Custom Events Sample skills (Work in progress)

Introduction

To demonstrate the magic of custom events, we have sample skills developed in NodeJS and Python, which show how custom events can be used alongside custom directives and how they can be parsed in the skill lambda.

Gadget side implementation (using Python):

Sending a custom event from Python based Gadget (PiGadget):

1. Use the following ***gadgetManagerGadgetEvent.proto*** protobuf file:

gadgetManagerGadgetEvent.proto

```
syntax = "proto3";

package gadgetManager;

option java_package = "com.amazon.proto.avs.v20160207.gadgetManager";

option java_outer_classname = "GadgetEvent";

import "eventHeader.proto";

message GadgetEventProto {
Event event = 1;
    message Event {
        string payload = 2;
        header.EventHeaderProto header = 1;
    }
}
```

2. Compile the ***gadgetManagerGadgetEvent.proto*** using the how-to-compile-the-proto-files guide OR download the following already compiled file ***gadgetManagerGadgetEvent_pb2.py***

3. Add the capability for supporting custom event in your ***PiGadgetConfig.json***

```
{
  "configuration": {
    "supportedTypes": []
  },
  "interface": "StatusGaugeGadget",
  "type": "AlexaInterface",
  "version": "1.0"
},
{
  "configuration": {
    "supportedTypes": []
  },
  "interface": "TTSGadget",
  "type": "AlexaInterface",
  "version": "1.0"
}
```

4. In your python environment, import ***gadgetManagerGadgetEvent_pb2***

```
import gadgetManagerGadgetEvent_pb2
```

NOTE: Import path might vary depending on the package structure on your RaspPi.
For e.g. in PiGadget, we do the following in *proto_gadget_base.py*:

```
from protobuf.proto.GadgetManager.GadgetEvent import
gadgetManagerGadgetEvent_pb2
```

5. Use the following function ***send_gadget_event()*** to send custom events which takes *namespace*, *name*, and *payload* as parameters.

```
def send_gadget_event(self, namespace, name, payload):
    print("Generating and sending gadget event")
    pb_msg = gadgetManagerGadgetEvent_pb2.GadgetEventProto()
    pb_msg.event.header.namespace = namespace
    pb_msg.event.header.name = name
    pb_msg.event.header.messageId = ""
    pb_msg.event.payload = payload

    custom_msg = list(pb_msg.SerializeToString())
    self.send_payload(custom_msg)
```

NOTE: *self.send_payload()* is the function used to send payload from the gadget to the EFD. Refer this code to see how it is implemented in PiGadget.

6. Define the payload for the custom event

for StatusGadget

```
payload = json.dumps({
    "status": 'Happy to help',
    "query": False
    })
```

OR

for TTSGadget

```
payload = json.dumps({
    "text": "Hello World!!",
    "mic_open": False,
    "extend_session": True
    })
```

NOTE: More information on the parameters being sent is covered later in the document.

7. Send the custom event using the *send_gadget_event()* function defined in step 5, where *namespace* = *interface* and *name* = *type* as defined in step 3.

for StatusGadget

```
self.send_gadget_event("StatusGadget", "SendStatus", payload)
```

OR

for TTSGadget

```
self.send_gadget_event("TTSGadget", "SpeakText", payload)
```

NodeJS based skill implementation

Receiving a custom event on the skill-end:

1. For receiving any custom event in the skill, a ***GameEngine InputHandler*** should be active. To start a *GameEngine InputHandler,* send the ***GameEngine.StartInputHandler*** directive as a response to any intent request (for e.g. Launch Intent).

Sample response

```
{
    'version': '1.0',
    'response': {
        'outputSpeech': {
            'type':'PlainText',
            'text': "Starting GameEngine Session"
        },
        'directives': [
            {
                'type': 'GameEngine.StartInputHandler',
                'timeout': 60000,
                'recognizers': {},
                'events': {
                    'timeoutEvent': {
                        'meets': [
                            'timed out'
                        ],
                        'reports': 'history',
                        'shouldEndInputHandler': True
                    }
                }
            }
        ]
    }
}
```

2. Parsing event to identify if it is a custom event:
   A custom event will have its request type = *'CustomInterfaceController.EventsReceived'*

```
if (event.request.type === 'CustomInterfaceController.EventsReceived')
{
...
}
```

3. Get the custom event from the request and based on the type of custom event received using *namespace* and *name* handle the received *payload*.

```
if (event.request.type === 'CustomInterfaceController.EventsReceived')
{
    let customEvent = event.request.events[0];
 let payload = JSON.parse(customEvent.payload);
    if (customEvent.header.namespace === "StatusGaugeGadget" &&
           customEvent.header.name === "SendStatus") {
         return statusGadget.handleCustomEvent(payload);
       } else if (customEvent.header.namespace === "TTSGadget" &&
           customEvent.header.name === "SpeakText") {
         return ttsGadget.handleCustomEvent(payload);
       }
}
```

4. Handle the received custom event:

Handling Status Gadget custom event

```
exports.handleCustomEvent = function (payload) {
   let speakText = "";
   if (payload.query) {
       if (payload.status != null) {
           speakText = "Your current status is " + payload.status;
       } else {
           return {
                'version': '1.0',
                'response': {
                    'outputSpeech': {
                        'type':'PlainText',
                        'text': "Status isn't set yet. What is your
status?"
                    },
                    'shouldEndSession': False
                }
            }
       }
   }
   else {
       speakText = "Status was successfully set to " + payload.status;
   }

   return {
        'version': '1.0',
        'response': {
            'outputSpeech': {
                'type':'PlainText',
                'text': speakText
            }
        }
    }
}
```

Handling TTS Gadget custom event

```
exports.handleCustomEvent = function (payload) {
   let speakText = payload.text;
   let shouldExtendSession = payload.extend_session;
   let shouldOpenMic = payload.mic_open;

   let response = {}
   if (shouldOpenMic) {
       // open the mic for input
       response.shouldEndSession = false;
       if (speakText != "") {
           response.outputSpeech = {
              'type':'PlainText',
              'text': speakText + "<break time=\'1s\'/> Would you like
```

```
to continue?"
            }
        }
    } else if (speakText != "") {
        // speak the text without opening the mic
        response.outputSpeech = {
           'type':'PlainText',
           'text': speakText
         }
    }

    if (shouldExtendSession == null) {
        // let the current GameEngine InputHandler continue
        return {
           'version': '1.0',
           'response': response
         }
    } else if (shouldExtendSession) {
        // Extend the skill session by 30 seconds by starting another
input handler
        response.directives = [
             {
                 'type': 'GameEngine.StartInputHandler',
                 'timeout': 30000,
                 'recognizers': {},
                 'events': {
                     'timeoutEvent': {
                         'meets': [
                             'timed out'
                         ],
                         'reports': 'history',
                         'shouldEndInputHandler': True
                     }
                 }
             }
         ]
    } else if (!shouldExtendSession) {
        // Stop the skill session
        response.shouldEndSession(true);
    }

    return {
       'version': '1.0',
```

```
        'response': response
      }
  }
```

Testing some use cases:

StatusGadget:

- **Getting a confirmation in form of custom event from the gadget, when a status is set by sending a custom directive.**
  Ask skill "Alexa, ask *Slingshot* to set my status as Busy"
  Skill responds "Okay, sending custom directive to Gadget"
  Status is set on the gadget and custom event is sent back to skill "Status was successfully set to Busy"

Custom Event sample

```
{
  "header": {
    "namespace" : "StatusGaugeGadget",
    "name" : "SendStatus"
  },
  "payload" : "{\"status\": \"Busy\", \"query\": false}",
  "endpoint":{
    "endpointId":"endpoint_id"
  }
}
```

- **Query skill to get the status that has already been set on the Status gadget.**
  Ask skill "Alexa, ask *Slingshot* what is my status?"
  Skill responds "Okay, sending custom directive to Gadget"
  Status is returned on the gadget and custom event is sent back to skill "Your current status is Busy"

Custom Event sample

```
{
  "header": {
    "namespace" : "StatusGaugeGadget",
    "name" : "SendStatus"
  },
  "payload" : "{\"status\": \"Busy\", \"query\": true}",
  "endpoint":{
    "endpointId":"endpoint_id"
  }
}
```

- **Query skill to get the status when no status has been set on the Status gadget yet.**
  Ask skill "Alexa, ask *Slingshot* what is my status?"
  Skill responds "Okay, sending custom directive to Gadget"
  Status is returned on the gadget and custom event is sent back to skill "Status isn't set yet. What is your status?"
  Respond "Traveling"
  Skill responds "Okay, sending custom directive to Gadget"
  Status is set on the gadget and custom event is sent back to skill "Status was successfully set to Traveling"

Custom Event sample

```
{
  "header": {
    "namespace" : "StatusGaugeGadget",
    "name" : "SendStatus"
  },
  "payload" : "{\"status\": null, \"query\": true}",
  "endpoint":{
    "endpointId":"endpoint_id"
  }
}
```

TTSGadget:

TTSGadget was designed to test use cases like sending a TTS message from gadget to the skill; opening the mic for input and extending skill session using *GameEngine.InputHandler* as responses to receipt of custom event, and combination of these functionalities.

PiGadget command:

```
speak "text" [-m (open mic)] [-e (extend session) | -s (stop session)]
```

"text" : Text to be spoken. If "" (empty text) is sent, *outputSpeech* won't be sent in the skill's response.

-m (open mic) : Opens mic for input. If empty text ("") is sent, mic is opened without speech.

-e (extend session) : Extends the session by 30 seconds by sending another *GameEngine.StartInputHandler*.

-s (stop session) : Stops the skill session. Cannot be sent with *-m* flag.

Use cases:

- **Send a custom event from the gadget to just speak some text.**
  Command on PiGadget `speak "Hello world!"`
  Skill says "Hello world"

Custom Event sample

```
{
  "header": {
    "namespace" : "TTSGadget",
    "name" : "SpeakText"
  },
  "payload" : "{\"text\": \"Hello world\", \"mic_open\": false,
\"extend_session\": null}",
  "endpoint":{
    "endpointId":"endpoint_id"
  }
}
```

- **Send a custom event from the gadget to speak some text and open the mic for input.**

Command on PiGadget `speak "Hello world!" -m`
Skill says "Hello world" <pauses for a sec> "Would you like to continue?"
If "yes" extends skill session by sending another *GameEngine.startInputHandler* directive, if "no" kills the skill session.

Custom Event sample

```
{
  "header": {
    "namespace" : "TTSGadget",
    "name" : "SpeakText"
  },
  "payload" : "{\"text\": \"Hello world\", \"mic_open\": true,
\"extend_session\": null}",
  "endpoint":{
    "endpointId":"endpoint_id"
  }
}
```

- **Send a custom event from the gadget to just open the mic for input.**
  Command on PiGadget `speak "" -m`
  Skill just opens the mic for input.

Custom Event sample

```
{
  "header": {
    "namespace" : "TTSGadget",
    "name" : "SpeakText"
  },
  "payload" : "{\"text\": \"\", \"mic_open\": true,
\"extend_session\": null}",
  "endpoint":{
    "endpointId":"endpoint_id"
  }
}
```

- **Send a custom event from the gadget to extend the skill session (can be used in combination with any of the above use cases).**
  Command on PiGadget `speak "Hello world!" -e`
  Skill says "Hello world"
  Skill extends the skill session by sending another *GameEngine.startInputHandler* directive.

Custom Event sample

```
{
  "header": {
    "namespace" : "TTSGadget",
    "name" : "SpeakText"
  },
  "payload" : "{\"text\": \"Hello world\", \"mic_open\": false,
\"extend_session\": true}",
  "endpoint":{
    "endpointId":"endpoint_id"
  }
}
```

- **Send a custom event from the gadget to kill the skill session (cannot be used with parameters *-m* and *-e*).**
  Command on PiGadget `speak "Hello world!" -s`
  Skill says "Hello world"
  Skill ends the skill session using *shouldEndSession* as *true*.

Custom Event sample

```
{
  "header": {
    "namespace" : "TTSGadget",
    "name" : "SpeakText"
  },
  "payload" : "{\"text\": \"Hello world\", \"mic_open\": false,
\"extend_session\": false}",
  "endpoint":{
    "endpointId":"endpoint_id"
  }
}
```

- **Send more than 1 custom events in a single InputHandler session.**
  Send this command on PiGadget multiple times within 10 seconds `speak "Hello world!"`
  Skill successfully receives and processes each custom event and speaks the text sent in the custom event.

- **Open mic either on launch or as a response to a custom event and send another custom event while the mic is open.**
  Skill successfully receives and processes the custom event and speaks the text sent in the custom event while the mic is left open.

- **Send multiple custom events with long texts to test if the custom event interrupts previous speech response.**
  YET TO TEST.