Salesforce Agent & Rover Integration Workshop

Unlocking the Power of Data + AI with Salesforce Agents

This workshop provides hands-on instructions for setting up a Salesforce Agent to interact with a physical device, "Rover," simulating a smart recycling sorting process.

Workshop Objectives

- Verify the connection and functionality between the Salesforce environment and the Rover device.
- Configure Salesforce Agent features, including a new Prompt Template, Case Classification, and an Employee Agent.
- Understand the physical and software components of the Rover device (e.g., sensors and Python scripts).
- Test the end-to-end process by creating and classifying service cases in Salesforce.

Test and Verify "Rover" is Connecting and Working with Device

Before proceeding, ensure the physical Rover device is powered on, connected to the network, and the integration service (not detailed here) is running to allow communication between Salesforce and the device.

Verification Step	Expected Outcome	Status
Rover Power On	Status LED is green/solid	
Network Connection	Rover IP is reachable	
Integration Service	Service log shows "Connected to Rover"	

Initial Setup:

Pybricks Firmware Installation

Steps:

- Go to https://code.pybricks.com
- Click the Bluetooth icon, select your hub, and install
- If Bluetooth scanning doesn't work:
  - Go to `chrome://flags`
  - Enable Experimental Web Platform Features
- If Bluetooth still fails:
  - Hold hub Bluetooth button
  - Plug USB cable into laptop (flashing pink → green → blue)
- On Windows 11, you might need the WinUSB driver via Pybricks troubleshooting wizard

---

Laptop Setup

Create environment & install dependencies

```
python3 -m venv venv
```

```
source venv/bin/activate
pip install -r requirements.txt
```

Required packages

- pybricksdev
- grpcio
- fastavro
- aiohttp

- requests
-

https://github.com/mikeletulle/LegoWorkshop/

Salesforce Admin Configuration

1. Ensure Agents are Enabled

1. In Salesforce Setup, use the Quick Find box to search for **Einstein Setup**.
2. Toggle the switch to **Turn on Einstein** (if not already enabled).
3. Use the Quick Find box to search for and select **Agentforce Agents**.
4. Toggle the switch to **Turn on Agentforce**.
5. Ensure the **Agentforce (Default) Agent** is enabled.

2. Create Brand New Prompt Template

This template is of type Flex and will be used later.

1. In Salesforce Setup, search for **Prompt Builder**.
2. Click **New Prompt Template**.
3. Set the **Name**: Support Classify the case.
4. Set the **API Name**: Support_Classify_the_case.
5. Select **Prompt Template Type**: Flex.
6. **Instruction Text**: Update this with the exact content later.
7. Click **Save**.

3. Create New Field "Support Classification"

Create a custom picklist field on the Case object for classification.

1. In Salesforce Setup, navigate to **Object Manager** > **Case** > **Fields & Relationships**.
2. Click **New** to create a custom field.
3. Select **Picklist** for the Data Type.
4. **Field Label**: Support Classification.

5. **Field Name**: Support_Classification.
6. Enter the following values, each on a new line:
    - RECYCLING_OK
    - CONTAMINATED
    - INSPECTION
7. Set field-level security and add the field to the necessary page layouts.
8. Click **Save**.

5. Creation of a new platform Events

Search for Platform Events in the setup

LEGO_Command__e

Field	Purpose
`Command__c`	`RECYCLING_OK`, `CONTAMINATED`, `INSPECTION`
`Case_Id__c`	originating Case ID

LEGO_Robot_Status__e

Field	Purpose
`Case_Id__c`	echoed ID
`Message__c`	e.g., “Target Zone Reached”

---

5. Create Workflow to Publish Platform Event

This workflow simulates the Rover itself publishing an event to Salesforce when an action is completed, such as a sorting error. We will assume a platform event `Rover_Action__e` exists.

1. In Salesforce Setup, search for **Flow**.
2. Click **New Flow**.
3. Select **Record-Triggered Flow** and click **Create**.
4. **Object**: Case.
5. **Trigger the Flow When**: A record is created or updated.
6. **Entry Conditions**: `Support_Classification__c` equals 'CONTAMINATED'.
7. **Optimize the Flow For**: Actions and Related Records (After Save).
8. Add an **Action** element.
9. **Action Type**: Platform Event > `Rover_Action__e`.
10. **Set Field Values**: `Action_Type__c` = 'Contamination_Detected'.
11. **Set Field Values**: `Case_ID__c` = Case.Id.
12. **Label**: Publish Contamination Event.
13. Click **Save** and **Activate**.

5. Create New Employee Agent (Topic & Action Creation)

Create a dedicated agent to handle employee inquiries about the Rover.

1. In Salesforce Setup, search for **Agentforce Agent**.
2. Click **Open in Builder** for **Einstein Copilot**.
3. Create a **New Topic**.
    - **Name**: Employee Rover Guide.
    - **Classification Description**: Provides guidance to employees about the Rover device, including status checks and how to report issues.
    - **Scope**: Only answer questions about the Rover's operational status and guide employees on case creation.
    - **Instructions**: Use the `Support_Classify_the_case` template for all status inquiries. Always ask the employee for the device ID.
4. Add the **Action**:
    - Click **New Action**.
    - **Action Name**: Check_Rover_Health. (This assumes a custom action/Apex is configured to retrieve the Rover health status data).
5. Link the `Support_Classify_the_case` template to this Topic.
6. Click **Save** and **Activate** the agent.

6. Test Employee Agent

1. Open the Service Console.
2. Initiate a new conversation with the Employee Agent.
3. **Test Utterance**: "What is the status of Rover ID `Person`?"
4. Verify the agent uses the custom prompt template and references the (simulated) status.

7. Connected App (External Client App)

Navigate to External Client App Manager in the Setup

External Client App Name: Lego Agent Bridge

Contact Email
Distribution State: Local

Enable oAuth Setting:
Callback URL: http://localhost:8080/callback

Scopes:

- Manage user data via APIs (`api`)
- Full access (`full`)
- Refresh tokens (`refresh_token`, `offline_access`)
- Access the Salesforce API Platform (`sfap_api`)

Uncheck all the checkboxes in **Flow Enablement and Security** section

Rover Configuration

1. Explain Parts of Rover

The Rover is comprised of several key components:

- **Main Processor/Controller**: The brain of the Rover, running the Python scripts.
- **Motor/Actuators**: Components responsible for movement and sorting arm operation.
- **Distance Sensor**: Used to detect the presence of an object in the sorting area.
- **Color Sensor**: Used to identify the type of material (e.g., plastic, glass, metal).
- **Sorting Mechanism**: The physical arm/gate that directs the item into the correct bin.

2. Add Distance Sensor and Color Sensor

Physically connect the following sensors to the main controller:

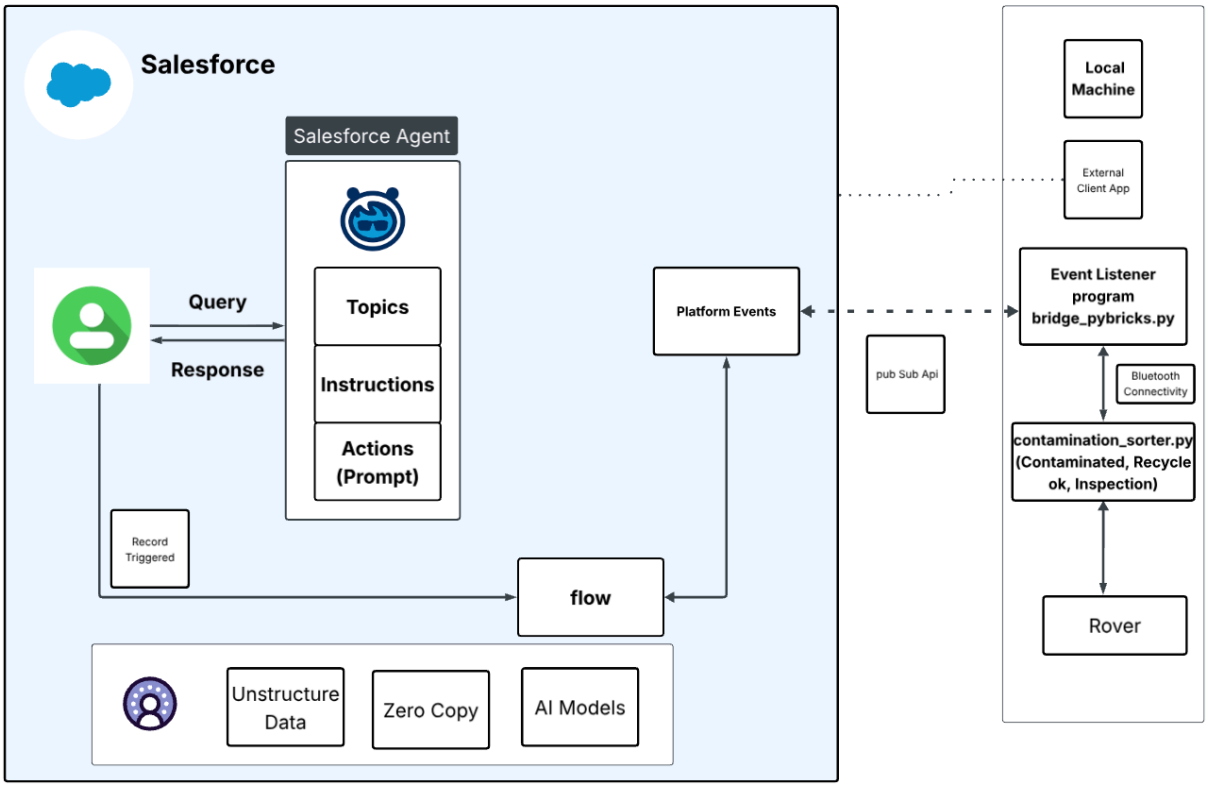
Sensor Type	Port	Notes

Color Sensor	Port B	Mounted downward on the black grabber arm to detect colored zones (see photo).
Ultrasonic Sensor	Port F	Detects obstacles and allows stopping ~2 inches before the box.

3. How It Works – Architecture Overview

1. Salesforce
    - Agent or automation publishes a *LEGO_Command__e* event:
        - `Command__c` = `RECYCLING_OK`, `CONTAMINATED`, or `INSPECTION`
        - `Case_Id__c` = the originating Case Id
    - The robot returns a *LEGO_Robot_Status__e* event:
        - `Case_Id__c`
        - `Message__c` (e.g., “Target Zone Reached”)
2. Python Bridge (`bridge_pybricks.py`)
    - Subscribes to Salesforce Pub/Sub
    - When a command arrives:
        - Runs `pybricksdev run ble contamination_sorter.py`
        - Parses `STATUS:` output lines
        - Publishes a Robot Status PE back to Salesforce
3. Pybricks Program (`contamination_sorter.py`)
    - Uses reflection values, not color names
    - Detects RED / YELLOW / GREEN zones
    - Drives to the appropriate zone
    - Prints `STATUS:` lines that the bridge forwards to Salesforce
    -

Architecture diagram



4. Run Color Calibration

The color sensor requires calibration for accurate color identification as it depends on the intensity of the light in the room.

Use color_calibrater.py to measure reflection values on each zone. pybricksdev run ble color_calibrater.py and notice REF values on each color and replace in contamination_sorter.py as shown here:

Example classifier

```
# ========================================
# --- COLOR CALIBRATION VALUES (EDIT HERE) ---
# ========================================
CAL_RED    = 5
CAL_GREEN  = 11
CAL_YELLOW = 13.5  # Range 13-15

# Filter out readings that are way off
```

5. Review All Python Scripts

Review the following scripts on the Rover's controller:

- **main_loop.py**: The primary script that runs the sorting logic. It continuously polls the distance sensor and calls `sorter_scripts.py`.
- **integration_client.py**: Handles communication with the Salesforce platform, sending status updates and receiving commands.
- **sensor_reader.py**: A library containing functions to read data from the Distance and Color sensors.

5. Review Sorter Scripts

Examine the material identification and sorting logic within `sorter_scripts.py`:

```
None
```

Running the System

Start bridge:

```
python bridge_pybricks.py
```

Send command via Salesforce Debug Anonymous (Apex).

Test with Anonymous Apex

```
LEGO_Command__e eventRecord = new LEGO_Command__e(
    Command__c = 'INSPECTION',
    Case_Id__c = 'Your SF CASE ID'
);
Database.SaveResult sr = EventBus.publish(eventRecord);
System.debug('SR: ' + sr);
```

Testing End to End:

Create the following cases in the Salesforce Service cloud to test the end-to-end agent and classification flow.

Scenario	Case Subject	Support Classification (Expected)
Recycling_OK	Rover has successfully completed a large batch of sorting	RECYCLING_OK
Contamination	Rover detected an unknown item and is currently paused	CONTAMINATED
Inspection	Rover needs routine maintenance and sensor check	INSPECTION

1. **Create 3 different service cases with 3 different scenarios for Recycling_OK, Contamination, Inspection**
    - Open the **Service cloud** application.
    - Create 3 new cases with following fields:
    - ----------
    - Subject: Prohibited Waste / Hazardous Material Alert
    - With Description like "Driver alerted the dispatcher that the contents of the last commercial pickup (Site ID: 900 Industrial Park, Asset ID 55112 - 8-yard compactor) appears to be highly contaminated. The load contains multiple sealed 5-gallon buckets marked with chemical warning labels (possibly paint thinner or solvent) and several discarded fluorescent light tubes. The material has been flagged and isolated at the yard. Requires hazmat team notification and segregation for special disposal."
    - ----------
    - Subject: Missed Collection (Residential)
    - With Description like ""Customer called stating their gray residential waste cart (Asset ID 10045) was missed on the scheduled route yesterday morning. They confirmed the bin was placed out by 7:00 AM and accessible at the curb. We verified that no exception tag was left. The bin is full and needs to be emptied ASAP. Issue is a standard missed pickup requiring a make-up collection."
    - "
    - ----------

- Subject: Access Problem / Unidentified Debris
- With Description like “Driver reported being unable to service the commercial property at 450 King Street for the past three weeks. The assigned bin is behind a newly installed chain-link fence. We have no key or access code on file. Driver noted that the area is very congested and suggests a smaller bin size might be required if the current location is permanent. Needs a supervisor site inspection to assess access requirements and potentially relocate the asset."
-

Running the System

Start bridge:

```
python bridge_pybricks.py
```

Send command via Salesforce Debug Anonymous (Apex).

Project Files

- `bridge_pybricks.py`
- `salesforce_pubsub.py`
- `contamination_sorter.py`
- `color_calibrater.py`
- `sf_login.py`
- `sf_config.json` (local only)
- `robot_board.png`
- `sensor_placement.png`