# Neato Web Service Interaction

Raja Software

Last updated: 04/01/2014

|  |  |  |
| --- | --- | --- |
| Date | Version | Author/Comments |
| 03/28/2014 | 0.1 | Initial draft – focus on smartphone web service interaction |
| 04/01/2014 | 0.2 | Added more details, discussed notifications |

# Overview

This document is intended for the all the developers on the Neato project to get the overall perspective of all the important web service calls made from SmartApps and User.

# Robot Profile Details

Before we get started on individual web services, let us look into how data is communicated between multiple SmartApps (logged in with same user) and the associated robot, and how the state of the robot is synchronized.

The web services involved are:

<http://neatostaging.rajatogo.com/api/rest/json?method=robot.set_profile_details3> and <http://neatostaging.rajatogo.com/api/rest/json?method=robot.get_profile_details2>

The robot profile, on the server, maintains a key-value pair along with a server timestamp of when the value associated with the key was changed.

The keys are defined for various properties which maintain a State of the robot as well as to have a command structure sent VIA server so that other SmartApps can be in sync.

Whenever a SmartApp or a Robot updates a value for any key in the profile, a XMPP data changed notification is sent to all the online user-devices for the robot and the robot itself.

This notification is parsed and the receiver calls: <http://neatostaging.rajatogo.com/api/rest/json?method=robot.get_profile_details2> to update itself.

Every receiver maintains a “Key-Timestamp” pair in the local storage (SQLite, file) to know the version of the state received. It does not store the value of the key in the local storage.

The following diagram illustrates the mechanism through which a state is propagated between different systems:

Step3.

Get Robot Profile Details

Step1.

Set Robot Profile Details Call

Step2.

XMPP Notification

Robot

SmartApp

Step3.

Get Robot Profile Details

SmartApp

Step 1: The SmartApp user changes a robot profile key (name, robot command or schedule state etc)

Step 2: Server sends a XMPP message to the online users for the robot as well as the robot.

Step 3: The details are retrieved by the receiver and processed.

After retrieving the data from the server, the timestamps for each key is compared with the timestamps of the keys in the local storage and the “changed” keys are notified to the SmartApp user.

Example:

If SmartApp UserA is using a robot RobotB:

1. UserA sends a Start Cleaning Command.
2. Set Robot Profile Details will be called with the appropriate key for start cleaning along with the cleaning parameters as the value for the key.
3. The server will update the value as well as update the timestamp.
4. The RobotB will receive a data changed notification.
5. The RobotB will retrieve the data from the server and compare the local timestamps with the server details and process the “Changed” key(s) which in this case is the Cleaning command.
6. It will update the current state of the RobotB on the server which again is propagated to all online users of the RobotB.

There are multiple keys which are set using this design:

1. Schedule State of the robot
2. Name
3. Cleaning Command
4. Robot Current State
5. Intend To Drive
6. Online/Offline Status of the robot.
7. Notifications and Errors sent by the robot.

Whenever any of the keys is changed, it will fire a notification to the online systems so that the change will be propagated.

If a SmartApp comes online after some time, it can know the last state updated of the robot, the name (if changed), schedule changes as well as the schedule state of the robot; even if the robot is offline at that point.

# SmartApp – Web Service Interaction

# User Related Web Services

This part will describe in detail about what happens if a particular phone-gap plugin API is called and what all web services are called in the background:

## Create User

**Phonegap Plugin API**: UserMgr.prototype.createUser3

**Plugin**:

1. Web service: <http://neatostaging.rajatogo.com/api/rest/json?method=user.create3>

This web service creates the user on the server along with other necessary details. The

Web service will also return a authentication token for use.

1. After a web service success:

<http://neatostaging.rajatogo.com/api/rest/json?method=user.get_user_account_details>

is called to retrieve all the user details

<http://neatostaging.rajatogo.com/api/rest/json?method=user.set_attributes>

is called to set the user related device attributes on the server

Registration for push notifications is also done at this point.

## Login User

**Phonegap Plugin API**: UserMgr.prototype.loginUser

**Plugin**:

1. Web service called: <http://neatostaging.rajatogo.com/api/rest/json/?method=auth.get_user_auth_token>

With the success we get the authentication token for the user which will be saved locally.

1. If the server call is success, we get the user details and set attributes similar to create user:

<http://neatostaging.rajatogo.com/api/rest/json?method=user.get_user_account_details>

<http://neatostaging.rajatogo.com/api/rest/json?method=user.set_attributes>

Registration for push notifications is also done at this point.

## Resend Validation Mail

Phonegap Plugin API: UserMgr.prototype.resendValidationMail

Web service called: <http://neatostaging.rajatogo.com/api/rest/json?method=user.ResendValidationEmail>

Used to send the validation email to the user if not already validated

## Change Password

Phonegap Plugin API: UserMgr.prototype.changePassword

Web service called: <http://neatostaging.rajatogo.com/api/rest/json?method=user.change_password>

The web service is used to change the existing password of the user.

## Change Subscription / Country Code

Phonegap Plugin API: UserMgr.prototype .setUserAccountDetails

Web service called: <http://neatostaging.rajatogo.com/api/rest/json?method=user.set_account_details>

This web service is used to change the user details

## Forgot Password

Phonegap Plugin API: UserMgr.prototype.forgetPassword

Web service called: <http://neatostaging.rajatogo.com/api/rest/json/?method=user.forget_password>

The web service is used to send the next steps if the user forgot the password

## Link Robot

Phonegap Plugin API: UserMgr.prototype.linkRobot

Web service called: <http://neatostaging.rajatogo.com/api/rest/json?method=robot.link_to_robot>

The web service is used to link a robot with the user. This is done with linking code of the user.

## Get Associated Robots

Phonegap Plugin API: UserMgr.prototype.getAssociatedRobots

Web service called: <http://neatostaging.rajatogo.com/api/rest/json/?method=user.get_associated_robots>

This web service returns the list of associated robots with the user.

## Disassociate Robot

Phonegap Plugin API:

Web service called:http://neatostaging.rajatogo.com/api/rest/json?method=robot.clear\_robot\_association

This web service dis-associates the robot with the user.

## Get Push Notification Settings

Phonegap Plugin API: UserMgr.prototype.getNotificationSettings

Web service <http://neatostaging.rajatogo.com/api/rest/json?method=message.get_user_push_notification_options>

This web service returns the push notification options set by the user.

## Set Push Notification Settings

Phonegap Plugin API: UserMgr.prototype.turnNotificationOnoff

Web service: <http://neatostaging.rajatogo.com/api/rest/json?method=message.set_user_push_notification_options>

This web service sets the push notification options for the user. It can disable/enable any/all options.

# Cleaning Related Web Services

## Send Cleaning Commands (Non manual Mode): (Start/Stop/Pause/Resume/Send to base)

**Phonegap Plugins called -**

**RobotMgr.prototype.startCleaning**

**RobotMgr.prototype.stopCleaning**

**RobotMgr.prototype.pauseCleaning**

**RobotMgr.prototype.resumeCleaning**

**Plugin**

1. Web service: <http://neatostaging.rajatogo.com/api/rest/json?method=robot.set_profile_details3> is called to set the command on the server.
2. A data changed XMPP message is sent to all the associated user-devices as well as the robot about “data changed”.

This notification also has the “causeAgentId” which reflects uniquely on who initiated the change.

1. When this notification is received on the SmartApp who initiated the cleaning command, the notification is ignored as the change was done by the Smartapp (comparing the causeAgentId).

IF Robot is ONLINE:

1. Robot receives the data changed notification and calls

<http://neatostaging.rajatogo.com/api/rest/json?method=robot.get_profile_details2>

1. After processing the set command, if the robot state changes due the set command, the new state is added on the server by calling <http://neatostaging.rajatogo.com/api/rest/json?method=robot.set_profile_details3> with the new state.
2. This will again fire a data changed notification from the server VIA XMPP which will cause the SmartApp to get the server details and get the current state of the robot.

If Robot is OFFLINE

The command timer on the SmartApp will fire a reset for the cleaning command so that the robot doesn’t get it when the robot comes online after a long time (which could be anytime). It calls <http://neatostaging.rajatogo.com/api/rest/json?method=robot.set_profile_details3> to reset the cleaning command. The timer expiry happens currently after 3 minutes and is only applicable for

Cleaning commands.

## Start Cleaning Flow (Manual Mode)

### Intend to drive

Phonegap Plugin API: RobotMgr.prototype.intendToDrive

1. Web service:

<http://neatostaging.rajatogo.com/api/rest/json?method=robot.set_profile_details3>

This is the 1st step to start driving and this is a request to the robot to open the TCP server and provide an IP address to connect to.

1. Robot if online will receive the data changed notification, and will retrieve the profile details. It will set whether it is available to drive (which is the case unless it is in driving mode already) along with its IP address using <http://neatostaging.rajatogo.com/api/rest/json?method=robot.set_profile_details3>

The received data changed notification is tracked by the caller Smartapp to initiate a TCP

connection to the robot address.

1. Appropriate callbacks are given to the UI JS layer about connection success or failure.

### Start Manual Cleaning

Once the connection is formed, the Phonegap plugin API to be called are same as non-manual mode. They will work only when the connection is formed, otherwise will return an error to the plugin API now.

### Cancel Intend To Drive

Phonegap API: RobotMgr.prototype.cancelIntendToDrive

This will clear the "intend to drive” request from the server unless the connection has already formed.

Web service called: <http://neatostaging.rajatogo.com/api/rest/json?method=robot.set_profile_details3>

### Stop Robot Drive

Phonegap API: RobotMgr.prototype.stopRobotDrive

This disconnects the current connection to the robot if connected.

## Schedule Related Web Services

## Enable/Disable Schedule

Phonegap Plugin: RobotMgr.prototype.enableSchedule

Web service called: <http://neatostaging.rajatogo.com/api/rest/json?method=robot.set_profile_details3>

This will set the state for the schedule which will fire a data changed notification to the associated users and the robot in order to retrieve the correct state.

## Is Schedule Enabled

Phonegap Plugin: RobotMgr.prototype.isScheduleEnabled

Web service called:<http://neatostaging.rajatogo.com/api/rest/json?method=robot.get_profile_details2>

This will get the profile details and retrieve the current schedule state.

## Schedule

A robot schedule is maintained on the Server along with a version id. Whenever the schedule changes, the version id is incremented by the server. To update a schedule, the caller should have the current

schedule id, otherwise the server will not allow the update.

Schedule or a Schedule Group of individual schedule events is saved as a JSON on the server. A schedule event is an individual cleaning event defined inside the schedule.

### Get Schedule For Robot

Phonegap Plugin APIs: RobotMgr.prototype.getScheduleEvents

This calls the web service: <http://neatostaging.rajatogo.com/api/rest/json?method=robotschedule.get_schedule_based_on_type>

It gets the current schedule from the server.

As an implementation detail, it maintains it in a local database as well. This was done because of the requirement of the UI layer team for the locally changing individual schedule events.

The Server schedule copy is treated as the master copy and robot as well as the user should update the schedule on the server.

Calls communicating with local database which interact with a schedule event:

RobotMgr.prototype.createSchedule

RobotMgr.prototype.getScheduleEventData

RobotMgr.prototype.addScheduleEvent

RobotMgr.prototype.updateScheduleEvent

RobotMgr.prototype.deleteScheduleEvent

These calls will basically interact with the local copy of the schedule. Each call would update/add/delete the individual schedule event in the schedule group. Only when the local copy is updated on the server, it becomes the master copy.

Note that the local maintenance of the schedule in the smartapp is just an implementation detail, and it should in no way treated as the master copy while showing the user. The plugin method “RobotMgr.prototype.getScheduleEvents” should be called by the UI layer when showing the schedule to the user which basically fetches it from the server.

### Update Schedule to Server

Phonegap Plugin: RobotMgr.prototype.updateSchedule

This calls the web service: <http://neatostaging.rajatogo.com/api/rest/json?method=robotschedule.update_data>

It also sets the profile details with the key that the schedule is updated so that the other associated users as well as the robot knows about the profile change. For this it calls <http://neatostaging.rajatogo.com/api/rest/json?method=robot.set_profile_details3>.

## Set Robot Name

Phonegap plugin API: setRobotName2

Web service called: <http://neatostaging.rajatogo.com/api/rest/json?method=robot.set_profile_details3>

with the appropriate name for the robot. This will fire a notification to relevant entities.

## Get Robot Presense Status:

Phonegap plugin API: getRobotOnlineStatus

The webservice called is:

http://neatostaging.rajatogo.com/api/rest/json/?method=robot.get\_robot\_presence\_status

## Create Robot

The robot calls this method to create it’s instance on the server.

The webservice called is:

http://neatostaging.rajatogo.com/api/rest/json/?method=robot.create2

## Get Robot Details

The webservice used to get the robot details

http://neatostaging.rajatogo.com/api/rest/json/?method=robot.get\_details

## Request Linking Code

This webservice is called from the robot to generate a linking code:

Webservice called is:

http://neatostaging.rajatogo.com/api/rest/json/?method=robot.request\_link\_code

# Notifications

There are three types of notifications defined which are –

1. Push Notifications
2. Robot Notifications
3. Robot Errors

## Push Notifications

The push notifications are sent from the robot using:

<http://neatostaging.rajatogo.com/api/rest/json?method=message.send_notification_to_all_users_of_robot2>

This notification is sent to all the users of the robot.

Currently the supported push notifications are:

* Dirt bag full
* Robot is stuck
* Dirt Bin Missing
* Plug the cable
* Robot Cancel Error
* Cleaning is done

Each notification has a specific unique ID which is sent to the SmartApp.

The SmartApp has the option to disable these notifications (explained earlier: Set Push Notification Settings).

These notifications are received even when the application is NOT alive.

## Robot Messages: Notifications / Error

The robot notifications and errors are the messages which are to be propagated to the SmartApps when the applications is running and is in foreground.

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

### Notifications

Webservice Method Name: <http://neatostaging.rajatogo.com/api/rest/json?method=robot.set_profile_details3>

There is a specified key for the notifications and the robot should update the value on the server to set the notification object as the value for the key.

The actual notification is represented by a JSON String.

Currently the Robot 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

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

### Errors

Webservice Method Name: <http://neatostaging.rajatogo.com/api/rest/json?method=robot.set_profile_details3>

There is a specified key for the error and the robot should update the value on the server to set the error object as the value for the key.

The actual error is represented by a JSON String.

Currently the Simulator 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.

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

End.