Leveraging Custom Code Actions in HubSpot Workflows

Within HubSpot workflows, the Custom code action allows you to write and execute JavaScript or Python (in beta). This feature extends workflow capabilities both within and outside of HubSpot. For more details on HubSpot's APIs, refer to the latest or legacy developer documentation. Examples of common custom code actions can be found in HubSpot's Programmable Automation Use Cases.

Custom code actions use the Node 16.x runtime for JavaScript and Python 3.9 runtime for Python, with compute managed via serverless functions by HubSpot and AWS Lambda.

For general issues, contact HubSpot support. For code-specific problems, use the HubSpot Developer's Forum for troubleshooting tips and advice.

Node.js Supported Libraries

For Node.js custom code actions, the following libraries are available and can be loaded using the require() function:

- `@hubspot/api-client ^10`
- `async ^3.2.0`
- `aws-sdk ^2.744.0`
- `axios ^1.2.0`
- `lodash ^4.17.20`
- `mongoose ^6.8.0`
- `mysql ^2.18.1`
- `redis ^4.5.1`
- `request ^2.88.2`
- `bluebird ^3.7.2`
- `random-number-csprng ^1.0.2`
- `googleapis ^67.0.0`

These libraries enable various functionalities such as API interactions, data manipulation, and database connectivity.

Python Supported Libraries

For Python custom code actions, you can load the following libraries using an import statement formatted as `from [libraryname] import [item]`, such as `from redis.client import Redis`:

- `requests 2.28.2`
- `@hubspot/api-client ^8`
- `google-api-python-client 2.74.0`
- `mysql-connector-python 8.0.32`
- `redis 4.4.2`
- `nltk 3.8.1`

Additionally, standard library modules can be imported normally, such as `import os`.

Create Custom Code Options

To incorporate a custom code action into a workflow:

1. In your HubSpot account, go to **Automation** > **Workflows**.
2. Select an existing workflow or create a new one.
3. Click the **+ plus** icon to add a workflow action.
4. In the right panel, choose **Custom code**.

Setting Up Your Custom Code Action

1. In the right panel, configure your action:
    - By default, custom code actions use Node.js 16.x. To use Python (if in the beta), select Python from the Language dropdown menu.
2. To add a new secret (e.g., a private app access token):
    - Click **Add secret**. Ensure the app includes the necessary scopes (e.g., contacts, forms).
    - Enter the Secret name and Secret value in the dialog box and click **Save**.
    - To manage existing secrets, click **Manage secrets**.
3. To include properties in your custom code:
    - Click **Choose property** and select a property. Enter a Property name to use in your code.
    - To add more properties, click **Add property** (each must be unique and can be added only once, up to 50 properties).
    - To delete a property, click the delete icon.
4. Enter your JavaScript or Python in the code field.
5. To define data outputs for later workflow actions:
    - Under **Data outputs**, select the data type from the dropdown menu and enter a name for the data output.
    - To add more outputs, click **Add output**.
6. Click **Save** to finalize your setup.

Important Considerations for Custom Code Actions

- **Function Execution**: The `def main(event):` function is called when the code snippet action is executed.
- **Event Argument**: The `event` argument contains details about the workflow execution.
- **Callback Function**: In Node.js, use the `callback()` function within `exports.main` to pass data back to the workflow. This function is essential for returning output data and should be utilized appropriately.

Testing the Custom Code Action

To ensure your custom code runs as expected, test the action before activating the workflow. Start by selecting a record to test the code with, then execute the code. This test runs only the custom code, not other workflow actions.

Note: Changes will apply to the selected test record, so use a dedicated test record to avoid altering live data.

Steps to Test:

In the workflow timeline, click the custom code action.

At the bottom of the right sidebar, click **Test action** to expand the testing section.

To test your code, choose a record by clicking the [Object] dropdown menu and selecting the desired record.

If you're using previously formatted property values in the workflow, enter a **test value** for the formatted data.

Click **Test** to execute the code.

In the confirmation dialog, click **Test** again to confirm testing against the selected record.

After execution, the sidebar will display:

- **Status**: Indicates whether the custom code action succeeded or failed.
- **Data Outputs**: Shows the values generated for the defined outputs. Alerts will appear next to any undefined outputs, which need to be added for use later in the workflow.
- **Logs**: Provides details about the test, including memory usage and total runtime.
- To update your custom code action, click Create action to expand the action editor. Continue to update and test your code as needed.

When you're done testing the action, click Save to save your changes.

Caveats

When using Node.js for custom code actions, keep the following in mind:

- **Random Number Generation**: Using `Math.random` can result in identical numbers across executions due to time-based seeding. Instead, use the `random-number-csprng` library for secure, random number generation.
- **Variable Re-use**: Variables declared outside the `exports.main` function may be reused in future executions. To ensure unique logic or information per execution, declare such variables within `exports.main`.

For Python custom code actions:

- **Variable Re-use**: Similar to Node.js, variables declared outside `def main` may be reused. If altering a variable, declare it within `def main` with the `global` keyword.