Local Skills - HelloWorldpart1_TimeOfFlight

In the *Local Skills - Hello World* series of tutorials you'll learn everything you need to know to begin writing robust, intricate local skills for your Misty robot. We'll start off slow, taking it one step at a time until you've been exposed to the full breadth of Misty's capabilities and potential. You can use these tutorials with or without the information in the "Writing a Local Skill" doc, but they're better together.

In Part 1 we'll create a simple skill that will change Misty's chest LED, drive her forward for 10 seconds, and tell her to stop if she detects an object in her path. We'll go over how to send commands, subscribe to sensor events, and structure your skill data. Let's get started!

The code for local skills is comprised of two parts. The logic used to define how the skill functions is located in a `.js` file located in the `/Code` directory under `/Skills`. In addition to this, there is a corresponding `.json` file in the `/Meta` directory. These files must have the same name. Create a `.js` file and call it `HelloWorldpart1_TimeOfFlight.js`. Then create a `.json` file and give it the same name. Be sure that these are located in the proper directories as specified above.

HelloWorldpart1_TimeOfFlight.json

The `.json` file includes fields that set certain specifications for your skill. You'll need to create a GUID to uniquely identify your skill, which can be generated here. Use this to set the value of `UniqueId`. In the `Name` field, enter the name used for the `.js` file and the `.json` file to ensure they are referenced correctly. Many of the parameters below have default values that are automatically set. For now, use the values shown below for the remaining fields.

```
{
	"Name": "HelloWorldpart1_TimeOfFlight",
	"UniqueId": "e46c1b29-9d46-4f38-945f-673fa4b7c2bd",
	"Description": "Local 'Hello, World!' tutorial series, part 1.",
	"StartupRules": [ "Manual", "Robot" ],
	"Language": "javascript",
	"BroadcastMode": "verbose",
	"TimeoutInSeconds": 300,
	"CleanupOnCancel": false,
	"WriteToLog": false
}
```

HelloWorldpart1_TimeOfFlight.js

To issue any command to Misty in the local environment, we call methods on the `misty` object. Start by writing a debug message so we're notified when the skill is started. To do this, call `misty.Debug()` and pass in a meaningful message. These messages will show up in your browser's JavaScript console if you're using the Skill Runner tool that came with the SDK.

```
misty.Debug("starting skill local_helloworld1");
```

Now let's use the very simple `misty.ChangeLED()` function to control the color of your robot's chest LED. The method takes three arguments, which correspond to the RGB parameters required to specify the LED color as described here. The code below will turn the LED green (for go!)

```
misty.ChangeLED(0, 255, 0);
```

Then, we'll issue one of Misty's drive commands, `DriveTime()`. The `DriveTime()` command accepts three parameters: `linearVelocity`, `angularVelocity`, and `time`. You can learn more about how these parameters will affect Misty's movement in the documentation here. In this case, we want Misty to drive forward slowly in a straight line for 10 seconds, so we'll set `linearVelocity` = 10, `angularVelocity` = 0, and `time` = 10000 (the unit of measure for this parameter is milliseconds).

```
misty.DriveTime(50, 0, 10000);
```

Alright, so now we've instructed Misty to drive forward for 10 seconds and come to a stop. But there's a major flaw in this code. What if there is an object in Misty's way? We want her to come to a stop, rather than plowing through it. In order to handle this type of event, we need to look at data coming back from Misty's sensors. To do this we will subscribe to events from one of Misty's many websocket streams available to us: namely `TimeOfFlight`.

Once we have subscribed to `TimeOfFlight`, we'll receive event data back from Misty telling us how far objects are away from her. See the template for registering for an event below:

```
misty.RegisterEvent(string eventName, string messageType, int debounce, [bool
keepAlive = false], [string callbackRule = "synchronous"], [string skillToCall
= null]);
```

We call `RegisterEvent()` and pass in the name we want to designate for the event ("`FrontTOF`"), and the name of the websocket stream we are subscribing to

(“TimeOFFlight”). By default, when a callback triggers for an event, the event is automatically unregistered. Make sure your register event method matches the code snippet below.

```
misty.RegisterEvent("FrontTOF", "TimeOfFlight");
```

Before we register to the event in our code, we can add property comparison tests to filter the data we receive. In this example, the first property test checks that we are only looking at data from the time-of-flight sensor we’re concerned with. The field we’re testing is `SensorPosition` and we’re checking that the data received is only coming from the time-of-flight sensor in the front center of Misty’s base, pointing in her direction of travel. Therefore, we’ll only let through messages where `SensorPosition == Center`.

```
misty.AddPropertyTest("FrontTOF", "SensorPosition", "==", "Center", "string");
```

The second property test checks ensures we are only looking at data where the distance to the object detected is less than 0.2m. We don’t want our skill to react to things further away than about 6 inches, basically.

```
misty.AddPropertyTest("FrontTOF", "DistanceInMeters", "<=", 0.2, "double");
```

Whenever we subscribe to an event, we receive the data back within a callback function. This function is triggered whenever messages are sent that pass the property tests. The callback is automatically given the name `_<event>`. So in this case, the callback name is automatically set to `_FrontTOF`. The data is passed directly into the callback and can accessed through an argument passed into `_FrontTOF`, which we’ll call `data`.

```
function _FrontTOF(data) {
}
```

Once we receive the data via the callback, we can access the distance the object was detected and the sensor position it was detected at.

```
function _FrontTOF(data) {
   let frontTOF = data.PropertyTestResults[0].PropertyParent;
   misty.Debug(“Distance: ” + frontTOF.DistanceInMeters);
   misty.Debug(“Sensor Position: ” + frontTOF.SensorPosition);
```

```
}
```

Call `Stop()` to issue a stop command to Misty, then `ChangeLED()` and pass in the values (255, 0, 0) to turn the LED red (for stop!) and log a message to notify us that the skill has finished.

```
misty.Stop();
misty.ChangeLED(255, 0, 0);
misty.Debug("ending skill helloworld part1");
```

Congratulations! You have just written your first skill for Misty. Check out the info in “Writing a Local Skill” to learn how to use either Skill Runner or Postman to load your skill data onto Misty and run the skill from the browser. See the entire file below for reference.

```
// debug message to indicate the skill has started
misty.Debug("starting skill helloworld part1");

// issue commands to change LED and start driving
misty.ChangeLED(0, 255, 0); // green, GO!
misty.DriveTime(10, 0, 10000);

// register for TOF and add property tests
misty.AddPropertyTest("FrontTOF", "SensorPosition", "==", "Center", "string");
misty.AddPropertyTest("FrontTOF", "DistanceInMeters", "<=", 0.2, "double");
misty.RegisterEvent("FrontTOF", "TimeOfFlight");

// TOF callback
function _FrontTOF(data) {
    // grab property test results
    let frontTOF = data.PropertyTestResults[0].PropertyParent;

    // log distance object was detected and sensor
    misty.Debug(frontTOF.DistanceInMeters);
    misty.Debug(frontTOF.SensorPosition);

    misty.Stop();
    misty.ChangeLED(255, 0, 0); // red, STOP!
    misty.Debug("ending skill helloworld part1");
```

```
}
```