# Neato: Handling Robot Notifications & Online Status

Raja Software

Last updated: 04/03/2014

|  |  |  |
| --- | --- | --- |
| Date | Version | Author/Comments |
| 03/26/2014 | 0.1 | First draft for support of notifications/error from robot and robot online status |
| 04/03/2014 | 0.2 | Added examples for error and notification request/response |

# Overview

This document is intended for the robot team as well as the UI team to understand and handle the robot notifications as well as handle the online state change of robot.

The document discusses on how the robot can send notifications to the associated SmartApp and how the SmartApp can parse the notification. It also discusses about the support for knowing the online/offline status for robot.

# Robot: Send Notifications/Error

The robot messages have been distinguished into two types which are– “Notifications” and “Errors”.

## Notifications

There can be many notifications which the robot would want to send to the user for information.

This can be done by following steps:

**Webservice Method Name**: “robot.set\_profile\_details3”

[Note this method name is prefixed by the base json url for the server in use].

**The post parameters being**:

profile[robotNotificationMsg]: <the json string for the message as described above>

The other parameters which always go with the setRobotProfileDetails3 being:

notification\_flag: 1 [1 is to signify that the data changed notification should be sent to associated users]

serial\_number: <serial number of the robot>

source\_serial\_number: <serial number of the robot>

cause\_agent\_id: <serial number of the robot>

This will trigger a data changed notifications which will ask the SmartApp to make the call to get the latest notification queued for it.

The actual notification is represented by a JSON String.

Currently the Simulator sends a notification json as follows:

**{“messageID”: <id for the message>}**

The **messageID** is the unique identifier for the notification which the SmartApp should interpret and show to the user. More attributes can be added to this structure if needed. T

**The message ids for the notifications should be defined by the robot team for various notifications. The robot team can also add other attributes to this JSON object which the SmartApp UI should handle.**

**Example:**

If a robot “yana” wants to send a “Low Battery” notification with **messageID** being 10001.

|  |  |
| --- | --- |
| cause\_agent\_id | yana |
| profile[robotNotificationMsg] | {"messageID":10001} |
| notification\_flag | 1 |
| api\_key | 1e26686d806d82144a71ea9a99d1b3169adaad917 |
| source\_serial\_number | yana |
| serial\_number | yana |

This will pass the data changed notification to the SmartApps which then will process the notification.

## Errors

There can be many errors which the robot would want to send to the user for information.

This can be done by following steps:

**Webservice Method Name**: “robot.set\_profile\_details3”

[Note this method name is prefixed by the base json url for the server in use].

**The post parameters being**:

profile[robotErrorMsg]: <the json string for the message as described above>

The other parameters which always go with the setRobotProfileDetails3 being:

notification\_flag: 1

serial\_number: <serial number of the robot>

source\_serial\_number: <serial number of the robot>

cause\_agent\_id: <serial number of the robot>

This will trigger a data changed notifications which will make the SmartApp to make the call to get the latest error queued for it.

The actual error is represented by a JSON String.

Currently the robot sends an error JSON as follows (similar to the notification):

{“messageID”: <id for the message>}

The messageID is the unique identifier for the notification which the UI layer should interpret and show to the user. More attributes can be added to this structure if needed.

**The message ids for the errors should be defined by the robot team for various notifications. The robot team can also add other attributes to this JSON object which the SmartApp UI should handle.**

Example:

If the robot “yana” wants to send a “Cannot Find Base” error with messageID being 20001.

The webservice post parameters for setRobotProfileDetails3 would be:

|  |  |
| --- | --- |
| cause\_agent\_id | yana |
| notification\_flag | 1 |
| profile[robotErrorMsg] | {"messageID":20001} |
| api\_key | 1e26686d806d82144a71ea9a99d1b3169adaad917 |
| source\_serial\_number | yana |
| serial\_number | yana |

This will pass the data changed notification to the SmartApps which then will process the error.

# UI: Handling Robot Notifications/Errors

Once the plugin layer receives the notification/error it will send the notification/error to the UI layer.

The UI layer should register a callback (which it already does for other notifications) on registerNotifications2 in Robot manager plugin.

A general notification received as callback is:

{robotDataKeyId:"robotDataKeyId", robotId:"robotId", robotData:"robotData"}

The keyID for a notification and error are:

var ROBOT\_MESSAGE\_NOTIFICATION = 4013;

var ROBOT\_MESSAGE\_ERROR = 4014;

which are defined in neatosmartapphelper.js.

If a notification is received for these Key Ids, then the robotData consists of the notification json sent by the robot.

The format for the robotData is:

Notification: { robotNotification: {JSON notification string from robot}}

Error: {robotError: {JSON error string from robot}}

The current notification/error format as described above is: {“messageID”:<id for the message>}.

The id will be a unique identifier for the notification/error.

**The message ids for the notifications/errors should be defined by the robot team for various notifications. The robot team can also add other attributes to this JSON object which the SmartApp UI should handle.**

**Example**:

If the robot sends a low battery notification like explained in above section’s example, the UX will get a callback with the result json being:

{robotDataKeyId:4013, robotId:testrobot, robotData: { robotNotification: {messageID: 10001}}}

Here by the messageID the UX would show the appropriate notification to the UI.

# UI: Handling Robot Online/Offline

The server sends a notification to the associated users when the robot online status changes.

The UI layer should register a callback (which it already does for other notifications) on registerNotifications2 in Robot manager plugin.

A general notification received as callback is:

{robotDataKeyId:"robotDataKeyId", robotId:"robotId", robotData:"robotData"}

The keyID for a robot online/offline is:

var ROBOT\_ONLINE\_STATUS\_CHANGED = 4015.

The robotData format being:

{ online: 1} for robot online and { online: 0} for robot offline.

**Example**:

If the robot – “testrobot” goes offline the UX will get a notification as:

{robotDataKeyId: 4015, robotId:testrobot, robotData: {“online” : 0}

Here as the online status is 0, the UX should show the appropriate state to the user.

End.