Scenario 1: Simple Weather API Call

Skill Level: Beginner to Intermediate
Time to Complete: 45-60 minutes (lab included)
Prerequisites: Genesys Cloud org access, Architect admin role, browser

---

Executive Summary

This scenario teaches you how to:

1. Create a **data action** that calls an external REST API (Open-Meteo weather)
2. Configure the **three output paths** (Success, Failure, Timeout)
3. Parse JSON responses using **translation maps** and **Velocity macros**
4. Use the API response in an Architect flow to make a routing decision
5. Understand what happens when the API is slow, returns an error, or times out

Real-world use case: A financial services IVR asks for the caller's city, looks up current weather, and routes them:

- **Good weather** → Sales queue (outdoor account interest)
- **Bad weather** → Account services queue (likely indoor issues)
- **API timeout** → Queue directly (no time to look up weather)

Part 1: Conceptual Foundation

What Is a Data Action?

In Genesys Cloud, a **Data Action** is a reusable connector that:

- Makes an **HTTP request** to an external API
- Receives a **JSON response**

- Parses that response using **translation maps** (JSONPath) and **success templates** (Velocity)
- Returns structured data back to your Architect flow

Think of it like this:

```
Architect Flow → [HTTP GET] → Open-Meteo API → [JSON weather] → Translation Map → Flow Variable
                                    ↓
                              (Success/Failure/Timeout)
```

Why Three Output Paths?

Every data action execution has **three possible outcomes**:

Path	When It Fires	Example
Success	The action successfully communicated with the endpoint and received a result	Got temp=72°F, humidity=55%
Failure	Execution error, parsing error, network/connectivity issue, or integration-service timeout	Got 404 "location not found" or connection lost
Timeout	The Architect action exceeded the timeout configured on the action step	Architect waited for configured timeout, now proceeds

You should handle all three paths in your flow. Here's why:

The exact behavior when a path is not configured depends on how the rest of your flow is designed. In most cases, if you don't handle a path, callers experience:

- No audio response
- Unexpected transfer
- Flow error

By handling all three paths, you ensure:

- **Resilience:** Slow APIs don't break your flow (Timeout path routes gracefully)
- **Fallback logic:** Failure path can retry or route differently than Success
- **Caller experience:** Professional handling even when external systems fail
- **Predictability:** You control what happens in every scenario

Best practice is to handle all three paths explicitly. Genesys documents separate Success, Failure, and Timeout paths for Call Data Action steps and recommends keeping the timeout at 30 seconds or less.

Source: https://help.genesys.cloud/articles/failure-paths-architect/

Part 2: The API We're Using (Open-Meteo)

Why Open-Meteo?

- **Free for non-commercial use** — No API key, no registration required
- **Simple** — Standard REST + JSON (easy to learn)
- **Public API** — Maintained by open-source community
- **Learning-safe** — Ideal for lab exercises without cost concerns

Verified sources:

- No API key required and free for non-commercial use: https://open-meteo.com/
- Open-source community maintained: https://github.com/open-meteo/open-meteo

The Endpoint We'll Call

```
GET https://api.open-meteo.com/v1/forecast
  ?latitude=25.85
  &longitude=-100.27
  &current=temperature_2m,weather_code,wind_speed_10m
  &temperature_unit=fahrenheit
  &timezone=auto
```

What it means:

- **latitude, longitude** = Monterrey, Mexico (for our example)
- **current** = Only get current conditions (not hourly/daily forecast)
- **temperature_unit** = Return in Fahrenheit (for US callers)
- **timezone=auto** = Open-Meteo automatically determines timezone from coordinates

The Response (What We Get Back)

```
{
  "latitude": 25.85,
  "longitude": -100.27,
  "timezone": "America/Chicago",
  "current": {
    "time": "2026-03-14T15:30",
    "temperature_2m": 78,
    "weather_code": 0,
    "wind_speed_10m": 8.5
  },
  "current_units": {
    "temperature_2m": "°F",
    "wind_speed_10m": "km/h"
  }
}
```


Key fields:

- `current.temperature_2m` → Current temp (78°F)
- `current.weather_code` → WMO code (0=clear sky, 1=mainly clear, 45=foggy, etc.)
- `current.wind_speed_10m` → Wind speed at 10m height

(Note: The timezone and exact response values shown are illustrative. Open-Meteo determines the actual timezone from the coordinates you provide.)

Part 3: Building the Data Action in Genesys Cloud

Step 1: Create a Custom Data Action

Path in Genesys UI:

```
Admin → Integrations & Apps → Data Actions → New Action
```

Field	Value	Why
Name	`GetCurrentWeather`	Clear, verb-noun naming

Field	Value	Why
Category	`Weather`	Organizational (optional but good practice)
Integration	`Web Services Data Actions`	For calling external REST APIs

Click Create.

Step 2: Define the Input Contract

This tells Genesys what data your flow will **send to** the API call.

Add Input Fields:

Field Name	Type	Required?	Description	Example
`LATITUDE`	Number	Yes	Geographic latitude	`25.85`
`LONGITUDE`	Number	Yes	Geographic longitude	`-100.27`
`TEMPERATURE_UNIT`	String	No	fahrenheit or celsius	`fahrenheit`

UI Path: Click **Input Contract** → **Add Property**

Why these inputs?

- Latitude/longitude identify the location
- Temperature unit allows caller preference (US vs. Canada)
- We're parameterizing the API call so it can be reused for different cities

---

Step 3: Define the Request

This is the actual HTTP request that Genesys will send to Open-Meteo.

Request Type: `GET`

Request URL Template:

```
https://api.open-meteo.com/v1/forecast?latitude=${esc.url(input.LATITUDE)}&longitude=${esc.url(input.LONGITUDE)}&current=temperature_2m,weather_code,wind_speed_10m&temperature_unit=${esc.url(input.TEMPERATURE_UNIT)}&timezone=auto
```

URL Parameter Escaping Explanation:

Notice the `${esc.url(...)}` syntax around the input variables. This is a **best practice** for handling query parameters.

Why escape URL parameters?

- If a parameter contains special characters (spaces, `&`, `?`, `=`), they can break the URL
- Escaping converts them to URL-safe equivalents (e.g., space → `%20`)
- For this weather API, it's not strictly necessary (latitude/longitude are numbers), but it's good practice for any string parameters like `TEMPERATURE_UNIT`

Example: If someone passes `temperature_unit = "fahrenheit & celsius"`, without escaping the `&` would break the URL. With escaping, it becomes safe.

Source: https://help.genesys.cloud/articles/request-configuration-data-actions/

Headers: (optional, usually unnecessary for GET requests to this API)

```
Content-Type: application/json
User-Agent: GenesysCloud-IVR/1.0
```

Step 4: Define the Output Contract (Success Schema)

This tells Genesys what data structure you expect back on **success**.

Add Output Fields:

Field Name	Type	Description
`TEMPERATURE`	Number	Current temperature in requested unit
`WEATHER_CODE`	Number	WMO weather code (0=clear, 1=mostly clear, 45=foggy, etc.)
`WIND_SPEED`	Number	Wind speed in km/h
`TIMEZONE`	String	Timezone of the location

Why these outputs?

- These are the fields your Architect flow will actually **use** to make decisions
- Weather_code 0-10 = good weather → route to Sales
- Weather_code 45-67 = bad weather → route to Service

- Temperature helps personalize the agent greeting

Step 5: Configure the Response Parsing

This is the **critical part**. Here's where we convert the raw Open-Meteo JSON into the output contract fields.

Genesys provides two tools:

1. **Translation Map** (JSONPath) — Extract data from the response
2. **Success Template** (Velocity) — Format the output

Translation Map

For each output field, specify a **JSONPath expression** that extracts the value from the API response.

JSONPath is just a way to "point to" parts of JSON.

Example: To get the temperature from:

```
{
  "current": {
    "temperature_2m": 78
  }
}
```

Use JSONPath: `$.current.temperature_2m`

Breakdown:

- `$` = root of the JSON
- `.current` = go into the "current" object
- `.temperature_2m` = get the "temperature_2m" field
- **Result:** 78

Configure these Translation Map entries:

Output Field	JSONPath Expression	Default (if parsing fails)
`TEMPERATURE`	`$.current.temperature_2m`	`0`
`WEATHER_CODE`	`$.current.weather_code`	`99`
`WIND_SPEED`	`$.current.wind_speed_10m`	`0`

Output Field	JSONPath Expression	Default (if parsing fails)
`TIMEZONE`	`$.timezone`	`UTC`

UI Path: In the data action, click **Response Configuration** → **Translation Map** → Add each field

---

Success Template

After the translation map extracts values, the success template formats them into the final output.

Critical: The property names in your success template **must exactly match your output contract field names**.

Your output contract defined: `TEMPERATURE`, `WEATHER_CODE`, `WIND_SPEED`, `TIMEZONE` (all uppercase)

So your success template must return those same names:

```
{
  "TEMPERATURE": ${TEMPERATURE},
  "WEATHER_CODE": ${WEATHER_CODE},
  "WIND_SPEED": ${WIND_SPEED},
  "TIMEZONE": "${TIMEZONE}"
}
```

Why this matters: Genesys validates that the resolved response conforms to the success schema. If your template returns `"temperature"` but your schema expects `"TEMPERATURE"`, the response fails validation and triggers the Failure path.

Note: String fields need quotes (`"${TIMEZONE}"`), numbers don't (`${TEMPERATURE}`).

UI Path: In Response Configuration, scroll to **Success Template**, paste the above

Source: https://help.genesys.cloud/articles/response-configuration-data-actions/

Part 4: Configuration Reference (All Fields)

Data Action Configuration Checklist

Section	Field	Value	Required?
General	Name	`GetCurrentWeather`	☐ Yes
General	Category	`Weather`	☐ No
General	Integration	Web Services	☐ Yes
Input	LATITUDE	Number	☐ Yes
Input	LONGITUDE	Number	☐ Yes
Input	TEMPERATURE_UNIT	String	☐ No
Request	Type	GET	☐ Yes
Request	URL	`https://api.open-meteo.com/v1/forecast?...`	☐ Yes
Request	Headers	Content-Type: application/json	☐ No
Request	Timeout	`30` seconds	⚠ Important
Output	TEMPERATURE	Number	☐ Yes
Output	WEATHER_CODE	Number	☐ Yes
Output	WIND_SPEED	Number	☐ Yes
Output	TIMEZONE	String	☐ Yes
Response	Translation Map	JSONPath for each field	☐ Yes
Response	Success Template	Velocity template	☐ Yes

Timeout Configuration (Critical!)

Where: Data Action → Configuration → Timeout
Value: `30` seconds (reasonable for weather API)
Why 30, not 60?

- Default is 60, but in production IVR, calling a weather API shouldn't take that long
- If the API is slow, 30 seconds gives you time to fall through to Timeout path
- Better to route caller to queue quickly than make them wait 60 seconds

What happens if timeout fires:

- When the configured timeout is reached, Architect stops waiting and follows the Timeout path
- The flow proceeds to handle the Timeout path and routes the call accordingly

This is important for understanding flow design: if the timeout threshold is reached, Architect no longer waits for a response, regardless of whether the backend request completes.

Source: https://help.genesys.cloud/articles/add-call-data-action-data-action-task/

Part 5: Using the Data Action in Architect

Now we've created the data action. Let's use it in a flow.

The Flow Structure

```
Inbound Call (IVR)
    ↓
Play: "Which city do you call from? 1 for Monterrey, 2 for Mexico City"
    ↓
Collect Digits (1 or 2)
    ↓
Decision: Which city?
    ├─ 1 → Call GetCurrentWeather (latitude 25.85, longitude -100.27)
    └─ 2 → Call GetCurrentWeather (latitude 19.43, longitude -99.13)
    ↓
[GetCurrentWeather Data Action]
    ├─ Success → Check weather_code
    │    ├─ 0-10 (clear) → Play "Nice weather! Routing to Sales"
    │    └─ 45-99 (bad) → Play "Rainy day! Routing to Support"
    │
    ├─ Failure → Play "Unable to check weather"
    │
    └─ Timeout → Play "System busy, routing now"
    ↓
Transfer to Queue
```

Adding the Call Data Action Step in Architect

UI Path:

> Architect → [Open your flow]
> → Task Sequence → [Drag in] Call Data Action

Configure the Data Action Step:

Field	Value	Example
Action	GetCurrentWeather	(dropdown, select your action)
LATITUDE	(expression)	`25.85` or `flow.selectedLatitude`
LONGITUDE	(expression)	`-100.27` or `flow.selectedLongitude`
TEMPERATURE_UNIT	(expression)	`"fahrenheit"`

Important: These map to your Input Contract fields from Step 2.

Handling the Success Output Path

When the data action **succeeds**, the output fields are available to bind to Architect variables.

Important: The variable names are **YOUR choice** — you bind each output field to whatever variable name you want in the Call Data Action configuration.

How it works:

In your Call Data Action step configuration, you specify:

- Which output field from the data action (TEMPERATURE, WEATHER_CODE, etc.)
- Which Architect variable to store it in (your choice)

For example, you could configure:

- Output field `TEMPERATURE` → Bind to flow variable `CurrentTemp`
- Output field `WEATHER_CODE` → Bind to flow variable `WeatherCondition`
- Output field `WIND_SPEED` → Bind to flow variable `WindKmh`
- Output field `TIMEZONE` → Bind to flow variable `LocalZone`

Then later in your flow, you reference the variables you chose: `CurrentTemp`, `WeatherCondition`, etc.

In practice: In the Call Data Action configuration step, you'll see fields labeled **Success Outputs** where you define this binding.

```
Success Outputs:
  TEMPERATURE  → CurrentTemp
  WEATHER_CODE → WeatherCondition
  WIND_SPEED   → WindKmh
  TIMEZONE     → LocalZone
```

In the Success path, use those bound variables:

```
Decision: Check Weather Code
├─ If WeatherCondition < 11
│   → Play "Great weather! Routing to Sales"
│   → Transfer to Queue "Sales"
│
└─ Else
   → Play "Rainy weather. Routing to Support"
   → Transfer to Queue "Support"
```

Source: https://help.genesys.cloud/articles/add-call-data-action-data-action-task/

Handling the Failure Output Path

Failure = API returned error (4xx/5xx) OR response parsing failed

Example: Open-Meteo returns 400 if you request an invalid latitude.

In the Failure path:

```
Play Audio: "We couldn't check the weather. Routing you now."
Set Participant Data:
  → Set "WeatherStatus" = "Failed"
Transfer to Queue: "Default Queue"
```

Handling the Timeout Output Path

Timeout = Architect stopped waiting after the configured timeout

This happens when:

- Open-Meteo servers are overloaded
- Network is slow
- Genesys Edge has poor connectivity to the internet

In the Timeout path:

```
Play Audio: "System busy. Routing you immediately."
Set Participant Data:
  → Set "WeatherStatus" = "Timeout"
Transfer to Queue: "Default Queue"
```

Part 6: Step-by-Step Lab Instructions

Lab Setup

You'll need:

- □ Access to a Genesys Cloud organization (CX license minimum)
- □ Admin or Architect role
- □ A web browser (Chrome, Firefox, Safari)
- □ At least one queue configured (or use the default)

Time: 45-60 minutes

Lab Exercise A: Create the Data Action

Step 1: Navigate to Integrations

```
https://apps.mypurecloud.com
  → Admin (menu icon, top-right)
  → Integrations & Apps
  → Data Actions
  → New Action
```

Step 2: Fill in General Information

Field	Enter
Name	`GetCurrentWeather`
Category	`Weather`
Integration	Web Services Data Actions

Click **Create**.

Step 3: Configure Input Contract

Click **Input Contract** in the left menu.

Click **Add Property** three times:

Property 1:

- Field Name: `LATITUDE`
- Type: Number
- Required: ✓ Checked

Property 2:

- Field Name: `LONGITUDE`
- Type: Number
- Required: ✓ Checked

Property 3:

- Field Name: `TEMPERATURE_UNIT`
- Type: String
- Required: ☐ Unchecked
- Default: `fahrenheit`

Step 4: Configure Request

Click **Request** in the left menu.

Field	Value
Request Type	GET
Request URL Template	(paste below)

Paste this URL (all one line):

```
https://api.open-meteo.com/v1/forecast?latitude=${esc.url(input.LATITUDE)}&longitude=${esc.url(input.LONGITUDE)}&current=temperature_2m,weather_code,wind_speed_10m&temperature_unit=${esc.url(input.TEMPERATURE_UNIT)}&timezone=auto
```

Headers section (optional, but add for good practice):

```
Content-Type: application/json
```

Timeout: Set to `30`

Step 5: Configure Output Contract

Click **Output Contract** in the left menu.

Click **Add Property** four times:

Property 1:

- Field Name: `TEMPERATURE`
- Type: Number

Property 2:

- Field Name: `WEATHER_CODE`
- Type: Number

Property 3:

- Field Name: `WIND_SPEED`
- Type: Number

Property 4:

- Field Name: `TIMEZONE`
- Type: String

Step 6: Configure Response Parsing

Click **Response Configuration** in the left menu.

Translation Map section:

Click **Add Entry** and add these four entries:

Output Field	JSONPath Expression	Default
`TEMPERATURE`	`$.current.temperature_2m`	`0`
`WEATHER_CODE`	`$.current.weather_code`	`99`
`WIND_SPEED`	`$.current.wind_speed_10m`	`0`
`TIMEZONE`	`$.timezone`	`UTC`

Success Template section:

Scroll down. In the **Success Template** textarea, paste:

```
{
  "TEMPERATURE": ${TEMPERATURE},
  "WEATHER_CODE": ${WEATHER_CODE},
  "WIND_SPEED": ${WIND_SPEED},
  "TIMEZONE": "${TIMEZONE}"
}
```

Click **Save** (bottom of page).

Step 7: Test the Data Action

Still in the data action UI, click **Test** (top-right).

Input values:

- LATITUDE: `25.85`
- LONGITUDE: `-100.27`
- TEMPERATURE_UNIT: `fahrenheit`

Click **Run Action**.

Important: Test mode has a different timeout than production flows

In the data action **test UI**, if the action takes longer than **19 seconds**, the test will timeout and report a timeout result. This is different from production flows, where your configured timeout (e.g., 30 seconds) applies.

Why the difference? Genesys returns a test-mode report after 19 seconds so you get feedback in the UI without waiting a long time.

Expected output (Success path):

```
{
  "TEMPERATURE": 75,
  "WEATHER_CODE": 0,
  "WIND_SPEED": 8.5,
  "TIMEZONE": "America/Chicago"
}
```

 If you see this, your data action works!

If you see Failure:

- Check the URL is correct (no typos)
- Check the JSONPath expressions match the response structure
- Verify success template uses UPPERCASE field names (TEMPERATURE not temperature)

If you see Timeout:

- The API might be taking >19 seconds to respond (test mode timeout)
- Try again in 30 seconds
- In production flows, you can configure your own timeout (1-60 seconds), so the API would continue trying beyond 19 seconds

Source: https://help.mypurecloud.com/faqs/how-many-seconds-before-a-data-action-times-out/

Lab Exercise B: Use the Data Action in Architect

This part assumes you already have a basic Architect flow set up.

New to Architect? This is your first time:

```
Admin → Architect → Inbound Call Flows → New (blue button)
```

Name it: `Weather API Demo`
Inbound Route: Create a new route (leave defaults)
Queue: Default Queue (or any queue you have)

Click **Create**.

Step 1: Add the Call Data Action Step

In Architect:

```
Task Sequence → [Drag from left panel]
Find "Call Data Action" → Drag to canvas
```

Configure the Call Data Action:

- **Data Action:** Select `GetCurrentWeather` (dropdown)
- **LATITUDE:** Type `25.85`
- **LONGITUDE:** Type `-100.27`
- **TEMPERATURE_UNIT:** Type `fahrenheit`

Step 2: Bind Success Outputs

In the Call Data Action configuration, find the **Success Outputs** section.

Bind each output field to a flow variable:

- TEMPERATURE → `CurrentTemp`
- WEATHER_CODE → `WeatherCondition`
- WIND_SPEED → `WindKmh`
- TIMEZONE → `LocalZone`

(You can name these variables whatever you want.)

Step 3: Configure the Success Path

Right-click the **Success** output → **Configure**.

Add these steps in sequence:

Step 1: Play Audio

- Message Type: Text-to-Speech
- Message: "The current temperature is ${ CurrentTemp } degrees"

(Note: `CurrentTemp` is the Architect variable you bound the `TEMPERATURE` output to. If you used a different variable name, use that instead. The `${...}` syntax inserts the variable value into the prompt.)

Step 2: Transfer to Queue

- Queue: Choose any queue (e.g., "Default Queue")

Step 4: Configure the Failure Path

Right-click **Failure** → **Configure**.

Add one step:

Play Audio:

- Message: "We couldn't check the weather at this time. An agent will help you shortly."
- Then connect to Transfer to Queue (same queue)

Step 5: Configure the Timeout Path

Right-click **Timeout** → **Configure**.

Add one step:

Play Audio:

- Message: "Systems are busy. Connecting you now."
- Then connect to Transfer to Queue (same queue)

Step 6: Save and Publish

Click **Save** (top-left)
Click **Publish** (blue button, top-right)

Expected confirmation: "Flow published successfully"

Lab Exercise C: Test the Flow

Option 1: Simulate the Flow (Easiest)

In Architect:

```
Top-right menu → Debug Flow
```

This simulates the flow step-by-step. You can see:

- When the Call Data Action is called
- What values are returned
- Which output path fires

Option 2: Make a Real Call (If Phone Line Available)

If your Genesys org has an inbound phone number configured to route to this flow:

1. Call the number from your cell phone
2. Listen to the prompt
3. Observe the weather and the routing

Part 7: Understanding the Behavior in Detail

Scenario A: Good Weather, API Works

What happens:

```
Call comes in
    ↓
Call Data Action fires (30 second timer starts)
    ↓
Genesys makes HTTP GET to Open-Meteo
    ↓
Open-Meteo responds in 150ms with:
{
  "current": { "temperature_2m": 78, "weather_code": 0, ... },
  "timezone": "America/Chicago"
}
    ↓
Translation Map extracts: TEMPERATURE=78, WEATHER_CODE=0
    ↓
Success Template formats output
    ↓
SUCCESS path fires
    ↓
Flow plays "The current temperature is 78 degrees"
    ↓
Transfer to Queue
```

Duration: ~2 seconds total (150ms API + prompt playback)

Scenario B: Bad Weather, API Works

Same as Scenario A, but:

- Open-Meteo responds with weather_code=45 (foggy)
- Success path fires (API worked)
- Flow decision checks: "Is weather_code < 11?" → False
- Flow routes to Service queue instead of Sales

Scenario C: API Returns Error (404, 500, etc.)

What happens:

```
Call comes in
    ↓
Call Data Action fires
    ↓
Genesys makes HTTP GET
    ↓
Open-Meteo responds with HTTP 500 (server error)
OR HTTP 400 (bad request)
    ↓
Translation Map SKIPS (no data to extract)
    ↓
FAILURE path fires (not Success, not Timeout)
    ↓
Flow plays "We couldn't check the weather..."
    ↓
Transfer to Queue
```

Duration: ~2 seconds (API error response is fast)

Scenario D: API Timeout (Slow Network/Overloaded Server)

What happens:

```
Call comes in
    ↓
Call Data Action fires (30 second timeout window starts)
    ↓
Genesys makes HTTP GET request to Open-Meteo
    ↓
[WAITING... 5 seconds pass]
[WAITING... 10 seconds pass]
[WAITING... 20 seconds pass]
[WAITING... 29 seconds pass]
    ↓
30 second timeout FIRES
    ↓
Architect STOPS WAITING for the response
    ↓
TIMEOUT path fires immediately (not Success, not Failure)
    ↓
Flow plays "Systems are busy. Connecting you now..."
    ↓
Transfer to Queue
```

Important distinction:

When the configured timeout is reached, Architect stops waiting and proceeds down the Timeout path. This is different from an integration-service timeout (which would trigger the Failure path instead).

Why this matters:

- If you don't handle the Timeout path, callers wait for the full timeout duration before the flow exits with an error
- A short timeout (30 seconds) helps route callers faster than the default 60-second timeout
- Network or server delays are common, so the Timeout path is critical for production flows

Source: https://help.genesys.cloud/articles/add-call-data-action-data-action-task/

Duration: Caller experiences ~30 seconds wait, then the Timeout path executes.

Why This Matters

Bad design (missing Timeout path):

- Caller waits 30 seconds
- Timeout fires
- Flow doesn't know what to do
- IVR error or transfer fails
- Customer frustration

Good design (all three paths handled):

- Success: Fast, professional, data-driven
- Failure: Quick recovery, fallback routing
- Timeout: Professional message, no caller frustration

Part 8: Debugging When Things Go Wrong

Problem 1: Data Action Test Returns "Failure"

Symptoms:

- You click "Test" and see "Failure" path

Likely causes (in order):

1. **JSONPath expression is wrong**
   - The response structure doesn't match your translation map
   - Fix: In the test output, look at the raw response and adjust JSONPath
2. **Response has an error object**
   - Open-Meteo returned: `{ "error": true, "reason": "..." }`
   - This is technically a 400-level response

- Fix: Check your URL parameters (latitude, longitude, etc.)

3. **Response is wrapped in a data envelope**
   - Some APIs return: `{ "data": { "current": { ... } } }`
   - Your JSONPath expects `$.current`, but it's actually `$.data.current`
   - Fix: Adjust the JSONPath expression
4. **Success template field names don't match output contract**
   - Output contract expects `TEMPERATURE` but template has `temperature`
   - Fix: Ensure all success template field names are UPPERCASE

How to debug:

1. Run the test
2. Look at the **Raw Response** tab (should show actual JSON)
3. Copy the JSON into a JSONPath tester tool: https://jsonpath.com/
4. Test your JSONPath expressions against the actual response
5. Update your translation map

Problem 2: Data Action Test Returns "Timeout"

Symptoms:

- You click "Test" and see "Timeout" after ~19 seconds

Causes:

- The API is genuinely slow (network, server overload, geographic distance)
- In a production flow, your configured timeout may be too short for this particular API

Fix options:

1. Try the test again (might be temporary slowness)
2. In production flows, you can increase timeout to 45 seconds if needed
3. Add retry logic in your flow for Timeout path
4. Switch to a different API with better performance

Problem 3: Architect Flow Not Getting Data

Symptoms:

- Data action test shows Success with correct data
- But in Architect flow, variables are empty or wrong

Likely cause:

- You haven't bound the Success Outputs correctly
- You're referencing the wrong variable names

Fix:

- In Call Data Action configuration, verify you bound each output to a variable
- In Architect prompts, use: `${ YourVariableName }`
- In Architect decisions, use: `YourVariableName` (no ${ })

Problem 4: Response Doesn't Match Schema

Symptoms:

- Data action test returns Success, but output looks malformed
- Flow gets partial data or null values

Example (wrong):

```
{
 "temperature": "${TEMPERATURE}",
 "timezone": "${TIMEZONE}"
}
```

Problems:

- Field names are lowercase (should be TEMPERATURE, not temperature)
- Temperature is quoted (numbers should not have quotes)
- Both cause schema validation to fail

Fix:

```
{
 "TEMPERATURE": ${TEMPERATURE},
 "TIMEZONE": "${TIMEZONE}"
}
```

Strings get quotes, numbers don't. Field names must match output contract exactly.

Part 9: Real-World Variations

Variation 1: Allow Caller to Choose City

Instead of hardcoding latitude/longitude (25.85, -100.27), let the caller choose:

In Architect flow:

```
Play Audio: "Press 1 for Monterrey, 2 for Mexico City, 3 for Guadalajara"
Collect Input: Store in flow.SelectedCity
Decision: Which city?
├─ Case 1: Set flow.Lat=25.85, flow.Lon=-100.27
├─ Case 2: Set flow.Lat=19.43, flow.Lon=-99.13
└─ Case 3: Set flow.Lat=20.66, flow.Lon=-103.35
Call Data Action:
  LATITUDE = flow.Lat
  LONGITUDE = flow.Lon
  (now it's dynamic based on caller choice!)
```

Variation 2: Route Based on Temperature, Not Weather Code

Instead of weather code, use actual temperature:

In the Success path, Decision:

(Remember: use the variable names you bound the outputs to. Example shown using `CurrentTemp`):

```
If CurrentTemp > 80
  → Play "Hot weather! Connecting to Sales (outdoor focus)"
  → Queue: Sales
Else If CurrentTemp < 60
  → Play "Cold weather! Connecting to Support"
```

→ Queue: Support

Else

→ Play "Mild weather. Connecting to your requested queue"

→ Queue: General

Variation 3: Retry Logic (if API times out)

Some IVRs need weather data badly enough to retry:

Call GetCurrentWeather (timeout=30)

If Timeout:

→ Play "Checking again..."

→ Call GetCurrentWeather (timeout=30)

If Timeout Again:

→ Play "Systems busy, routing now"

→ Queue: Default

But: This risks making the caller wait 60+ seconds. Use sparingly.

Part 10: Key Takeaways

Concepts

Concept	Definition	Why It Matters
Data Action	Reusable HTTP + parsing connector	Code once, use in many flows
Input Contract	Parameters your flow provides to API	Lets you reuse the action for different cities
Output Contract	Structured fields returned to flow	Flow gets clean variables, not raw JSON
Translation Map	JSONPath to extract fields	Handles API response structure automatically
Success Template	Velocity formatting	Converts extracted fields to final output
Three Paths	Success, Failure, Timeout	Resilient IVR design

Concept	Definition	Why It Matters
Timeout	Architect stops waiting, action continues	Prevents endless waiting (concurrency implications)
Output Binding	Your choice of variable names	Flexible variable naming in Architect

Skills Acquired

- Create a data action from scratch
- Configure input and output contracts
- Write JSONPath expressions (translation map)
- Write Velocity templates (success template)
- Handle all three data action output paths
- Bind outputs to Architect variables
- Use data action output in Architect flows
- Debug when things go wrong
- Understand API performance and resilience

Production Checklist

Before deploying to production:

- ☐ Test data action 10+ times (success, failure, timeout)
- ☐ Verify timeout is reasonable (not too short, not too long)
- ☐ Handle all three output paths (success/failure/timeout)
- ☐ Add monitoring/logging (optional, but recommended)
- ☐ Document the flow (what data is being called, why)
- ☐ Test during high-traffic times (does API still respond fast?)
- ☐ Have a fallback (what if API goes down permanently?)

Part 11: Velocity Template Details (Advanced)

Escaping Special Characters

If your API response contains quotes or special JSON characters, you need to escape them.

Example (bad):

```
API returns: {"message": "Hello \"World\""}
```

If you paste this directly into the success template:

```
{
  "text": "${message}"
}
```

You get invalid JSON with unescaped quotes.

Solution: Use `$esc.jsonEncode()`

```
{
  "text": "$esc.jsonEncode(${message})"
}
```

Important Velocity macros:

- `$esc.jsonEncode($variable)` — Escapes quotes and special chars for JSON
- `$esc.jsonDecode($variable)` — Unescapes JSON-encoded strings
- `$esc.html($variable)` — Escapes for HTML (if sending to web)
- `$esc.url($variable)` — Escapes for URL parameters

When do you need this?

- API response has text with quotes: `"message": "Say \"Hi\" now"`
- API response has newlines in strings
- API response has backslashes

For our weather API, you probably don't need it (numbers and simple strings). But good to know!

Part 12: JSONPath Reference

Common JSONPath Expressions

Expression	Extracts	Example

`$.field`	Top-level field	`$.timezone` → `"America/Chicago"`
`$.object.field`	Nested field	`$.current.temperature_2m` → `78`
`$.array[0]`	First array element	`$.hourly.time[0]` → `"2026-03-14T00:00"`
`$.array[*].field`	Field from all array elements	`$.hourly.time[*]` → `["2026-03-14T00:00", "2026-03-14T01:00", ...]`

For Our Weather API

What You Want	JSONPath	Result
Current temperature	`$.current.temperature_2m`	`78`
Weather code	`$.current.weather_code`	`0`
Timezone	`$.timezone`	`"America/Chicago"`
Wind speed	`$.current.wind_speed_10m`	`8.5`
Latitude	`$.latitude`	`25.85`

Pro tip: Use https://jsonpath.com/ to test JSONPath expressions against actual API responses.

Part 13: Interview Preparation

Common Interview Questions

Question	Answer
What are the three data action output paths?	Success (API worked), Failure (error returned), Timeout (Architect stopped waiting)
What is a translation map?	JSONPath expressions that extract fields from raw API response
What is a success template?	Velocity template that formats translation map output into final response
How do you pass data from Architect to a data action?	Via the Input Contract (LATITUDE, LONGITUDE, etc.)
How do you get data back from a data action in Architect?	Bind the Success Outputs to Architect variables in the Call Data Action step, then reference those bound variables in your flow

Question	Answer
What's the difference between Failure and Timeout?	Failure = API returned error (4xx/5xx) or response parsing failed; Timeout = Architect stopped waiting after configured timeout, but the data action still executes in background
Why handle all three paths?	Best practice for resilience and professional caller experience. Success path handles happy path, Failure path handles errors gracefully, Timeout path prevents caller from waiting indefinitely
How do you test a data action?	Use the "Test" button in the data action UI, provide input values, observe output
What does JSONPath `$.current.temperature_2m` mean?	Go to root `$`, then `current` object, then `temperature_2m` field
How do you escape URL parameters?	Use `$esc.url()` syntax: `${esc.url(input.PARAMETER_NAME)}`

Hands-On Questions

Question 1: You're given a new API response. Write the JSONPath to extract the `temperature` value from this structure:

```
{
  "location": {
    "forecast": {
      "temperature": 72
    }
  }
}
```

Answer: `$.location.forecast.temperature`

Question 2: Your flow calls a data action. It times out 5% of the time. What should you do?

Answers (in order of preference):

1. Check if the API is slow at certain times (monitor data action metrics)
2. Increase timeout from 30 to 45 seconds
3. Add retry logic in the Timeout path
4. Contact the API provider about performance

Question 3: Your data action test shows Success, but the output is missing fields. What could be wrong?

Answers:

1. Translation map JSONPath is pointing to wrong fields
2. API response structure changed (test with raw response visible)
3. Success template has syntax errors (lowercase field names, wrong quoting)
4. Output contract fields don't match translation map entries

---

Part 14: Reference Links

Official Genesys Cloud Documentation

- **Data Actions Overview:** https://help.genesys.cloud/articles/about-call-data-actions/
- **Call Data Action:** https://help.genesys.cloud/articles/call-data-action/
- **Response Configuration:** https://help.genesys.cloud/articles/response-configuration-data-actions/
- **Velocity Macros:** https://help.genesys.cloud/articles/velocity-macros-data-actions/
- **Request Configuration:** https://help.genesys.cloud/articles/request-configuration-data-actions/
- **Architect Failure Paths:** https://help.genesys.cloud/articles/failure-paths-architect/
- **Test Data Actions:** https://help.genesys.cloud/articles/test-data-actions-integrations/
- **Data Action Timeout:** https://help.genesys.cloud/faqs/how-many-seconds-before-a-data-action-times-out/

Open-Meteo API Documentation

- **Main API Docs:** https://open-meteo.com/en/docs
- **GitHub Repository:** https://github.com/open-meteo/open-meteo

Helpful Tools

- **JSONPath Tester:** https://jsonpath.com/
- **JSON Formatter:** https://jsonformatter.org/
- **Velocity Template Language:** https://velocity.apache.org/

Document Metadata

Scenario: 1 of 12
Title: Simple Weather API Call

Skill Level: Beginner to Intermediate
Time to Complete: 45-60 minutes (including lab)
Last Updated: March 2026

END OF SCENARIO 1

Revision #3
Created 15 March 2026 01:18:19 by Cesar Gzz
Updated 15 March 2026 03:10:50 by Cesar Gzz