

Writing Advanced Formulas in Microsoft Flow

Ted Pattison
Power platform MVP



PHOENIX, AZ | OCTOBER 15-18, 2018



GitHub Repo

- All slides and demo files available for download
- <https://github.com/CriticalPathTraining/AdvancedFlowExpressions>

The screenshot shows the GitHub repository page for `CriticalPathTraining / AdvancedFlowExpressions`. The repository has 1 watch, 0 stars, and 0 forks. The main content area shows the repository description: "Demos and slides for the Advanced Flow Expressions talk". Below this, it indicates 2 commits, 1 branch, 0 releases, and 1 contributor. The repository is on the `master` branch, and there is a "New pull request" button. The file list shows the following files and their last update times:

File	Update Type	Time Ago
AppPackages	Updates	3 minutes ago
.gitattributes	Updates	3 minutes ago
Advanced Flow Expressions.pdf	updates	2 minutes ago
Advanced Flow Expressions.pptx	Updates	3 minutes ago
FlowExpressionSnippets.txt	Updates	3 minutes ago
RequestUrl.txt	Updates	3 minutes ago
WebApiSchema.txt	Updates	3 minutes ago
contactJSON.txt	Updates	3 minutes ago
jsonObject.txt	Updates	3 minutes ago

Agenda

- Flow Fundamentals
- Writing Flow Expressions
- Control of Flow
- Processing Data and Preparing Content
- Converting Between Types
- Advanced Techniques

Thanks for coming up with the idea for this session



Stephen Siciliano, Principal Group PM Manager



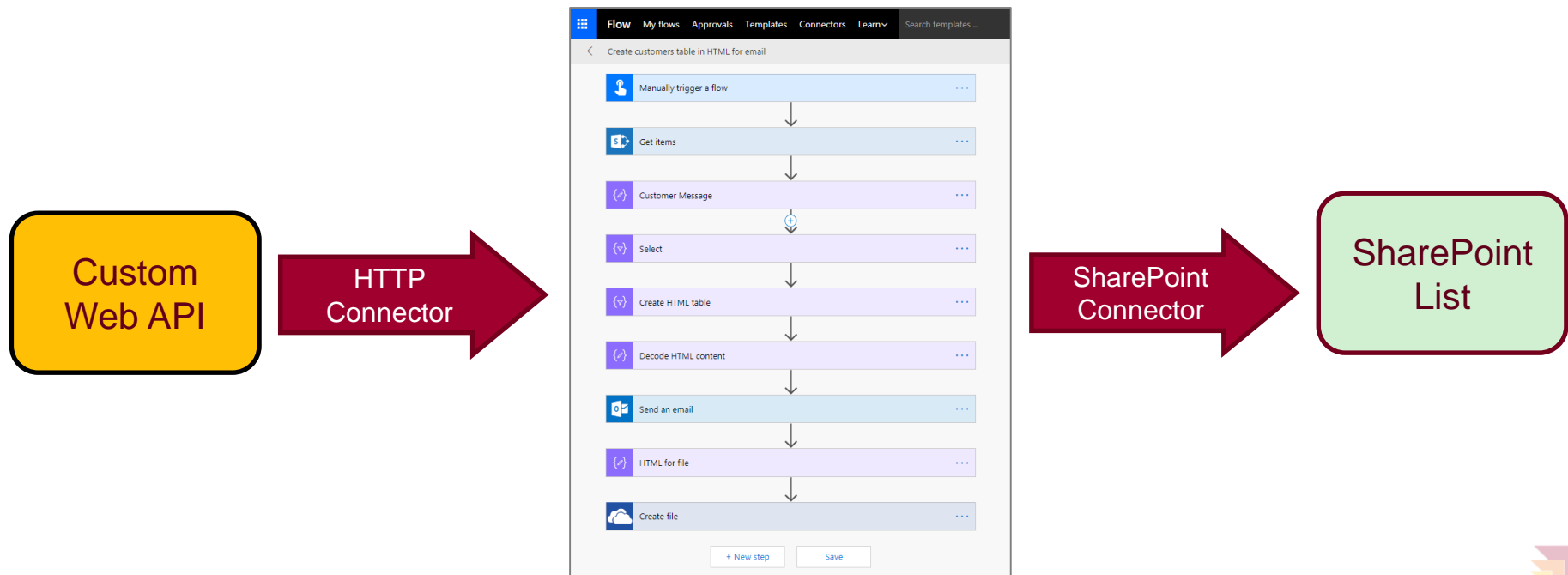
Deep Dive into PowerApps and Flow

- 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



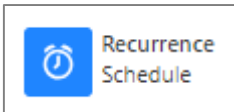
Building Blocks of Flow

- **Triggers** - events that start a flow
- **Actions** - tasks and operation executed by flow
- **Services** - sources and destinations for data
- **Connectors** - wrappers to communicate with service APIs

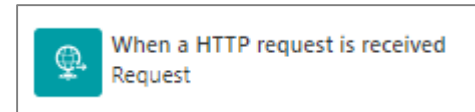
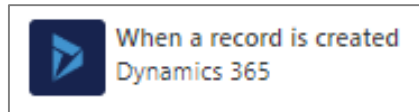
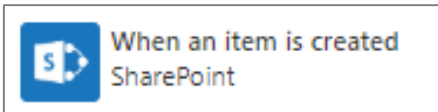


Flow Trigger Types

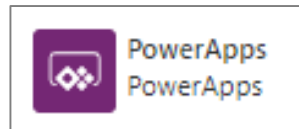
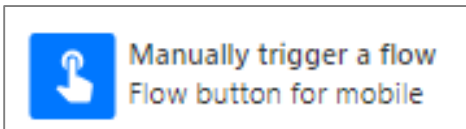
- Scheduled Flow Triggers
 - Runs periodically based on an interval



- Automated Flow Triggers
 - Runs when something happens

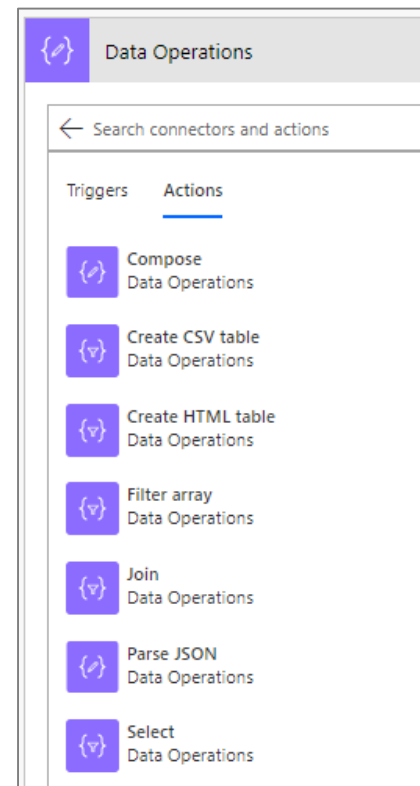
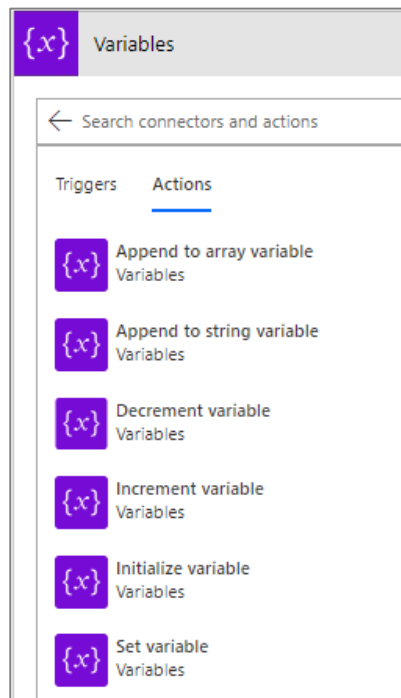
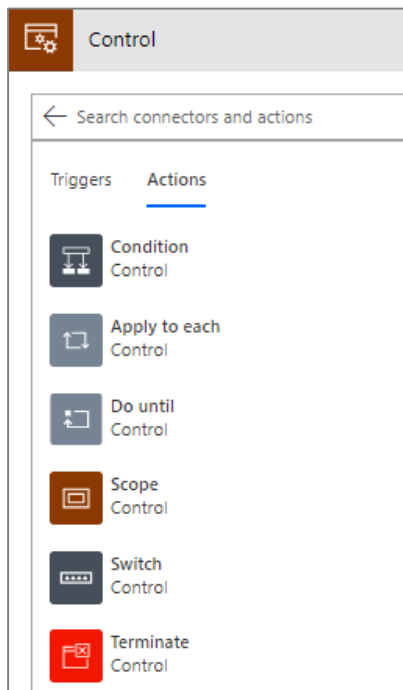


- On-demand Flow Triggers
 - Runs when a user clicks a button



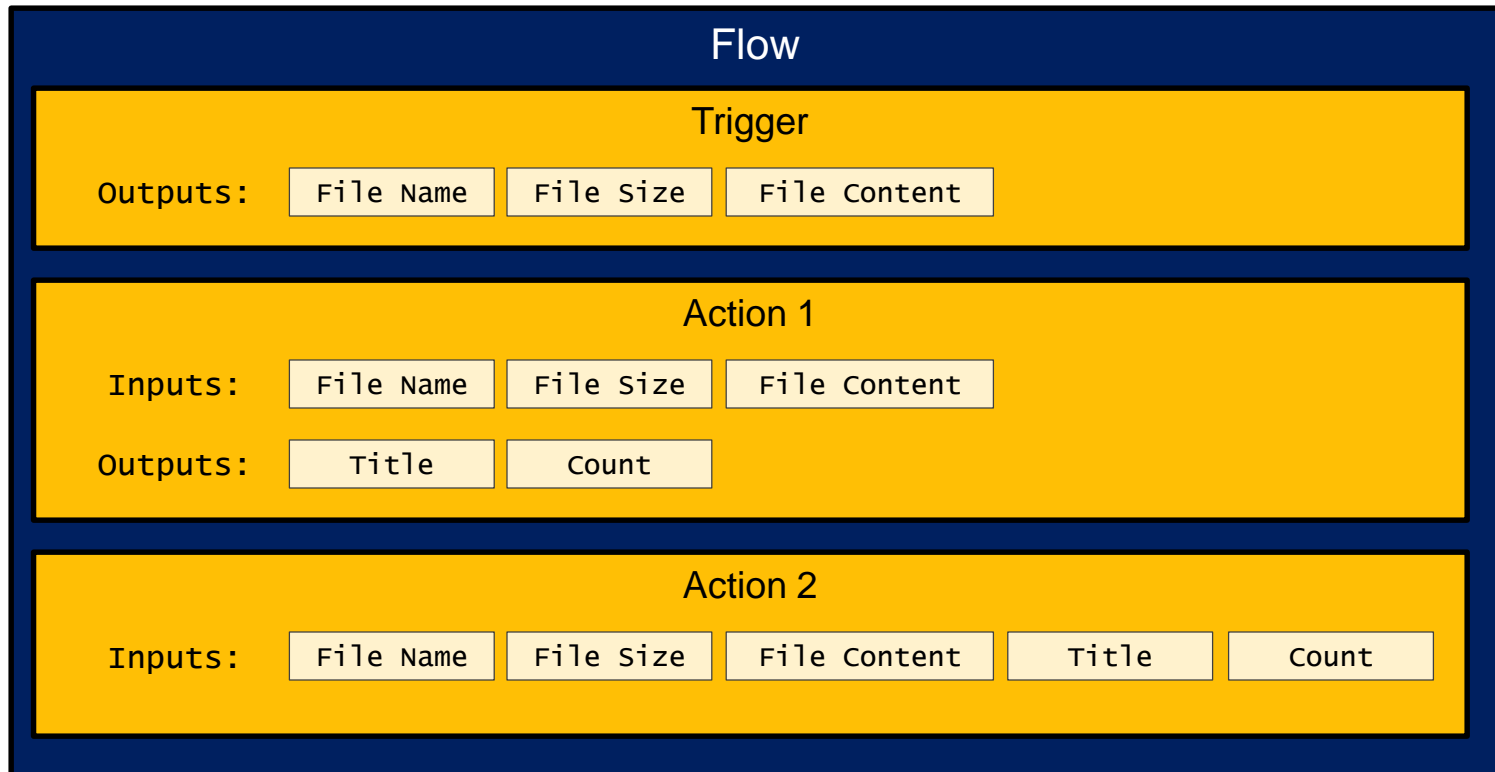
Core Action Categories

- **Control:** actions to provide control-of-flow
- **Variables:** actions to manage state within flow lifetime
- **Data operations:** action to process data & prepare content



Data Automatically Flows from Step to Step

- Data in flows added by step outputs
 - Data added in step output is available in later steps
 - It's easy to configure step input data using output data in previous steps
 - Certain outputs displayed/hidden based on types of input and output





DEMO

Demo 1

Changing the Flow Trigger Type

Agenda

- ✓ Flow Fundamentals
- Writing Flow Expressions
 - Control of Flow
 - Processing Data and Preparing Content
 - Converting Between Types
 - Advanced Techniques



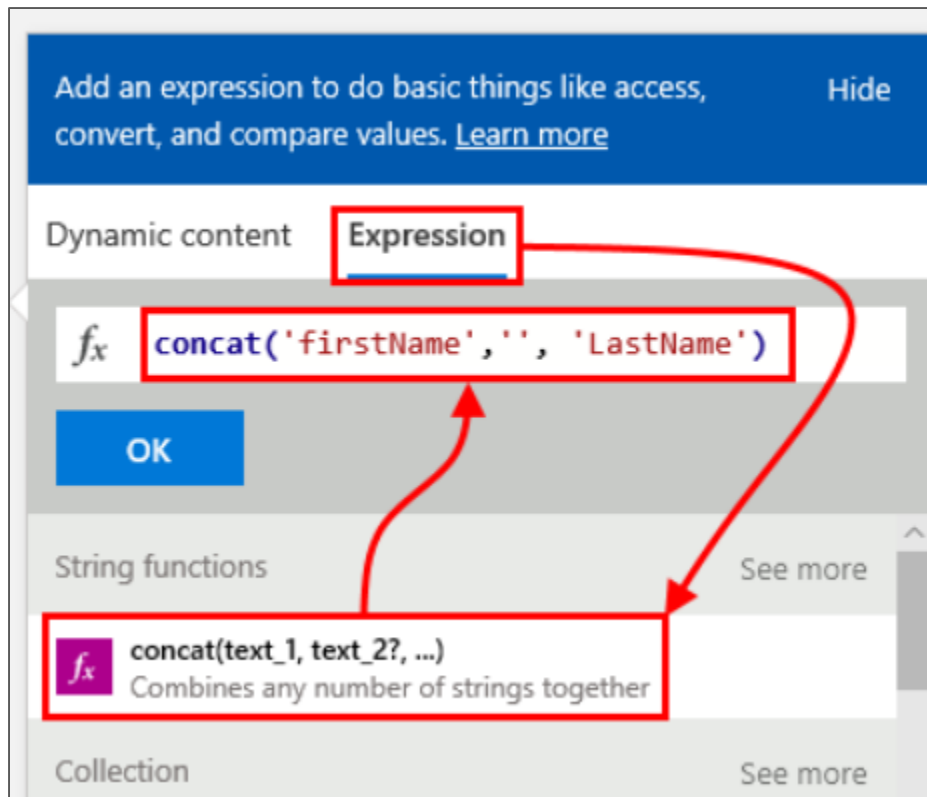
Writing Flow Expressions

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



Writing Expressions

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



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




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
 - No support for 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...





DEMO

Demo 2

Adding SharePoint List Items

Agenda

- ✓ Flow Fundamentals
- ✓ Writing Flow Expressions
- Control of Flow
 - Processing Data and Preparing Content
 - Converting Between Types
 - Advanced Techniques



Understanding Arrays in Flow


- Flow arrays are zero-based
 - Primitive value arrays

0	Daugherty
1	Hernandez
2	Mack
3	Wiley

- Object arrays

	Last Name	First Name	Company	Business Phone	Home Phone
0	Daugherty	Cindy	Wonka Industries	1(337)111-4444	1(337)111-7777
1	Hernandez	Zane	Vandelay Industries	1(757)666-3333	1(757)777-1111
2	Mack	Chang	Wonka Industries	1(480)111-4444	1(480)777-0000
3	Wiley	Ramona	Ecumena	1(201)777-8888	1(201)777-2222

Accessing an Array using ['value']

 Get items ⓘ ⋮

*Site Address ✕

*List Name ✕

Show advanced options ▾



`body('Get_items')?['value']`

	Last Name	First Name	Company	Business Phone	Home Phone
0	Daugherty	Cindy	Wonka Industries	1(337)111-4444	1(337)111-7777
1	Hernandez	Zane	Vandelay Industries	1(757)666-3333	1(757)777-1111
2	Mack	Chang	Wonka Industries	1(480)111-4444	1(480)777-0000
3	Wiley	Ramona	Ecumena	1(201)777-8888	1(201)777-2222



Retrieving List Items

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

The screenshot shows a Power Automate flow with two actions. The first action is 'Get items' (indicated by a downward arrow). The second action is 'Delete item' (indicated by a plus sign in a circle). The 'Delete item' action has the following configuration:

- *Site Address: Critical Path Training Labs Team Site - <https://msd0910.sharepoint.com/>
- *List Name: List1
- *Id: `body('Get_items')?['value'][0]['ID']` (with a dynamic content icon and a close button)

An 'Add dynamic content' button is visible at the bottom right of the 'Delete item' action card.

The screenshot shows the 'Dynamic content' pane with the 'Expression' tab selected. The expression entered is:

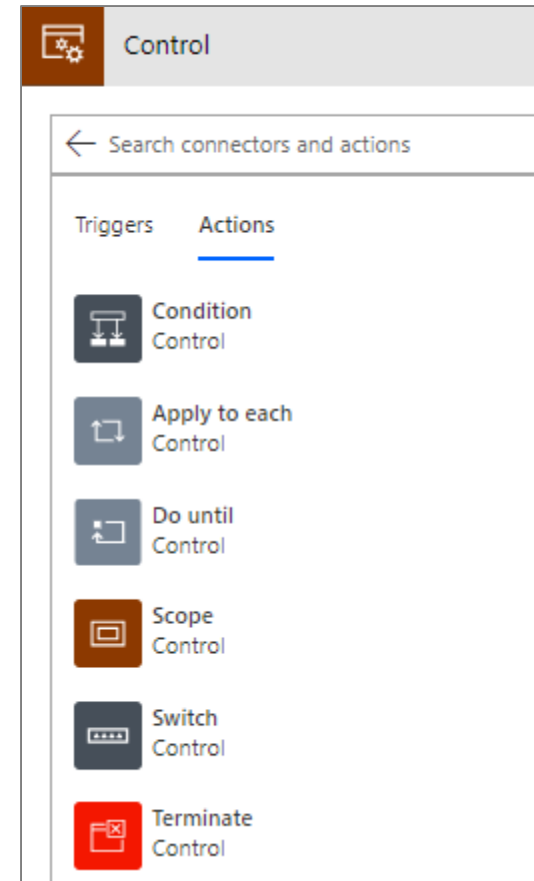
```
fx body('Get_items')?['value'][0]['ID']
```

An 'Update' button is located below the expression field.

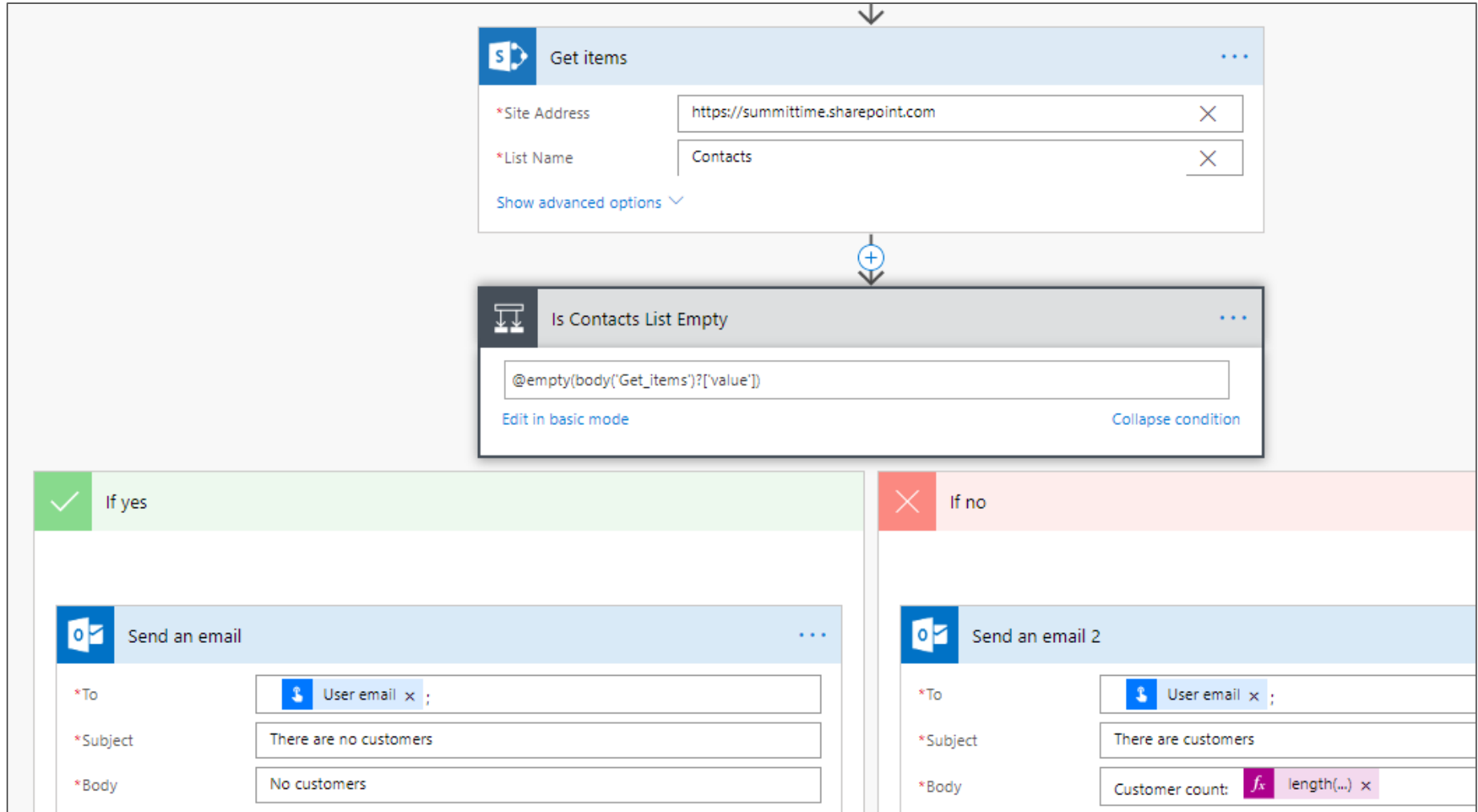


Control of Flow

- Condition
 - Provides logical structure for If Then Else
- Apply to each
 - Enumerate through collection (e.g. list items)
- Do until
 - Repeat until condition changes
- Scope
 - Create an action container with a private scope
- Switch
 - Select Case flow
- Terminate
 - Completes a flow

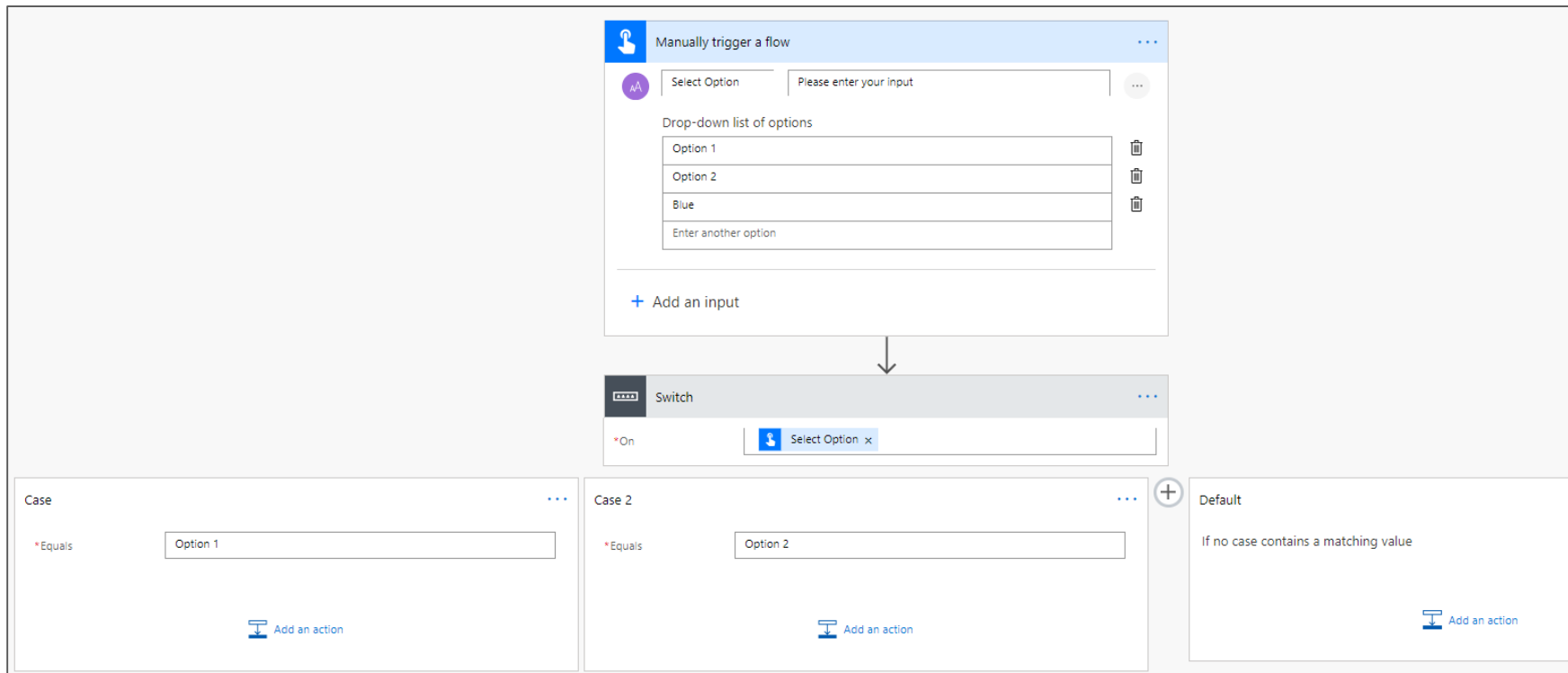


Condition Action



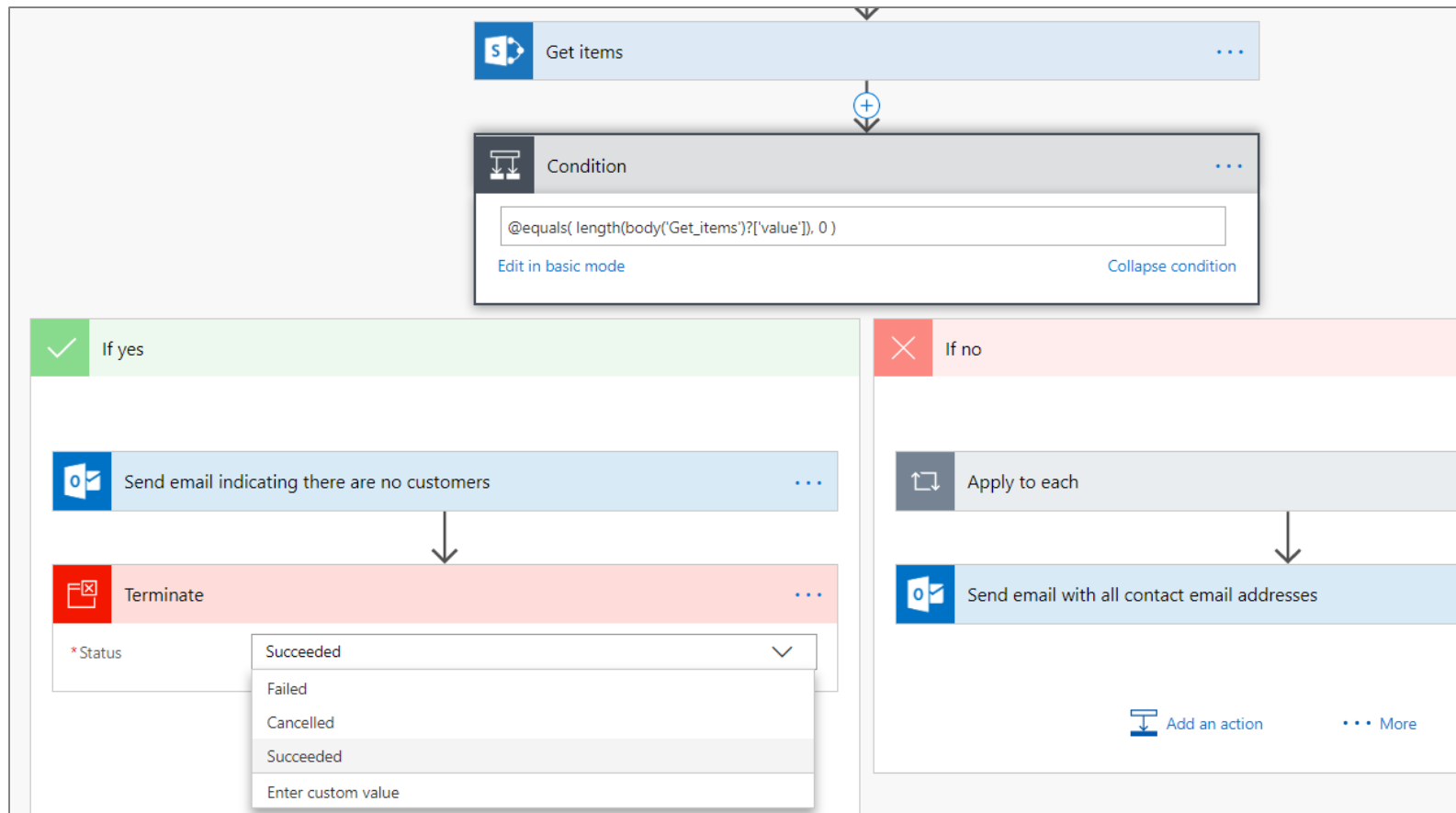
Switch Action

- Switch actions provides cases
 - Each case represents separate execution path
 - Only one execution path will execute
 - Default case executes when no other case is matched



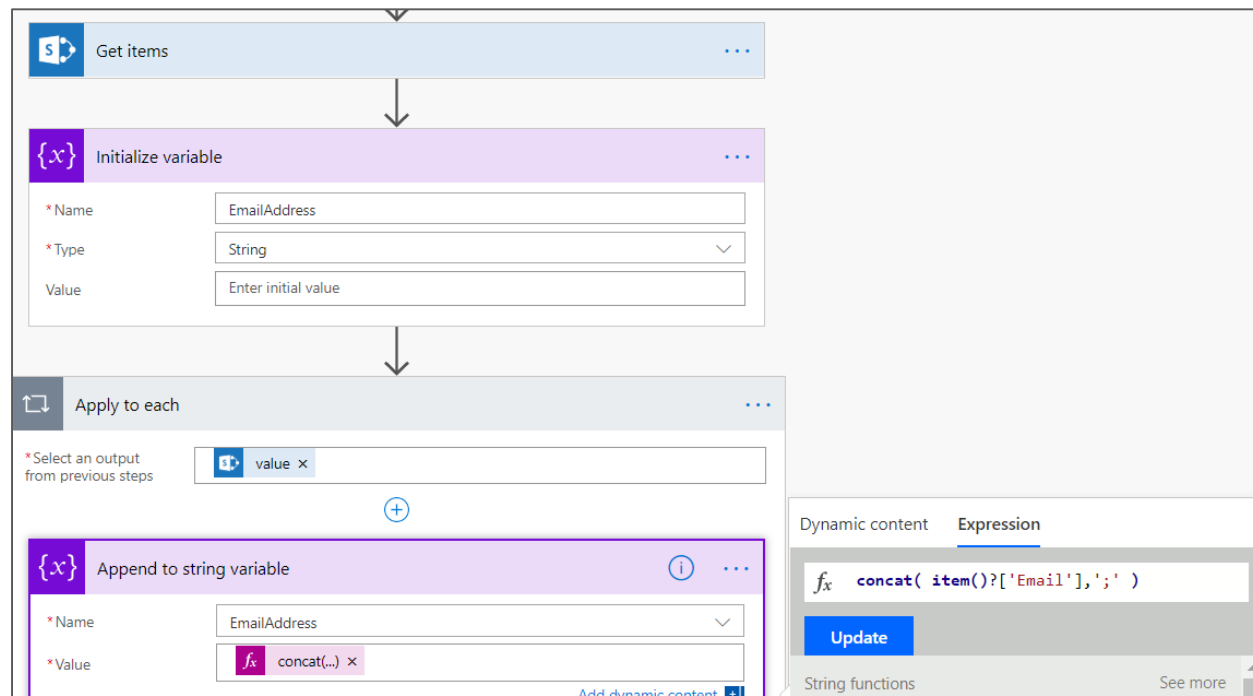
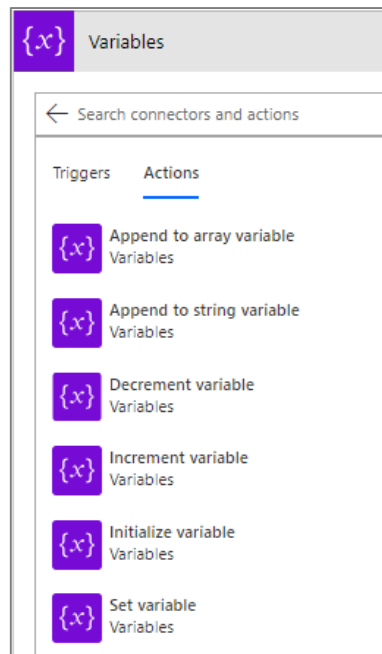
Terminate action

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

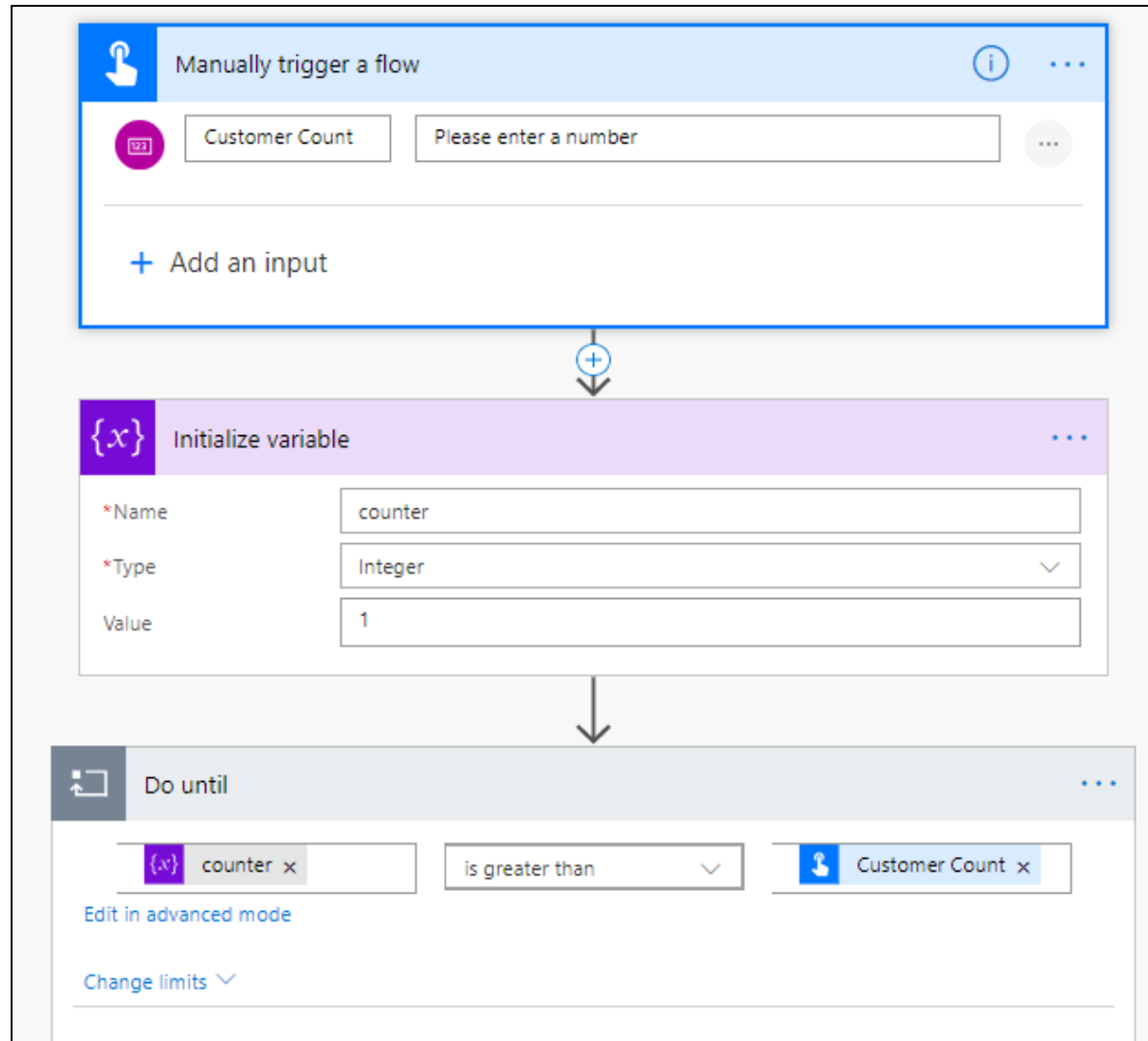


Tracking State using Variables

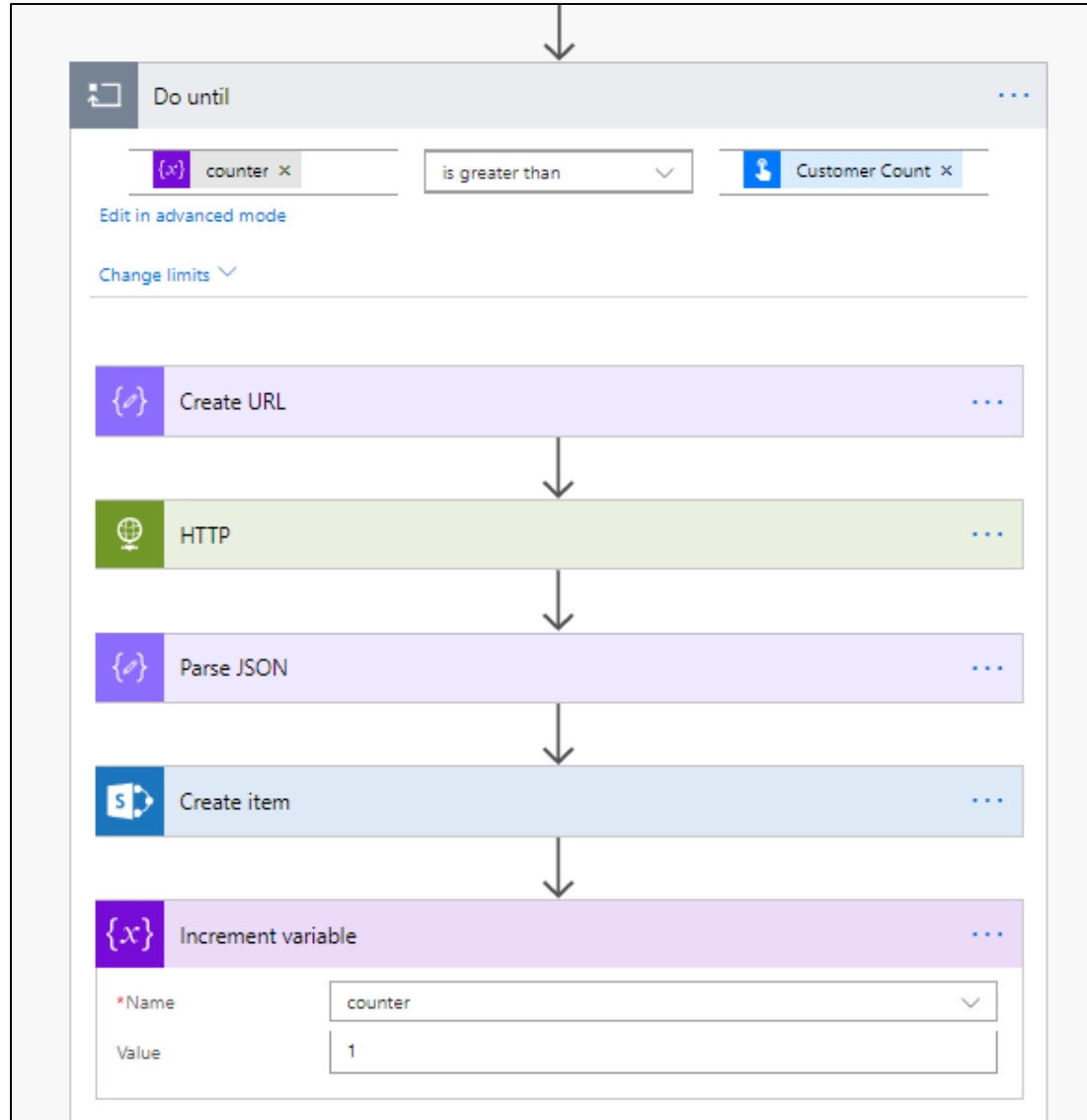
- Variables used to track state during flow lifetime
 - Initialize Variable used to create variable with Type and Value
 - Other variable actions uses to update variable values
 - By default, variable stored within flow until end of flow lifetime
 - Variables can be initialized inside Scope action to reduce lifetime



Do Until Action with Counter Variable



Executing Operations inside Do Until Loop



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. The first step, 'Get items', is highlighted with a blue border. It contains two dropdown menus: 'Site Address' with the value 'https://msd0910.sharepoint.com/' and 'List Name' with the value '0769e0e8-aec5-4441-8c88-6283a6799718'. Below these is a 'Show advanced options' link. A connector arrow points down to the 'Apply to each' step, which is highlighted with a grey border. This step has a dropdown menu for 'Select an output from previous steps' showing 'value'. Below this is another 'Delete item' step, also highlighted with a blue border. It contains three dropdown menus: 'Site Address' with 'Critical Path Training Labs Team Site - https://msd0910.sharepoint.com/', 'List Name' with 'List1', and 'Id' with 'ID'. At the bottom of the editor, there are three buttons: 'Add an action', 'Add a condition', and 'More'.

Get items

* Site Address

* List Name

Show advanced options

Apply to each

* Select an output from previous steps

Delete item

* Site Address

* List Name

* Id

Add an action Add a condition More





DEMO

Demo 3

Apply to Each and Do Until

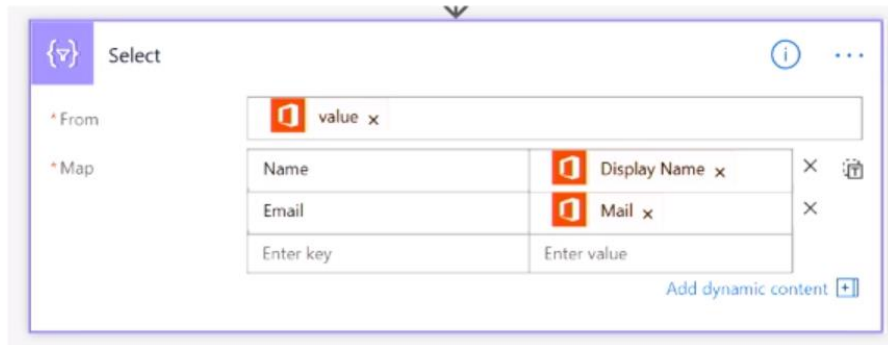
Agenda

- ✓ Flow Fundamentals
- ✓ Writing Flow Expressions
- ✓ Control of Flow
- Processing Data and Preparing Content
 - Converting Between Types
 - Advanced Techniques

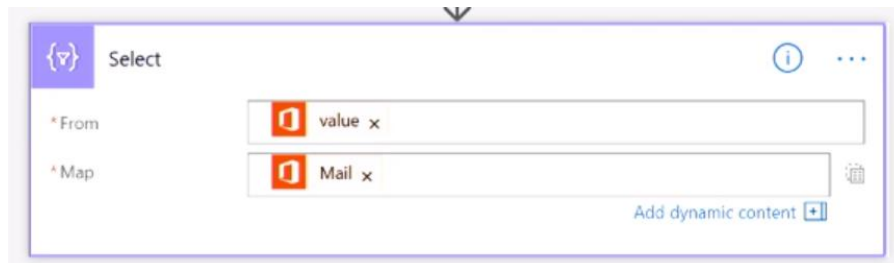


Transforming Arrays

- 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)



Converting an Array using Select

The screenshot displays a Power Automate flow with the following steps:

- Manually trigger a flow**: The starting trigger.
- Get items**: Retrieves data from a source.
- Email Array**: A variable of type Array. It contains:
 - *From**: A dynamic content field showing `value`.
 - *Map**: A dynamic content field showing `Email Address`.
- Get Parsed Email Addresses**: An action that uses the `join(...)` function to convert the array into a string.

The **Get Parsed Email Addresses** action is configured with the following inputs:

- *Inputs**: `join(...)`

The right-hand pane shows the **Expression** tab for the `join(...)` function, with the following code:

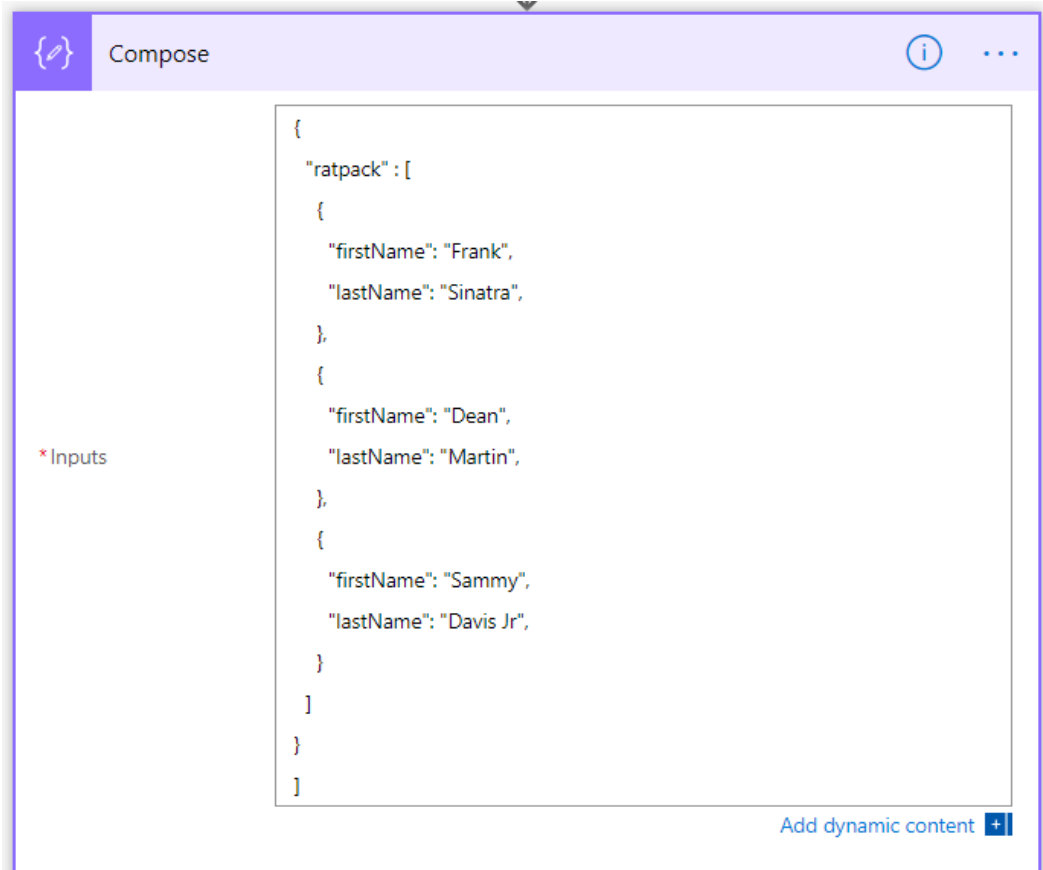
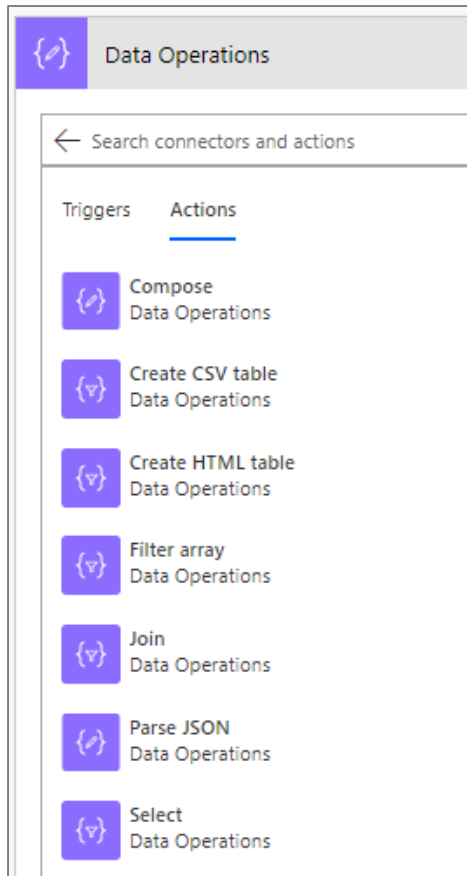
```
fx join(body('Email_Array'),';')
```

The **Update** button is visible at the bottom of the expression pane.



Data Operations

- Used to process data and prepare content





DEMO

Demo 4

Generating an HTML Table

Agenda

- ✓ Flow Fundamentals
- ✓ Writing Flow Expressions
- ✓ Control of Flow
- ✓ Processing Data and Preparing Content
- Converting Between Types
- 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



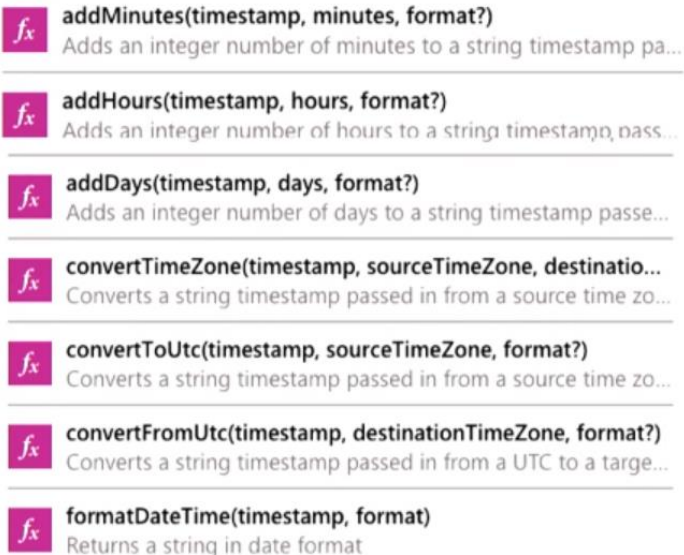
Flow Type Conversion Matrix

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

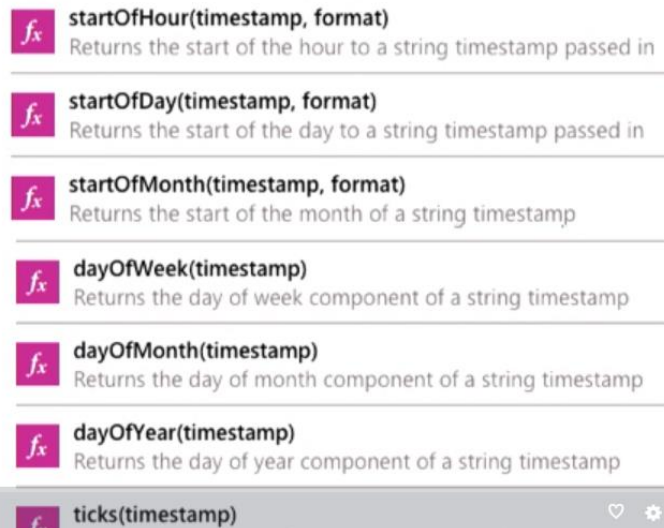
fx **addDays(timestamp, days, format?)**
Adds an integer number of days to a string timestamp passed in

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)**



dataUriToBinary()

- PowerApps photos require conversion
 - Allows you to upload photos to SharePoint
 - Accomplished using **dataUriToBinary()** function

```
dataUriToBinary(triggerBody()['Createfile_FileContent'])
```

The screenshot displays a PowerApps flow interface. The first step is 'Get File Name', which uses the 'concat(...)' function. The second step is 'Create file', which is configured with the following fields:

- Site Address:** Critical Path Training Labs Team Site - <https://msd0910.sharepoint.com/>
- Folder Path:** /My Photos
- File Name:** Output
- File Content:** dataUriToBinary(triggerBody()['Createfile_FileContent'])

On the right side, the 'Dynamic content' pane is open, showing the 'Expression' tab. It displays the function `dataUriToBinary(triggerBody()['Createfile_FileContent'])` and includes an 'Update' button. Below this, a list of 'String functions' is visible, including 'concat(text_1, text_2?, ...)' with a description: 'Combines any number of strings together.'



DEMO

Demo 5

Uploading Mobile Photos to SharePoint

Agenda

- ✓ Flow Fundamentals
- ✓ Writing Flow Expressions
- ✓ Control of Flow
- ✓ Processing Data and Preparing Content
- ✓ Converting Between Types
- Advanced Techniques



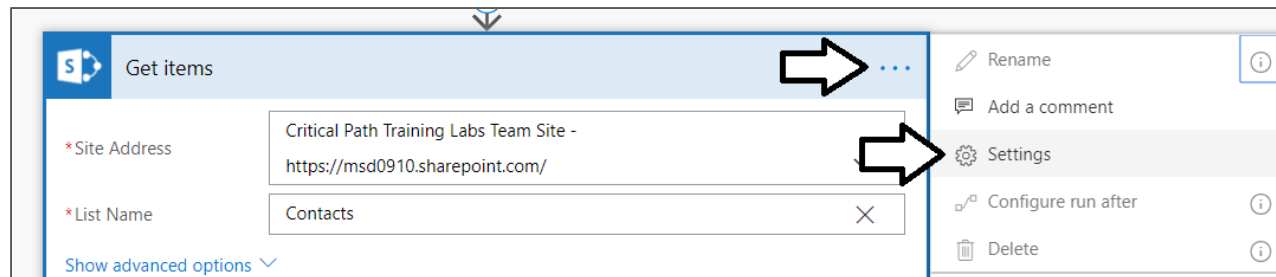
Normal action execution

- Standard behavior of a flow
 - Action steps execute in sequential order
 - Flow terminates if error occurs (failure or timeout)
- After flow runs, every action left 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 ⓘ:

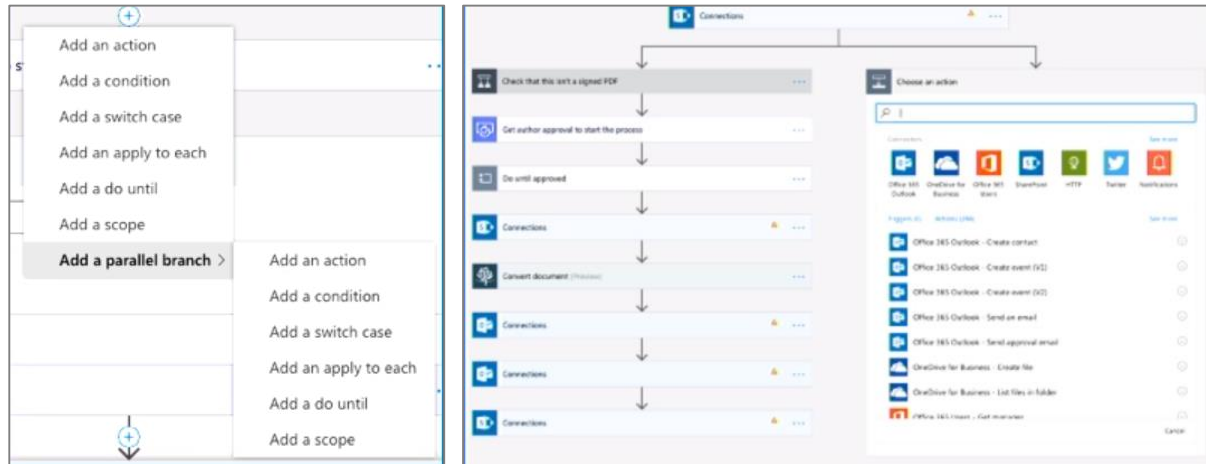
Count:

Done **Cancel**

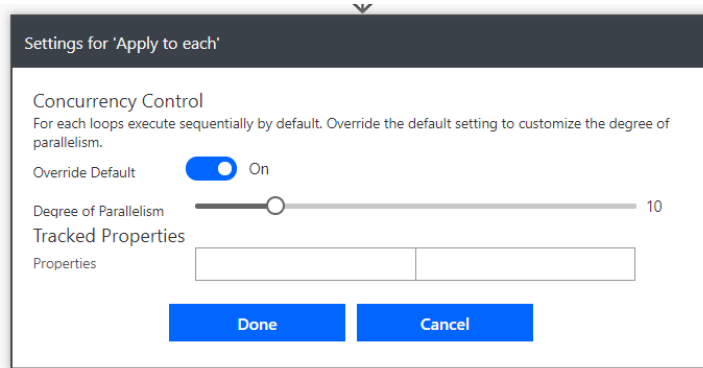


Parallel Execution

- Add parallel branch from above using ⊕

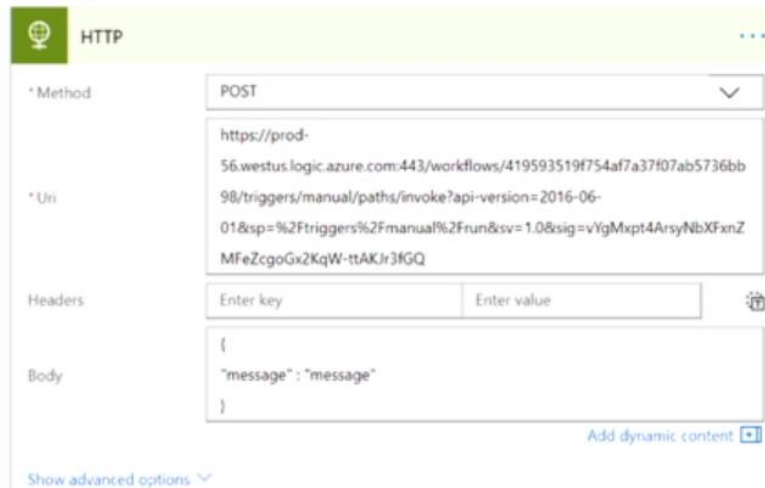


- Apply to each is sequential by default
 - Adding parallel execute to Apply to each



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" and is set to the POST method. The URI is a long URL starting with "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=vYgMxpt4ArsyNbXfanZMFeZcgoGx2KqW-ttAKJr3fGQ". The body is a JSON object with a "message" property. There are links for "Show advanced options" and "Add dynamic content".

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=vYgMxpt4ArsyNbXfanZMFeZcgoGx2KqW-ttAKJr3fGQ`

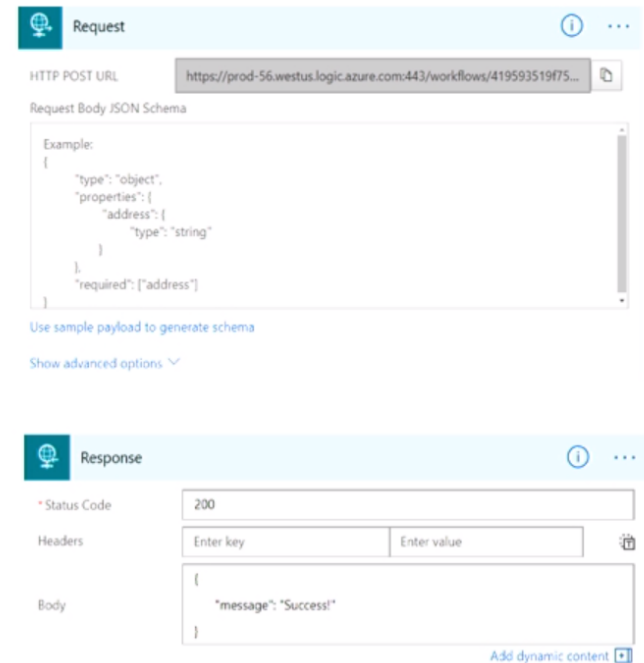
Headers: Enter key Enter value

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

Show advanced options Show advanced options Add dynamic content



In the child workflow:



The screenshot shows the configuration for an HTTP action in a child workflow. The action is named "Request" and is set to the POST method. The URI is a long URL starting with "https://prod-56.westus.logic.azure.com:443/workflows/419593519f75...". The body is a JSON object with a "message" property. There are links for "Show advanced options" and "Add dynamic content".

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



Summary

- ✓ Flow Fundamentals
- ✓ Writing Flow Expressions
- ✓ Control of Flow
- ✓ Processing Data and Preparing Content
- ✓ Converting Between Types
- ✓ Advanced Techniques

