Exosense Server Manual

Copyright 2012, 2013 Feuerlabs, Inc. All rights reserved.

2013-02-04

Contents

1 References **4**

2 Revisions **5**

3 Introduction **6**

4 Notes about versions **7**

5 Exosense Components **8**

5.1 Accounts 9

5.2 Users 10

5.3 Roles 11

5.4 Device 12

5.5 Device Type 13

5.6 Package 14

5.7 Protocol Plugin 16

5.8 Device Group 17

5.9 Exosense Built-In Yang Specifications 19

6 Component Relationships **21**

6.1 Relationship between Accounts, Users and Roles 22

6.2 [Deprecated in 2.x] Relationship between Configuration Sets and Devices 23

6.3 [Added in 2.x] Relationship between Devices, Device Types and Packages 24

6.4 Relationship between Device Groups and Packages 27

7 Use Case Sequencing **28**

7.1 [Added in 2.x] Sending Configuration Data To Device 29

7.2 Device Transaction sequence 30

7.3 Device Group Transaction sequence 33

8 JSON-RPC Protocol Overview **36**

8.1 General concepts 37

8.2 Authentication of JSON-RPC commands Authentication is done through 38

8.3 Device Transaction Queuing 39

8.4 Exosense JSON-RPC Command Role Assignment 40

8.5 Assigning roles to custom JSON-RPC commands 42

1 References

[1] http://www.jsonrpc.org/specification [2] http://tools.ietf.org/html/rfc6020

2 Revisions

Revision	Date	Description
A1	2013-02-05	Initial draft

Table 1: Revisions

3 Introduction

This manual provides instructions on how to operate the Exosense Server system.

The manual is divided into the following main sections:

1. **Exosense Components**
   The overall architecture of the system is provided with a description of each component. The provisioning system, configuration management system, and package management system are described.

2. **Component Relationships and Sequences**
   Describes which components are related to each other and how they interact.

3. **Use Case Sequencing**
   Describes the sequence of events for the major use cases executed by Exosense

4. **Exosense Operation**
   Procedures for installing, starting, stopping, upgrading, and backing up the system are documented.

5. **Exosense Management**
   Describes how to manage various components such as devices, accounts, users, packages, etc.

6. **Writing Custom Protocol Plugins**
   The process of implementing, building and deploying custom plugin protocols implementing proprietary device protocols is documented.

7. **JSON-RPC Core Protocol Reference**
   The core JSON-RPC protocol is documented together with formal specs and examples.

4 Notes about versions

This document describes version 1.x and version 2.x. Version 2.x adds package management and deprecate configuration sets in favor of a package/device-group based configuration management design.

Throughout the documents, the following semantics is used to differentiate between 1.x and 2.x.

1. [**added in 2.x**]
   This functionality is not available in 1.x and has been added to 2.x.
2. [**extended in 2.x**]
   This functionality has been extended in 2.x, but is still backward compatible with 1.x.
3. [**modified in 2.x**]
   This functionality has been modified in 2.x in a way that is not backward compatible with 1.x.
4. [**deprecated in 2.x**]
   This functionality is available in 1.x, but has been removed in 2.x.

5 Exosense Components

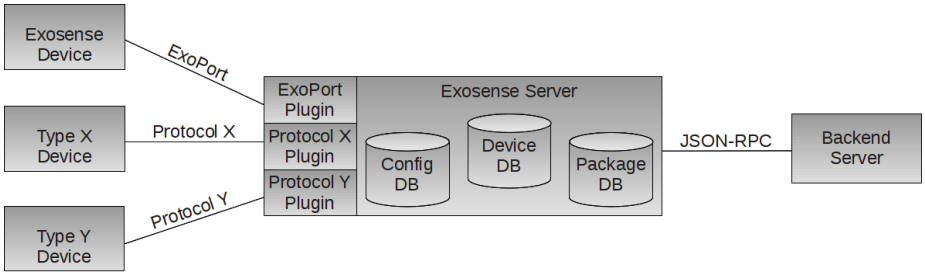

Figure 1: Exosense Server Components

The Exosense Server (Exosense in short) manages devices, traffic, configuration and software packages in large networks of connected devices. The server provides an end-to-end integration between the backend server and devices in the field.

By default, Exosense implements the ExoPort protocol to talk to devices running the Exosense Device stack, but it can easily be extended with client-developed plugins to host multiple custom protocols.

The backend server interfaces the Exosense Server through a JSON-RPC based protocol. This protocol can be extended with RPCs that are to be forwarded by the Exosense Server to target devices, thus allowing the backend server to remotely invoke functionality on a device. All protocol extensions are handled through the Yang specifications that are uploaded to the server with no programming outside the customer-developed protocol plugin.

Finally, Exosense can install, upgrade, and remove packages on a device. These packages are uploaded to the Exosense over JSON-RPC. [Added in 2.0]

All communication with a device can be queued for later forwarding, allowing the device to call in to the server and receive all pending commands and updates scheduled for it.

5.1 Accounts

An account is a top level entity used to manage an Exosense Server System. Accounts are managed through a set of JSON-RPC commands described in the Exosense Reference Manual. An account hosts the following items in Exosense

1. **Devices**
   A provisioned device is always owned by a single account.
2. **[Extended in 2.x] Device Types**
   A device type ties a device to a protocol and a set of packages that can be installed on it.
3. **Device Groups**
   A collection of devices that can be managed as a single unit.
4. **[Added in 2.x] Packages**
   A package contains a software package to be installed and executed on one or more devices, and with the RPC and configuration specification for that software.
5. **[Deprecated in 2.x] Configuration Sets**
   A configuration set contains configuration data to be sent to one or more devices, together with a yang specification describing the format for the data.
6. **JSON-RPC execution permission**
   While the built-in JSON-RPC commands used to manage the Exosense Server have pre-defined execution permissions, extended JSON-RPC commands that are forwarded to devices and device groups can have customized permissions through the use of roles.

When an account is created by sysadm user (see Users), an initial user will be created for that account as well. This account-level sysadmin will have its user name will set to [account]-admin, where [account] is name of the account given to the exodm:create-account called. The account admin user is used to manage the given account and create additional users to be given access to it.

When an Exosense Server is installed, an initial account, exodm, is automatically created together with its account admin, exodm-admin.

All account management is done using a set of built in JSON-RPC commands supported by the server. When issuing these commands, the credentials of the initial sysadm and subsequent account sysadm users are provided as authorization.

5.2 Users

A user is an entity used to control access to devices, device types, device groups, configuration sets, packages, and JSON-RPC commands created under an account. All access to these objects are managed through a set of built in JSON-RPC commands supported by the Exosense Server. The credentials of a user are provided with a JSON-RPC command to authorize its execution on the Exosense Server.

A user can optionally be given access to more than one account, allowing it to manage devices, device groups, and other entities owned by multiple accounts.

When the system is installed a single pre-defined user, sysadm, (not an account) is set-up with a default password. The sysadm user can then be used to create additional accounts and their users.

5.3 Roles

A role is an entity that has configurable execution rights of specifics RPCs in a package. When execution rights are assigned to a role through the exodm:add-role-execution-permission call, an account and, a package, and an RPC name are specified. Users can be then be given access to the account/package/RPC by assigning the user to the role through the exodm:add-user-to-role.

Once the role has been assigned execution right of a given RPC in a package owned by an account, the user can use the JSON-RPC interface to access that RPC on all devices, owned by the account, that has the package installed.

The roles available for execution rights and user assignment are predefined and have predefined execution rights for the built in Exosense JSON-RPC commands. In addition to the built in commands, the predefined roles can be freely assigned execution rights for RPCs in packages.

5.4 Device

A device is a unit managed by an Exosense Server. A device is always associated with a device id, a device type, and an account owning the device.

The device id is a string identifier that is unique across all devices for a given account. The device is associated with one of the provisioned device types.

[Deprecated in 2.x] A device can be associated with one or more configuration sets that defines the configuration data to be sent to the device.

[Added in 2.x] A device can be associated with one or more packages that defines software that can be sent to, installed on, and executed on a device together with the software's API and configuration data structure.

Once a device has been provisioned, it will be recognized by Exosense when it calls in. In cases where the network supports it, the device can also be contacted by Exosense through outbound communication.

A device can optionally be configured with a notification URL, which is an destination address to receive JSON-RPC commands originated from the device. If a device sends a RPC command to the Exosense server, it will forward it to the provisioned notification URL as a JSON-RPC command. The reply to that JSON-RPC command will be send back to the device.

Please note that the notification-URL of a device will not be used when transaction notifications are sent back from the device or Exosense in response to a request sent from the backend server. Such transactions have a transaction-id element that is reused when the recipient (a device or the Exosense Server) wants to send continuous update notifications on the progress of the original request. The notification-URL of a device will, in other words, only be used to forward requests originating in the device.

Please note that a device can also be a member of a device group, which in its turn can have its own notification URL specified. In cases where both the device and an associated device group have a notification URL setup, the device's URL will take precedence.

5.5 Device Type

A device type is setup via JSON-RPC and defines the properties of a device. The device type has the following properties:

1. **Name**
   A symbolic reference used by JSON-RPC commands to identify the device type.
2. **Device Attributes**
   Device attributes, in addition to the mandatory device id and device type, are specified, to be used by the protocol plugin. These attributes are not pushed to the device itself, but instead remains as data that can be used by a server protocol plugin and also returned through an exodm:lookup-device call
3. **Protocol Plugin**
   Specifies the protocol plugin to employ in order to communicate with the device.
4. **[Added in 2.0] Allowed Packages**
   Contains a list of package names that devices of this type can install. See Package chapter for details.

5.6 Package

[Added in 2.0]

A package is a software image that can be transferred and installed on one or more devices. While the Exosense server only forwards a package to its destined devices without any consideration of their format, packages are usually of the RPM, Debian, Opackage, or other standard variety.

A package is uploaded to the Server through the JSON-RPC call exodm:create-package. In addition to the (opaque) package image itself, a package has the following properties:

1. **Name**
   A symbolic reference used by JSON-RPC commands to identify the package. This name is unique for all packages owned by the creating account.

2. **A Yang Specification for JSON-RPC API and Configuration Data**
   A Yang specification that maps out all the JSON-RPCs that can be sent to the Exosense server in order to be forwarded to a device running the given package. The server will forward the JSON-RPC command to the Protocol Plugin specified by the device type for the device. The plugin is expected to forward the command to the device targeted by the JSON-RPC command. Likewise, the client can send RPCs to the server, which will forward them to the notification URL of the Device or its Device Group. The Yang specification also defines the configuration data structure. When a package is installed on a device, configuration data is created for that device/package combination. Specific values to be set in the configuration data are provided with the package install command.

3. **Dependencies**
   This is a (possibly zero-length) list of other packages that must be installed on the device in order for this package to be successfully installed and executed. The dependecy list is used by the Exosense server to determine if additional packages are needed on a device when an install command is executed.

The difference between device attributes, specified by a device type, and device configuration data, specified by a package, is that attributes are only used by the Exosense Server's protocol plugin; they are never transmitted to a device. Hence, attributes tend to contain information such as network addresses, encryption keys, callout schedules, location, and other information of interest to the backend server and the protocol plugin.

Configuration data, on the other hand, is transmitted to the target device where it will affect the behavior of a package running on the device. The server's

only involvement is to validate the format of the data against the relevant yang specification prior to sending it out to its destination devices.

5.7 Protocol Plugin

The protocol plugin is responsible for encoding and decoding traffic to and from the device. It usually also listens to a specific network address (or port) to receive incoming traffic from devices in the field. When Exosense needs to send configuration data or RPCs to the device, it is the Protocol Plugin's responsibility to translate the internal representation of these two operations to a format that can be understood by the device.

The protocol plugin also, with support from Exosense, has the notion of sessions and queues with data to be sent to the device when it becomes available for communication (i.e. when it calls in).

A protocol plugin is deployed in an Exosense server as an extension that registers itself with the server. Once registered, Device Types can reference it by name in order to direct configuration and RPC commands to it.

[Added in 2.x] The protocol plugin will be also be invoked when a package is to be transmitted to a device. The plugin will receive the package name and the target device id to transmit the package to. The Exosense Server provides chunking of the package image, allowing the plugin to retrieve sections of the image from the server, transmit them to the device, and inform the server that the chunks have been received and stored by the device. When the final chunk has been transferred, the protocol plugin can command the device to validate and install the now complete package.

5.8 Device Group

A device group is a collection of devices that are sharing the same packages and configuration data. When a package is installed, or configuration data is set, for a device group, the package/config data will be sent out to all member devices of the group.

A device group is identified by a group ID string, which must be unique across all devices and groups for a given account. In other words, a group cannot have the same ID as another group or another device provisioned under the same account.

Configuration data can be set for individual devices in a device group, thus allowing default configuration data for all device members to be set on a device group level, while individual devices can still be personalized with network addresses, encryption keys, etc.

Please note that all configuration data transmits, either to an individual device, or to a device group that cascades to all its member devices, will overwrite any existing configuration data stored in the device. It is up to the device administrator to arrange the configuration updates so that they do not overwrite each other in unintended ways.

All member devices in a device group must have the same device type. The device type of the first device added as a group member will be used to validate all subsequent device memberships.

A device group can optionally be configured with a notification URL, which is an address to serve RPC requests originated from the device. If a device sends a RPC request to the Exosense server, it will forward it to the provisioned notification URL as a JSON-RPC command. The reply to that JSON-RPC command will be send back to the device.

If a device is a member of multiple device groups, incoming RPC commands from the device will be sent to the the notification URL of all groups. The reply to be sent back to the originating device will be read from the notification URL of the device group marked as primary. The replies received from all other notification URLs outside that of the primary group will be silently dropped by the Exosense Server.

A group is marked as primary through an argument to the exodm:add-device-group-member command.

If the device itself has a notification URL specified, any RPCs received from the device will be sent to that URL as well as all associated device groups' notification URLs. However, the reply will always be read back from the device's notification URL, thus ignoring the reply from any primary device group notification URLs.

Please note that the notification-URL of a device group will not be used when transaction notifications are sent back from the device or Exosense in response to

a request sent from the backend server. Such transactions have a transaction-id element that is reused when the recipient (a device or the Exosense Server) wants to send continuous update notifications on the progress of the original request. The notification-URL of a device will, in other words, only be used to forward requests originating in the device.

Figure 2 shows an example where several devices sends RPC commands, such as Ping and Alarm, to the Exosense Server, which forwards them to notification URLs associated with device groups. Device 2 in the example is a member of both group A and B, but has group A marked as primary. Thus, the Status Report RPC from Device 2 goes to group A and group B, and the reply to send back to the device is read from the notification URL of group A (http://groupa.com). The reply from group B's notification URL (http://groupb.com) is silently dropped.

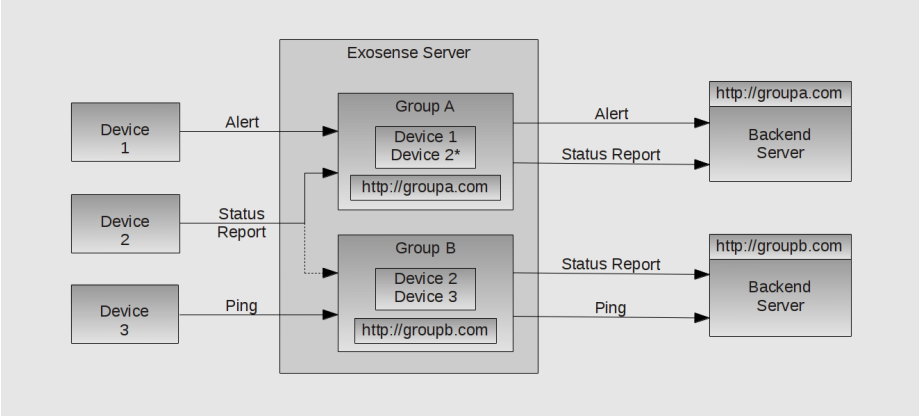

Figure 2: Device Group - Inbound RPC calls

Device groups can also be used to route a single JSON-RPC from the backend server to multiple devices by addressing the JSON-RPC to a group instead of to a single device. In that case, multiple notifications will be sent back in response to the original JSON-RPC call. Each notification will describe the success or failure of the command for a specific device that is a member of the group.

Figure 3 shows an example of such a call flow.

If a device is member of multiple groups, configuration written to one group may overwrite the configuration data written to the other group, leading to unpredictable results. It is up to the backend system and administrator to manage devices and groups to avoid this.

When a device, which is a member of multiple groups, sends an RPC to the Exosense server, the group that the device was last added to as a member will receive the corresponding JSON-RPC call (unless the device itself is configured with a notification).

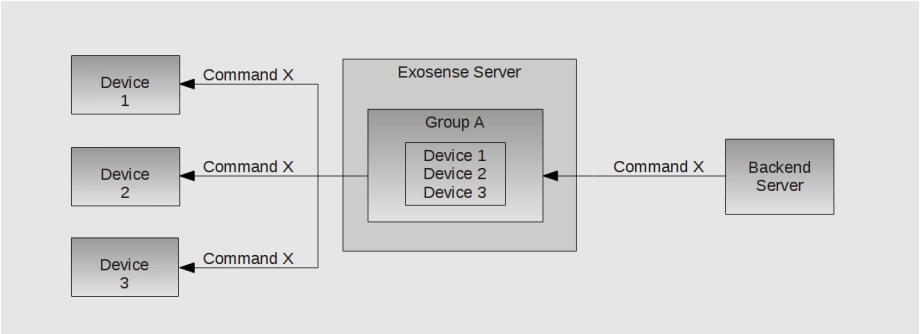

Figure 3: Device Group - Outbound RPC calls

5.9 Exosense Built-In Yang Specifications

Yang[2] is used to describe schemas for RPCs, device attributes, and configuration data

Exosense is shipped with three core Yang modules:

1. **ietf-inet-types.yang**
   Defines standard types such as IP addresses, host names etc. This file is distributed under the BSD license as a standard component maintained by IETF.

2. **exodm.yang**
   Top level yang file that includes the files below to form a complete API for the Exosense Server.

3. **exosense.yang**
   Defines standard types used throughout the Exosense Server such as status codes, transaction id's etc. This file, distributed under MPL-2.0 license by Feuerlabs, is included by exodm.yang (see below) and custom protocol and configuration data specifications. This file is usually included by other yang files that specify RPCs and custom configuration data schemas.

4. **exodm_user.yang**
   Defines the API to create, modify, delete, and list users.

5. **exodm_config_set.yang**
   Defines the API to create, modify, delete and list configuration set, and manage their member devices.

6. **exodm_device.yang**
   Defines the API to create, modify, delete and list devices.

7. **exodm_device_group.yang**
   Defines the API to create, modify, delete and list device groups, and manage their member devices

8. **exodm_device_types.yang**
   Defines the API to create, modify, delete and list device groups, and list all members of a type.

9. **exodm_package.yang**
   Defines the API to create, list, and delete packages, and manage the list of device types that can have them installed.

10. **exodm_package.yang**
    Defines the API to create, list, and delete packages uploaded in to the Exosense Server.

11. **exodm_yang_module.yang**
    Defines the API to create, list, and delete yang modules uploaded to the Exosense Server.

12. **exodm_role_module.yang**
    Defines the API to create, list, and delete yang modules uploaded to the Exosense Server.

[Added in 2.x] In addition to these, Packages are associated with their own yang specifications to describe the configuration data and RPCs supported by the package. These yang specifications are uploaded to the Exosense Server through the exodm:create-yang-module JSON-RPC command defined in the exodm.yang core module.

[Deprecated in 2.x] In addition to these, [Configuration Sets] are associated with their own yang specifications to describe the configuration data and RPCs supported by the devices that are members of a set. These yang specifications are uploaded to the Exosense Server through the exodm:create-yang-module JSON-RPC command defined in the exodm_yang_module.yang core module.

6 Component Relationships

6.1 Relationship between Accounts, Users and Roles

Accounts, Users and Roles are combined to define which users can do what with devices owned by a specific account.

Figure 4 shows the relationship between the three entities.

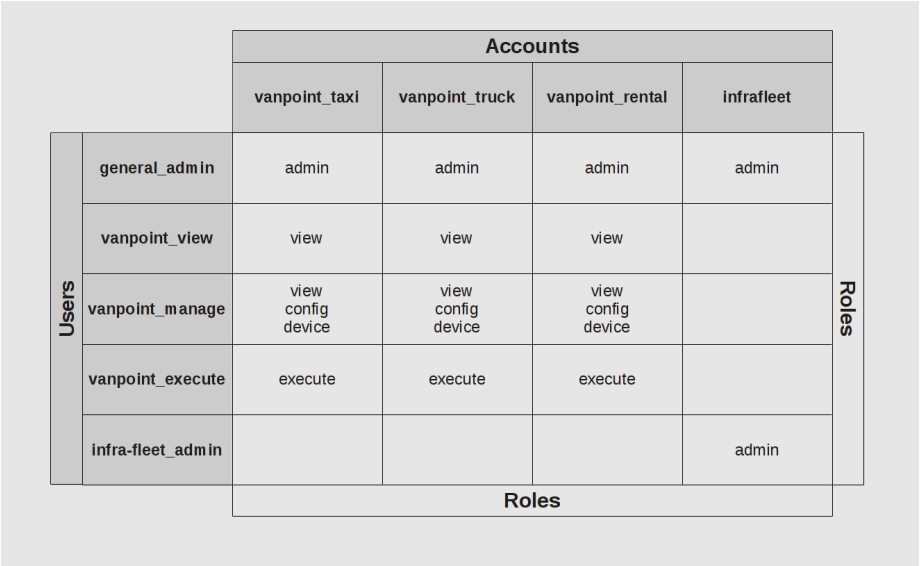

Figure 4: Accounts, Users, Roles

Users are given access to accounts through the exodm:add-user-to-account command. Once access has been given, the is yser assigned roles within that account through the exodm:add-user-to-role command.

Roles are, as a final step, given execution rights to specific RPCs in specific packages installed on devices onwed by the account. This is done through the exodm:add-role-execution-rights command.

Once a role has been assigned to a specific RPC, within a package. All users given that role within an account can execute the RPC on a device (owned by the account) that happens to have the package installed.

Please note that roles defines the execution rights for its assigned users on a specific account. If the user is member of multiple accounts, the account is specified together with the user credentials to indicate the account the JSON-RPC command should be executed on.

In addition to the RPCs of a package, the Exosense server itself supports a number of JSON-RPCs for device, account, user, and package management. These RPCs are in the exodm module, and have execution rights assigned to various roles as described in the Exosense Command Roles chapter

6.2 [Deprecated in 2.x] Relationship between Configuration Sets and Devices

A configuration set defines configuration data and an RPC interface that is supported by all the set's member devices.

The format of a configuration set is determined by an associated yang specification uploaded to the server.

Figure 5 shows a diagram showing the relationship between a configuration set (A), its member devices (Device 1, Device 2, Device 3), and a yang specification (basic_cfg.yang).

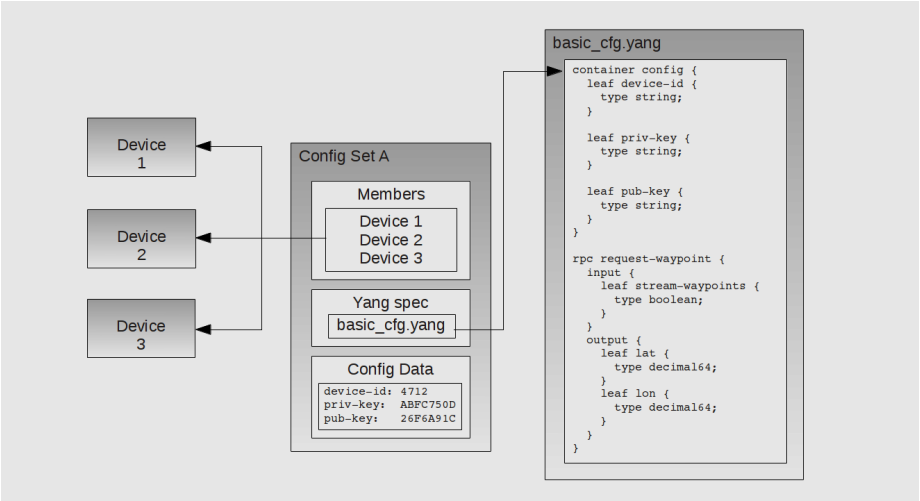

Figure 5: Configuration Set

The basic_cfg.yang specification defines that Device 1, 2 and 3 can accept a device-id, a private and a public key as configuration data. The member devices also accept an RPC, request-waypoint with the single argument stream-waypoints. The RPC returns a lat/lon position as a result.

exodm:update-config-set can be called to modify the configuration values stored in the set. When exodm:push-config-set is called, the values in the config set is transferred to all the set's member devices.

6.3 [Added in 2.x] Relationship between Devices, Device Types and Packages

Figure 6 shows an example schematics of how three devices are associated with their device types, allowed packages, and, by extension, their associated protocol plugins and config/RPC yang specifications.

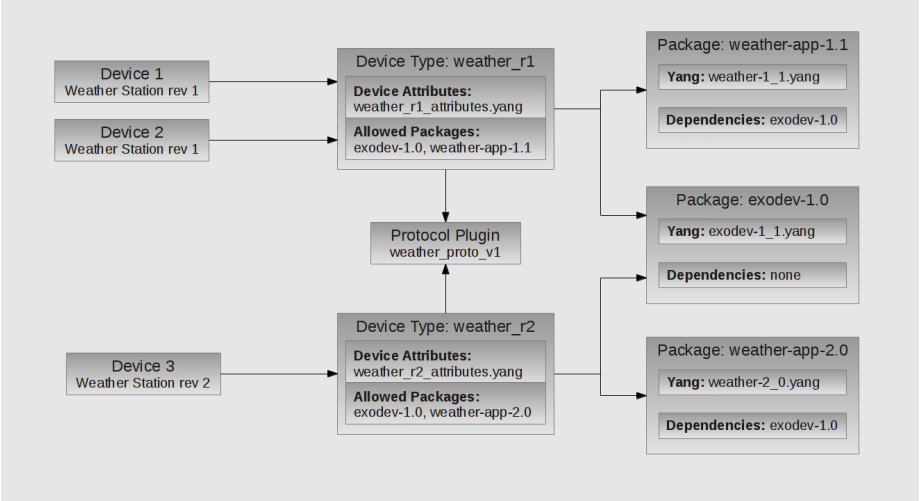

Figure 6: Device, Device Types, Protocol Plugins, and Packages

The characteristics of the three devices are the following

1. **Device 1 and 2**
   These two devices, which are weather stations of hardware revision 1, uses the weather_proto_v1 protocol plugin for communication. The device attributes are specified through the weather_r1_attributes.yang Yang specification used by the weather_r1 device type. The weather_r1 device type also specifies that two packages can be installed on devices of this type: exodev-1.0 and weather_app-1.1 .

2. **Device 3**
   This device, which is of hardware revision 2 weather station, is similar to device 1 and 2. The difference is that a number of device attributes have been added together with a modified configuration data and a few new RPC commands. The protocol used to communicate with the device, however, is still the same weather_proto_v1 as is used by Device 1 and 2. A separate device type, weather_r2, is created to handle the updated structures. This type uses its own device attribute Yang specification, weather_r2_attributes.yang. It also specifies that the weather_app-2.0, tailored for the second hardware revision is the only allowed package on the

device on top of exodev-1.0. Weather_app-2.0 defines the modified configuration data and added RPC commands through its weather-2_0.yang specification. The older weather_app-1.1 only works on revision 1 and is not included over permitted packages for revision 2.

Figure 7 shows example is given of how Device 1 in the schematics above can have its device attributes, configuration data, and installed packages setup.

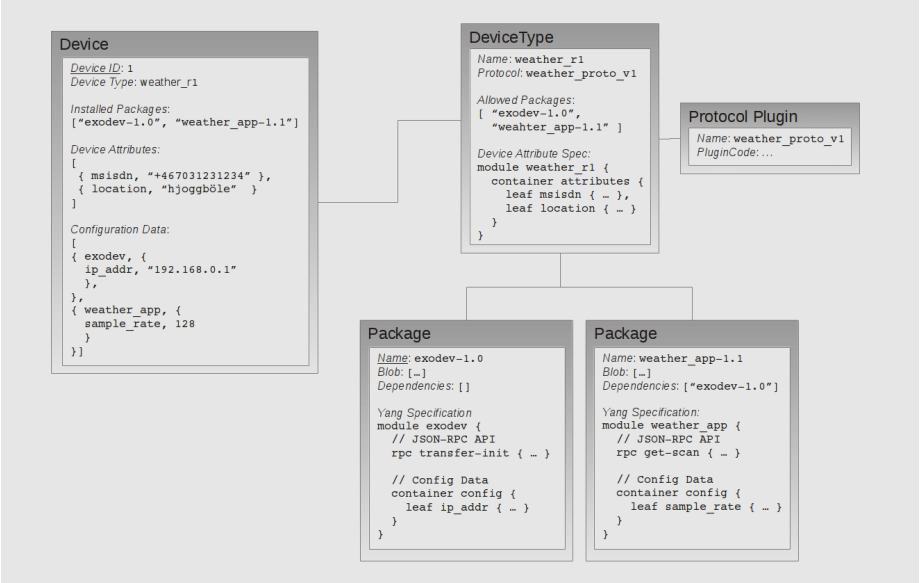

Figure 7: Exosense Device Example

The data stored for a device is as follows:

1. **Device ID**
   This is the account-wide unique ID for the device. References to the device in JSON-RPC commands are done through the Device ID. In this example the Device ID is set to "1".

2. **Device Type**
   Points out the type that this device is associated with; in this case weather_r1. The type, as described above provides the schema for the device attributes, the packages that can be installed on the device, and the protocol plugin to use to communicate with the device.

3. **Installed Packages**
   Contains references to the packages that have successfully been installed on device "1". The reference will be present in the Allowed Packages list of the weather_r1 Device Type, since those packages are the only ones

allowed to be installed on the device. The package install operation has triggered the creation of the Configuration Data described in point 5 below.

4. **Device Attributes**
   An set of attributes with a structure retrieved from the Device Attribute Spec of the weather_r1 Device Type. The msisdn and location elements have been set for the device to a phone number and a very small town in Sweden.

5. **Configuration Data**
   Since the configuration data is managed on a per-package level, it has been broken down into two sections; one for the installed exodev-1.0 package, and one for the weather_app-1.1 package. The structure of each section is taken from the Config Specification of its corresponding Package. As a result, the exodev section contains an ip address element set to "192.168.0.1", and the weather_app section contains a sample rate element set to 128.

6. **RPCs (not shown)**
   In addition to the entities described above, device "1" will also accept two different RPC calls being sent to it over the weather_proto_v1 Protocol Plugin. The first RPC, transfer-init, is specified by the exodev-1.0 package installed on the device. The second RPC, get-scan, is specified by the weather_app-1.1 package also installed on the device. Both these calls will be routed to their handling package when they are received by the device.

6.4 Relationship between Device Groups and Packages

Figure 8 shows a variant of the earlier example, with a Device Group containing two devices together with their relationship to device types, protocol plugins and packages.

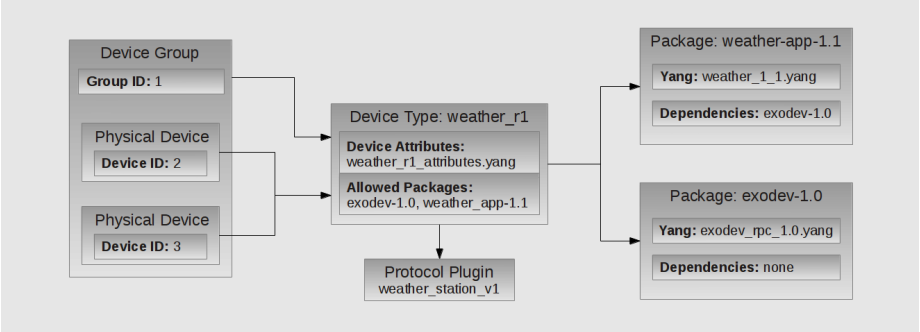

Figure 8: Exosense Device Group Example

The device group, with a group ID set to 1, contains the devices with device ID 2 and 3. Any device added to a device group must have the same device type as the group itself.

In this example, any packages installed, upgraded, or deleted from the group will have its action mirrored to all member devices. Likewise any configuration updates made to the device group will be pushed out to its hosted member devices.

Please note, however, that the opposite is not true. If device 2 has its configuration modified directly, without going through the device group, those changes will not be pushed to its hosting group.

7 Use Case Sequencing

7.1 [Added in 2.x] Sending Configuration Data To Device

Configuration data is set per Device and Package. A package has a Yang specification for the schema of its configuration data. When a package is installed on a device, configuration data will be setup in the server for the package on the given device and pushed out to it.

When configuration data is modified through JSON-RPC commands, it will be done for a specific package on a specific device. If the same configuration data is to be sent to multiple devices, a device group can be setup as the destination for the data.

The initial configuration data is automatically created and installed when a package is installed on a device. Updates to existing configuration data, on the other hand goes through a specific cycle:

1. **Configuration data is updated on the server**
   One or more update-config JSON-RPC commands are sent to the Exosense Server with updates to values in the configuration data tree. These changes are stored in the server without being sent to the device.

2. **Configuration data is committed to be pushed to the device**
   Once all the updates are completed to a set of configuration data, a push command is sent to the server through a push-config-data JSON-RPC command. This causes the updated data to be queued for the given device as a [Device Transaction].

3. **Transaction is sent to the device**
   Once a device is on-line, the transaction with the configuration data updates is sent to it.

4. **Transaction is executed on the device**
   When the configuration update transaction has been received and validated on the device, it will be executed to update the local configuration data base.

5. **Device sends transaction acknowledgment to server**
   Immediately after the transaction has completed on the device, it queues a transaction confirmation to the server. Depending on the communication characteristics, this confirmation may be sent back immediately, or the next time the device comes on-line.

7.2 Device Transaction sequence

All configuration data updates, package operations, and RPC commands are handled as transactions by the Exosense Server. This means that the operation is either in progress, has completed successfully, or has failed. In addition to this, the operation has to complete or fail in whole, leaving no partially modified data behind.

The server uses the device transaction concept to track the state of each device managed by it. Each update to configuration data and packages are broken down into discrete transactional changes which are yet to be completed, have completed or have failed, allowing the server to transmit retries, report failures and calculate transactional deltas between two states.

Each transaction is given a transaction id, which is unique for each transaction on the given device.

Figure 9 shows a sequence diagram showing how configuration data is updated on device "1", from [Example 1], in a transactional manner.

The steps outlined in the sequence diagram are as follows:

1. **First configuration update**
   The backend server requests that the ip_addr of the exosense-1.1 package (see [Example]) is to be set to "10.0.0.1" on the device.

2. **Prepare transaction to device.**
   The JSON-RPC module forwards the first configuration update as a transaction prepare command to the device queue.

3. **Ack on first configuration update**
   The device queue confirms the configuration update, through the JSON-RPC module, to the backend server.

4. **Second configuration update**
   The backend server requests that the location of the weather_app-1.0 package (see [Example]) is to be set to "Bro" on device.

5. **Prepare transaction to device.**
   The JSON-RPC module forwards the second configuration update as a transaction prepare command to the device queue.

6. **Ack on second configuration update**
   The device queue confirms the second configuration update, through the JSON-RPC module, to the backend server.

7. **Request config data to be pushed to device**
   The backend server sends a push-config command to the Exosense server to initiate transmission of the two configuration updates to the device.

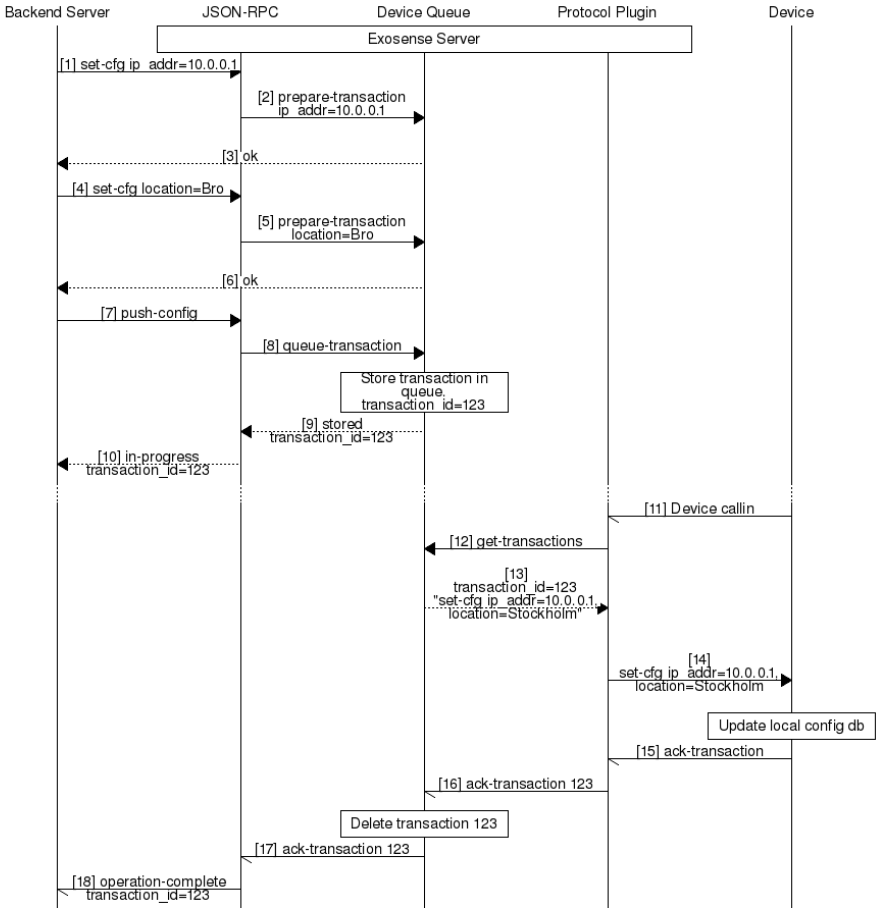

Figure 9: Device Transaction Sequence Diagram

8. **Queue both configuration updates as a transaction**
   The Exosense server creates a single transaction, containing the updates from step 1 and 4 above, and queues it for transmission to the target device when it becomes available. The transaction is assigned the device-unique id "123"

9. **Confirm that the transaction has been queued**
   The device queue acknowledges to the JSON-RPC module that the transaction has been queued with id 123.

10. **Confirm that the transaction has been queued**
    The confirmation is forwarded by the JSON-RPC module to the backend server.

11. **Device connects to Exosense server**
    The device, either triggered by a signal sent to it, or through a scheduled event, connects to the Exosense server. The protocol plugin listening to the given address/port receives the connection and initiates communication with the device.

12. **Request queued transactions**
    The plugin sends a get-transactions call to the device queue associated with the connected device.

13. **Return queued transactions**
    The transaction created in step 8 above is returned to the protocol plugin to be forwarded to the device.

14. **Forward transaction to device**
    The protocol plugin transmits the transaction to the device using its native protocol. The device, having received the transaction, processes it and updates its local configuration database.

15. **Confirm transaction to protocol plugin**
    The device acknowledges to the protocol plugin that the transaction was successfully executed.

16. **Confirm transaction to device queue**
    The protocol plugin forwards the acknowledgment for the transaction to the device queue. The device queue reacts by marking the transaction as complete and removes it.

17. **Confirm transaction to JSON-RPC module**
    The transaction acknowledgment is forwarded to the JSON-RPC module.

18. **Confirm transaction to backend server**
    The transaction acknowledgment is forwarded to the JSON-RPC module.

7.3 Device Group Transaction sequence

Transactions can be queued to a device group instead of a specific device. When the transaction targets a device group, the transaction will be cascaded out to all devices hosted by the group. Each cascaded transaction will be treated as an individual device transaction.

Figure 10 shows a sequence diagram showing how a transaction sent to the device group shown in [Device Group Example] is processed. Please note that some steps, such as the set config JSON-RPC calls and device callin details have been omitted for clarity.

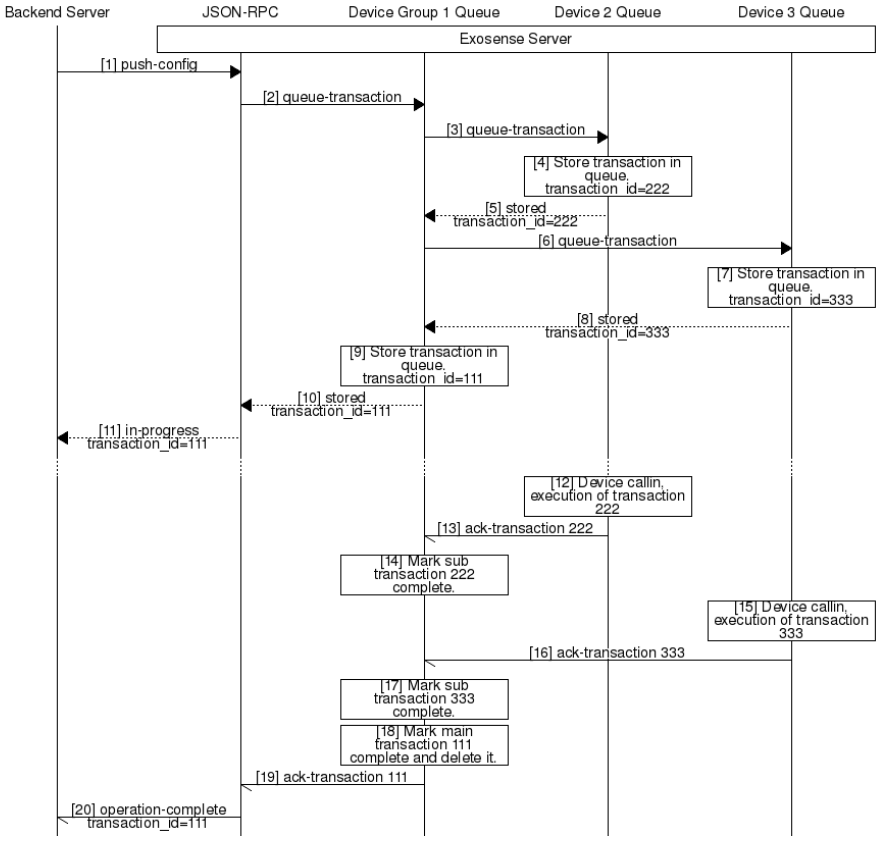

Figure 10: Device Group Transaction Sequence Diagram

The steps outlined in the sequence diagram are as follows:

1. **Request config data to be pushed to device group**
   The backend server sends a push-config command to the Exosense server

to initiate transmission of the two configuration updates to it and all its hosted devices.

2. **Queue all prepared configuration updates as a transaction**
   The Exosense server creates a single transaction, containing all configuration updates (not shown) sends it to be queued for the device group.

3. **Cascade transaction to device 2**
   The device group queue forwards the transaction received to the queue of device 2.

4. **Store transaction for device 2**
   The transaction to device 2 is stored and assigned a transaction ID of 222. This ID is unique for device 2.

5. **Confirm device 2 transaction store**
   Device queue 2 confirms that the transaction has been queued for device 2 and assigned a transaction id of 222. The device group queue will record the received transaction ID. (See step 9)

6. **Cascade transaction to device 3**
   The device group queue forwards the transaction received to the queue of device 3.

7. **Store transaction for device 3**
   The transaction to device 3 is stored and assigned a transaction ID of 333. This ID is unique for device 3.

8. **Confirm device 3 transaction store**
   Device queue 3 confirms that the transaction has been queued for device 3 and assigned a transaction id of 333. The device group queue will record the received transaction ID. (See step 9)

9. **Store transaction for device 1**
   A transaction is created for device group 1. This transaction will be completed once both sub-transactions 222 and 333, sent to device 2 and 3, have been completed.

10. **Confirm that the transaction has been queued**
    Device Group 1 queue acknowledges to the JSON-RPC module that the transaction has been queued with id 111. Sub-transactions 222 and 333 are hidden within the system and are not reported back to JSON-RPC or the backend server.

11. **Confirm that the transaction has been queued**
    The confirmation is forwarded by the JSON-RPC module to the backend server.

12. **Device 2 connects to Exosense server**
    Device 2 connects to the server, receives and executes sub-transaction 222, and sends a confirmation back to the p device 2 queue through the protocol plugin (not shown).

13. **Confirm transaction to device group 1 queue**
    Device 2 queue acknowledges to device group 1 queue that sub-transaction 222 has been executed on the device.

14. **Mark sub-transaction 222 as complete**
    The device group 1 queue will mark sub-transaction 222 as executed, leaving sub-transaction 333 left to execute before the main transaction 111 in device group 1 queue is also complete.

15. **Device 3 connects to Exosense server**
    Device 3 connects to the server, receives and executes sub-transaction 333, and sends a confirmation back to the device 3 queue through the protocol plugin (not shown).

16. **Confirm transaction to device group 1 queue**
    Device 3 queue acknowledges to device group 1 queue that sub-transaction 333 has been executed on the device.

17. **Mark sub-transaction 333 as complete**
    The device group 1 queue will mark sub-transaction 333 as executed, leaving no further sub-transactions to be executed in order for transaction 111 in device group 1 queue to be complete.

18. **Mark transaction 111 as complete**
    Device group 1 queue marks transaction 111 as complete.

19. **Confirm transaction 111 to JSON-RPC module**
    The acknowledgment for transaction 111 is forwarded to the JSON-RPC module.

20. **Confirm transaction 111 to backend server**
    The acknowledgment for transaction 111 is forwarded to the backend server.

8 JSON-RPC Protocol Overview

8.1 General concepts

The JSON-RPC interface conforms to the JSON-RPC 2.0 standard [1].

Since calls invoking a device may take a long time (minutes to weeks) to execute, a callback scheme is used to notify the backend server of the progress of a call.

Figure 11 shows the general flow of data.

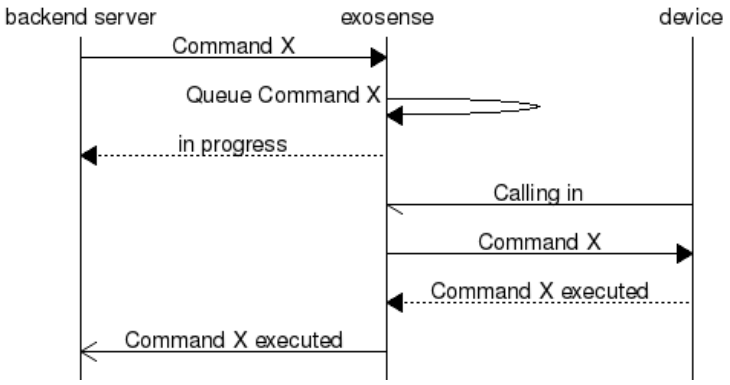

Figure 11: JSON RPC General Flow of Data

8.2 Authentication of JSON-RPC commands Authentication is done through

HTTP basic access authentication, providing the name and password of a provisioned user as credentials for the JSON-RPC command. Since these credentials are transferred in plaintext, HTTPS or an encrypted tunnel is recommended for communication between the backend server and the Exosense Server unless both operate in a trusted environment.

The authenticating user is associated with one or more accounts, each owning its own set of devices, groups, packages, etc. In cases where a user is associated with more than one account, each request sent to the Exosense Server must specify which account the request is targeting. The Exosense Server will use the user credentials to determine if the user is assigned to the account (through an exodm:add-user-to-account), and if the user is also assigned to a role within that account (through exodm:add-user-to-role) that has execution rights to the given RPC under the account (assigned through exodm:add-role-execution-rights).

If the user passes all validations, the RPC will be executed by the Exosense server, or, if applicable, forwarded to the targeted device or group.

8.3 Device Transaction Queuing

When transactions, such as a configuration set or an RPC command, is to be transmitted to a device, three scenarios may unfold.

1. **The device is on-line**
   This will be the case if the device has permanent connectivity and a well known (IP) address. The other case is if the device has periodical connectivity but happens to be on-line when the data is to be transmitted.
2. **The device is offline, but can be brought on-line**
   An offline device can, depending on the network it is on, be brought on-line by calling it, sending a wake up SMS, or through other out-of-band mechanisms.
3. **The device is offline and we cannot bring it on-line**
   In some cases, only the device can initiate a connection to the Exosense server. This usually happens at regular intervals or when a specific event, such as an alert, occurs in the device.

In case 2 and 3 above the Exosense Server will queue any data destined for the device until either the device comes on-line, or a timeout occurs. The timeout interval is specified by the JSON-RPC command that initiated the data transfer.

If a command targets multiple devices (such as push-config-set) the data will be queued on a per-device basis, thus ensuring that each recipient device gets the data when it becomes available.

8.4 Exosense JSON-RPC Command Role Assignment

Below is the execution right matrix for the roles and the built in JSON-RPC commands.

Please note that the roles defines the execution rights for its member users on a specific account. If the user is only a member of a single account, the JSON-RPC command will be executed on that account. If the user is member of multiple accounts, the account is specified together with the user as a part of the authorization credentials to indicate which account the command applies to.

JSON-RPC	master	admin	view	device	config	execute
exodm:create-account	X					
exodm:update-account	X					
exodm:delete-account	X					
exodm:add-user-to-account	X	X				
exodm:remove-user-to-account	X	X				
exodm:list-account-users	X	X	X			
exodm:create-user	X	X				
exodm:delete-user	X	X				
exodm:list-user	X	X	X			
exodm:create-device-type	X	X		X		
exodm:update-device-type	X	X		X		
exodm:delete-device-type	X	X		X		
exodm:list-device-types	X	X	X	X		
exodm:list-device-type-members	X	X	X	X		
exodm:create-device-group	X	X		X		
exodm:update-device-group	X	X		X		
exodm:delete-device-group	X	X		X		
exodm:list-device-groups	X	X	X	X		
exodm:list-device-group-members	X	X	X	X		
exodm:add-device-group-members	X	X		X		
exodm:remove-device-group-members	X	X		X		
exodm:create-yang-module	X	X				

exodm:delete-yang-module	X	X			
exodm:list-yang-modules	X	X	X		
exodm:create-package	X	X			
exodm:add-package-members	X	X		X	
exodm:push-package	X	X		X	
exodm:list-packages	X	X	X	X	
exodm:get-package-status	X	X	X	X	
exodm:create-config-set	X	X		X	X
exodm:update-config-set	X	X		X	X
exodm:delete-config-set	X	X		X	X
exodm:list-config-sets	X	X	X	X	X
exodm:list-config-set-members	X	X	X	X	X
exodm:add-config-set-members	X	X		X	X
exodm:remove-config-set-members	X	X		X	X
exodm:push-config-set	X	X		X	X
exodm:create-device	X	X		X	
exodm:lookup-device	X	X	X	X	
exodm:list-devices	X	X	X	X	
exodm:delete-device	X	X		X	
exodm:add-user-to-role	X	X			
exodm:remove-user-from-role	X	X			
exodm:add-execution-permission	X	X			
exodm:remove-execution-permission	X	X			
exodm:list-execution-permissions	X	X	X		

Table 2: JSON-RPC built-in commands and roles

8.5 Assigning roles to custom JSON-RPC commands

A package installed on a device can have an RPC interface, specified through an YANG module, associated with it. In order for the API to be accessed, as JSON-RPC commands, on the server, the authorizing user needs to gain access to the given package.

Access is provided in two steps.

First, a user is assigned to a role in the account owning the device. Role assignment is done through the exodm:add-user-to-role command that takes a user name and a role as its arguments. Once a user has been assigned to a role, that role is given execution rights to the RPCs associated with a package. After this, the given user can execute the package RPCs on all devices with the package installed on them.