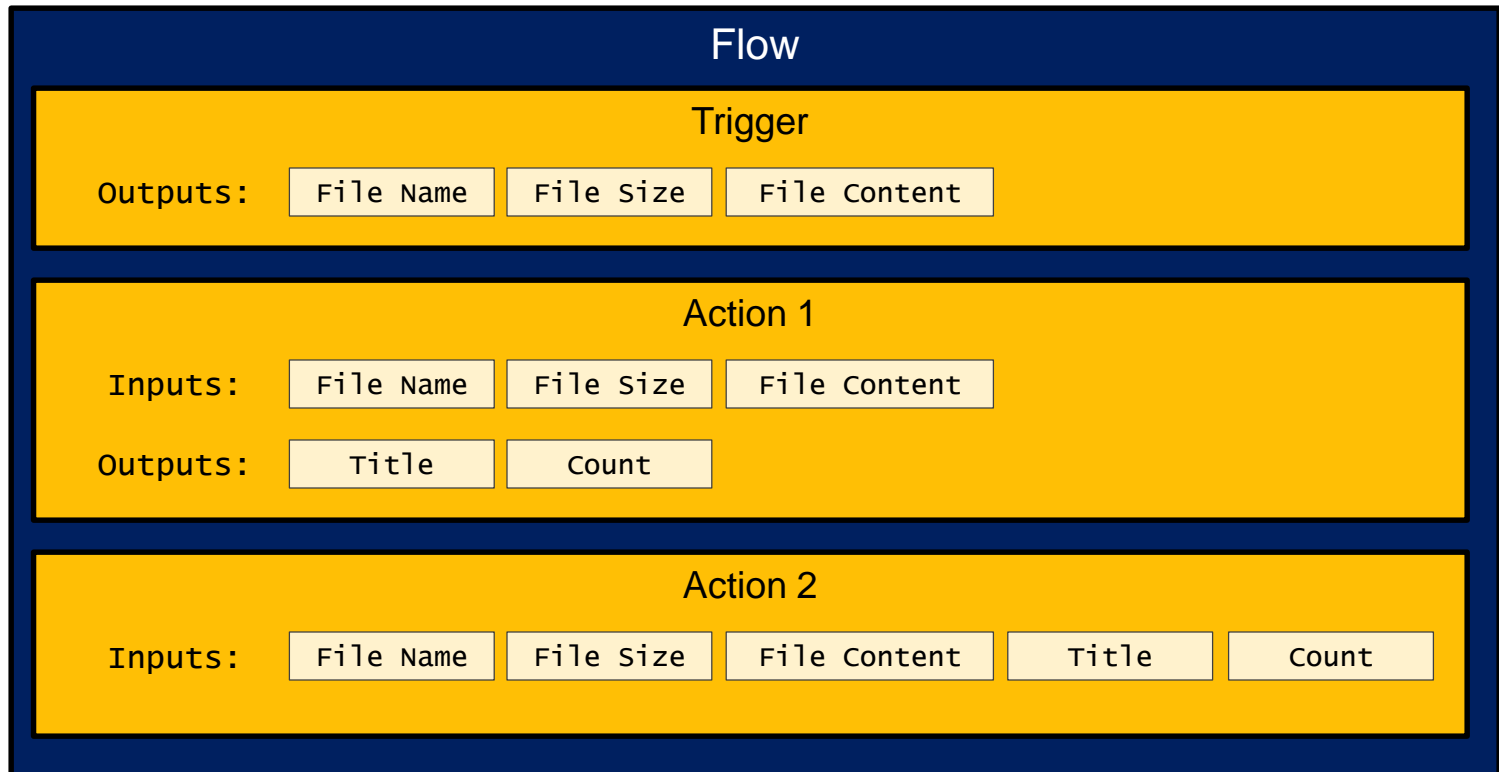
Agenda

- ✓ Flow Fundamentals
- Writing Flow Expressions
 - Converting Between Types
 - Working with Arrays
 - Advanced Techniques



Flowing Data

- Data in flows added by steps
 - Data added in step is available in later steps
 - Add Dynamic Content select outputs from previous steps
 - Certain outputs shows up based on types of input and output



Writing Flow Expressions

- Scenarios for writing Flow expressions
 - Convert types
 - Perform simple inline calculations
 - Perform string manipulation
 - Generate a GUID or a random number
 - Handling optional values
 - Writing conditional statements using "If" statements
 - Working with lists



Workflow Definition Language (WDL)

- Flow expressions written in Workflow Definition Language
 - Same language used in Azure Logic Apps
 - WDL is more powerful yet more complicated than PowerApps
 - WDL does not overload operators like PowerApps does
 - WDL requires single quotes instead of double quotes

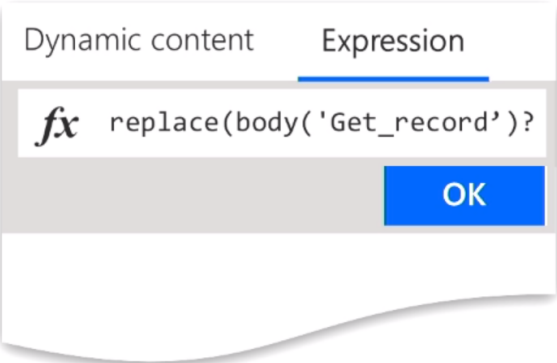
<code>fx "this is a test"</code>	Invalid: no double quotes
<code>fx 'this is a test'</code>	Valid
<code>fx 'this is a ' + 'test'</code>	Invalid : + operator not supported
<code>fx 'this is a ' & 'test'</code>	Invalid : & operator not supported
<code>fx concat('this is a ', 'test')</code>	Valid



Using the Add Dynamic Content

How to create expressions

1  Add Dynamic Content

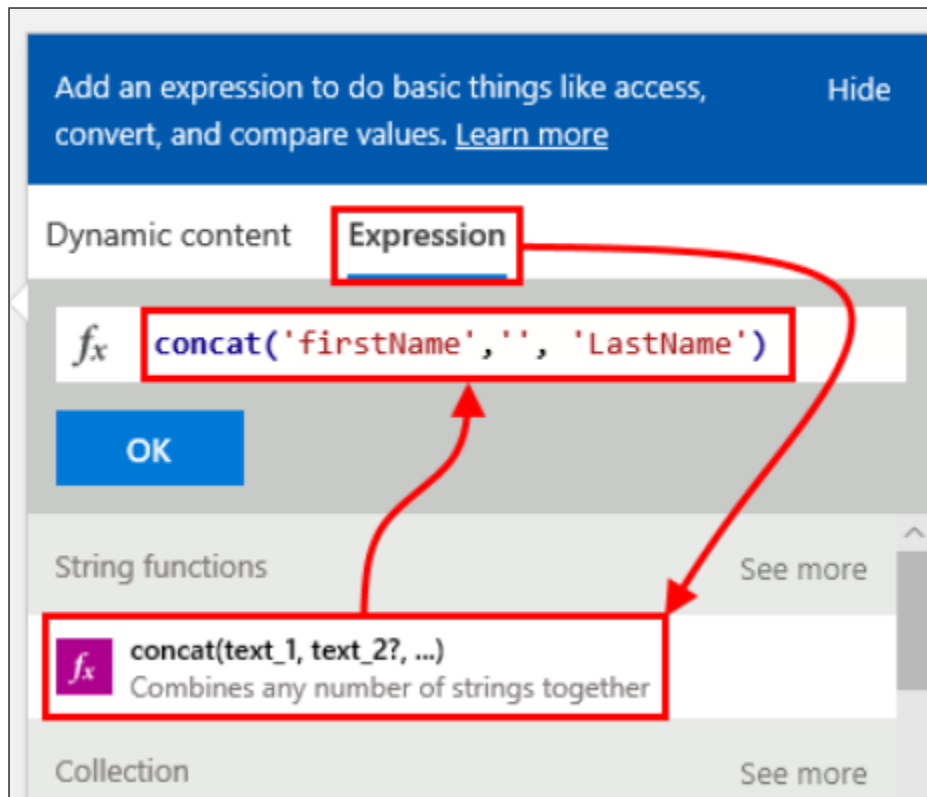
2 
Dynamic content Expression
fx replace(body('Get_record'))?
OK

3 
Action name
Inputs *fx* replace(...) ×



Writing Expressions

- Expressions written in fx textbox
- Click OK to enter expressions




Working with Strings

- Parse text together using **concat()**
- Parse out text using **substring()**
- Convert casing using **toLowerCase()** and **toUpperCase()**
- Search string using **indexOf** and **startsWith()**
- Create new GUID identifier using **guid()**

 **concat(text_1, text_2?, ...)**
Combines any number of strings together

 **substring(text, startIndex, length)**
Returns a subset of characters from a string

 **replace(text, oldText, newText)**
Replaces a string with a given string


 **guid()**
Generates a globally unique string (GUID)


 **toLowerCase(text)**
Converts a string to lowercase using the casing rules of the i...

 **toUpperCase(text)**
Converts a string to uppercase using the casing rules of the i...

 **indexOf(text, searchText)**
Returns the first index of a value within a string (case-insensi...

 **lastIndexOf(text, searchText)**
Returns the last index of a value within a string (case-insensit...

 **startsWith(text, searchText)**
Checks if the string starts with a value (case-insensitive, invar...

 **endsWith(text, searchText)**
Checks if the string ends with a value (case-insensitive, invari...



Performing Arithmetic Operations

- You cannot use standard arithmetic operators
 - This includes familiar operators such as **+**, **-**, *****, **/**
 - This does not work: **2 + 2**
 - This works: **add(2, 2)**

fx **min(collection or item1, item2?, ...)**
Returns the minimum value in the input array of numbers

fx **max(collection or item1, item2?, ...)**
Returns the maximum value in the input array of numbers

fx **rand(minValue, maxValue)**
Generates a random integer within the specified range (inclu...

fx **add(summand_1, summand_2)**
Returns the result from adding the two numbers

fx **sub(minuend, subtrahend)**
Returns the result from subtracting two numbers

fx **mul(multiplicand_1, multiplicand_2)**
Returns the result from multiplying the two numbers

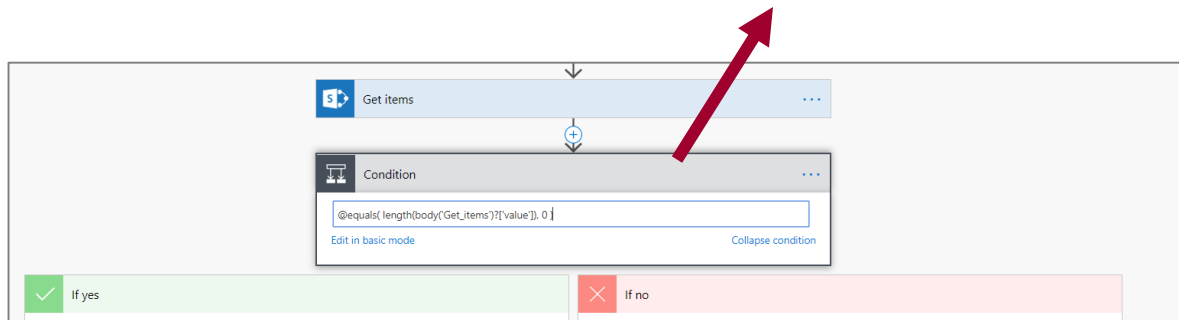
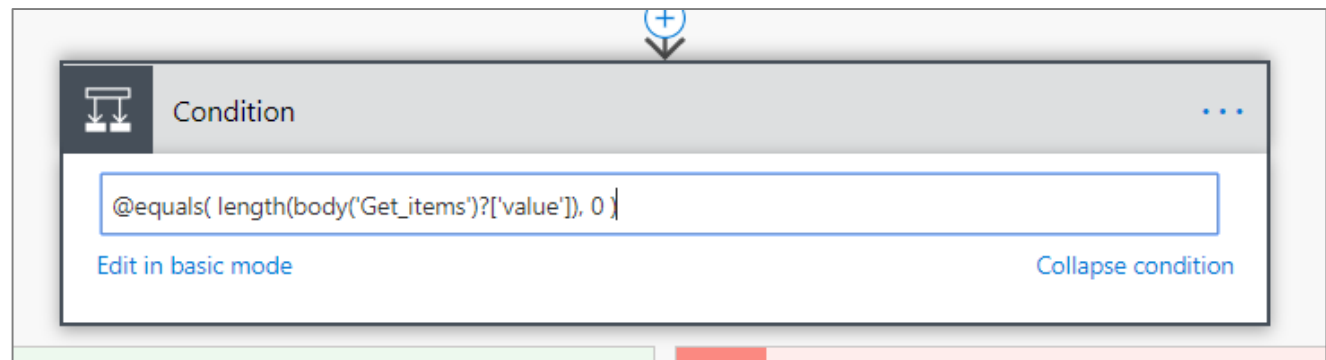
fx **div(dividend, divisor)**
Returns the result from dividing the two numbers

fx **mod(dividend, divisor)**
Returns the remainder after dividing the two numbers (mod...



Using Boolean Expressions in a Condition

- Do as much as you can in basic mode
 - Then select Edit in advanced mode
 - Enter expression as Boolean expression
 - Unlike in Dynamic Content, you must begin with @



Escaping Rules

- Escaping often required when writing expressions

Using strings with a single quote character:
`@substring('It' 's A Great Day!',1,5)`

Using spaces in action names:
`@body('Name_with_spaces')`

Building an object you need to escape '@':

```
{  
  "@@odata.type" : ""  
}
```



Agenda

- ✓ Flow Fundamentals
- ✓ Writing Flow Expressions
- Converting Between Types
 - Working with Arrays
 - Advanced Techniques



Handling Type Conversion

- Some conversion is automatic
 - Sometimes conversions are performed for you
 - In other cases, you must explicitly convert between types

**string(value)**

Convert the parameter to a string

**float(value)**

Convert the parameter argument to a floating-point number

**bool(value)**

Convert the parameter to a Boolean

**base64(value)**

Returns the base 64 representation of the input string

**base64ToBinary(value)**

Returns a binary representation of a base 64 encoded string

**base64ToString(value)**

Returns a string representation of a base 64 encoded string

**binary(value)**

Returns a binary representation of a value

**dataUriToBinary(value)**

Returns a binary representation of a data URI

**dataUriToString(value)**

Returns a string representation of a data URI

**dataUri(value)**

Returns a data URI of a value

**uriComponent(value)**

Returns a URI encoded representation of a value

**uriComponentToBinary(value)**

Returns a binary representation of a URI encoded string

**uriComponentToString(value)**

Returns a string representation of a URI encoded string



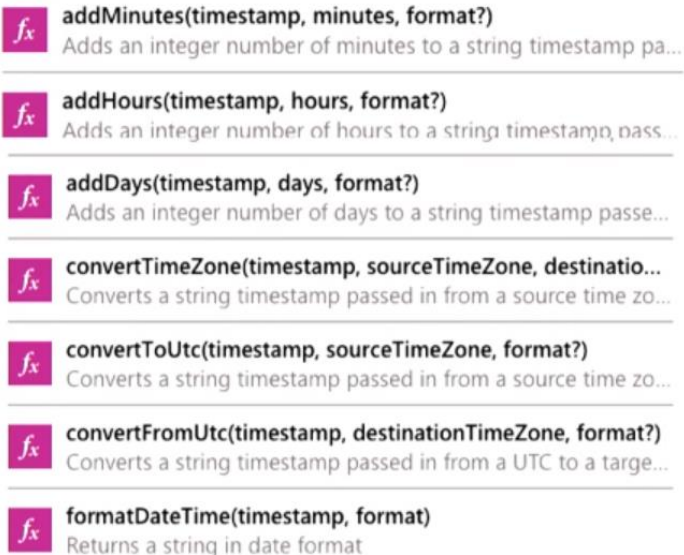
Flow Type Conversion Matrix

To From	UI adds automatically					Floating-point	Integer	Bool.	Array	JSON Object	XML content
	String	Base 64	Binary content	Data URI	URI comp.						
String	Yes	base64()	binary()	dataUri()	uriComponent()	float()	int()	bool()	split() json()	json()	xml()
Base 64	base64ToString()	Yes	base64ToBinary()	*	*	*	*	*	*	*	*
Binary content	string()	base64()	Yes	dataUri()	uriComponent()	*	*	*	*	*	*
Data URI	dataUriToString()	*	dataUriToBinary()	Yes	*	*	*	*	*	*	*
URI comp.	uriComponentToString()	*	uriComponentToBinary()	*	Yes	*	*	*	*	*	*
Floating-point	Yes	base64()	binary()	dataUri()	uriComponent()	Yes	No	No	No	No	No
Integer	Yes	base64()	binary()	dataUri()	uriComponent()	Yes	Yes	No	No	No	No
Bool.	Yes	base64()	binary()	dataUri()	uriComponent()	No	No	Yes	No	No	No
Array	join() string()	*	*	*	*	No	No	No	Select Action	Select or Compose	xml()
JSON object	string()	*	*	*	*	No	No	No	Select or Compose	Compose Action	xml()
XML content	string()	*	*	*	*	No	No	No	xpath()	xpath()	Logic apps only



Working with Dates and Time

- Get Greenwich Meantime using **utcnow()**
- Use **add*()** functions to move time back/forward
- **convertTimeZone()** used to handle local times
- **formatDateTime()** used to format



A screenshot of a code editor window displaying a list of date and time functions. Each function is preceded by a small icon of a pink square with a white 'fx' symbol. The functions listed are: **addMinutes(timestamp, minutes, format?)** (Adds an integer number of minutes to a string timestamp passed in), **addHours(timestamp, hours, format?)** (Adds an integer number of hours to a string timestamp passed in), **addDays(timestamp, days, format?)** (Adds an integer number of days to a string timestamp passed in), **convertTimeZone(timestamp, sourceTimeZone, destinationTimeZone, format?)** (Converts a string timestamp passed in from a source time zone to a destination time zone), **convertToUtc(timestamp, sourceTimeZone, format?)** (Converts a string timestamp passed in from a source time zone to UTC), **convertFromUtc(timestamp, destinationTimeZone, format?)** (Converts a string timestamp passed in from a UTC to a target time zone), and **formatDateTime(timestamp, format)** (Returns a string in date format).

fx **addMinutes(timestamp, minutes, format?)**
Adds an integer number of minutes to a string timestamp passed in

fx **addHours(timestamp, hours, format?)**
Adds an integer number of hours to a string timestamp passed in

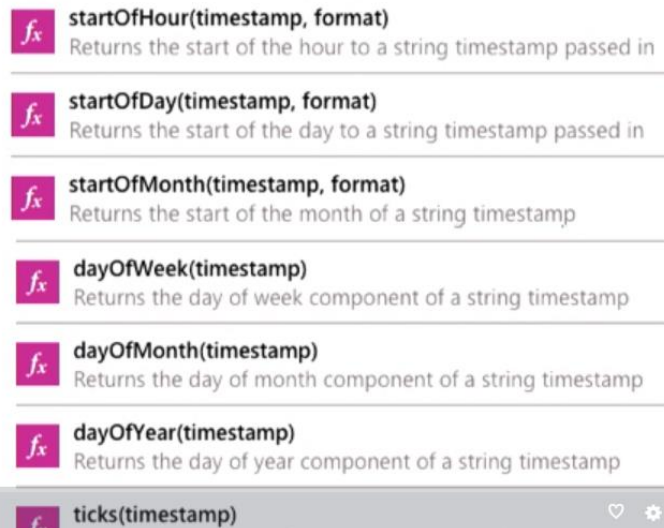
fx **addDays(timestamp, days, format?)**
Adds an integer number of days to a string timestamp passed in

fx **convertTimeZone(timestamp, sourceTimeZone, destinationTimeZone, format?)**
Converts a string timestamp passed in from a source time zone to a destination time zone

fx **convertToUtc(timestamp, sourceTimeZone, format?)**
Converts a string timestamp passed in from a source time zone to UTC

fx **convertFromUtc(timestamp, destinationTimeZone, format?)**
Converts a string timestamp passed in from a UTC to a target time zone

fx **formatDateTime(timestamp, format)**
Returns a string in date format



A screenshot of a code editor window displaying a list of date and time functions. Each function is preceded by a small icon of a pink square with a white 'fx' symbol. The functions listed are: **startOfHour(timestamp, format)** (Returns the start of the hour to a string timestamp passed in), **startOfDay(timestamp, format)** (Returns the start of the day to a string timestamp passed in), **startOfMonth(timestamp, format)** (Returns the start of the month of a string timestamp), **dayOfWeek(timestamp)** (Returns the day of week component of a string timestamp), **dayOfMonth(timestamp)** (Returns the day of month component of a string timestamp), **dayOfYear(timestamp)** (Returns the day of year component of a string timestamp), and **ticks(timestamp)**. The **ticks(timestamp)** function is highlighted with a grey background. At the bottom right of the editor, there are icons for a heart, a gear, a speaker, and a star.

fx **startOfHour(timestamp, format)**
Returns the start of the hour to a string timestamp passed in

fx **startOfDay(timestamp, format)**
Returns the start of the day to a string timestamp passed in

fx **startOfMonth(timestamp, format)**
Returns the start of the month of a string timestamp

fx **dayOfWeek(timestamp)**
Returns the day of week component of a string timestamp

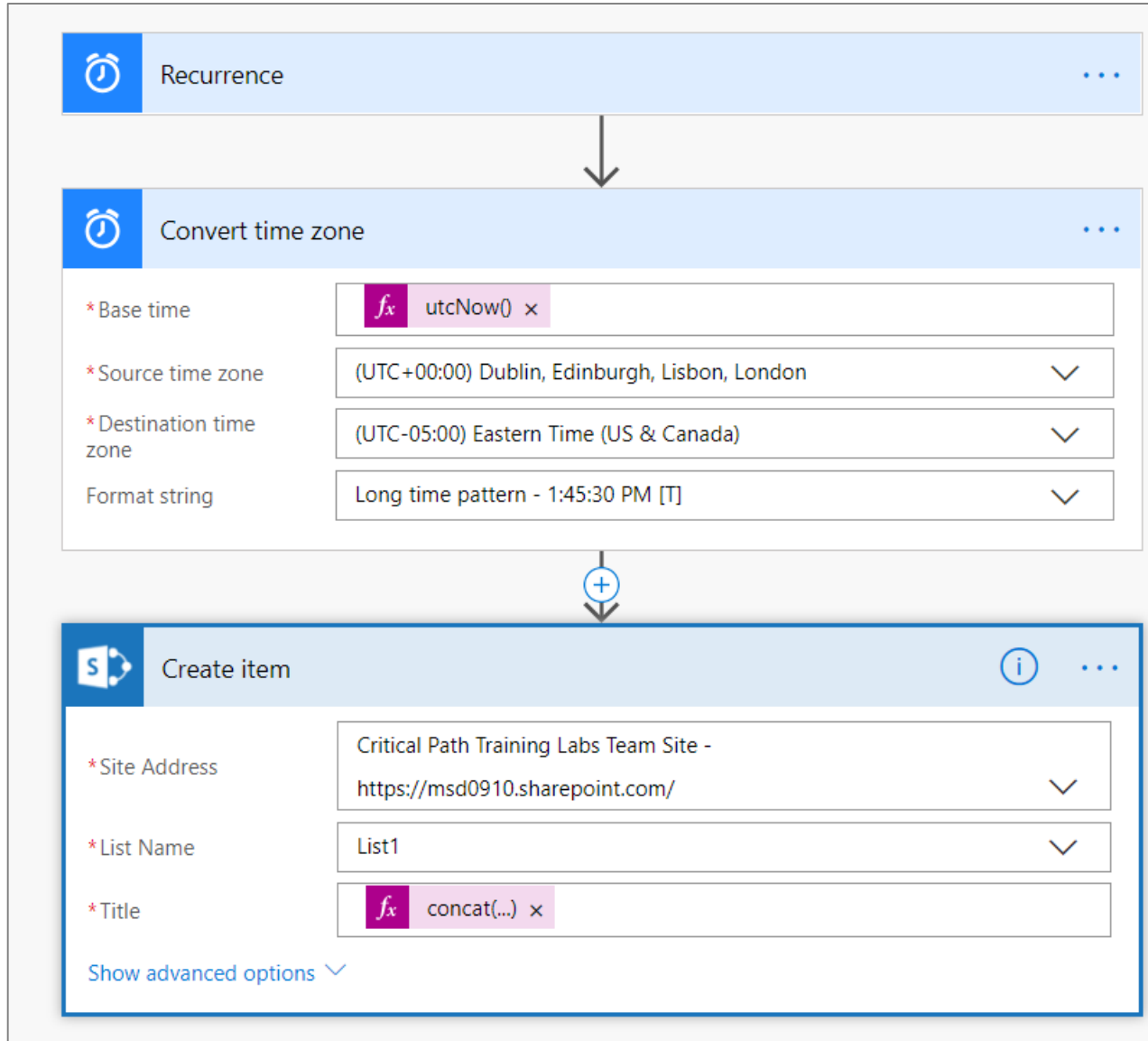
fx **dayOfMonth(timestamp)**
Returns the day of month component of a string timestamp

fx **dayOfYear(timestamp)**
Returns the day of year component of a string timestamp

fx **ticks(timestamp)**



Using Convert time zone



Agenda

- ✓ Flow Fundamentals
- ✓ Writing Flow Expressions
- ✓ Converting Between Types
- Working with Arrays
- Advanced Techniques



Working with Arrays in Flow

- Scenarios for working with arrays
 - Repeat action of each item in SharePoint list
 - Retrieve single item from a list
 - Filter list items

Jane	8.29	ID: 123 Name: Jane Mail: J@a.c
Dan	4.89	ID: 888 Name: Bob Mail: B@a.c
Steve	0.28	ID: 456 Name: Dave Mail: D@a.c



Using Apply to Each

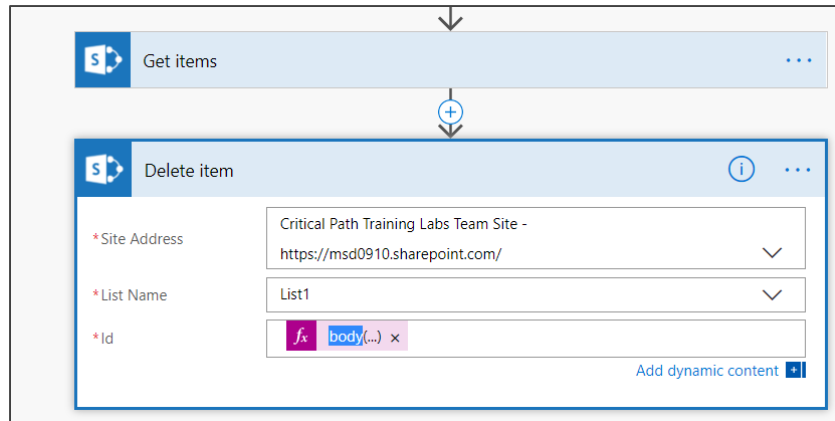
- Automatically added when list is used from output
- Destination step enumerates over list items

The screenshot displays a workflow editor interface. At the top, a 'Get items' step is configured with 'Site Address' set to 'https://msd0910.sharepoint.com/' and 'List Name' set to '0769e0e8-aec5-4441-8c88-6283a6799718'. Below this, an 'Apply to each' step is shown, which is automatically added when a list is used from an output. The 'Apply to each' step has a dropdown menu showing 'value x'. Below the 'Apply to each' step, a 'Delete item' step is configured with 'Site Address' set to 'Critical Path Training Labs Team Site - https://msd0910.sharepoint.com/', 'List Name' set to 'List1', and 'Id' set to 'ID x'. At the bottom of the editor, there are buttons for 'Add an action', 'Add a condition', and 'More'.

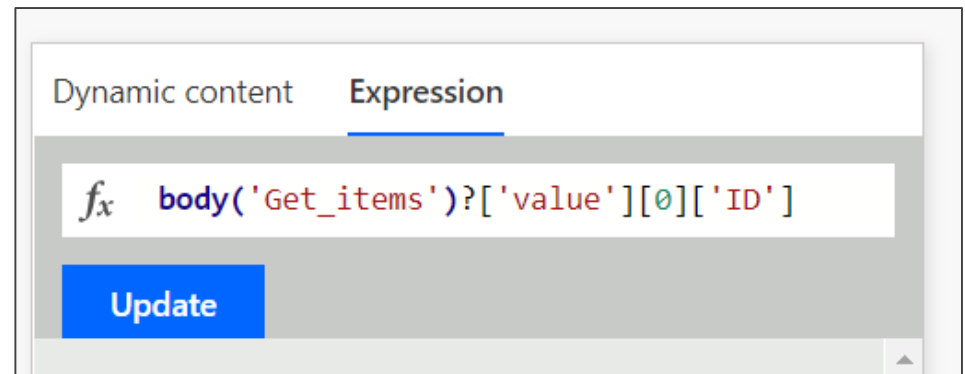


Retrieving List Items

- Use **first()** and **last()** to get lead at head or tail
- Individual items retrieved using zero-based array syntax
 - String array - `body('Get_items')?[0]`
 - SharePoint list - `body('Get_items')?['value'][0]['ID']`



The screenshot shows a Power Automate flow. The first action is 'Get items' (SharePoint connector). An arrow points down to the second action, 'Delete item' (SharePoint connector). The 'Delete item' action has three input fields: '* Site Address' with the value 'Critical Path Training Labs Team Site - https://msd0910.sharepoint.com/', '* List Name' with the value 'List1', and '* Id' with a dynamic content expression `body('...')?['ID']`. There is an 'Add dynamic content' button at the bottom right of the 'Delete item' action.



The screenshot shows the 'Dynamic content' pane in Power Automate. The 'Expression' tab is selected. The expression entered is `body('Get_items')?['value'][0]['ID']`. Below the expression is a blue 'Update' button.



Convert a List to a String

- Join
 - - Use join for simple tasks such as combining email
 - Use must work with string array and not an object array
- Create HTML/CSV Table
 - Convert list of objects into tabular display format

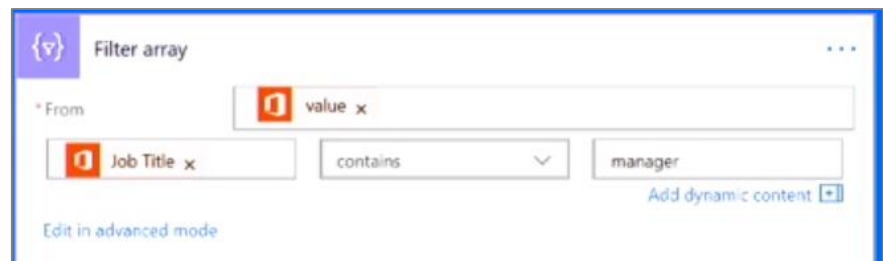


Filtering

- Best to have connect perform filtering
 - Often requires use of OData query string parameters
- There is also a built-in Flow action named **Filter array**
 - Select the Array in the From field
 - Can use simple or advanced mode just like in condition
 - If you need the first N items, use **take()** and **skip()**



The screenshot shows the 'Get items' action configuration in Microsoft Flow. It includes fields for 'Site Address' (Process Simple Partners - https://microsoft.sharepoint.com/teams/prosipart), 'List Name' (TechReadyDemoList), 'Filter Query' (Title eq 'string'), 'Order By' (An ODATA orderBy query for specifying the order of entries), and 'Top Count' (Total number of entries to retrieve (default = all)).

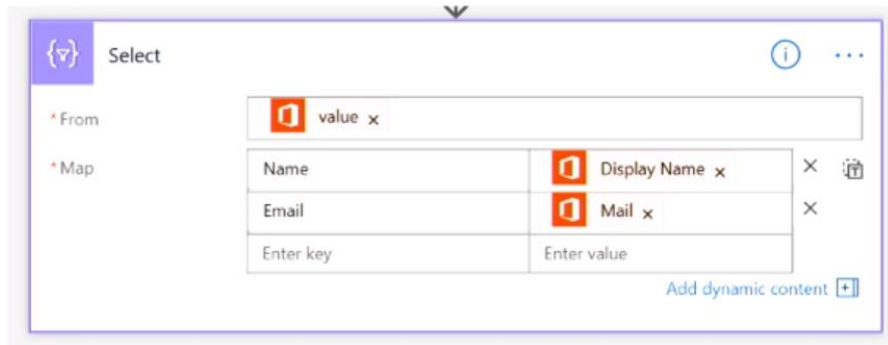


The screenshot shows the 'Filter array' action configuration in Microsoft Flow. It includes a 'From' field with a 'value x' placeholder, a 'Job Title x' field, a 'contains' operator, and a 'manager' field. There are also links for 'Edit in advanced mode' and 'Add dynamic content'.



Transforming Arrays

- Use Select action
 - Two input modes: fill key-value pairs or typing directly
- Create array object objects
 - Useful for passing array to another action

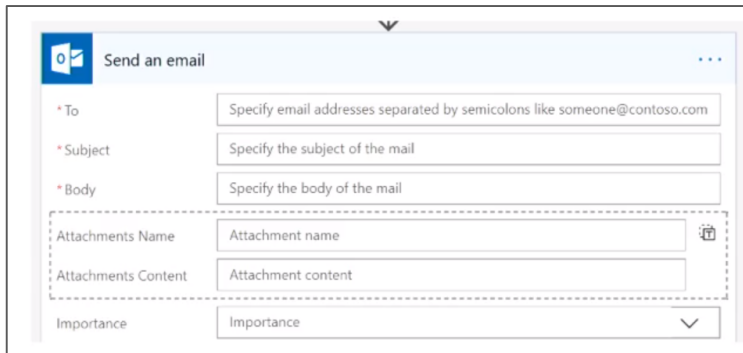


- Create a simple array of strings, numbers, Booleans, etc
 - Useful for creating simple list (e.g. email addresses)



Passing Data into Arrays

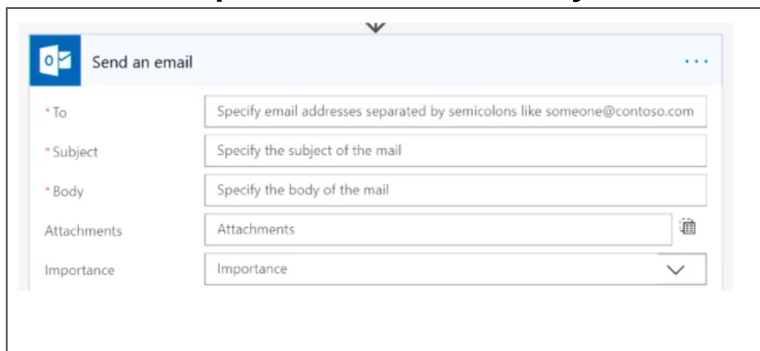
- Two options to pass data to an array
 - Hardcode list into an action
 - Create list dynamically at runtime
- Fill in each property of object and get one item created



The screenshot shows a 'Send an email' form with the following fields:

- To:** Specify email addresses separated by semicolons like someone@contoso.com
- Subject:** Specify the subject of the mail
- Body:** Specify the body of the mail
- Attachments Name:** Attachment name
- Attachments Content:** Attachment content
- Importance:** Importance (dropdown menu)

- Need to pass in an Array that exactly matches the fields



The screenshot shows a 'Send an email' form with the following fields:

- To:** Specify email addresses separated by semicolons like someone@contoso.com
- Subject:** Specify the subject of the mail
- Body:** Specify the body of the mail
- Attachments:** Attachments
- Importance:** Importance (dropdown menu)



Checking List Containment or Duplicates

- You can use `contains()` to check if something is in array
- You can use `union()` with array twice to get unique items



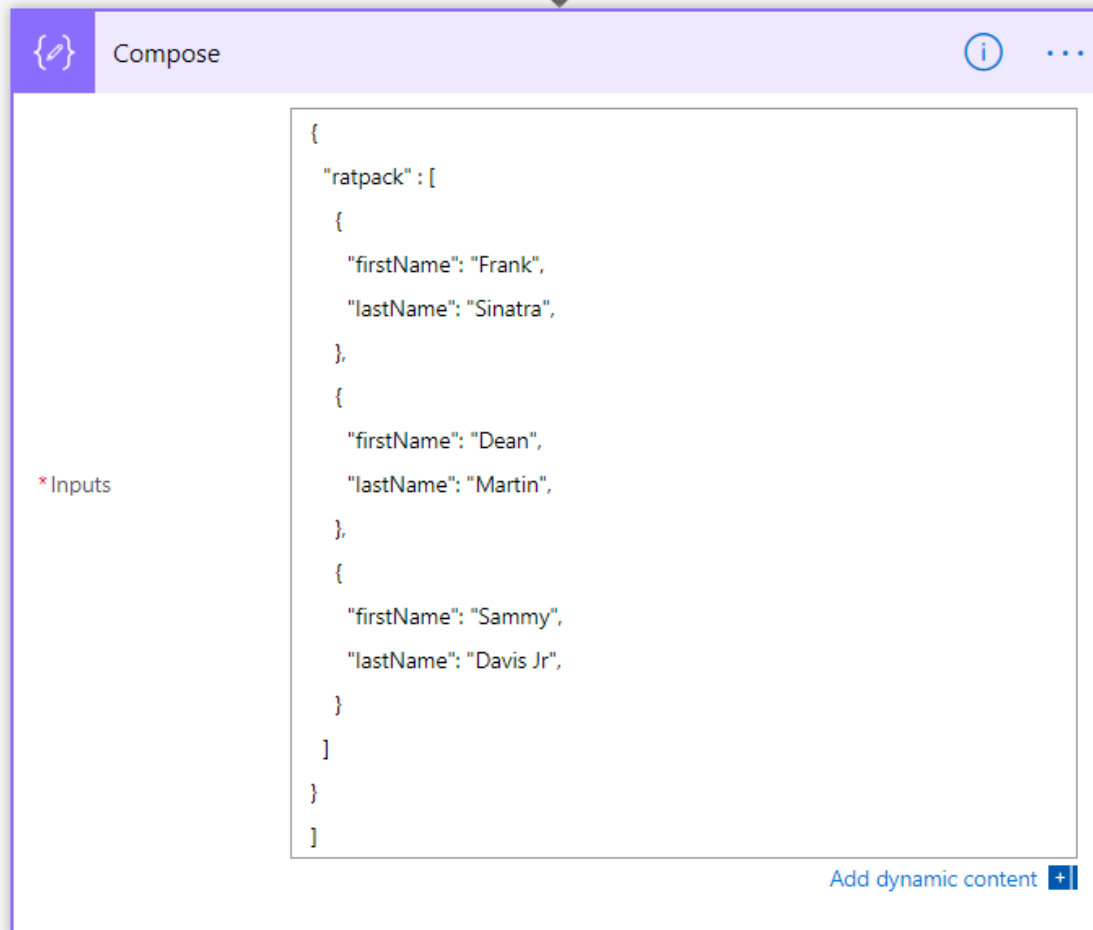
Agenda

- ✓ Flow Fundamentals
- ✓ Writing Flow Expressions
- ✓ Converting Between Types
- ✓ Working with Arrays
- Advanced Techniques



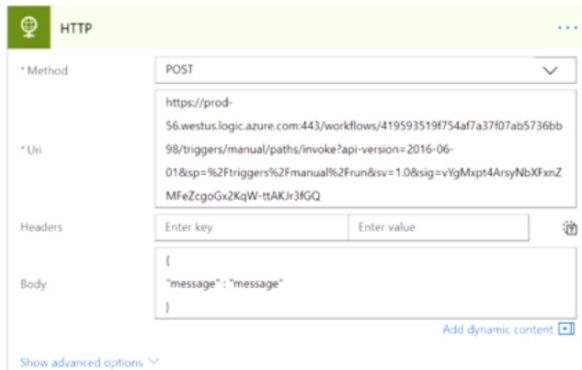
Building object directly

- You can use **Compose** action to create object or array



Calling Flows using the HTTP action

In the parent workflow:



The screenshot shows the configuration for an HTTP action in a parent workflow. The action is named "HTTP". The method is set to "POST". The URL is a long Azure Logic App endpoint. The body is a JSON object with a "message" property.

Method: POST

Uri: `https://prod-56.westus.logic.azure.com:443/workflows/419593519f754af7a37f07ab5736bb98/triggers/manual/paths/invoke?api-version=2016-06-01&sp=%2Ftriggers%2Fmanual%2Frun&sv=1.0&sig=vYgMxpt4ArsyNbXfnZMFeZcgoGx2KqW-ttAKJr3KGQ`

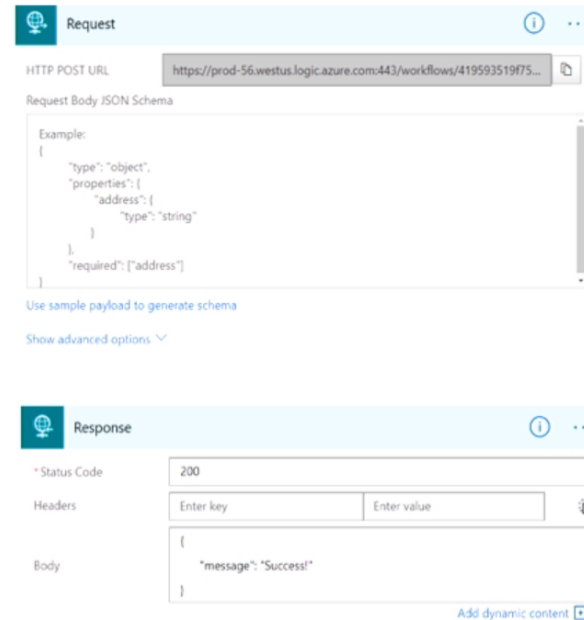
Headers: Enter key | Enter value

Body: `{ "message": "message" }`

[Add dynamic content](#)

[Show advanced options](#)

In the child workflow:



The screenshot shows the configuration for a Request and Response action in a child workflow. The action is named "Request". The HTTP POST URL is the same Azure Logic App endpoint. The Request Body JSON Schema is defined with an example object. The Response configuration shows a status code of 200 and a body with a "message" property.

Request

HTTP POST URL: `https://prod-56.westus.logic.azure.com:443/workflows/419593519f75...`

Request Body JSON Schema

Example:

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

[Use sample payload to generate schema](#)

[Show advanced options](#)

Response

Status Code: 200

Headers: Enter key | Enter value

Body: `{ "message": "Success!" }`

[Add dynamic content](#)



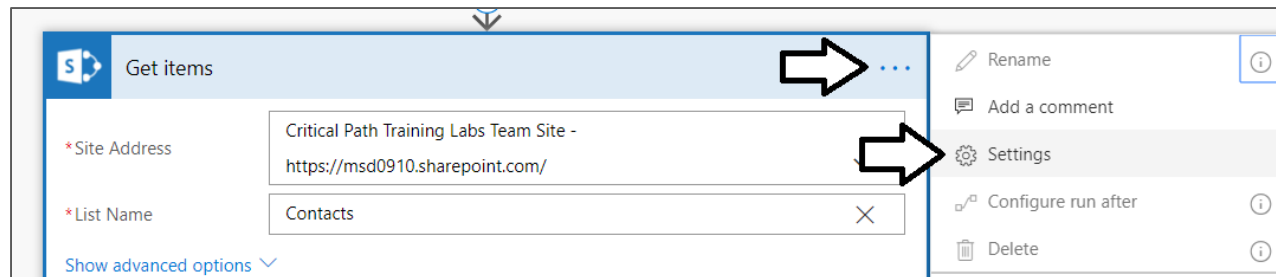
Normal action execution

- Standard behavior of a flow
 - Action steps execute in sequential order
 - Flow terminates if error occurs (failure or timeout)
- After flow runs, every action is in 1 of 4 possible states



Action Settings


- Settings let you configure
 - Async Actions
 - Timeouts
 - Retry Policy
 - Sequential Behavior
 - And more!




Error Handling

- Select the **Run after** option from action menu
 - Choose which error conditions, the arrow will turn dotted red
 - Use parallels for errors that are not at end of flow
 - Retry policy by default handles transient failures
 - Recommended to select exponential as they last a long time

'Handle duplicate file names' should run after:

 This step will only run if the flow couldn't create the file because there's already another one with the same name. It will add some numbers to the end of the file name to make it unique.

 **Create file**
Failed

☐ is successful
☒ has failed
☐ is skipped
☐ has timed out

Done **Cancel**

Retry Policy

A retry policy applies to intermittent failures, characterized as HTTP status codes 408, 429, and 5xx, in addition to any connectivity exceptions. The default retry policy is to retry 4 times with a 20 second delay between each attempt.

Retry policy type:

Interval ⓘ:

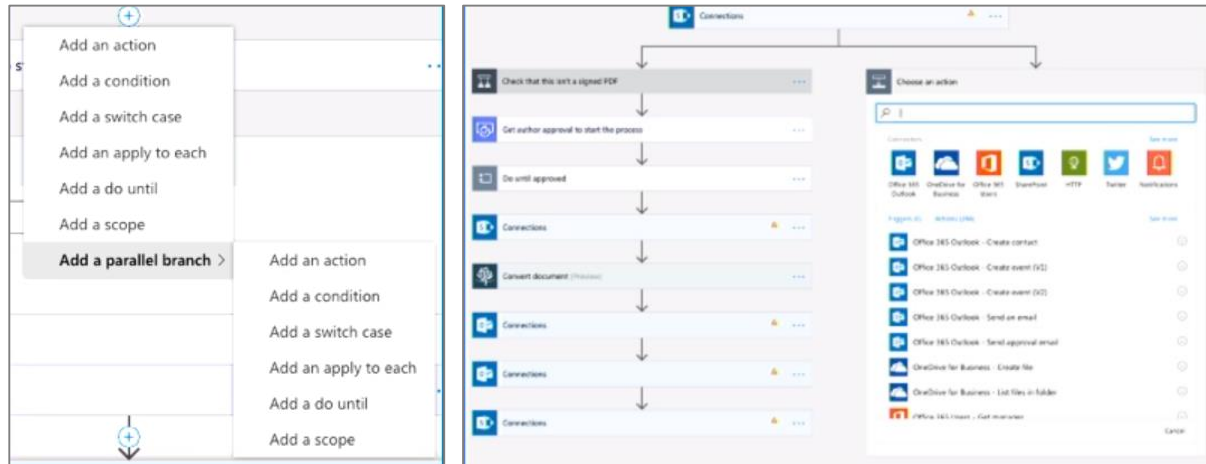
Count:

Done **Cancel**

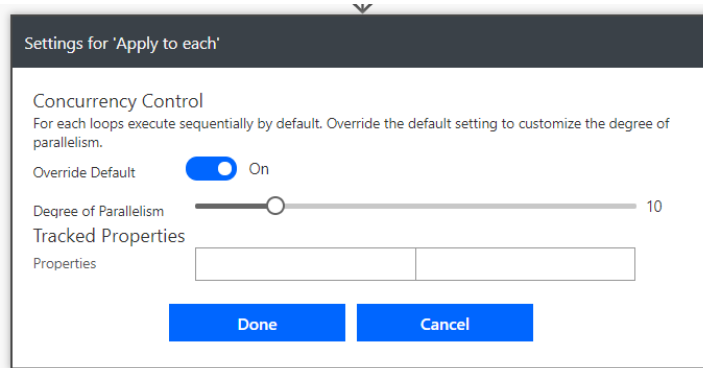


Parallel Execution

- Add parallel branch from above using ⊕

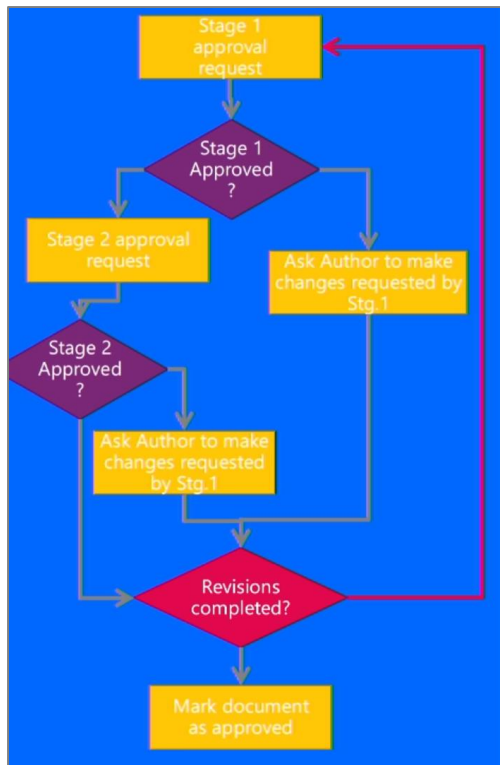


- Apply to each is sequential by default
 - Adding parallel execute to Apply to each



Looping

- Do until actions
 - Provides loop until a condition becomes true
 - Useful when implementing approval process



Variables

- Using variables
 - Always define the variable type and initialize it
 - Used for counters and Boolean conditions
 - Useful to append content in to strings & arrays

The screenshot displays a workflow editor with three main steps:

- Get items:** The first step in the workflow.
- Initialize variable:** A step where a variable is defined.
 - Name:** EmailAddress
 - Type:** String
 - Value:** Enter initial value
- Apply to each:** A loop step that applies an action to each item from the previous step.
 - Select an output from previous steps:** value x
 - Append to string variable:** An action that appends content to the variable.
 - Name:** EmailAddress
 - Value:** concat(...)

On the right side, a **Dynamic content** pane is open, showing the **Expression** tab. It displays the following expression:

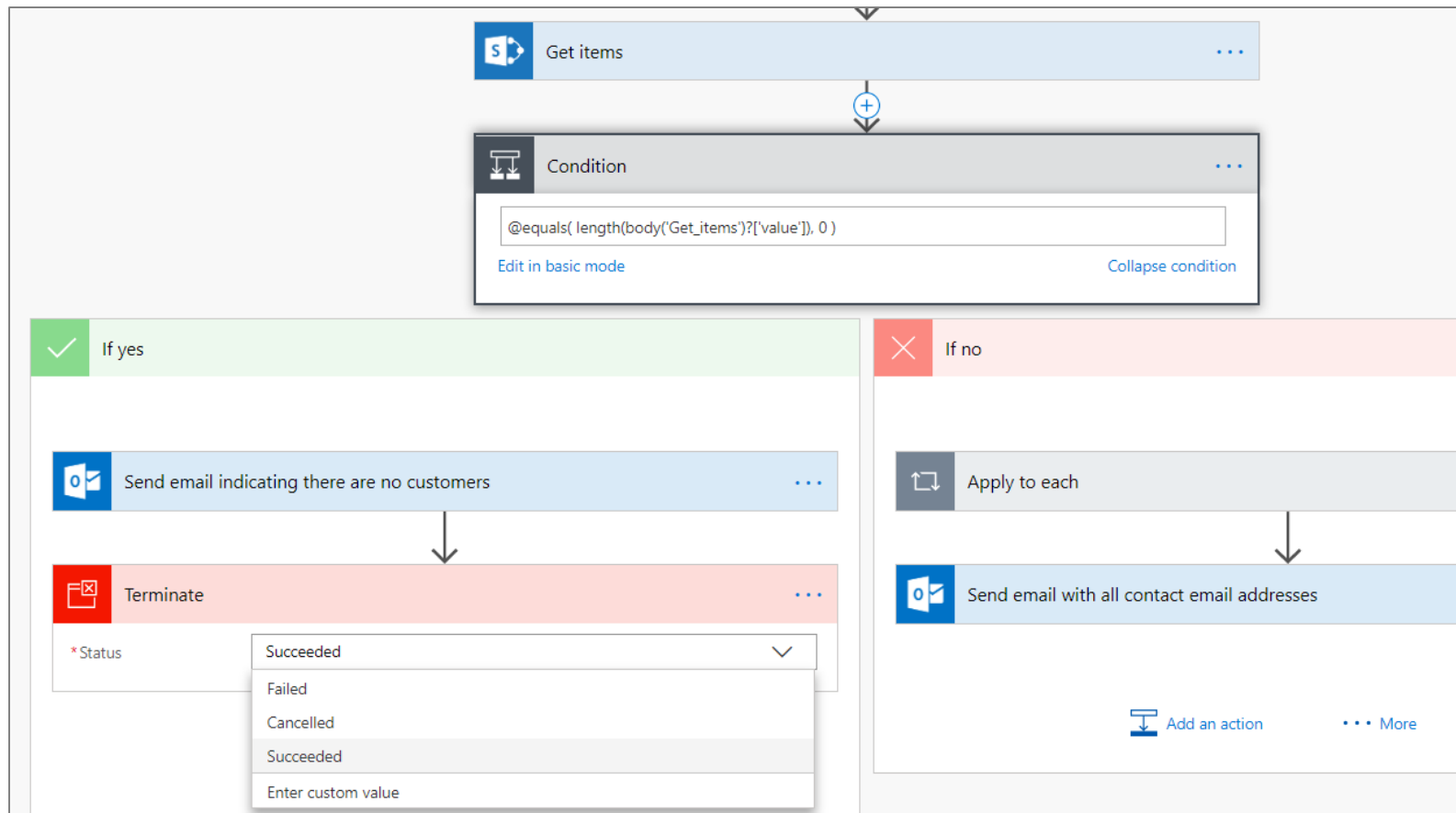
```
fx concat( item()?['Email'], ';' )
```

Below the expression, there is an **Update** button. At the bottom, a list of **String functions** is visible, including `concat(text_1, text_2?, ...)`.



Terminate action

- Used to stop a flow at any point
 - Terminate status can be set to Succeeded, Cancelled, Failed



Summary

- ✓ Flow Fundamentals
- ✓ Writing Flow Expressions
- ✓ Converting Between Types
- ✓ Working with Arrays
- ✓ Advanced Techniques



Deep Dive into PowerApps and Flow

- Two action-packed days of building PowerApps and Flows
 1. Getting Started with PowerApps Studio
 2. Designing PowerApps using Advanced Techniques
 3. Building PowerApps for SharePoint Online
 4. Introduction to Microsoft Flow
 5. Designing Flows to Automate an Approval Process
 6. Building PowerApps and Flows for Power BI
 7. Working with the Common Data Service for Apps
 8. Managing Application Lifecycle with PowerApps and Flow
- More info
 - <https://www.criticalpathtraining.com>
 - info@criticalpathtraining.com

