SPC: BCTA Clicked Map Manual Guide

Yingjie Feng

December 5, 2025

Introduction

The purpose of this manual guide is to provide a comprehensive walkthrough for creating the **BCTA Microtransit Click-Event Map**, an interactive web mapping application designed to visualize internal and external origin–destination (OD) trips from Beaver County. This tool enhances a static ArcGIS Pro map by integrating **ArcGIS Maps SDK for JavaScript**, enabling users to click on block groups and instantly view trip patterns. The map dynamically displays OD-pairs using a color-coded symbology and updates in real time based on user-selected filters such as trip type, day of the week, and time period (optional, and can be amplified with more fields).

This guide is intended for the SPC Model & Data Team, who may need to replicate or adapt the workflow for similar transportation planning or accessibility analysis projects. It covers all major steps, from preparing raw OD-pair data to publishing the web map and adding interactive JavaScript functionality. This manual assumes a basic working knowledge of Excel, ArcGIS Pro, and introductory programming in Python and JavaScript. For those seeking additional guidance, links to external tutorials will be provided to support further learning and understanding.

To follow along, you will need a combination of GIS software and programming tools:

- ArcGIS Pro for spatial data processing and publishing.
- Python (with relevant libraries such as `pandas`) for data cleaning and formatting.
- ArcGIS Maps SDK for JavaScript for building the web application.
- A modern web browser (e.g., Chrome, Edge, Firefox) for testing the application.

The workflow is organized into three main stages:

1. Excel & Python – Preparing and formatting OD-pair datasets.
2. ArcGIS Pro – Creating and publishing the block group layers and OD data service.
3. ArcGIS JavaScript SDK – Developing the interactive click-event functionality.

By the end of this process, you will have a fully functional web map that allows users to explore spatial travel patterns dynamically, supporting deeper insights into regional microtransit demand.

Methodology

Excel & Python

The first stage of the workflow focuses on preparing the OD-pair dataset so that it can be efficiently queried and visualized in the click-event map. This involves organizing the StreetLight output into a structured table and performing data type adjustments to ensure compatibility with ArcGIS and the JavaScript application.

Begin by creating an Excel table containing at least the following required fields:

- **Origin Block Group ID** – Ensure this field is in **string format**. If the original data stores it as a number, create a new column or convert it to preserve leading zeros (e.g., `420070001001`).
- **Destination Block Group ID** – Also in string format to match with spatial join operations later.
- **Trips** – The numeric value representing the number of trips between the origin and destination.

In addition to the required fields, you may include optional attributes such as day of week, time period, or any other segmentation variables that allow for more granular filtering within the interactive map. These extra fields will later enable dropdown menu filtering without having to regenerate the dataset.

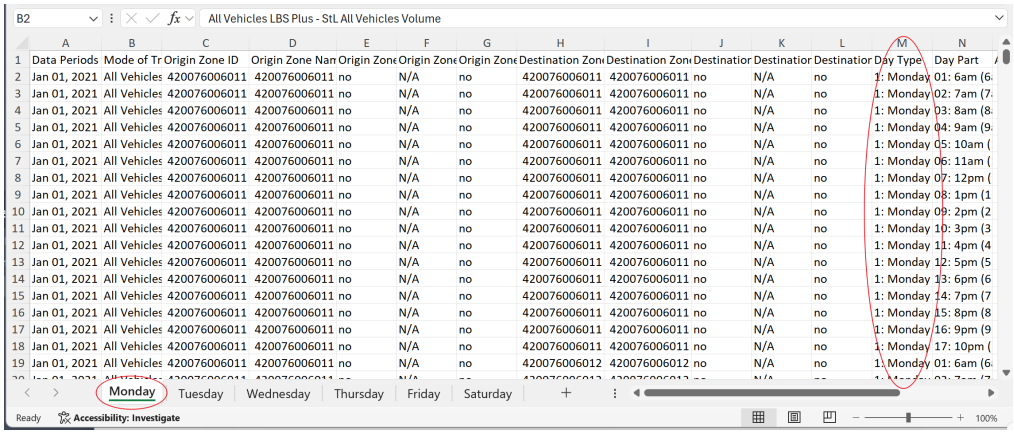
Once the table is structured in Excel, optionally use Python (with relevant libraries such as `pandas`) to perform key cleaning and formatting steps. This can include:

- Standardizing column names to a consistent format for later script references.
- Removing or handling missing values.
- Verifying that all Block Group IDs conform to the expected 12-digit FIPS code format.
- Exporting the cleaned dataset to an Excel (`.xlsx`) file ready for ArcGIS Pro ingestion.

By the end of this step, you will have a clean, well-structured dataset that serves as the foundation for linking OD-pair attributes with spatial geometries in the next stage of the workflow.

Tips: If your dataset is stored in a wide format (for example, with separate trip columns for each day or time period), consider using the `melt()` function from Python's Pandas library to convert it into a long-table format. This structure is generally easier to filter and query within the interactive map application.

For very large datasets that exceed Excel's row limit, separate the data into multiple sheets based on a chosen attribute. For example, you can split trip volumes into different sheets for each day of the week (e.g., Monday through Saturday) while keeping the same column structure in each sheet. This makes it easier to manage and load the data during later processing steps.



ArcGIS Pro

In this stage, we bring together the spatial layers and the OD-pair data so they can be linked for interactive queries later in the web application. The first step is to **merge** all origin and destination block group layers into a single feature layer. This ensures that every block group polygon—whether used as an origin or a destination—is stored in one continuous geometry layer for consistent querying.

Next, use the **"Excel to Table"** geoprocessing tool in ArcGIS Pro to import the cleaned StreetLight dataset you prepared earlier in Excel or Python. This will create a standalone table within your geodatabase that can be linked to the block group geometries.

The most important step is to create a Relationship Class:

- **Required** Relationship – Link the merged block group feature layer to the OD-pairs table using the Origin Block Group ID as the key with **One to Many (1:M)** relationship. This ensures that when a user clicks an origin in the map, the application can query its corresponding destination records from the OD table.
- (Optional Relationship) – Create a secondary relationship between the origin-only block group layer and the OD-pairs table (also keyed on the Origin Block Group ID). This can be used later to apply a distinct symbology for origins versus destinations, making it easier to visually differentiate them.

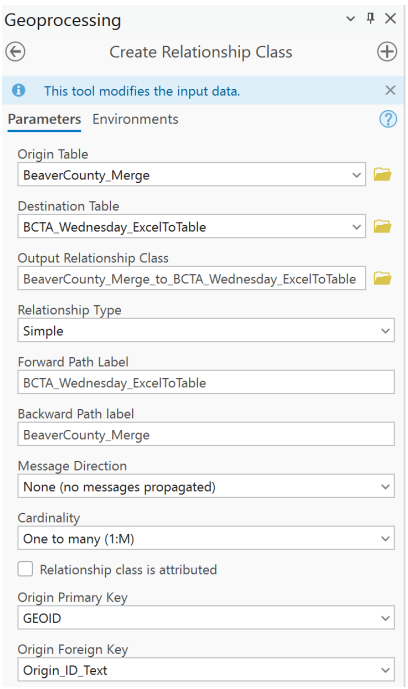


Figure 1: Required Relationship between Merged layer and Wednesday Table

Once the relationships are in place, share the merged feature layer as a web layer in ArcGIS Online. Make sure to set the sharing level to **Everyone (public)** so the JavaScript application can retrieve the data through REST API calls. After these, we have created the basic feature layer for creating further interactive features with ArcGIS SDK Maps for Javascript. Make sure to check the content for which the layer has successfully created relationships with all needed tables. If so, click **publish**.

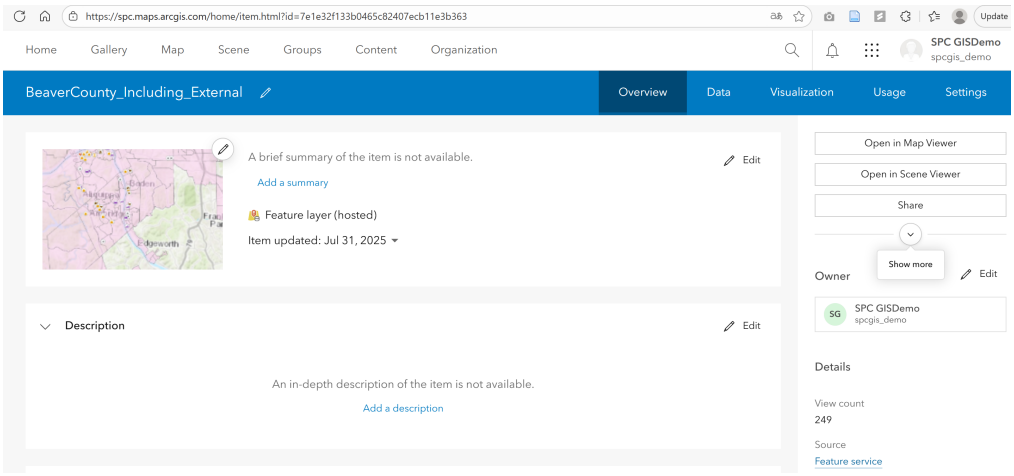


Figure 2: Example of the Shared Layer in AGOL (1/2)

In the page of ArcGIS Online Content, we can find this shared web layer, with the **URL**s in Detail section which guides you to ArcGIS REST Services Directory FeatureServer and Layers. We will directly use URLs for further API call in ArcGIS Maps SDK for Javascript.

Tips: If you are working with a separated Excel dataset (e.g., one sheet per day of the week), you will need to upload each sheet individually using the "Excel to Table" tool, then merge them back into one consolidated table in ArcGIS Pro. While keeping them separated also works, a single merged table simplifies queries, and ArcGIS Pro's geodatabase tables can handle far more records than Excel.

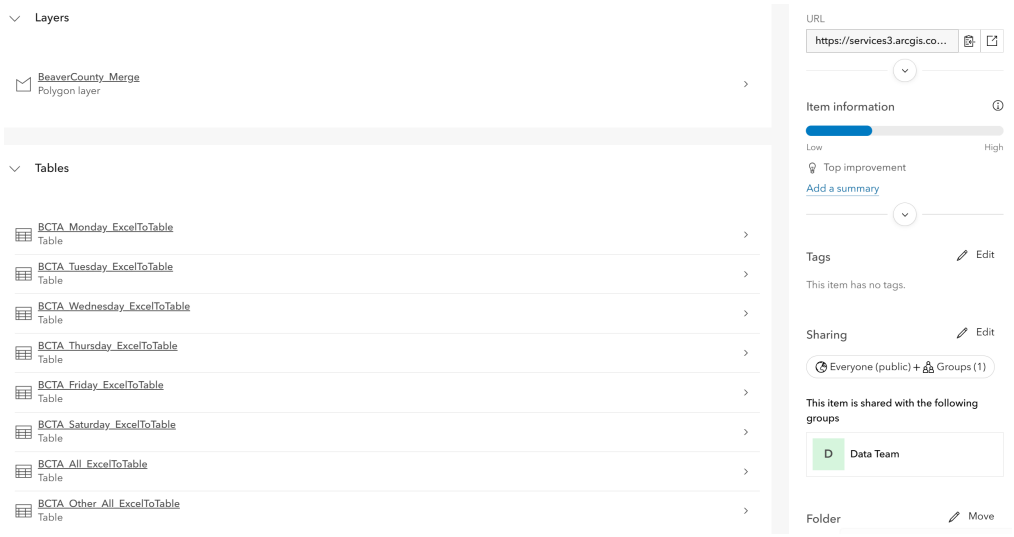
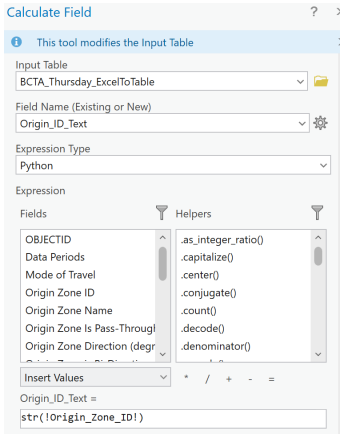


Figure 3: Example of the Shared Layer in AGOL (2/2)

If you want the origin block groups to appear visually distinct from destination block groups, complete the optional relationship step so you can publish an origin-only layer with a unique symbology.

If the Origin Block Group ID or Destination Block Group ID were not converted to string format earlier, you can fix this directly in ArcGIS Pro by adding a new field and calculating it with an expression like:

```
str(!Your Origin Block Group ID field!)
```



ArcGIS Maps SDK for JavaScript

What it is and why we need it

ArcGIS Maps SDK for JavaScript turns our hosted ArcGIS Online layers into an interactive, browser-based application. **We use it to connect click events on block groups to OD queries, apply user-selected filters (trip type, day, time), aggregate results on the fly, and render destinations with a consistent class-break color scale.** This elevates a static map into a responsive OD explorer for planning and analysis.

Flow chart of the JS app

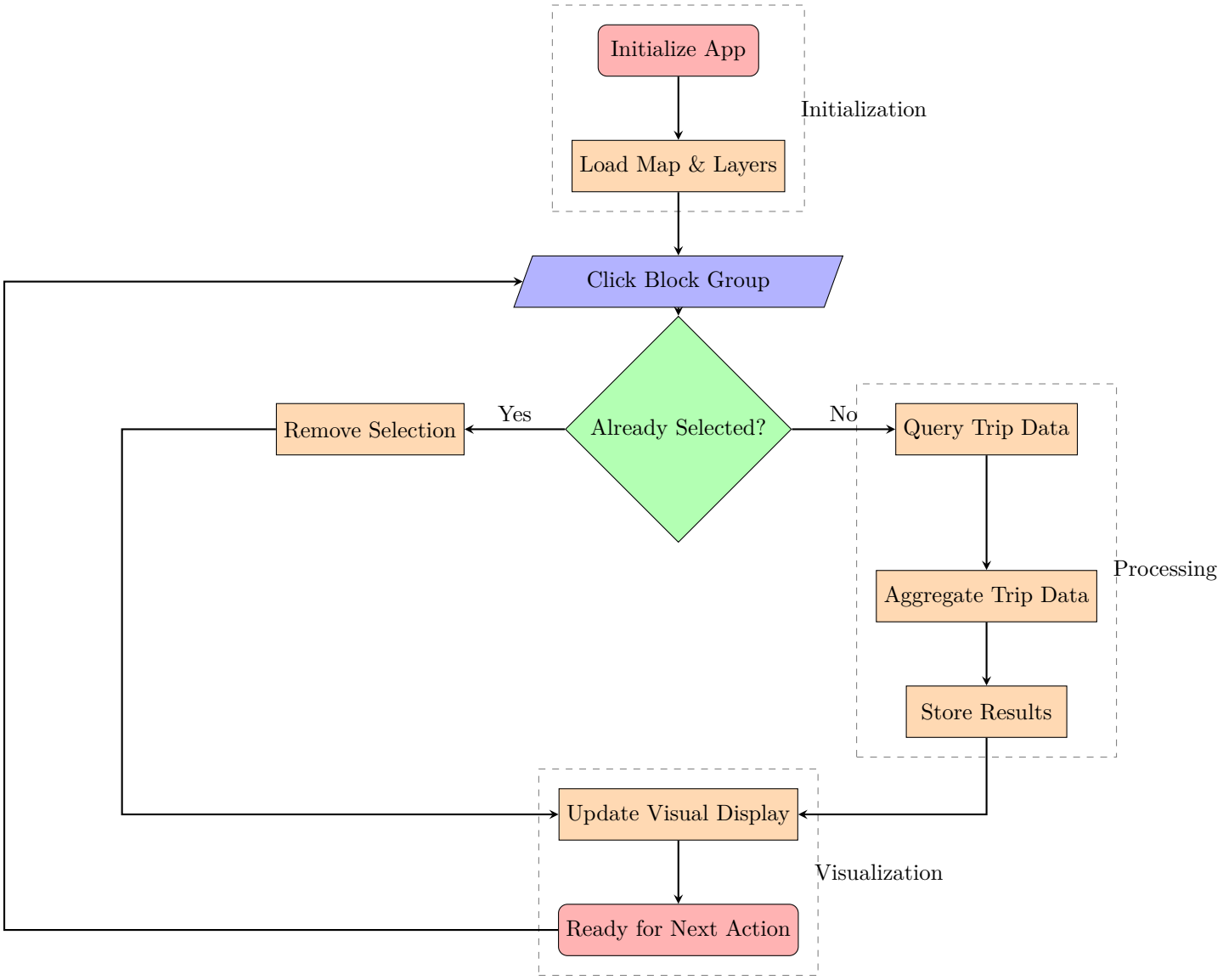


Figure 4: Workflow for Aliquippa / Ambridge / Beaver Falls Click-Event Map (No Filters)

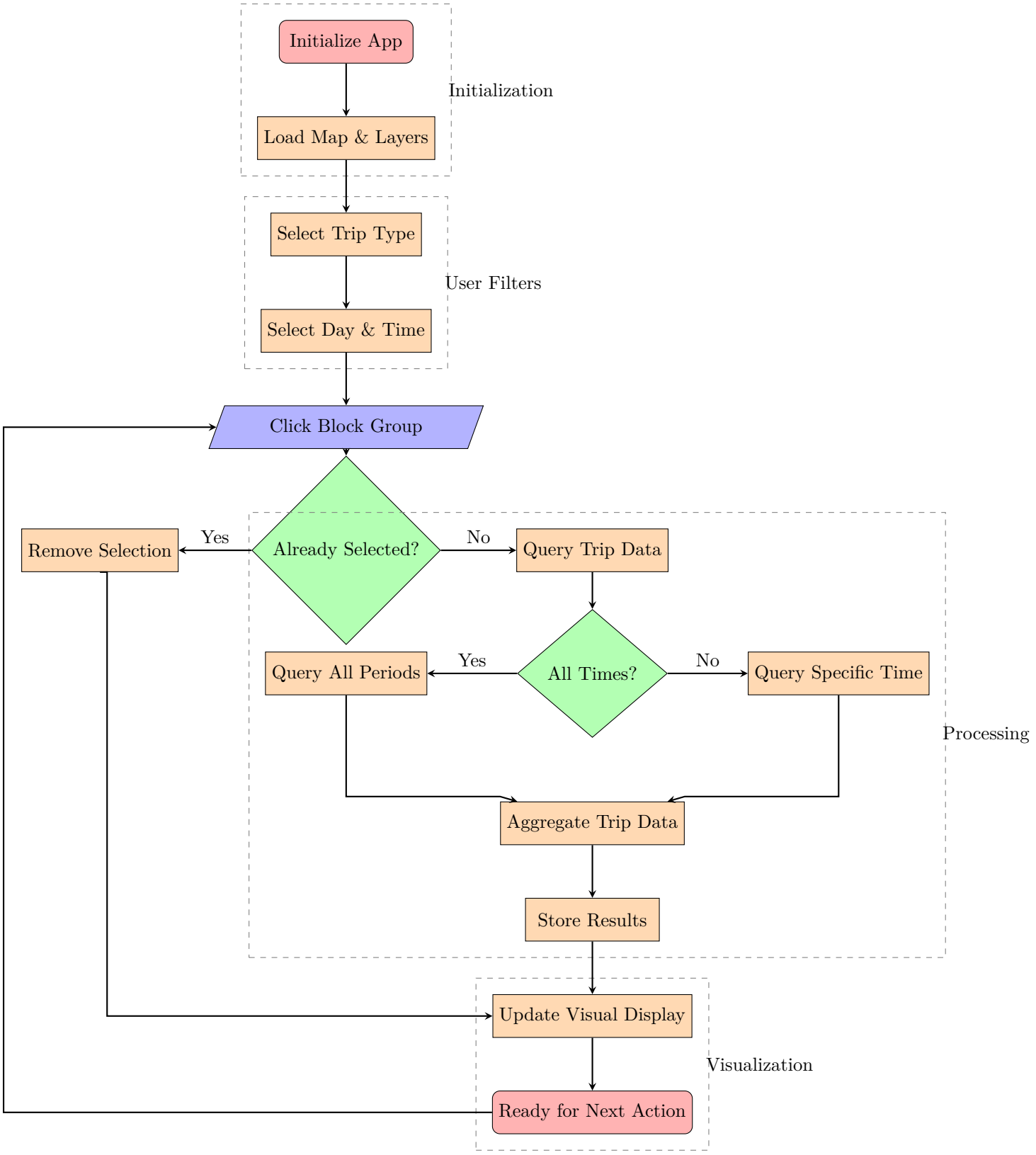


Figure 5: Workflow for Beaver County Click-Event Map

Design walkthrough (step by step)

Initialization → *Initialize App & Load Map & Layers* When the web application first loads, the ArcGIS JavaScript API libraries are imported. These provide the building blocks for displaying and interacting with maps in a web browser. A base `Map` object is created with a neutral basemap (e.g., light gray) so that thematic layers stand out clearly. A `MapView` is then initialized, which serves as the "map window" that controls what geographic area is visible, the zoom level, and the on-screen interface.

Block Group Layer Architecture The application uses three separate *instances* of the same Block Group Feature Service (layer 0) from ArcGIS Online, each serving a different purpose:

1. **Context Layer (`blockGroupOutlineLayer`)** Purpose: Provides geographic context for navigation. Styling: Light green fill (`[180, 230, 180, 0.6]`) with green outlines. Role: Always visible as the base geometry showing all Block Group boundaries.
2. **Symbolized Layer (`blockGroupTripsLayer`)** Purpose: Defines the color scheme and legend for trip volumes. Styling: Class-breaks renderer with five categories for trip counts. Role: Provides the symbology reference used later for dynamically coloring destinations.
3. **Interactive Query Layer (`beaverCountyBG`)** Purpose: Used for hit testing and attribute retrieval when a user clicks on the map. Role: Identifies the clicked Block Group (via GEOID) and retrieves its geometry for selection highlighting.

Additional Data Layers Beyond the Block Group geometries, the application uses two non-visual layers for OD trip data:

- **Origin–Destination Data Layer (`odTable`)** Purpose: Contains actual trip records between origins and destinations. Dynamic Source: Switches between Layer 7 (*Internal*) – trips within Beaver County Layer 8 (*External*) – trips to outside areas. Note: This layer is never displayed directly — it is queried to retrieve trip counts matching the user's filters.

Rendering Architecture While Feature Layers are used for geometry display and data retrieval, the *actual* trip visualization is drawn using the JavaScript API's **GraphicsLayer**. This approach allows:

- Fully dynamic updates when the user changes filters or selections.
- Highlighting only the relevant destinations without redrawing the entire feature layer.
- Using the class-break symbology from the `blockGroupTripsLayer` to ensure visual consistency.

Complete Layer Structure Summary:

1. Block Group Base Layer — context, green styling.
2. Block Group Symbolized Layer — defines legend and class breaks.
3. Block Group Query Layer — for feature identification.
4. OD Table Data Layer — trip data (internal or external).
5. Graphics Layer — draws the dynamic trip results and selection highlights.

This multi-layer approach ensures the application can support complex interactive behavior while maintaining performance and clear visual hierarchy.

User filters → *Select Trip Type, Select Day & Time* Before clicking anything, the user chooses from three filter controls:

- **Trip Type** — Internal trips (within Beaver County) or External trips (from Beaver County to outside areas). This determines which OD dataset (service layer) the application will use. For example, Internal queries go to `/FeatureServer/7`, External queries to `/FeatureServer/8`.
- **Day of Week** — The specific day or a grouped category (e.g., "Mon–Sat" for weekday averages). Selecting the day activates the time control.
- **Time Period** — Either a single hour bin or `ALL` to represent the whole day period defined in the dataset (e.g., 6 a.m.–11 p.m.).

The JavaScript code translates these selections into a filter expression that will be passed to the OD service. In ArcGIS terms, this is a `definitionExpression`—a SQL-like WHERE clause that restricts the records returned.

Changing any filter automatically clears any previously selected origins and their destination results, ensuring the map only shows data consistent with the new criteria.

Click handling → *Click Block Group* Once filters are set, the user clicks anywhere on the map. The application runs a "hit test" — a spatial query that checks what features intersect the mouse click location. If the click falls inside a Block Group polygon from the BG layer, the feature's attributes are retrieved. Most importantly, the **12-digit Block Group GEOID** is extracted. This ID is what links the geometry to the OD records in the related table.

If the click doesn't intersect a BG polygon, the event is ignored and the map waits for the next click.

Selection logic → *Already Selected?* → *Remove Selection* The application maintains a list (or set) of currently selected origin BGs. If the clicked GEOID is **already in the set**, this means the user is toggling it off. The GEOID is removed from the set, and the application immediately recalculates the combined results from any remaining selected origins using cached data (no new queries are needed). The map is refreshed to remove the red outline for that origin and to adjust destination symbology accordingly.

Query path → *Already Selected?* → *Query Trip Data* If the clicked GEOID is **not already in the set**, the application adds it to the set and sends a query to the appropriate OD service endpoint (as determined by Trip Type). The query is built with:

- `Origin_ID_Text = '<clicked GEOID>'`
- The current Day filter, e.g., `Day_Type = 'Monday'` or `Day_Type IN ('Monday',...,'Saturday')`
- The Time filter, unless Time = `ALL`, in which case all times are allowed for aggregation later.

This query returns the OD records for that origin, with each row containing a `Destination_Zone_ID` and a trip count field.

Time branching → *All Times?* → *Query All Periods / Query Specific Time* If Time = `ALL`, the application requests all time bins for the chosen day. The trip values are later summed across those bins during aggregation. If a specific hour is selected, the query only returns rows for that hour's bin, eliminating the need for further temporal aggregation.

Processing → *Aggregate Trip Data & Store Results* Regardless of which branch is taken, the raw query results are grouped by `Destination_Zone_ID`, and the trip counts are summed to produce a total number of trips from the clicked origin to each destination BG.

The aggregated results are stored in memory (e.g., `tripData[origin GEOID]`). This way, if the user deselects and reselects the same origin without changing filters, the application can pull the results directly from cache without re-querying the server.

Visualization → *Update Visual Display* With aggregation complete, the map display is refreshed:

1. All previous destination graphics are cleared.
2. Each destination BG in the results is drawn as a polygon graphic with a fill color determined by the same class-breaks used in the published layer's renderer (e.g., light yellow for low trip counts, deep red for high trip counts).
3. Selected origin BGs are outlined in thick red for emphasis.

The summary panel in the UI is updated to list each selected origin and its total outbound trips (the sum of all its destination trips). The legend title is updated to match the active Trip Type ("Within Beaver County" or "To External Areas").

A floating tooltip follows the mouse pointer and updates dynamically:

- Hovering over a selected origin shows "Selected Origin" and its outbound trips, plus inbound trips from other selected origins if applicable.

- Hovering over another BG (when at least one origin is selected) shows inbound trips from the selected origins to that BG.
- Hovering when no origins are selected prompts the user to click a BG to select it.

Event loop → *Ready for Next Action* Once the display is updated, the application waits for the next user action:

- A **new click** re-enters the selection logic: if it's a new origin, query and aggregate; if it's already selected, remove it.
- A **filter change** restarts from the filter stage: clear all selections, reapply the `definitionExpression`, and wait for new clicks.

This continuous loop—filter, click, query if needed, aggregate, merge, display—keeps the application's visualization synchronized with user intent at all times.

Conclusion

The methodology has been structured to not only document the specific Beaver County and Aliquippa/Ambridge/Beaver Falls implementations, but also to present a generalizable framework that can be adapted to any region and dataset with similar needs.

Key takeaways include:

- **Layer setup in ArcGIS JS**: how to instantiate feature layers from ArcGIS Online URLs, reuse the same source service for multiple purposes (context display, symbology reference, interactive querying), and employ a graphics layer for dynamic, client-side rendering.
- **Data preparation best practices**: ensuring consistent formats for origin/destination IDs, leveraging `pandas` for efficient reshaping and cleaning, and designing a schema that supports flexible filtering.
- **Interactive design logic**: translating user actions—clicks and filter changes—into feature queries, aggregating results on the fly, and updating visualizations without unnecessary server requests.
- **Deployment workflow**: hosting the JavaScript app with GitHub Pages, adjusting URLs and configuration for different datasets, and optionally registering the hosted app in ArcGIS Online for broader sharing and integration.

By following this guide, SPC GIS professionals can rapidly stand up interactive OD visualization tools tailored to exploring study areas and data sources. The outlined practices—such as separating layer roles, caching query results, and using graphics layers for responsiveness—are applicable far beyond the specific examples here, and can be extended with additional interface elements, more complex filtering, or integration with other APIs.

If you have questions, encounter issues, or wish to collaborate on enhancements to this workflow, please feel free to reach out directly at:

`fengyingjie703@gmail.com`

Your feedback and shared use cases will help improve and expand this methodology for the wider GIS community.