# Neato SmartApps Backend DB Schema

Raja Software

Last updated: 03/24/2014

|  |  |  |
| --- | --- | --- |
| Date | Version | Author/Comments |
| 03/24/2014 | 0.1 | Captures all DB tables and a brief description of each of them. |
| 03/26/2014 | 0.2 | Added PhpMyAdmin details, formatting |

# 

# Summary

This document gives an overview of the DB schema that is used by the backend. It briefly explains the table and their relationship with each other. This document also has an Entity Relationship Diagram. Additionally this document explains how you can use PHPMyAdmin to explore the DB schema further.

# Tables

This section lists all the tables and briefly explains their usage. The listed tables are not in alphabetical order. First all the configuration tables are listed and then the robots, users and message related tables are listed.

#### sites

This table stores the sites (Neato/Vorwerk) and can store the configuration parameter against each of these sites.

#### api\_users

This table has API key and secret key. These are the keys that are used by the consumer to make the API calls.

#### app\_configuration

Eventually, we plan to move all the configurations to DB and give Admin an easy way to edit these configuration parameters. This table stores few configuration parameters (grace period, should email be validated etc)

#### app\_info

This table stores the SmartApp releases. CheckForUpgrade API queries this table and returns the upgrades, if available, with the path to binary.

#### upgrade\_status

All the upgrade statuses are available in this table and it is used in app versions. Entries are, “No Upgrade Available”, “Mandatory Upgrade Available” and “Optional Upgrade Available”

#### country\_code\_list

This table has all the supported countries. If you do not want to support a country, you can remove the entries from this table.

#### roles

This table has all the available user roles in the system. For the time being we have admin, user and support roles.

#### push\_notification\_types

This table holds all the push notification types in the system (dirt bag full, i'am stuck etc). This is used when we are sending a push notification.

#### ws\_logging

This table stores all the API calls (request, response, date userID) and can be enabled by a config parameter. This table is mainly used for the debugging.

#### robots

This is the table that stores the robot related information. Id of this table is used across the robots related tables.

#### robot\_types

We are supporting different robot types. This way we can store the configuration parameters at the robot type level that can be overridden at the robot level. It has 2 types by default, basic and advanced and new robot types can be added.

#### robot\_type\_metadata

This table stores meta data like values for sleep time and wake up time for this specific robot type.

#### robot\_robot\_types

We are supporting different robot types. This way we can store the configuration parameters at the robot type level that can be overridden at the robot level as well. Although 1 robot can be only in 1 robot type but we have kept it in a different table to ensure that in future we can support 1 robot in multiple robot types. In the current implementation, we have only 1 robot - 1 robot type association stored in this table though.

#### robot\_maps

One robot can have multiple maps. This table has reference of the robot's map. ID of this table is used to store the blob/XML version information.

#### robot\_atlas

1 robot can have many atlases and this table stores information about this. 1 Atlas can have 1 XML file and 1 blob data file. That is captured in the grid and XML tables.

#### robot\_map\_blob\_data\_versions

Robot's map has XML and Blob data. This table stores the version for the blob. This helps us in handling the case where a user tries to update the blob data without having the most recent version with him. This has reference to the robot map table's ID.

#### robot\_map\_xml\_data\_versions

Robot's map has XML and Blob data. This table stores the version for the map's XML. This helps us in handling the case where a user tries to update the XML without having the most recent version with him. This has reference to the robot map table's ID.

#### atlas\_grid\_image

Robot atlas has XML and Grid images. This table stores the information about the grid which is usually a png/jpg file.

#### robot\_config\_key\_values

This is loosely coupled way to add configuration related key/value pair for robot so that API consumer can create any key and store values against them.

#### robot\_key\_values

Similarly consumer can store key value pairs against each robot.

#### robot\_linking\_code

This table has all the linking code that can be used by the user to link the robot with user.

#### alive\_robot

This table stores the serial numbers of all the robots that are online. Ping API call results in entry in this table.

#### robot\_ping\_log

This table stores all the pings that are made by the Robot using the Ping APIs.

#### robot\_user\_association\_tokens

This table stores the linking code that is used by a robot to link with a user.

#### robot\_schedules

This table stores the schedules for robots. ID of this table is used in the version tables for schedule blob and schedule XML data.

#### robot\_schedule\_blob\_data\_versions

Robot's schedule has XML and Blob data. This table stores the version for the blob. This helps us in handling the case where a user tries to update the schedule blob data without having the most recent version with him. This has reference to the robot schedule table's ID.

#### robot\_schedule\_xml\_data\_versions

Robot's schedule has XML and Blob data. This table stores the version for the schedule's XML. This helps us in handling the case where a user tries to update the schedule XML without having the most recent version with him. This has reference to the robot schedule table's ID.

#### users

This table stores the user information. Id of this table is used across the tables for user association.  

#### user\_role

Although we are supporting 1 role per user but we are still keeping the user's role in a different table so that we can support multiple roles per user.

#### users\_api\_sessions

This table stores all the available session for a user. This table helps us in keeping multiple sessions active for a user across the devices.

#### user\_devices

This table stores the user's device information. This is used in the CheckForUpgrade API calls.

#### device\_details

This table stores the devices that are being used by the users. This table also has the operating system of these devices.

#### user\_push\_notification\_preferences

This table stores the type of notification a user wants to receive. Before sending notifications to a user, it is checked against this table.

#### users\_robots

This table stores the user-robots association. Now we are supporting just 1 user per robot but we are still keeping in different table so that we can always support multiple users per robot.

#### notification\_logs

This table stores the log of iOS/GCM notifications and can be used in future to find out the audit trail of all the notifications that went out of the system.

#### notification\_registration\_id\_logs

As user might get new registration IDs, this table stores the log of the registration id with mapping of old to new. Using this table along with notification logs table, we can find out the actual receiver of the notification at any given point of time.

#### notification\_registrations

This table stores all the registration keys (GCM/IOS) that are stored for a user. These keys are used to send the notifications.

#### xmpp\_message\_logs

All the messages that are sent over ejabberd are logged in the in this table.

#### xmpp\_notification\_via\_mq

RabbitMQ is used to send XMPP messages. When we have to send the message, RabbitMQ consumer picks it up from this table and sends the XMPP message and also logs it in the xmpp\_message\_logs table.

#### online\_chat\_ids

In the ejabberd we have configured callbacks for online/offline events. These events result in either an addition or a removal from this table. When we have to find out who all is online, instead of querying the ejabberd, we query this table and get faster results.

# Deprecated Tables

#### users\_socialservices

This table stores the user/social service information. For example it has values like user ID X has Facebook ID 123. As we have disabled end user login, we would be deprecating this table.

#### socialservicetypes

This table stores all the social services (Facebook/Google) that are supported by the application. Now we are not using this table. As we have disabled end user login, we would be deprecating this table.

#### robot\_custom\_data

This table provides a loosely coupled way to store the custom data (just like map/atlas files) for each robot. As we are not storing the custom data at the Robot level any more, we would be deprecating this table.

#### robot\_custom\_data\_types

This table categorizes the custom data in 2 categories.  As we are not storing the custom data at the Robot level any more, we would be deprecating this table.

#### robot\_customs

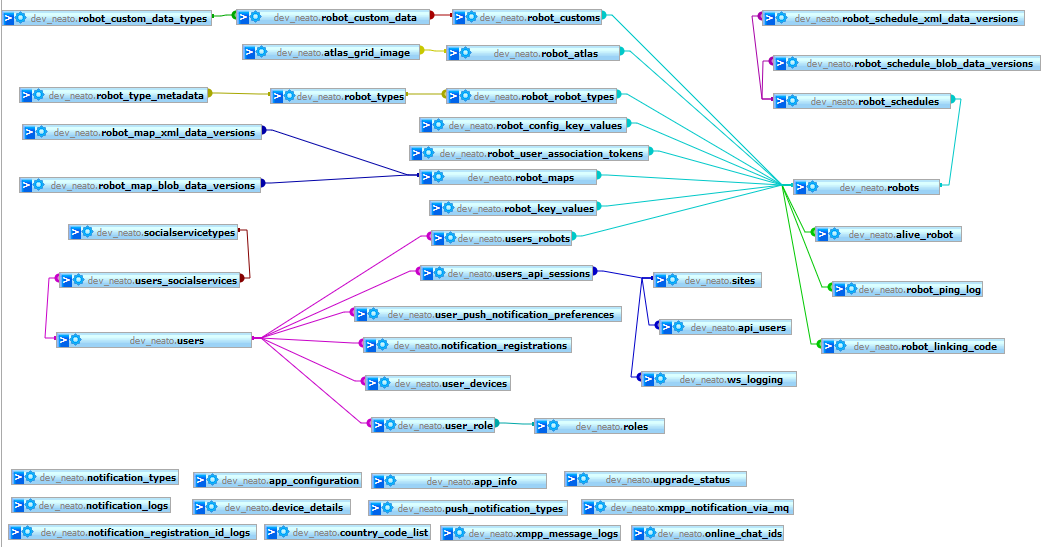
This table actually stores the value for the custom data for the robot. This is more to do with the data like files etc. As we are not storing the custom data at the Robot level any more, we would be deprecating this table.

#### notification\_types

There are 3 types of notifications that we support, system generated, SOS and related to activities.

# Entity Relationship Diagram

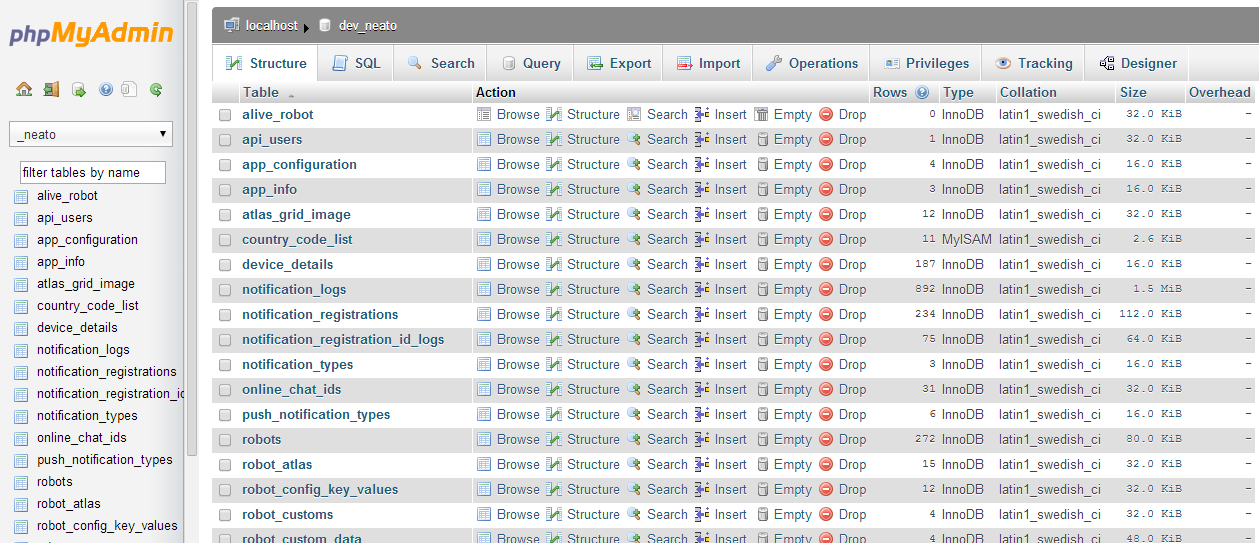
The diagram below shows the relationships between various tables.



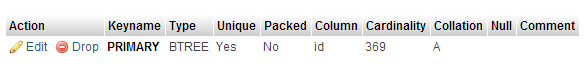
# PHPMyAdmin

We have enabled viewing of the MySQL database over the HTTP using PHPMyAdmin. You can view the MySQL database at <http://rajatogo.com/phpmyadmin/index.php> URL.

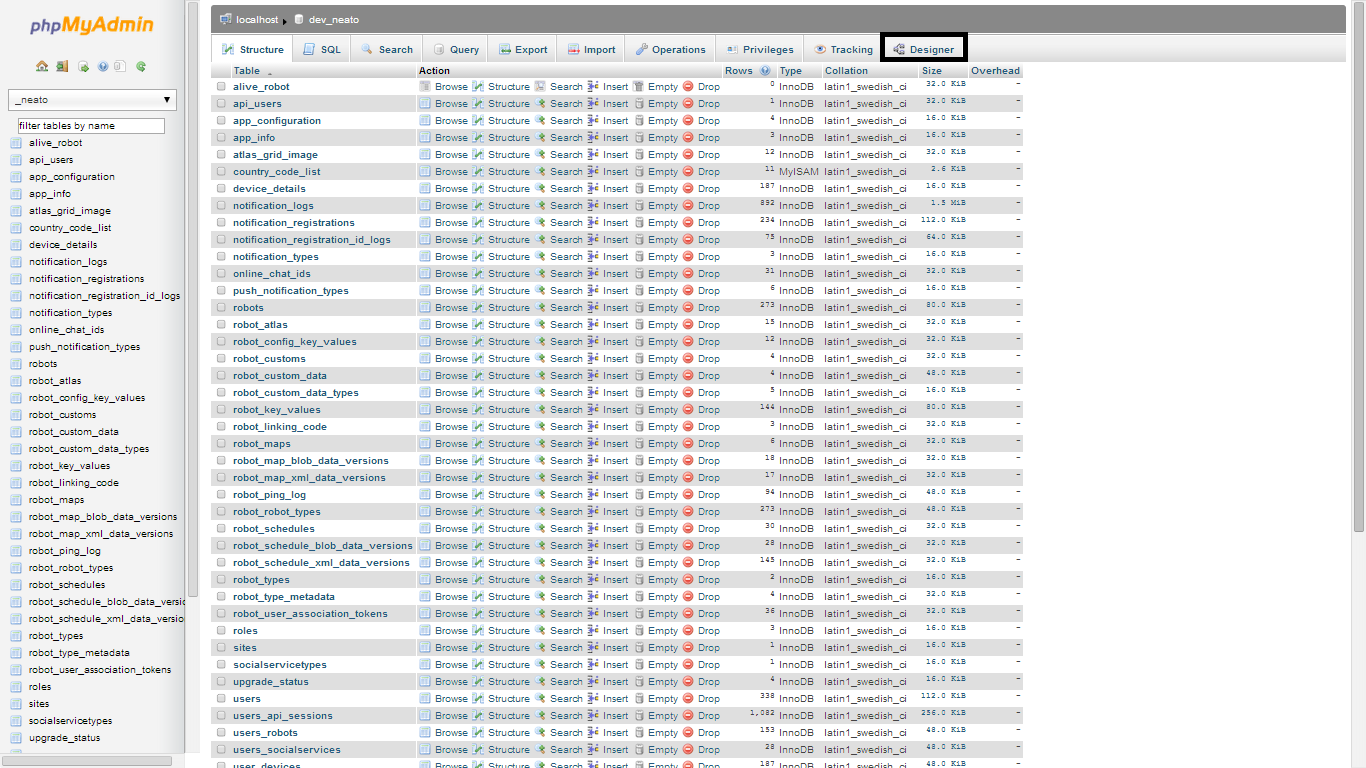
In PHPMyAdmin, first select the Database and then you can view all the tables inside the database. From this screen you can browse the table and see the structure of the table.



When you can click on any of these tables and view the data and the schema of this table. When you select table, at the bottom of the page, you can view all the relationships and the indexes applied on this specific table.



The entity relationship diagram that is shown above is available at the DB level. In the PHPMyAdmin, when you select a DB, you would see a menu item called Designer as described below.



PHPMyAdmin has a decent help manual and you can find out more about it at

<http://www.phpmyadmin.net/home_page/docs.php>

End.