WORKSHOP

All-In! A Deep-Dive On Model-Driven Power Apps

Business Logic For Model-Driven Apps

Part III

PPCC 2025

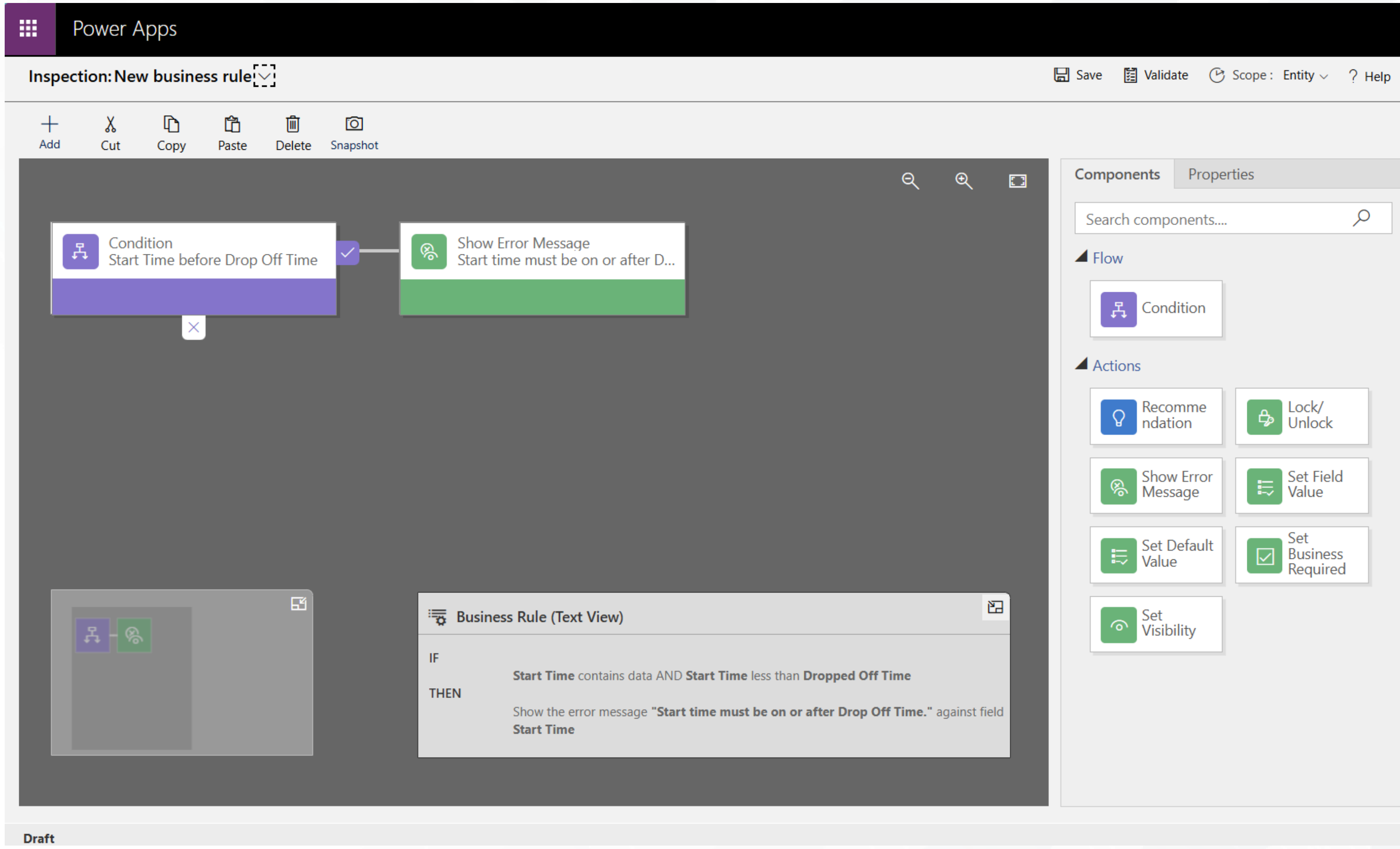
Why Business Logic Matters

- Ensure data integrity and consistency regardless of how the data is accessed
- Enforce rules on the client (user interface) and/or the server (back-end)
- Automate processes and integrations
- Improve UX by surfacing real-time feedback to users

No-Code/Low-Code Business Logic

Business Rules

- Declarative, no-code rule engine
- Runs client-side (form) or server side (on save)
- Column validations, show/hide logic, set required levels
- Real-time



Business Rule (Text View)

IF

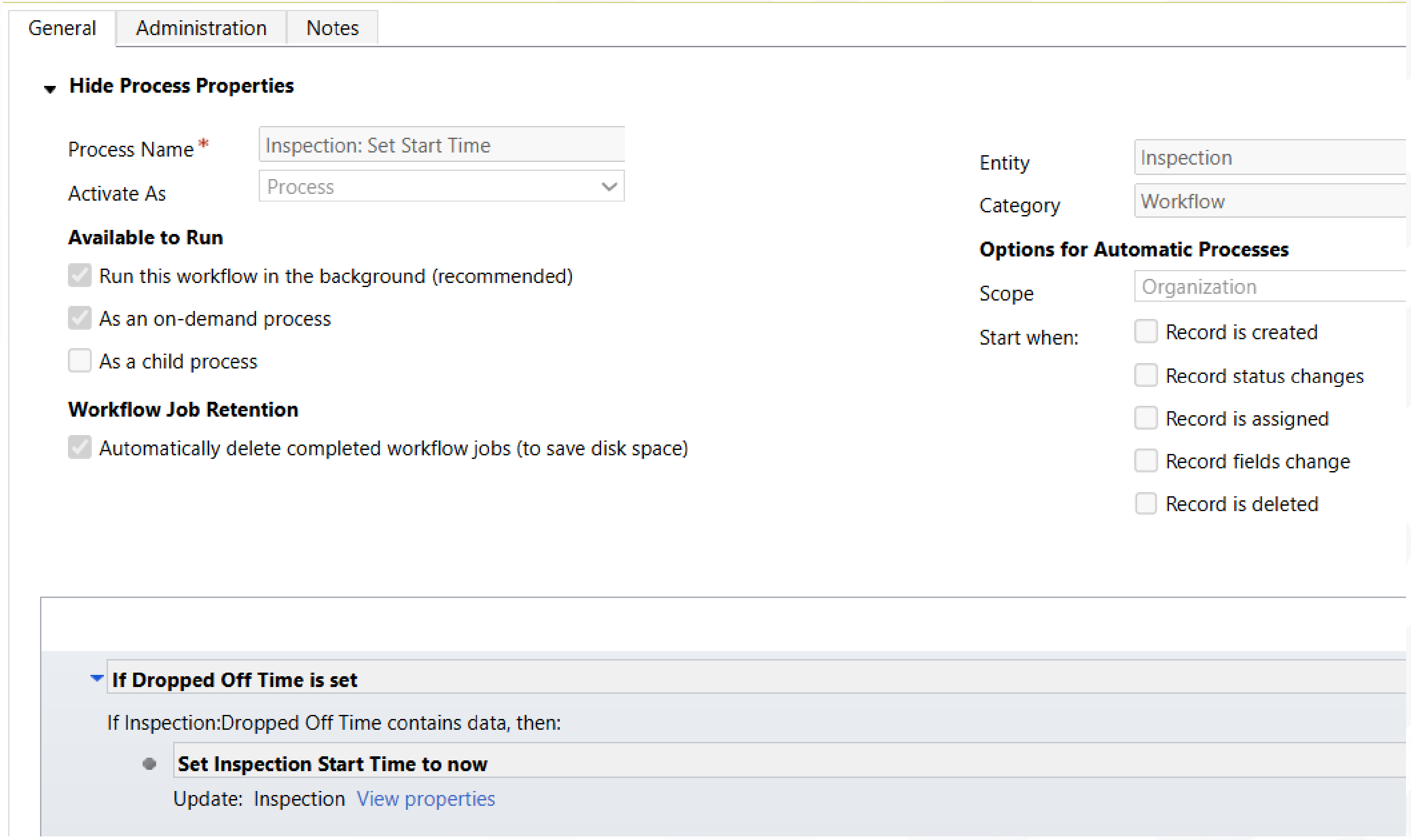
Start Time contains data AND **Start Time** less than **Dropped Off Time**

THEN

Show the error message **"Start time must be on or after Drop Off Time."** against field **Start Time**

Classic Dataverse Workflows

- Server-side processes
- Can run synchronously (real-time) or asynchronously (background)
- Automated triggers (create/update/delete) or on-demand
- Can run before an update/delete occurs or after a create/update/delete
- Can perform operations on parent records
- Transactional/rollback support
- Supports branching/conditions
- Dataverse only interaction



Hide Process Properties

Process Name *	Inspection: Set Start Time
Activate As	Process

Available to Run

- [✓] Run this workflow in the background (recommended)
- [✓] As an on-demand process
- [ ] As a child process

Workflow Job Retention

- [✓] Automatically delete completed workflow jobs (to save disk space)

Entity	Inspection
Category	Workflow

Options for Automatic Processes

Scope	Organization

Start when:

- [ ] Record is created
- [ ] Record status changes
- [ ] Record is assigned
- [ ] Record fields change
- [ ] Record is deleted

If Dropped Off Time is set

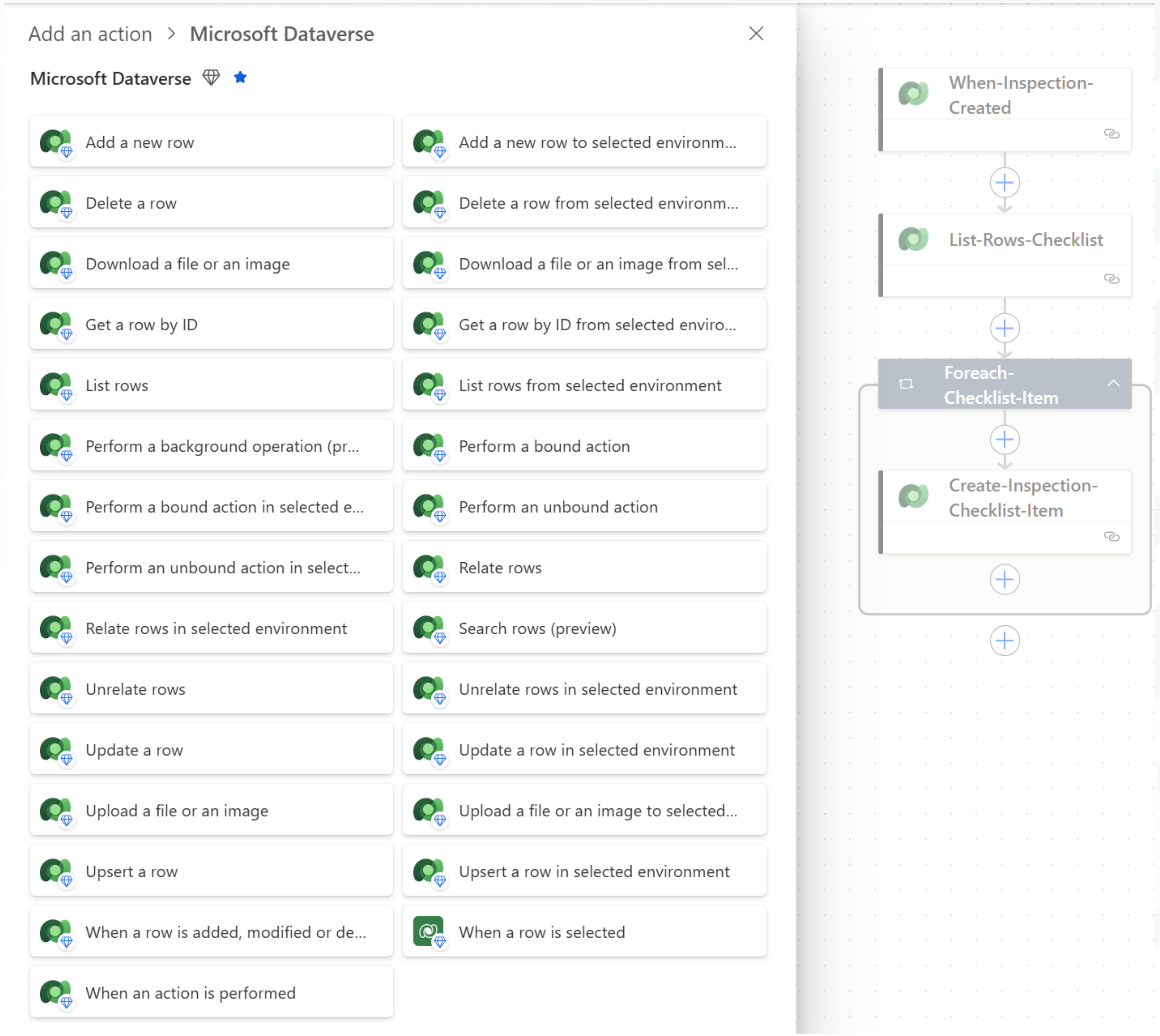
If Inspection:Dropped Off Time contains data, then:

Set Inspection Start Time to now

Update: Inspection View properties

Power Automate Cloud Flows

- First-class support for Dataverse
- Trigger on data events, actions, business process flow step, when a row selected
- Actions can modify records, retrieve/search rows, perform actions, work with files
- Runs Asynchronously
- Allows more complex orchestration and branching than classic workflows
- Integration with other systems through connectors



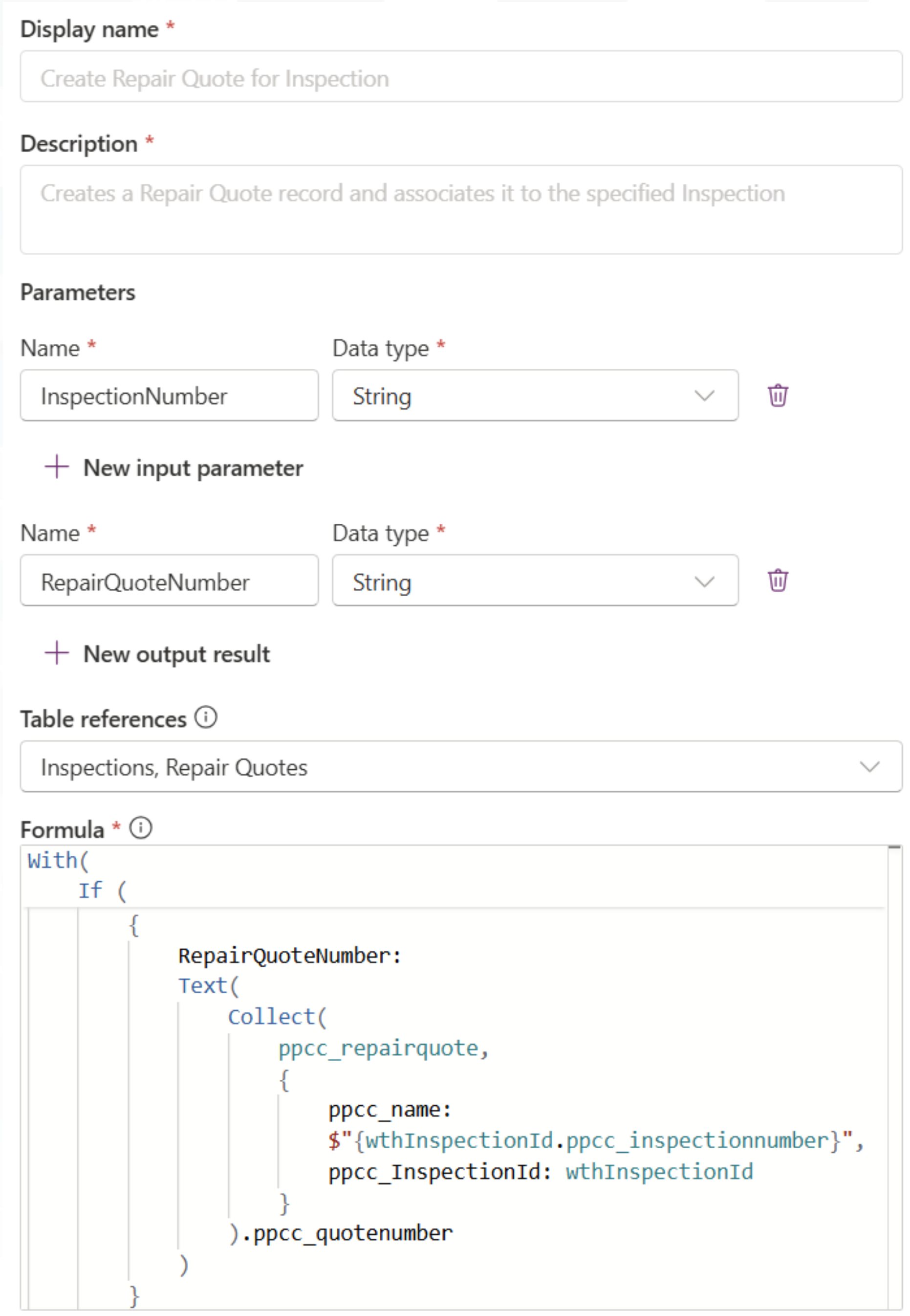
Add an action > Microsoft Dataverse

Microsoft Dataverse

Add a new row	Add a new row to selected environm...
Delete a row	Delete a row from selected environm...
Download a file or an image	Download a file or an image from sel...
Get a row by ID	Get a row by ID from selected enviro...
List rows	List rows from selected environment
Perform a background operation (pr...	Perform a bound action
Perform a bound action in selected e...	Perform an unbound action
Perform an unbound action in select...	Relate rows
Relate rows in selected environment	Search rows (preview)
Unrelate rows	Unrelate rows in selected environment
Update a row	Update a row in selected environment
Upload a file or an image	Upload a file or an image to selected...
Upsert a row	Upsert a row in selected environment
When a row is added, modified or de...	When a row is selected
When an action is performed	

Dataverse (Power Fx) Functions

- Great for encapsulating reusable and/or complex business logic
- Authored using **Power Fx**
- Runs on-demand **synchronously**
- Execute inside the Dataverse Event Pipeline that C# Plugins and Custom APIs do
- Built in **transactions** with automatic rollback on error (Dataverse operations only)
- Call from Power Automate, Canvas Apps, C# plugins, or JavaScript same as a Custom API / Custom Action



Display name *

Create Repair Quote for Inspection

Description *

Creates a Repair Quote record and associates it to the specified Inspection

Parameters

Name *	Data type *
InspectionNumber	String

New input parameter

Name *	Data type *
RepairQuoteNumber	String

New output result

Table references

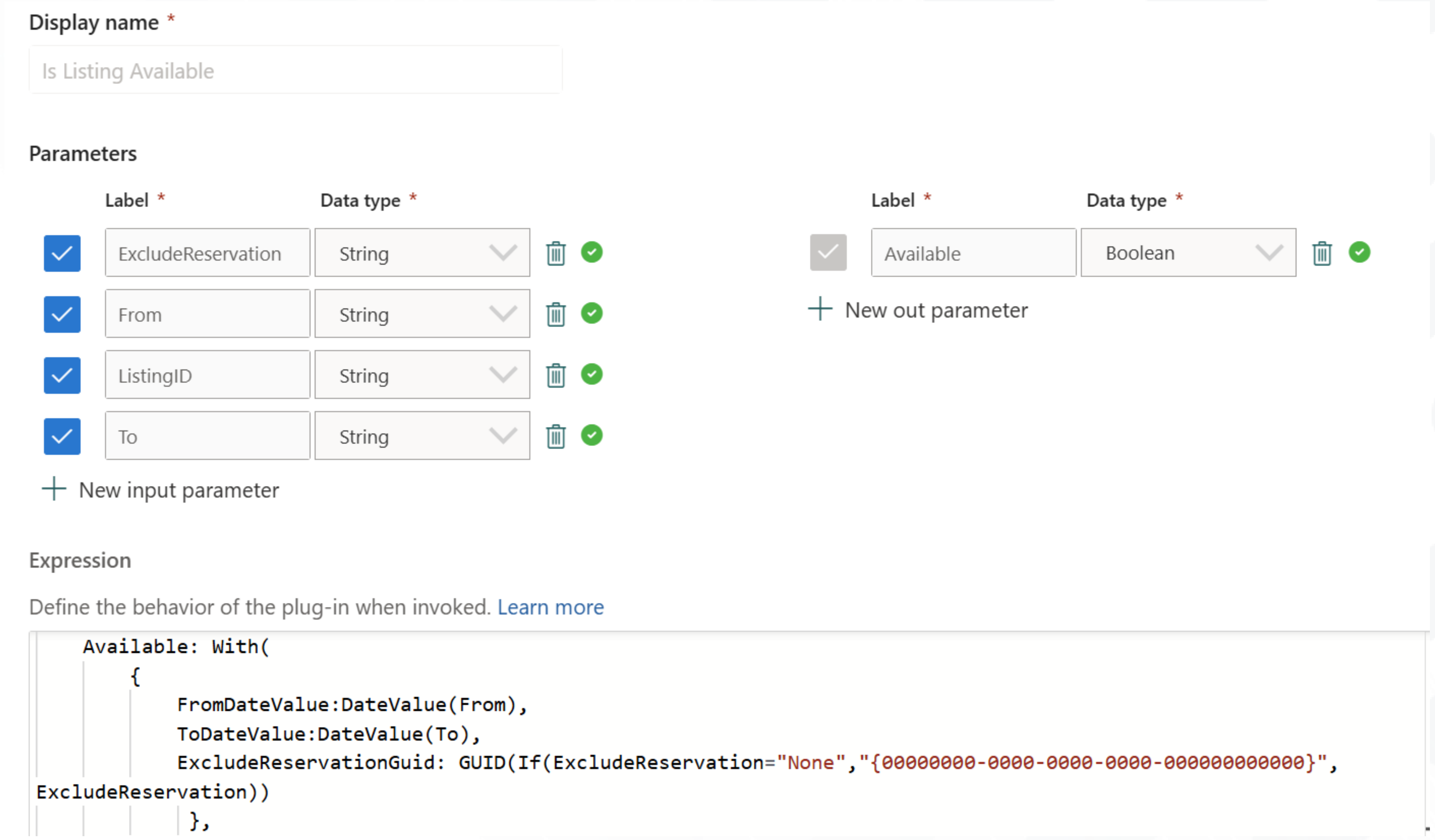
Inspections, Repair Quotes

Formula *

```
With(
    If (
        {
            RepairQuoteNumber:
            Text(
                Collect(
                    ppcc_repairquote,
                    {
                        ppcc_name:
                        $"{wthInspectionId.ppcc_inspectionnumber}",
                        ppcc_InspectionId: wthInspectionId
                    }
                ).ppcc_quotenumber
            )
        }
```

Dataverse Low-Code Plugins (Power Fx)

- Runs synchronous logic
- Written in Power Fx
- Automated low-code plugins trigger on CRUD events
- Can use Connectors to access other data sources
- Requires the Dataverse Accelerator to author
- Still in public preview



Display name *

Is Listing Available

Parameters

	Label *	Data type *
✓	ExcludeReservation	String
✓	From	String
✓	ListingID	String
✓	To	String

New input parameter

	Label *	Data type *
✓	Available	Boolean

New out parameter

Expression

Define the behavior of the plug-in when invoked. Learn more

```
    Available: With(
        {
            FromDateValue:DateValue(From),
            ToDateValue:DateValue(To),
            ExcludeReservationGuid: GUID(If(ExcludeReservation="None","{00000000-0000-0000-0000-000000000000}",
ExcludeReservation))
            },
```

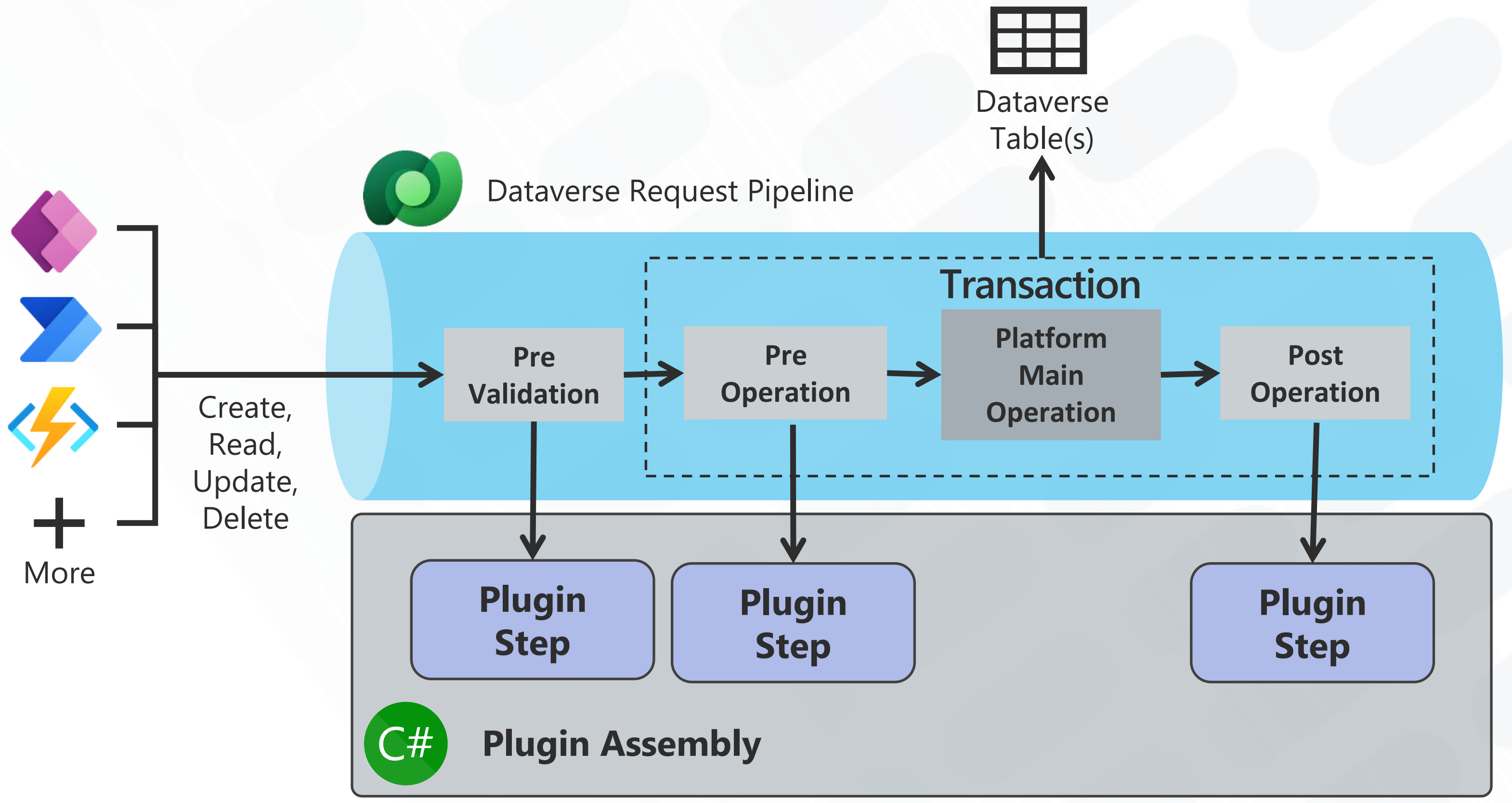
Pro-Code Business Logic

JavaScript Web Resources

- Used to extend the user interface in model-driven apps.
- Requires JavaScript and/or TypeScript knowledge
- Runs only on the client
- Can be registered on Forms, Ribbon Commands, SiteMap, and HTML web resources
- Dynamic UI changes, custom dialogs, or calls to Actions
- Best when you need complex business logic in the UX not possible with business rules

C# Dataverse Plugins

- Adds logic before/during/after a Dataverse transaction
- Written in C# (.NET Framework)
- Logic will execute irrespective of where the Dataverse API is called from
- Best for Fast and Synchronous logic that benefits from executing inside the Dataverse context
- Synchronous (impacts UI performance) or Asynchronous (queued for later execution)



Comparing Options

Choosing the Right Approach

Approach	Low-Code / Code-First	Client Side / Server Side	Triggers	Sync/Async	When to use
Business Rules	Low-Code	Both	Form field change Record save	Sync	Column validation/logic in UX and/or sever
Classic Workflows	Low-Code	Server Side	Automated On-Demand	Both	Real-time automation after record changes
Power Automate Flows	Low-Code	Cross-system	Automated On-Demand Scheduled	Async	Orchestration in near-real time or scheduled External data sources
Power Fx Functions	Low-Code	Server Side	On-Demand	Sync	Centralize and encapsulate related steps into an atomic action
Business Process Flow	Low-Code	Client Side	N/A	Sync	For visually guided processes that are linear
JavaScript Web Resources	Code-First	Client Side	Form field change Record save	Sync	Complex UI logic with form sections, sub grids, related records
C# Plugins	Code-First	Server Side	Automated	Sync/Async	Complex server-side event processing and transactions
Custom APIs	Code-First	Server Side	On-Demand	Sync	Complex server side processing that can be triggered on-demand

Demo & Lab

Best Practices

Business Logic: General Best Practices ⭐

- Always perform validation server-side to enforce business logic regardless of the client
- Prefer asynchronous logic (Power Automate) over synchronous (Real-Time workflows) to prevent long execution time blocking the user interface (UI)
- Prefer no-code/low-code methods before code-first alternatives

Business Rules: Best Practices ⭐

- Ensure referenced columns are **present on a form** (hidden is fine)
- Set the scope to **Entity** to enforce business logic regardless where the record is created/modified (e.g. Canvas Apps, Power Automate Flows, Data import, etc.)
- Minimize the number of rules that target the same field to prevent unexpected behavior
- Prefix name with a numbering schema and activate the business rules in that order to prevent inconsistent behavior between environments

Classic Dataverse Workflows: Best Practices ⭐

- Limit the number of real-time workflows on the same table
- Limit the columns that trigger updates
- Use the before stage for updates if pre-updated values need to be accessed
- Use child workflows to encapsulate steps used by multiple workflows
- Use Power Automate cloud flows if the business logic doesn’t need to be real-time

Power Automate Cloud Flows: Best Practices ⭐

- Limit the columns used to trigger create/update events
- Limit the number of columns returned in retrieve
- Use Expand in List Rows or Get a Row to retrieve related rows in a single API call
- Use FetchXml with the List Rows action to:
    - Reduce API calls through multiple levels of expand/join
    - Perform join queries when there are no formal relationships between two tables
- Move a set Dataverse operations that should be transactional into a Dataverse function where possible

Q&A Time ⏰

🙋‍♂️ 🙋‍♀️

Softball questions only. Unless you have a difficult problem.