Public Preview: UiPath Robot JS SDK

User Guide

Contents

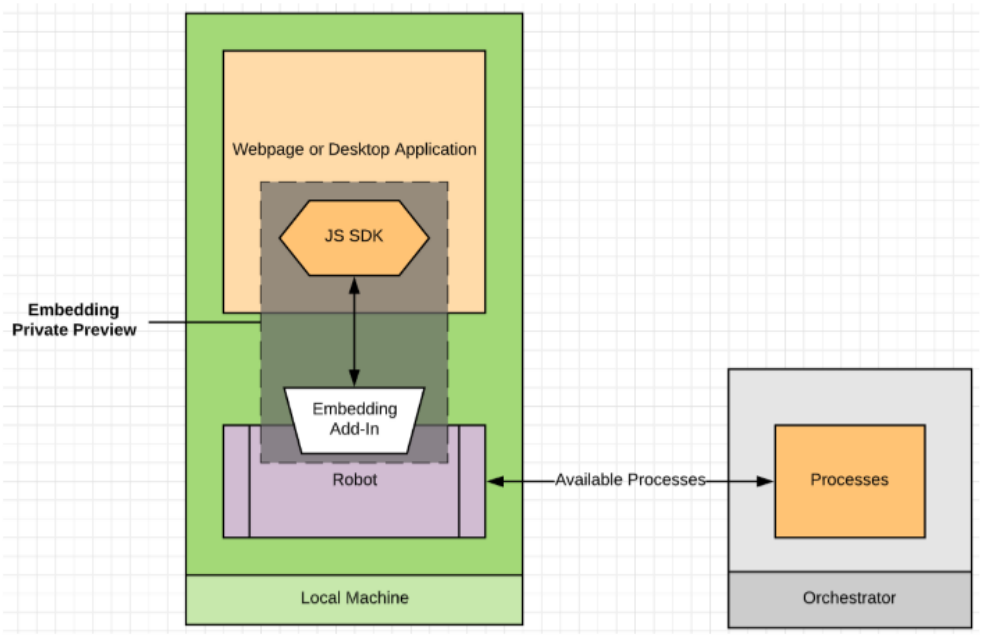
Overview

To run attended automation today, users must know which process to click in their tray – and are unable to specify inputs to those processes or do anything with the output. Robot.js unlocks the ability to embed UiPath processes into your desktop applications and web-apps in the browser with a simple JavaScript SDK. It also enables developers to create business-user-friendly interfaces to run processes in context of their work.

Technical Overview

How does it work? The Robot Add-In exposes a secure communication layer between your web browser and UiPath's Robot. This preview has two components:

- Desktop Installer for the JavaScript Add-on for Robot: An Add-in for UiPath's Robot
- JavaScript SDK: A library to include in your web or desktop application



This is a better experience for developers and for business users running attended automation. In addition, Embedding has the following benefits:

- There is no need to associate user with a specific robot to run attended automation.
- Run a process based on the context of your business application and act on the process result. a. Pass in input arguments to your process and get back output arguments
- Speed. Everything is happening locally!
- Status updates: Provide progress/status updates from a process inline in your applications

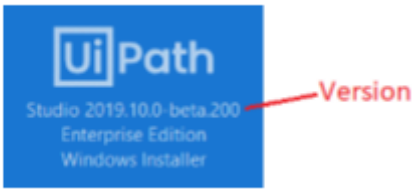
Pre-requisites

Processes and robots in Orchestrator for the purposes of this example, we'll run process that are in your Cloud Orchestrator, but any orchestrator will work.

- Begin by navigating to https://platform.uipath.com
- Next, create a service instance by navigating to the Services tab in the menu on the left and clicking on the Add service button.
- Finally, add your robots and publish your processes to the UiPath Cloud Platform.

This preview requires Studio and Robot that are at least 19.10 or later. To check your version, follow the steps below.

Check the version by launching UiPath studio. On the bottom left side of the start screen, the version and install type will be displayed:



If your version of studio is 19.10 or greater, then continue. Otherwise downloaded the latest version from your Cloud Portal Resource Center

Installation Steps

Visit https://robotjs.uipath.com/download to download the JavaScript Add-on for Robot

Using the Robot.js sample

For the purposes of this example, we've provided a sample process and web app to showcase Robot.js - Download and extract the "Boilerplate Sample" from https://robotjs.uipath.com/samples

- Open the 'Workflow folder and using UiPath Studio
- Publish the package to your Cloud Orchestrator
- Add the process it in your Robot's environment.

Once you have your robots and processes in Cloud Orchestrator, we can run the sample.

For security reasons, the sample web app must be hosted on a web server. If you don't have a web server to deploy to, or just want to try the example locally, you can host the sample via localhost:

Running the sample locally

There are many ways to host a localhost server and you should use whatever method you are most comfortable with. Here are two common methods:

Localhost server via Node Install the http-server module via npm

> npm install -g lite-server

Then from the root of your sample's directory run:

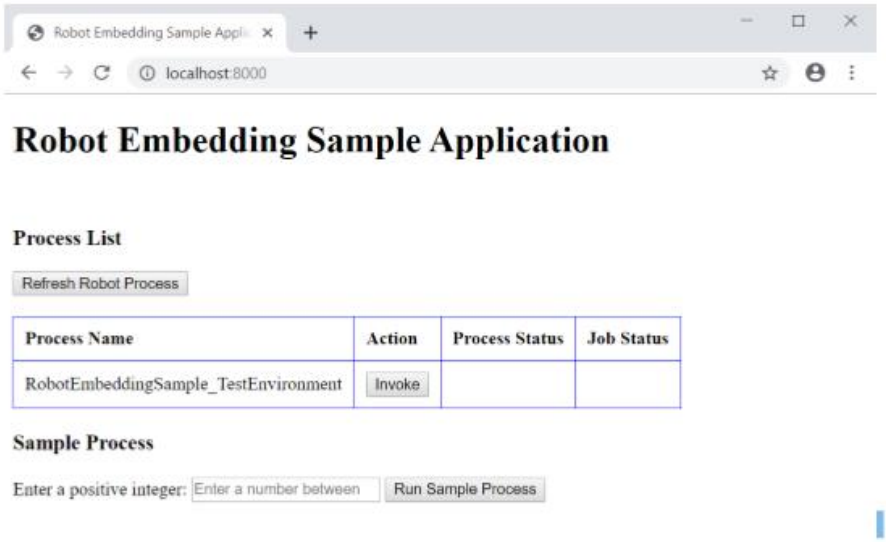
> lite-server .

Localhost server via Python 2.0 Then from the root of your sample's directory run:

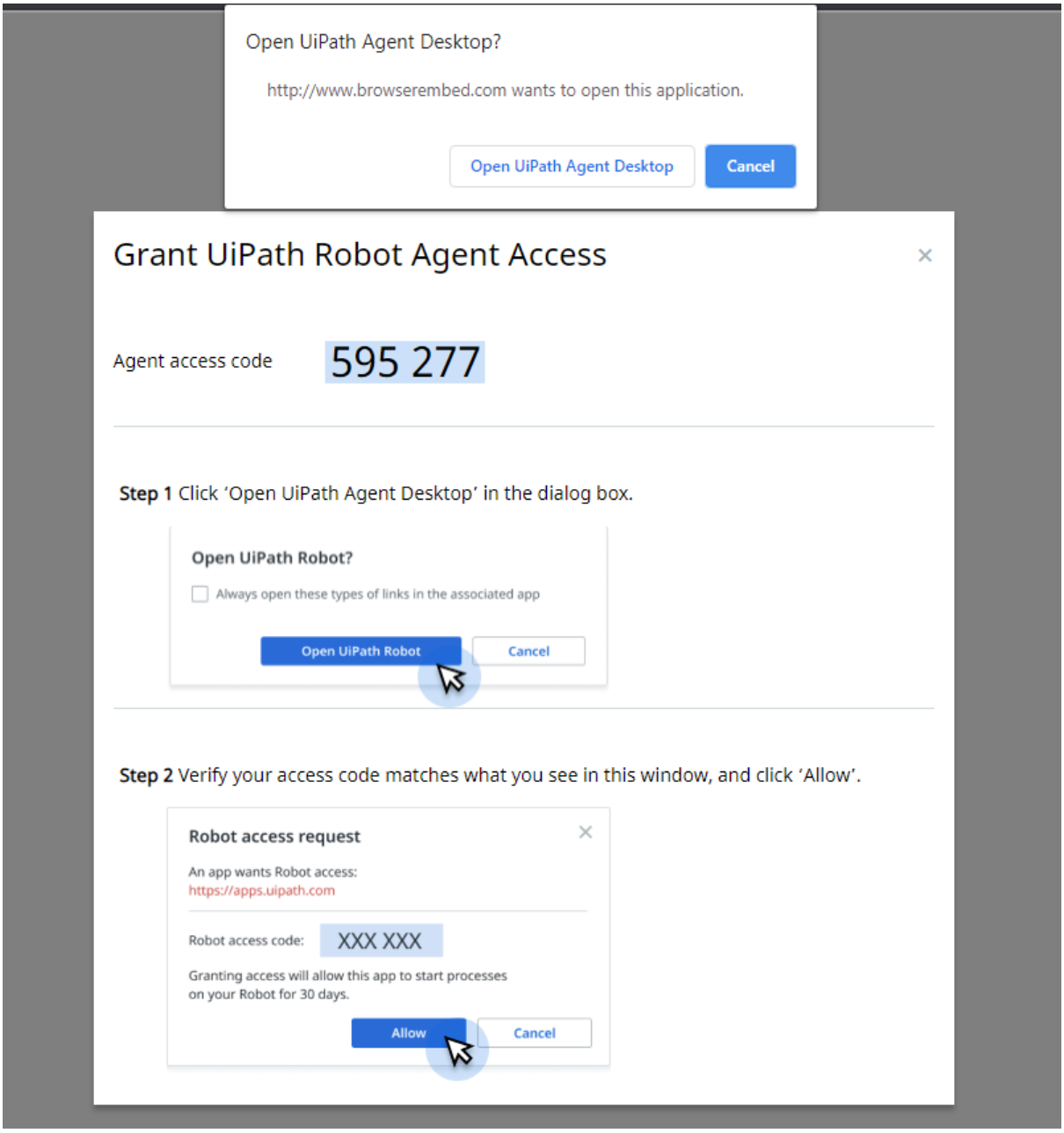
> python -m SimpleHTTPServer

Launch and Test

- Launch the hosted version of the sample (on the web or on localhost using your selected port).

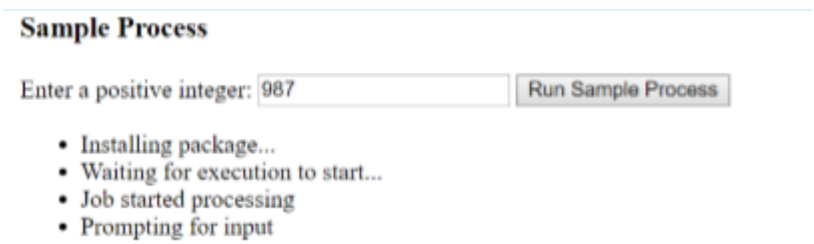


- You should see a prompt to launch your local Robot. Follow the instructions on screen and grant access. This will allow your website to communicate with the local robot, without having to prompt for consent again for 30 days.



After granting consent, you should see the RobotEmbeddingSample in the process list.

- Enter a number and click 'Run Sample Process'.
  - The process will start locally
  - You will see a status message that says "Prompting for input"



  - The process will open a popup. Enter a number into that popup and press enter.

- The sample app will show the sum of the two numbers.

Usage Guide

This section is intended for developers integrating Robots into their applications. For the full documentation see SDK Specifications.

Including the SDK

To use Robot.js, you must first download and include the JavaScript SDK. For the purposes of this example, the SDK is included in the sample web app. You can download the latest version of the SDK here: UiPathRobot.js

<script src=" https://download.uipath.com/js/1.1.1/UiPathRobot.js"/>

Tips

- For security reasons, your web app must be hosted on a web server (or localhost).
- The local robot can not be accessed via a static HTML file.
- Your consent token is stored in the browser and cannot be modified by JavaScript.
- To see the consent window again clear out your cookies and refresh your webpage.
- Arguments being passed into the process are case sensitive.
- Arguments that are Complex types must be passed in using a JSON serialized version of the .NET object (see Orchestrator API documentation for more details)

Known issues

- If your process updates its status too frequently, some status updates might be lost. Mitigation: try increasing your polling rate, eg: UiPathRobot.init(100)
- Making a call on the SDK before window.onload has run will result in an error.
- Every time a process starts, you will see the get status messages:
  - Process is available
  - Installing package
  - Status: Job started processing

SDK Specifications

UiPath javascript library lets you run robot processes from web pages. It is also available as a standalone javascript file on below CDN path.

Versioned:
<script src="https://download.uipath.com/1.1.0/UiPathRobot.js"/>

Latest:
<script src="https://download.uipath.com/Latest/UiPathRobot.js"/>

Also available in NPM:
```
npm install -save @uipath/robot
```

List of methods and properties accessible.

Init

```
/**
* Init method.
* @returns {IRobotSDK} instance
*/
init(): IRobotSDK;
```

Optional init method which returns `IRobotSDK` instance enabling developers to store it as a variable for further use.

Usage:

```
const robot = UiPathRobot.init();
```

Get Processes

```
/**
* Method to retrieve all published robot processes on users local machine.
* @returns Deferred promise of type RobotProcess[] which will be resolved/rejected based on http response.
*/
getProcesses(): Promise<Array<RobotProcess>>;
```

Method to retrieve all published robots on users local machine. Returns a promise which contains the array of all the published robot processes on the machine.

Usage:

```
// Prints all published robots to browser console

const robot = UiPathRobot.init();
robot.getProcesses()
.then(result => {
    for (let i = 0; i < result.length; i++) {
        console.log(result[i].name);
    }
}, err => {
  console.log(err);
});
```

Start Job

```
/**
* Method to invoke a robot process.
* @param job Job object containing all information about the robot process to run.
* @returns Deferred promise which is resolved with job result when robot process completes.
*/
startJob(job: Job): Promise<JobResult>;
```

Method to run a robot process by passing robot process id and in-arguments if any. Returns a promise which will be resolved once the robot process completes execution.

Usage:

```
// Example 1 - Start a job using process id
let arguments = {
    "input1" : 23,
    "input2" : 23,
    "operation" : "add"
};
const robot = UiPathRobot.init();
robot.getProcesses()
.then(processes => {
    let calculatorProcess = processes.find(p => p.name.includes('Calculator'));
    let job = new Job(calculatorProcess.id, arguments);
    robot.startJob(job).then(result => {
        console.log(result.Sum);
    }, err => {
        console.log(err);
    })
}, err => {
  console.log(err);
});
```

On

```
/**
* Method to attach event handlers on the SDK.
* @param eventName Available SDK events are 'consent-prompt', 'missing-components'.
* @param eventHanlder Event handler callback function called when event occurs.
*/
on(eventName: string, callback: (argument?: any) => void): void;
```

Method to attach event handlers on the SDK. SDK has an inbuilt consent overlay shown every time consent is required. Such events can be overridden with a custom handler of developers choice.

Usage:

```
// Example -
// Instead of showing default consent overlay, the consent code will
// be written to the console.
// Instead of showing error overlay when required components are missing,
// the error is shown on the console

const robot = UiPathRobot.init();
robot.on('consent-prompt', (consentCode) => { console.log(consentCode) });
robot.on('missing-components', () => { console.log('Missing components') });
```

Process.Start

```
class RobotProcess {

/**
 * Start the robot process
 * @param inArguments JSON object. In arguments to be passed to the robot if any.
 */
start: (inArguments?: any) => JobPromise;

}
```

Method to start a robot process by passing in-arguments if any. Returns a promise which will be resolved once the robot process completes execution.

Usage:

```
// Example - Start a process
let arguments = {
  "input1" : 23,
  "input2" : 23,
  "operation" : "add"
};
const robot = UiPathRobot.init();
robot.getProcesses()
.then(processes => {
      let process = processes.find(p => p.name.includes('Calculator'));
      process.start(arguments).then(result => {
            console.log(result.Sum);
      }, err => {
            console.log(err);
      })
}, err => {
    console.log(err);
});
```

Process.Start.OnStatus

```
class JobPromise {

  /**
      * Method to attach job status event handlers.
      * @param eventHanlder Attaches callback which is invoked on job status change.
      */
  onStatus: (eventHanlder: (argument?: any) => void) => this;

}
class RobotProcess {
  start: (inArguments?: any) => JobPromise;
}
```

Method to start a robot process and attach status callbacks by passing callback handler. Returns a promise which will be resolved once the robot process completes execution.

Usage:

```
// Example - Start a process and attach a status callback
let arguments = {
    "input1" : 23,
    "input2" : 23,
    "operation" : "add"
};
let globalHandler = (status) => { console.log(status) };
const robot = UiPathRobot.init();
robot.getProcesses()
.then(processes => {
  let statusHandler = (status) => { console.log('Calculator Process Status -' + status) };
    let process = processes.find(p => p.name.includes('Calculator'));
    process.start(arguments)
    .onStatus(globalHandler)
    .onStatus(statusHandler)
    .then(result => {
        console.log(result.Sum);
    }, err => {
        console.log(err);
    })
}, err => {
  console.log(err);
});
```

Job.On

```
class Job {

  /**
      * Method to attach event handler on the job.
      * @param eventName Available job events are 'status'.
      * @param eventHanlder Event handler callback function called when event occurs.
      */
   on(eventName: string, eventHanlder: (argument?: any) => void): void;

}
```

Method to attach an event handler to get on going process statuses.

Usage:

```
// Example - Start a job using process id and attach a callback to
// get ongoing process status. All statuses from 'Report Status' activity is
// passed along to the SDK.
let arguments = {
     "input1" : 23,
     "input2" : 23,
     "operation" : "add"
};
const robot = UiPathRobot.init();
robot.getProcesses().then(processes => {
     let calculatorProcess = processes.find(p => p.name.includes('Calculator'));
     let job = new Job(calculatorProcess.id, arguments);
   job.on('status', (robotStatus) => { console.log(robotStatus) });
     robot.startJob(job).then(result => {
          console.log(result.Sum);
     }, err => {
          console.log(err);
     })
}, err => {
   console.log(err);
});
```

Settings

```
class Settings {
   portNumber: number;
   pollTimeInterval: number;
}
```

Settings property on the SDK to edit any default settings

Usage:

```
// Example
const robot = UiPathRobot.init();
robot.settings.portNumber = 1234; // Default port number is 2323
robot.settings.pollTimeInterval = 1000; // Default poll interval time is 250ms
```

Model Definitions

Settings

```
class Settings {
    portNumber: number;
    pollTimeInterval: number;
}
```

RobotProcess

```
class RobotProcess {
    id: string;
    name: string;

    constructor(id: string, name: string);

    start: (inArguments?: any) => JobPromise;
}
```

IRobotSDK

```
interface IRobotSDK {
    settings: Settings;
    getProcesses(): Promise<Array<RobotProcess>>;
    init(): IRobotSDK;
    on(eventName: string, callback: (argument?: any) => void): void;
    startJob(job: Job): Promise<JobResult>;
}
```

JobResult

```
/**
 * Job result model containing all output arguments from the process.
 * Empty if there are no out arguments.
 */
class JobResult {
   [Key: string]: any;
}
```

Job

```
class Job {
   processId: string;
   argument?: any;
   jobId: string;

   constructor(processId: string, argument?: any);
   on(eventName: string, eventHandler: (argument?: any) => void): void;
}
```