sdl

SDL Server Guides

Document current as of 04/13/2022 10:53 AM.

Overview

This document provides the information for creating and integrating the SmartDeviceLink (SDL) server component with the mobile libraries and vehicle's Head Unit (HU).

The Policy Server's main purpose is to curate policy tables composed of rules and permissions defined by a vehicle's OEM. Each vehicle will download its Policy Table and use it to govern SDL behaviors.

Abbreviations and Definitions

Abbreviations used in this document are collected in the table below

ABBREVIATION	MEANING
BT	Bluetooth
GUI	Graphical User Interface
HMI	Human Machine Interface
IVI	In Vehicle Infotainment
JSON	JavaScript Object Notation
OEM	Original Equipment Manufacturer
RPC	Remote Procedure Call
SDE	Software Development Environment
SDL	SmartDeviceLink
SEE	Software Engineering Environment
TTS	Text To Speech
VDOP	Vertical Dilution of Precision
VR	Voice Recognition

Introduction

Below are the API routes that the Policy Server exposes.

There may be CORS issues in the case where a separate web app needs to make API calls to the policy server, such as an HMI using the `applications/store` route. For cases like these, CORS is enabled. Preflight is also enabled for sufficiently complex POST requests. See below for the routes which have CORS or preflight enabled by default. The full list of routes and their middleware can also be seen in the project's `app/v1/app.js` file.

ROUTE	CORS	PREFLIGHT
POST /api/v1/staging/policy	true	true
POST /api/v1/production/policy	true	true
GET /api/v1/applications/store	true	false
GET /api/v1/applications/store/staging	true	false

`POST /api/v1/login`

If basic authentication is enabled, the Policy Server UI opens a login page on startup which will call this route. The Policy Server will then validate that the entered password matches the one set up by the server maintainer.

`GET /api/v1/applications` **&** `GET /api/v1/groups` **&** `GET /api/v1/messages`

Retrieves information regarding applications, functional groups, or consumer friendly messages. An **id** (or additionally **uuid** for applications) can be specified so as to retrieve information for a specific item. Functional groups and consumer messages can be set to return templates containing all necessary information on that item being stored in the

database. Applications can be filtered by approval status. If no parameters are specified `/applications` will return the latest version of each app, `/groups` and `/messages` will return the latest version of all functional groups or consumer messages in either production or staging mode.

`POST /api/v1/applications/action`

Updates an application's approval status. In the future this route will also notify the app's developer via email of the change in approval status.

`POST /api/v1/applications/auto`

If an application has been set to automatically approve all future updates then this route will validate the app uuid and update the approval status. In the future this route will also notify the app's developer via email of the change in approval status.

`POST /api/v1/applications/administrator`

This route updates whether an app will have access to administrator functional groups.

`POST /api/v1/applications/passthrough`

This route updates whether an app will be able to send unknown RPCs through App Service RPC Passsthrough.

`POST /api/v1/applications/hybrid`

This route updates the hybrid preference of an app.

`PUT /api/v1/applications/rpcEncryption`

This route updates whether an app should have RPC Encryption enabled.

`PUT /api/v1/applications/service/permission`

This route modifies the App Services permissions of an application.

`POST & GET /api/v1/applications/certificate/get`

This route queries the Policy Server database for an app's certificate and returns it, unless it's expired. If it is expired a 400 response is returned. Either appId or AppId is required in the query or the json body of the request.

Example requests:

```
GET /api/v1/applications/certificate/get?appId=31cc4209-79e7-4704-9ec4-3b485d3eeb93
```

OR

```
POST /api/v1/applications/certificate/get
{
   appId: 31cc4209-79e7-4704-9ec4-3b485d3eeb93
}
```

Response:


```
{
   "meta": {
      "request_id": "427a7fb4-f2f1-44d6-8c2b-e7d927790960",
      "code": 200,
      "message": null
   },
   "data": {
      "certificate": "MIIKMQIBAzCCCf..."
   }
}
```

The certificate is a Base64 encoded string containing the pkcs12 certificate. This contains the certificate and private key and can be read using an openssl library with the password provided as `CERTIFICATE_PASSPHRASE` in your server's .env settings.

Example using openssl (note that the cert is a Base64 string and the `` `CERTIFICATE_PASSPHRASE `` is used to read the pkcs12 certificate):

```
echo "MIIKMQIBAzCCCf..." | base64 -D > app-cert.p12 && openssl pkcs12 -nokeys -in app-cert.p12 -passin pass:CERTIFICATE_PASSPHRASE
```

`POST /api/v1/applications/certificate`

This route updates the pkcs12 certificate of an application in the database.

`GET /api/v1/applications/groups`

Returns the functional groups for which a given application has access.

`PUT /api/v1/applications/groups`

Updates the functional groups for which a given application has access.

`GET /api/v1/applications/store`

Retrieves approved, embedded application information, filterable by `uuid` or by `transport_type`. The possible values for `transport_type` are `webengine` and `websocket`. The return object includes app bundle information such as the location of the bundle and its file size, compressed and uncompressed. The logic of where to store these app packages is customizable by the policy server. See the `customizable/webengine-bundle/index.js` file for details.

`GET /api/v1/applications/store/staging`

Does the same thing as /api/v1/applications/store, but returns apps whose approval statuses are in staging.

`POST /api/v1/webhook`

This is the route that should be specified on a company's page on the SDL Developer Portal (in the box titled Webhook URL under Company Info) to be hit by the SHAID server when an app has been updated.

`POST /api/v1/staging/policy` **&** `POST /api/v1/production/policy`

These are the routes sdl_core's default Policy Table should use when requesting a Policy Table update with either `/staging` or `/production` specified.
Given a "shortened" Policy Table, the Policy Server will use that information to automatically construct a full Policy Table response and return it to the requester.

`GET /api/v1/policy/preview`

This is the route hit by the Policy Server UI requesting a preview of the Policy Table. A variable **environment** indicates whether it is to be staging or production.

`POST /api/v1/policy/apps`

The Policy Server UI makes a request to this route which returns an example Policy Table segment for a particular app.

`POST /api/v1/permissions/update`

The route updates the available permissions and permission relationships from SHAID.

`GET /api/v1/permissions/unmapped`

This route returns a list of permissions that are currently not attributed to any functional groups.

`POST /api/v1/groups` & `POST /api/v1/messages`

These routes are hit by the Policy Server UI to update a functional group's/consumer message's information or to change its deleted status.

`GET /api/v1/groups/names` & `GET /api/v1/messages/names`

These routes return the names of all functional groups or consumer friendly messages recognized by the Policy Server.

`POST /api/v1/groups/promote` & `POST /api/v1/messages/promote`

These routes are hit by the Policy Server UI to promote a functional group or consumer message from staging to production. If the functional group has a user consent prompt associated with it then the consent prompt must be promoted to production before promoting the functional group.

`POST /api/v1/messages/update`

This route updates the Policy Server's list of languages.

`GET /api/v1/module`

This route will return either the staging or production module config object.

`POST /api/v1/module`

This route will update the staging module config object on record.

`POST /api/v1/module/promote`

This route will promote the current staging module config to production.

`POST /api/v1/security/certificate`

This route will return a PEM certificate. It is used by the UI when generating app certificates and must be provided a private key in order to function.

`POST /api/v1/security/private`

This route will return a RSA private key. It is used by the UI when generating an application's private keys.

`POST /api/v1/vehicle-data`

This route will add or update a custom vehicle data item.

`GET /api/v1/vehicle-data`

This route will return a list of custom vehicle data items filtered by status and optionally by id.

`POST /api/v1/vehicle-data/promote`

This route will promote the custom vehicle data on staging to production.

`GET /api/v1/vehicle-data/type`

This route will return a list of all the data types and custom vehicle data parameter types on record.

User Interface Pages

These are API routes that are accessed by the Policy Server user interface.

`/applications`

The Applications page.

`/applications/:id`

The App Details page with information regarding an app specified by the **id**. The Applications page documentation contains more information pertaining to this page.

`/policytable`

The View Policy Table page.

`/functionalgroups`

The Functional Groups page.

`/functionalgroups/manage`

The Functional Group Details page with information regarding a functional group that is specified by an **id**. The Functional Groups page documentation contains more information pertaining to this page.

`/consumermessages`

The Consumer Friendly Messages page.

`/consumermessages/manage`

The Consumer Message Details page with information regarding a consumer message that is specified by an **id**. The Consumer Messages page documentation contains more information pertaining to this page.

`/vehicledata`

The Custom Vehicle Data page.

`/moduleconfig`

The Module Config page.

`/about`

The About page.

Docker Install (For versions 3.0+)

Docker Compose Installation

Docker Engine is required to be installed. The `docker` folder in the policy server contains all the files needed to set up the policy server through docker images. The `docker-compose.yml` file will spin up the server, the Postgres database, and the Redis database and automatically connect them all. The policy server is made available on `http://localhost:3000`.

Environment Variables

An `.env` file is expected in the `docker` directory, and the Dockerfile will pull in all environment variables from that file, just like how the policy server uses the `.env` file in the root directory. The Dockerfile uses the remote sdl_server repository instead of the local installation. The branch can be changed by changing the `docker-compose.yml` file's arg VERSION value: its default is the master branch.

The following are notable `.env` variables to the docker environment. They are not a comprehensive list. The usual variables such as `SHAID_PUBLIC_KEY` and `SHAID_SECR`

`ET_KEY` are still required for usage. Connection to postgres and redis is automatic and no further configuration is required for them, such as setting environment variables.

NAME	TYPE	USAGE	DESCRIPTION
DB_HOST	String	Postgres	Please do not use this value. It is predefined to work with Docker Compose
DB_PASSWORD	String	Postgres	Not required to be set. Defaults to "postgres"
DB_USER	String	Postgres	Not required to be set. Defaults to "postgres"
DB_DATABASE	String	Postgres	Not required to be set. Defaults to "postgres"
CACHE_HOST	String	Redis	Please do not set this value. It is predefined to work with Docker Compose
BUCKET_NAME	String	WebEngine app support	The name of the S3 bucket to store app bundles
AWS_REGION	String	WebEngine app support	The region of the S3 bucket
AWS_ACCESS_KEY_ID	String	WebEngine app support	AWS credentials to allow S3 usage. These are exclusive to the docker install of the policy server!

NAME	TYPE	USAGE	DESCRIPTION
AWS_SECRET_ACCESS_KEY	String	WebEngine app support	AWS credentials to allow S3 usage. These are exclusive to the docker install of the policy server!

Note the nearly empty `keys` subfolder. Insert your own key and pem files meant for the certificate generation feature and SSL connections in there, and the contents will be copied into the docker container policy server's `customizable/ca` folder and `customizable/ssl` folder. You will still need the necessary environment variables to activate certificate generation and SSL connections respectively.

Commands

You need to run the following commands in the docker directory of the project.

To start a new or existing cluster, remembering to rebuild the policy server image in case of .env changes (Make sure you are in the `docker` folder of the policy server):
`docker compose up --build`
Use Ctrl+C once to stop all the docker containers.

To tear down a cluster without removing the volume (this will delete the database contents!):
`docker compose down`

To tear down a cluster and remove the volume (this will delete the database contents!):
`docker compose down -v`

Read the rest of this page if you wish to launch the server without the use of Docker.

Normal Installation

Prerequisites

The following must be installed before installation of the Policy Server can begin:

PROJECT	VERSION
Postgres	9.6+
Node.js	8.12.0+
NPM	3.0.0+

Note: For policy server major version 2, be aware it will not function if the Node.js version is 13 or higher.

You must also acquire a set of SHAID API keys. These are made available to level 4 OEM members through the developer portal.

NOTE: Be careful not to use sets of SHAID API keys from multiple vendors. Some Policy Server actions (like changing the auto-approval status of an app) will attempt to send information back to SHAID and if the wrong SHAID API keys are used then the action may fail.

Setup Guide

Download the project to your current directory.

```
git clone https://github.com/smartdevicelink/sdl_server.git
cd sdl_server
```

The recommended branch to use is master, which should be used by default. Install dependencies.

```
npm install
```

The Policy Server requires a SQL database, and currently the only supported implementation is PostgreSQL. In the next section, we will cover how to get one running locally.

PostgreSQL Installation (Mac)

To install PostgreSQL on a Mac with Homebrew, run the following command in a Terminal window:

```
brew install postgresql
```

Then run the following command to start PostgreSQL, and ensure that you won't need to start it again in case your system resets:

```
pg_ctl -D /usr/local/var/postgres start && brew services start postgresql
```

You can run the following command to know if you have PostgreSQL and also check that you are running the most recent version:

```
psql -V
```

PostgreSQL Installation (Ubuntu)

To install PostgreSQL in Ubuntu, run the following commands from the PostgreSQL documentation:

```
# Create the file repository configuration:
sudo sh -c 'echo "deb http://apt.postgresql.org/pub/repos/apt $(lsb_release -cs)-pgdg main" > /etc/apt/sources.list.d/pgdg.list'

# Import the repository signing key:
wget --quiet -O - https://www.postgresql.org/media/keys/ACCC4CF8.asc | sudo apt-key add -

# Update the package lists:
sudo apt-get update

# Install the latest version of PostgreSQL.
# If you want a specific version, use 'postgresql-12' or similar instead of 'postgresql':
sudo apt-get -y install postgresql
```

You can run the following command to know if you have PostgreSQL and also check that you are running the most recent version:

```
psql -V
```

Logging in to PostgreSQL

In order to start creating users and databases, you will have to log in to PostgreSQL. It comes with a `postgres` user that should have no password by default. Run the following command to log in as the `postgres` user:

```
psql -U postgres
```

```

```

If you're prompted for a password but have not yet set one, you'll have to locate and modify your pg_hba.conf file. Find the line that contains

```
local  all     postgres     peer
```

Update it to contain

```
local  all     postgres     trust
```

Then restart postgres and attempt to log in to postgres again

```
sudo service postgresql restart
psql -U postgres
```

Creating the PostgreSQL Database

You should now be in the postgres command-line interface. You can type `help` to get more info. If you want to continue using the `postgres` user, you can add a password with the following command:

```
ALTER USER postgres WITH PASSWORD '<password>';
```

If you want to create a new user, run the following commands to create one with a password and give them super user access:

```
CREATE USER <username> WITH PASSWORD '<password>';
ALTER USER <username> WITH SUPERUSER;
```

Alternatively you can use the GRANT command to limit the user's permissions. In the future, you can log in to PostgreSQL using this new user. Next, you'll need to run the following command to add a new database for the Policy Server to manage:

```
CREATE DATABASE <database_name>;
```

This database will be where the Policy Server stores all of its data pertaining to policy table generation. Remember to save your PostgreSQL username, password, and database name so you can use them in the next section. To exit the PostgreSQL CLI, simply type `quit` and hit Enter.

Environment Variables

Once you set up a database (locally or remotely) you'll need to supply the Policy Server with some environment variables. This Policy Server uses the dotenv module, meaning you can write all your environment variables in a `.env` file located in the root directory of the Policy Server. The Policy Server will load the variables at `.env`. `.env` files will not be tracked by Git.

There are several settings that can be configured for Policy Server usage. See below for explanations on the purpose of each of them.

Basic Environment Variables

NAME	TYPE	EXAMPLE	DESCRIPTION
POLICY_SERVER_HOST	String	testing.com	The hostname or public IP address which the server runs on
POLICY_SERVER_PORT	Number	3000	The port which the server runs on. It is optional and the default is 3000
DB_USER	String	postgres	The name of the user to allow the server to access the database
DB_DATABASE	String	postgres	The name of the database where policy and app data is stored
DB_PASSWORD	String	password	The password used to log into the database
DB_HOST	String	rds-database.com	The host name or IP address of the database
DB_PORT	Number	5432	The port number of the database
TEST_PG_USER	String	postgres	Same as DB_USER but for specifically running tests via `npm run test`

NAME	TYPE	EXAMPLE	DESCRIPTION
TEST_PG_DATABASE	String	postgres	Same as DB_DATABASE but for specifically running tests via `npm run test`
TEST_PG_PASSWORD	String	password	Same as DB_PASSWORD but for specifically running tests via `npm run test`
TEST_PG_HOST	String	rds-database.com	Same as DB_HOST but for specifically running tests via `npm run test`
TEST_PG_PORT	Number	5432	Same as DB_PORT but for specifically running tests via `npm run test`

SHAID Environment Variables

NAME	TYPE	DESCRIPTION
SHAID_PUBLIC_KEY	String	A public key given to you through the developer portal that allows access to SHAID endpoints.
SHAID_SECRET_KEY	String	A secret key given to you through the developer portal that allows access to SHAID endpoints.
SHAID_URL	String	The location of the SHAID server. The default value will query the production SHAID server. It is not recommended to change this value.

Caching Environment Variables

NAME	TYPE	EXAMPLE	DESCRIPTION
CACHE_MODULE	String	Redis	The name of the caching module to use. Currently supports null (no caching, default) or "redis"
CACHE_HOST	String	redis-host.com	The host name or IP address of the cache server
CACHE_PORT	Number	6379	The port number of the cache server
CACHE_PASSWORD	String	password	The password used to log into the cache server

Emailing Environment Variables

NAME	TYPE	EXAMPLE	DESCRIPTION
SMTP_HOST	String	smpt-host.com	The host name or IP address of an SMTP server to use for email notifications. A null value implies that outgoing emails are disabled
SMTP_PORT	Number	25	The port number of the SMTP server. The default is 25
SMTP_USERNAME	String	smtp	The username of the optional SMTP user
SMTP_PASSWORD	String	password	The password of the optional SMTP user
SMTP_FROM	String	example@email.com	The email address which emails are sent from. A null value implies that outgoing emails are disabled
NOTIFY_APP_REVIEW_FREQUENCY	String Enum (DISABLED, REALTIME)	REALTIME	The frequency of which outgoing emails should be sent to notify the OEM of new apps ready for review. The default is DISABLED

NAME	TYPE	EXAMPLE	DESCRIPTION
NOTIFY_APP_REVIEW_EMAILS	String with comma-separated values	example1@email.com,example2@email.com	A comma-separated list of email addresses to send an email to when new apps are ready for review

Mandatory Certificate and Encryption Environment Variables

NAME	TYPE	EXAMPLE	DESCRIPTION
CA_PRIVATE_KEY_FILENAME	String	CA.key	The filename of your .key file generated, to be placed in `customizable/ca/`
CA_CERTIFICATE_FILENAME	String	CA.pem	The filename of your .pem file generated, to be placed in `customizable/ca/`
CERTIFICATE_PASSPHRASE	String	password	A secret password used for every certificate generated
CERTIFICATE_COMMON_NAME	String	*.company.com	Default information of the issuer's fully qualified domain name to secure

Optional Certificate and Encryption Environment Variables

NAME	TYPE	EXAMPLE	DESCRIPTION
POLICY_SERVER_PORT_SSL	Number	443	The port which the server should listen for SSL connections on (typically 443). It is optional and the default is `null` (do not listen for SSL connections)
SSL_CERTIFICATE_FILENAME	String	file.pem	The filename of the SSL certificate located in `./customizable/ssl`. Required if a value is set for `POLICY_SERVER_PORT_SSL`
SSL_PRIVATE_KEY_FILENAME	String	file.key	The filename of the SSL certificate's private key located in `./customizable/ssl`. Required if a value is set for `POLICY_SERVER_PORT_SSL`
PRIVATE_KEY_BITSIZE	Number	2048	The size of the private keys generated. Default 2048

NAME	TYPE	EXAMPLE	DESCRIPTION
PRIVATE_KEY_CIPHER	String	des3	The type of cipher to use for encryption/decryption. Defaults to "des3"
CERTIFICATE_COUNTRY	String	US	Default information of the issuer's country (two-letter ISO code)
CERTIFICATE_STATE	String	Michigan	Default information of the issuer's state
CERTIFICATE_LOCALITY	String	Royal Oak	Default information of the issuer's city
CERTIFICATE_ORGANIZATION	String	Livio	Default information of the issuer's legal company name
CERTIFICATE_ORGANIZATION_UNIT	String	Human Resources	Default information of the issuer's company's branch
CERTIFICATE_EMAIL_ADDRESS	String	example@email.com	Default information of the issuer's email address
CERTIFICATE_HASH	String	sha256	The cryptographic hash function to use. Defaults to 'sha256'

NAME	TYPE	EXAMPLE	DESCRIPTION
CERTIFICATE_DAYS	Number	7	The number of days until the certificate expires. Defaults to 7
ENCRYPTION_REQUIRED	Boolean	true	Whether or not to require RPC encryption for auto-approved app versions. Defaults to "false"
MODULE_CONFIG_ENCRYPT_CERT_BUNDLE	Boolean	true	Whether to package the module config's certificate and private key into a pkcs12 bundle string using the CERTIFICATE_PASSPHRASE. If false (default), it will just be a concatenation of the certificate and the private key

Miscellaneous Environment Variables

NAME	TYPE	EXAMPLE	DESCRIPTION
AUTO_APPROVE_ALL_APPS	Boolean	true	Whether or not to auto-approve all app versions received by SHAID (except for blacklisted apps). Defaults to "false"

The Policy Server comes with migration scripts that can be run using npm scripts. You can see a list of all the possible scripts by looking in `package.json`, but these are the most important ones:

- `start-server`: Runs the migration up script which initializes data in the database and starts the Policy Server
- `dev` or `start`: Starts the dev server with hot reloading so any changes made to the UI are instantly updated in the browser

NOTE: Using the dev server can cause CORS issues when connecting to the API so it should only be used when testing UI changes.

- `build`: Generates a new staging/production build using webpack. Not required to be used if you're using the start-server script.
- `lint`: Parses the Policy Server code and checks for syntactical or stylistic errors.
- `test`: Runs the unit tests packaged with the project. Uses the `TEST_` database environment variables to modify the database. **This will clear all policy server data when running! Make sure you use a database you do not mind being cleared!**
- `db-migrate-up`: Runs all migrations on the database.
- `db-migrate-reset`: Runs migration downs and clears the database.

Run the following command to finalize set up and start the server.

```
npm run start-server
```

Verify that it started properly by navigating to your configured host and port, or to the default address: `http://localhost:3000/`

Now you have a Policy Server running!

- If you wish to enable caching with an unofficially supported datastore, you may create a custom cache module. Do so by creating a folder inside `custom/cache` with the name of your module. Put your implementation in a file named `index.js` inside of your module's folder. Your module should export the following functions:
    - `get(key, callback)`: Receives a value from the cache stored at key.
    - `set(key, value, callback)`: Sets a value in the cache stored at key.
    - `del(key, callback)`: Deletes a value from the cache stored at key.
    - `flushall(callback)`: Deletes all data previously set in the cache.
- Set your `CACHE_` environment variables to correspond with your new datastore solution and access information.

Security

For your convenience, we have implemented the following security features into the Policy Server.

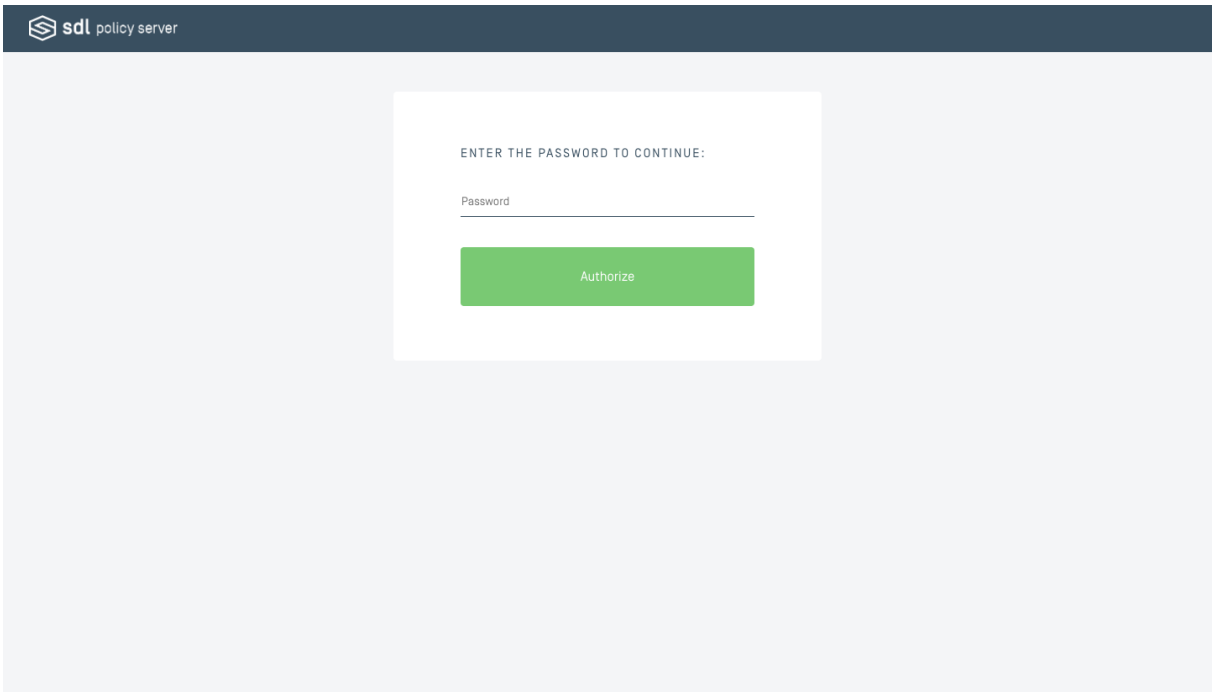
HTTPS Connections (SSL/TLS)

HTTPS connections (disabled by default) can be enabled by doing the following:
Store your SSL Certificate and Private Key files in the `./customizable/ssl` directory
Set your `POLICY_SERVER_PORT_SSL` environment variable to your desired secure port (typically 443)
Set your `SSL_CERTIFICATE_FILENAME` environment variable to the filename of your SSL Certificate file
Set your `SSL_PRIVATE_KEY_FILENAME` environment variable to the filename of your Private Key file
If you are unable to modify your environment variables, you may define these settings in the `./settings.js` configuration file
Restart your Policy Server and navigate to your server's hostname on the secure port!

Basic Authentication

You may optionally require your Policy Server administrators to enter a password before being able to access the user interface. We recommend using a more secure method of authentication in accordance to your company's IT security standards, but provide this basic authentication feature for convenience.

By default, basic authentication is disabled. To enable it, simply set your `AUTH_TYPE` environment variable to `basic` and your `BASIC_AUTH_PASSWORD` environment variable to a password of your choice, then restart your Policy Server. If you are unable to modify your environment variables, you may define these settings in the `./settings.js` configuration file.



Policy Table Encryption

You may wish to encrypt your Policy Table when in transit to/from SDL Core. To achieve this, we've implemented skeleton methods to house your custom encryption logic. The Policy Table JSON object (array) is passed to these methods so you can run encryption and decryption transformations against it. By default, these methods perform no transformations.

The customizable Policy Table skeleton `encryptPolicyTable` and `decryptPolicyTable` methods are located in the Policy Server project at the following file path: `./customizabl`

`e/encryption/index.js`

If you modify this skeleton method to implement Policy Table encryption on your Policy Server, you will also need to implement corresponding cryptography logic via the `crypt` and `decrypt` methods in your build of SDL Core. These methods are available in the `sample_policy_manager.py` file of SDL Core.

Configurable CA Key and Certificate Creation

If you are attempting to use encrypted RPCs with SDL Core, you will need to have certificates for both Core and the Mobile Proxy. Generating the CA key and certificate files will have to be done manually (see below). After they are created and certificate generation is enabled, additional ones can be created via the Policy Server UI. The Policy Server uses a wrapper for OpenSSL to provide the same options that would normally be provided when directly dealing with OpenSSL.

PREREQUISITES

OpenSSL version 1.1.0+ must be installed. The source files can be found here along with instructions for installation.

Once OpenSSL is properly installed, you'll need to take the necessary steps to establish a certificate authority. The CA will be responsible for signing all certificates created by the policy server. This can be done by simply entering the following two commands into any terminal:

COMMAND	EXPLANATION
openssl genrsa -out CA.key 2048	This creates a 2048 bit RSA private key and saves it in the file "CA.key". It will later be used for signing certificates.
openssl req -x509 -new -nodes -key CA.key -sha256 -days 3650 -out CA.pem	This creates a certificate in the file name "CA.pem" that will be used in the creation of additional certificates. It is set to expire after 10 years. OpenSSL will then prompt you for further information.

The CA files will then need to be relocated to the `./customizable/ca` folder and their file names will need to be specified in the `.env` file.

The following environment variables are the most relevant for getting the policy server set up to start creating certificates on its own:

VARIABLE	IS MANDATORY	DESCRIPTION
CA_PRIVATE_KEY_FILENAME	true	The filename of your .key file generated, to be placed in customizable/ca/
CA_CERTIFICATE_FILENAME	true	The filename of your .pem file generated, to be placed in customizable/ca/
CERTIFICATE_PASSPHRASE	true	A secret password used for every certificate generated.
CERTIFICATE_COMMON_NAME	true	Default information of the issuer's fully qualified domain name to secure
PRIVATE_KEY_BITSIZE	false	The size of the private keys generated. Defaults to 2048.
PRIVATE_KEY_CIPHER	false	The type of cipher to use for encryption/decryption. Defaults to "des3".
CERTIFICATE_COUNTRY	false	Default information of the issuer's country (two-letter ISO code).
CERTIFICATE_STATE	false	Default information of the issuer's state.
CERTIFICATE_LOCALITY	false	Default information of the issuer's city.
CERTIFICATE_ORGANIZATION	false	Default information of the issuer's legal company name.

VARIABLE	IS MANDATORY	DESCRIPTION
CERTIFICATE_ORGANIZATION_UNIT	false	Default information of the issuer's company's branch.
CERTIFICATE_EMAIL_ADDRESS	false	Default information of the issuer's email address
CERTIFICATE_HASH	false	The cryptographic hash function to use. Defaults to sha256.
CERTIFICATE_DAYS	false	The number of days until the certificate expires. Defaults to 7 days.
MODULE_CONFIG_ENCRYPT_CERT_BUNDLE	false	Whether to package the module config's certificate and private key into a pkcs12 bundle string using the CERTIFICATE_PASSPHRASE. If false (default), it will just be a concatenation of the certificate and the private key

To know if this process was successful and if your policy server is now capable of generating keys and certificates, check the About page to see if certificate generation is enabled.

Retrieving the Certificates

SDL Core's certificate is stored in the module_config of the policy table and is updated via a Policy Table Update. For an app to retrieve its certificate, it must make either a `GET` or `POST` request to the `/api/v1/applications/certificate/get` endpoint. See the API documentation for more details.

On Startup

When the Policy Server starts up, it will try to update its current information by using external sources such as SHAID. It will do the following:

- Update the permission list and permission relationships. These permissions include RPCs, vehicle parameters and module types.
- Synchronize the app categories from SHAID.
- Update language information. Language code information is retrieved from the SDL RPC spec, specified in `settings.js`. This is used for the consumer friendly messages object.
- Query and store SHAID applications. The Policy Server will grab new or updated application information from SHAID and store it in the Policy Server's database.
- Pull in changes from new releases of the RPC spec, if there are any, and store its information.
- Check and renew certificates for the stored applications, if applicable.
- Check and renew the module config certificate, if applicable.
- After all tasks above have been completed, expose the UI and API routes for the Policy Server. It is important that the Policy Server receives all the information above before allowing requests from Core to happen.
- Set up cron jobs for updating permission information, for generating templates and for updating the languages. The Policy Server does not need a cron job for getting new application information from SHAID because of webhooks.

Occasionally, you may receive a banner on the bottom of the Policy Server UI indicating an update is available. When this occurs, we recommend following the update procedure below to ensure your version of the Policy Server is up-to-date with the latest patches and features.

First, use Git to pull the latest version of the Policy Server:

```
git pull
```

Then, update NPM modules with:

```
npm update
```

Finally, start the server using the typical method:

```
npm run start-server
```

Verify that it started properly by navigating to `http://localhost:3000/`

Now your updated Policy Server is up and running!

The Policy Server allows for some extra configuration through the use of custom modules. The Policy Server relies on these modules for logging and querying tasks, and so the ability to write a new module allows for great flexibility in how these tasks are handled.

Loggers

Only two named functions need to be exported in an object for a valid implementation: `info` and `error`. `info` accepts a string as its first parameter and is used to log non-error messages using the string. `error` accepts a string and is used for logging error messages using the string. Check the default `winston` module for an example.

Databases

Currently only PostgreSQL has been tested enough to be considered a usable type of database for the Policy Server. See the default `postgres` module for an example.

The migration scripts setup the tables necessary to contain all of the functional group info, consumer message info, country information, etc., and populates those tables with the initial data from a default Policy Table. Any updates to this data will come through as another migration up script and a download from the repository will be needed to get those changes. An alert will appear in the UI to notify the user when a new version of the Policy Server exists.

WebEngine Support

With the introduction of WebEngine applications, the Policy Server requires additional setup to be able to support them. This is because WebEngine apps need to be uploaded and servable for execution by supported HMIs, and the Policy Server provides the information necessary for downloading these app bundles. The implementation details are up to the Policy Server maintainer, as the method of hosting these WebEngine app bundles may depend on the environment and manner in which the Policy Server is running.

Getting Started

The entrypoint for the custom implementation starts in the `customizable/webengine-bundle/index.js` file in the project. In the single function stub `handleBundle` the URL of the app bundle is passed in. The goal is for the Policy Server to download the bundle from the passed in URL, extract the bundle to get the compressed and uncompressed file size data, and to host it in a publicly accessible location. The new URL for the WebEngine app bundle and its size information is expected to be returned in the `cb` argument for the `handleBundle` function, and that information will automatically be reflected in future calls to the `/api/v1/applications/store` route. Check the `customizable/webengine-bundle/index.js` file comments for specifics.

It is recommended that the app bundles are hosted in a dedicated online file-sharing service such as AWS's S3 buckets. These URLs are expected to be persistent and unchanging, even after Policy Server restarts or migrations.

S3 Storage Code Example

You may use this code snippet for reference on how to implement the `handleBundle` function. This implementation stores the app bundles on an S3 bucket, and assumes that your computer's credentials are set up to be authenticated with AWS, and that you have installed the `node-stream-zip` and `aws-sdk` node modules to the Policy Server.

```
// skeleton function for customized downloading and extracting of package
information
const request = require('request');
const fs = require('fs');
const UUID = require('uuid');
const AWS = require('aws-sdk');
const StreamZip = require('node-stream-zip');
AWS.config.update({region: 'us-east-1'});
const BUCKET_NAME = 'webengine-bundles';

/**
 * asynchronous function for downloading the bundle from the given url and
extracting its size information
 * @param package_url - a publicly accessible external url that's used to download
the bundle onto the Policy Server
 * @param cb - a callback function that expects two arguments
 *     if there was a failure in the process, it should be sent as the first argument. the
Policy Server will log it
 *     the second argument to return must follow the formatted object below
 *     {
 *         url: the Policy Server should save a copy of the app bundle somewhere
publicly accessible
 *             this url must be a full resolved url
 *         size_compressed_bytes: the number of bytes of the compressed downloaded
bundle
 *         size_decompressed_bytes: the number of bytes of the extracted downloaded
bundle
 *     }
 */
exports.handleBundle = function (package_url, cb) {
    let compressedSize = 0;
    let bucketUrl = '';
    const TMP_FILE_NAME = `${UUID.v4()}.zip`;

    // create a new bucket if it doesn't already exist
    new AWS.S3().createBucket({Bucket: BUCKET_NAME, ACL: 'public-read'}, err => {
        // OperationAborted errors are expected, as we are potentially
        // calling this API multiple times simultaneously
        if (err && err.code !== 'OperationAborted') {
            return cb(err);
        }
        // read the URL and save it to a buffer variable
        readUrlToBuffer(package_url)
            .then(zipBuffer => { // submit the file contents to S3
                compressedSize = zipBuffer.length;
                const randomString = UUID.v4();
                const fileName = `${randomString}.zip`;
                bucketUrl = `https://${BUCKET_NAME}.s3.amazonaws.com/${fileName}`;
                // make the bundle publicly accessible
                const objectParams = {Bucket: BUCKET_NAME, ACL: 'public-read', Key:
fileName, Body: zipBuffer};
                // Create object upload promise
```

```
            return new AWS.S3().putObject(objectParams).promise();
        })
        .then(() => { // unzip the contents of the bundle to get its uncompressed data
information
            return streamUrlToTmpFile(bucketUrl, TMP_FILE_NAME);
        })
        .then(() => {
            return unzipAndGetUncompressedSize(TMP_FILE_NAME);
        })
        .then(uncompressedSize => {
            // delete the tmp zip file
            fs.unlink(TMP_FILE_NAME, () => {
                // all the information has been collected
                cb(null, {
                    url: bucketUrl,
                    size_compressed_bytes: compressedSize,
                    size_decompressed_bytes: uncompressedSize
                });
            });
        })
        .catch(err => {
            // delete the tmp zip file
            fs.unlink(TMP_FILE_NAME, () => {
                cb(err);
            });
        });
    });
}

function unzipAndGetUncompressedSize (fileName) {
    let uncompressedSize = 0;

    return new Promise((resolve, reject) => {
        const zip = new StreamZip({
            file: fileName,
            skipEntryNameValidation: true
        });
        zip.on('ready', () => {
            // iterate through every unzipped entry and count up the file sizes
            for (const entry of Object.values(zip.entries())) {
                if (!entry.isDirectory) {
                    uncompressedSize += entry.size;
                }
            }
            // close the file once you're done
            zip.close()
            resolve(uncompressedSize);
        });

        // Handle errors
        zip.on('error', err => { reject(err) });
    });
}
```

```
function streamUrlToTmpFile (url, fileName) {
    return new Promise((resolve, reject) => {
        request(url)
            .pipe(fs.createWriteStream(fileName))
            .on('close', resolve);
    });
}

function readUrlToBuffer (url) {
    return new Promise((resolve, reject) => {
        let zipBuffer = [];

        request(url)
            .on('data', data => {
                zipBuffer.push(data);
            })
            .on('close', function () { // file fully downloaded
                // put the zip contents to a buffer
                resolve(Buffer.concat(zipBuffer));
            });
    })
}
```

User Interface

A majority of the modifications made to the Policy Table are done through SQL database queries. To make this easier, the Policy Server has a user interface that can be found by navigating to http://localhost:3000/ in a browser of your choice. There are four main pages to the Policy Server.

Applications

View Policy Table

Functional Groupings

Consumer Friendly Messages

Custom Vehicle Data

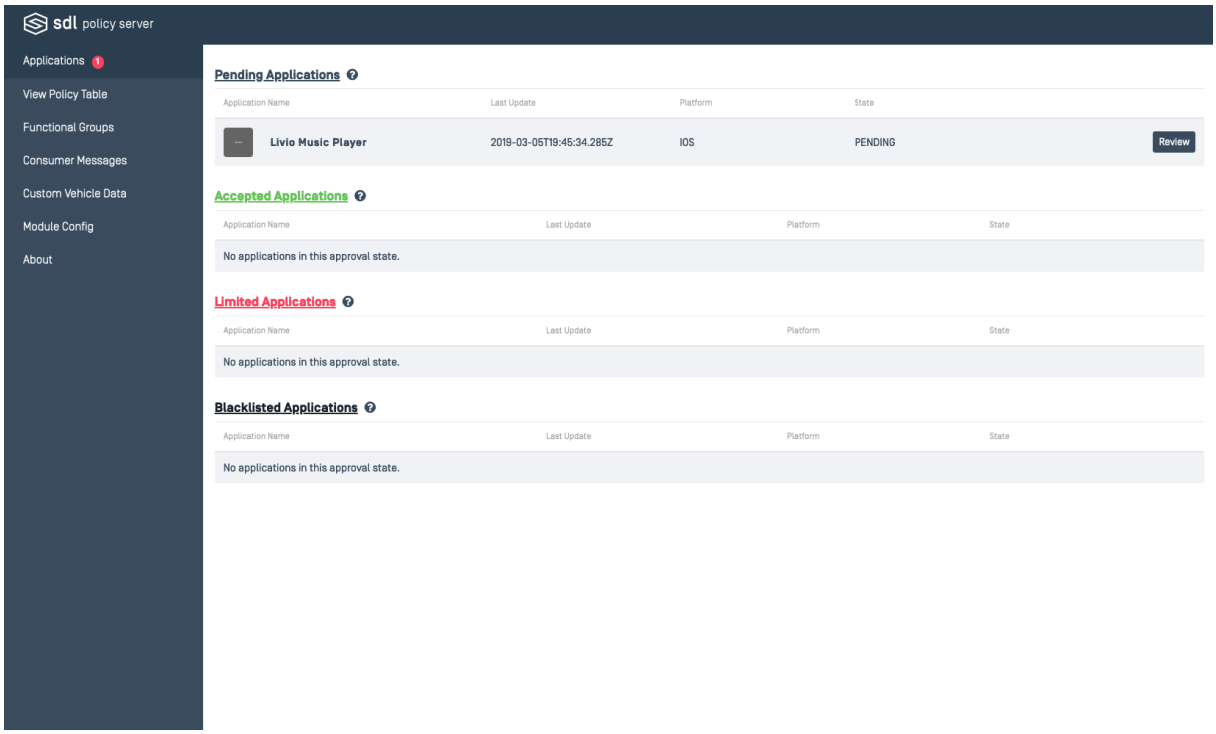
Vue.js

Vue.js is an open source JavaScript framework which the Policy Server uses in building the user interface. It allows the creation of multiple components of a similar structure. For the Policy Server, the larger components for building each page exist in the /src/components directory while the smaller and more numerous items are located in the /common subdirectory. Any files related to styling such as CSS, text fonts, and images, are in the /assets subdirectory. The basic HTML for the user interface can be found in the /ui/raw directory.

Webpack

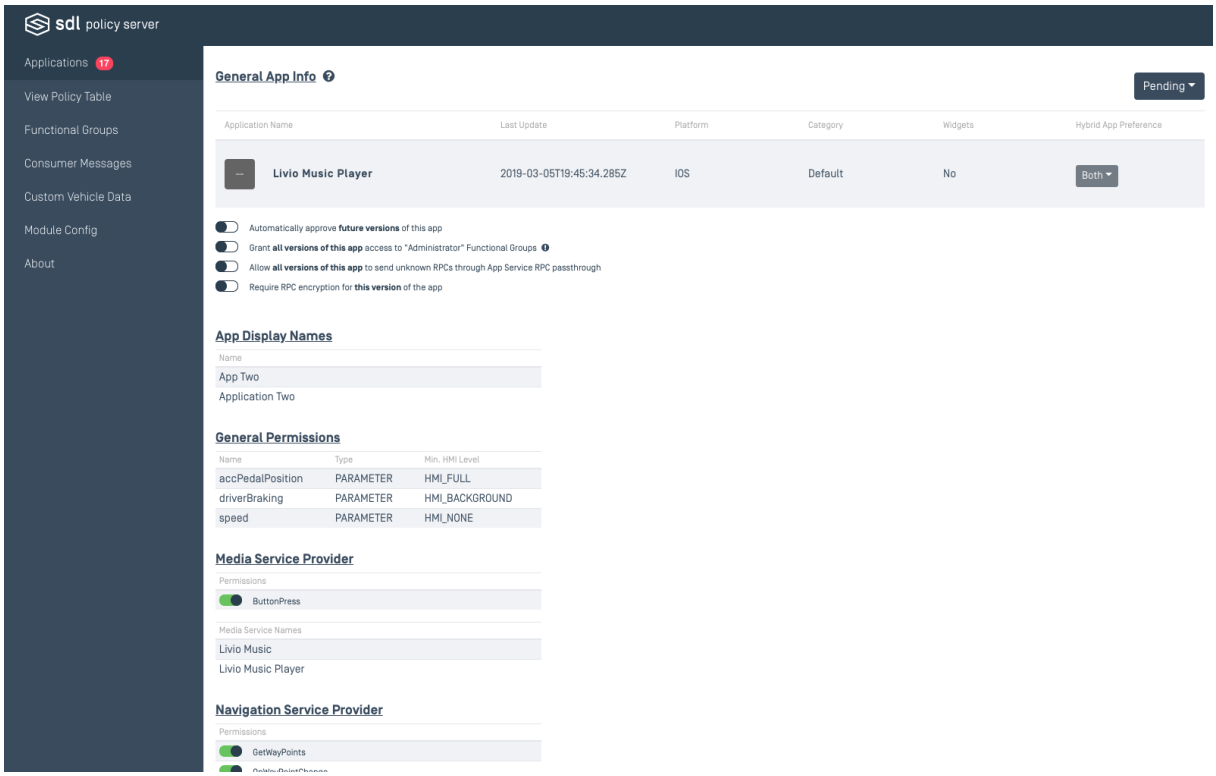
The Policy Server is an open source project giving the user the ability to customize the project to his/her specific needs. Webpack is used to bundle the files into a build and then the build files are executed. If any changes are made to the files before restarting the server, the build command (found in the package.json) must be run in the terminal to rebuild the project with the newly made changes. The /build folder contains all files associated with Webpack.

Applications



This page displays a list of applications pulled from the SHAID server. When initially added, apps will be pending approval. Reviewing each app will give the user a detailed page on the important information associated with the app such as the requested permissions, developer contact information, and preview of what its segment in the Policy Table would look like.

General App Info


sdl policy server
Applications 17
View Policy Table
Functional Groups
Consumer Messages
Custom Vehicle Data
Module Config
About
General App Info
Pending
Application Name
Last Update
Platform
Category
Widgets
Hybrid App Preference
Livio Music Player
2019-03-05T19:45:34.285Z
IOS
Default
No
Both
Automatically approve future versions of this app
Grant all versions of this app access to "Administrator" Functional Groups
Allow all versions of this app to send unknown RPCs through App Service RPC passthrough
Require RPC encryption for this version of the app
App Display Names
Name
App Two
Application Two
General Permissions
Name
Type
Min. HMI Level
accPedalPosition
PARAMETER
HMI_FULL
driverBraking
PARAMETER
HMI_BACKGROUND
speed
PARAMETER
HMI_NONE
Media Service Provider
Permissions
ButtonPress
Media Service Names
Livio Music
Livio Music Player
Navigation Service Provider
Permissions
GetWayPoints

PROPERTY	DEFINITION
Application Name	The String for which to identify the application.
Last Update	The timestamp from when the app information was most recently updated.
Platform	Android/IOS
Category	Specifies the type of application. eg. Media, Information, Social.
Widgets	Whether this app is requesting the use of widgets.
Hybrid App Preference	Which app to show on the HMI when the same app is detected on multiple platforms.
Endpoint	For cloud/embedded apps, the server endpoint of the app.
Transport Type	For cloud/embedded apps, the expected transport type of the server endpoint.

Toggles

TOGGLE	NOTES
Automatically approve future versions of this app	The current version will still need to be approved manually.
Grant all versions of this app access to "Administrator" Functional Groups	
Allow all versions of this app to send unknown RPCs through App Service RPC passthrough	
Require RPC encryption for this version of the app	

App Display Names

PROPERTY	DEFINITION
Name	Alternate strings to identify the application. The app's name must match one of these in order for it to connect to Core.

General Permissions

PROPERTY	DEFINITION
Name	Strings to identify the permission.
Type	RPC
Min. HMI Level	BACKGROUND/FULL/NONE/LIMITED

Service Provider

Service Provider options appear when an application has requested to be an App Service provider. OEMs may choose which RPCs/events the application is allowed to receive via the permission toggle switches. OEMs should note that disabling all the toggle switches does *not* revoke the application's general ability to act as an App Service Provider, but simply limits the app's abilities regarding that particular Service.

PROPERTY	DEFINITION
Permissions	An RPC/event related to the app's requested service.

Grant Proprietary Functional Groups

PROPERTY	DEFINITION
Functional Group Name	A functional group that is categorized as a proprietary functional group.

Developer Contact Info

PROPERTY	DEFINITION
Vendor	The name of the developer to contact regarding this application.
Email	The contact email for the Vendor.
Phone	The contact phone number for the Vendor.
Tech Email	The optional contact email for technical issues regarding the app.
Tech Phone	The optional contact phone number for technical issues.

Certificates

An application can have a private key and certificate associated with it, if certificate generation is enabled. The certificate is set up to auto renew one day before its expiration, but these values can also be manually renewed by clicking "Generate Key and Certificate", followed by clicking "Save Key and Certificate".

Policy Table Preview

This is an example of how the app and its required permissions will appear in the Policy Table.

```
{
  "nicknames": [
    "Livio Music",
    "Livio Music Player"
  ],
  "keep_context": true,
  "steal_focus": true,
  "priority": "NONE",
  "default_hmi": "NONE",
  "groups": [
    "AdministratorGroup",
    "AppServiceConsumerGroup",
    "AppServiceProviderGroup",
    "Base-4",
    "DialNumberOnlyGroup",
    "DrivingCharacteristics-3",
    "HapticGroup",
    "Notifications",
    "OnKeyboardInputOnlyGroup",
    "OnTouchEventOnlyGroup"
  ],
  "moduleType": [],
  "RequestType": [],
  "RequestSubType": [],
  "app_services": {
    "MEDIA": {
      "service_names": [
        "Livio Music",
        "Livio Music Player"
      ],
      "handled_rpcs": [
        {
          "function_id": 41
        }
      ]
    },
    "NAVIGATION": {
      "service_names": [
        "Livio",
        "Livio Music and Nav"
      ],
      "handled_rpcs": [
        {
          "function_id": 45
        },
        {
          "function_id": 32784
        },
        {
          "function_id": 46
        }
      ]
```

```
    }
  },
  "hybrid_app_preference": "MOBILE"
}
```

Significance of Approval States

The top right corner of the application's review page contains a drop down allowing the user to change the approval state of the application. See below for what each state signifies.

PENDING

New applications and updated applications that reach your SDL Policy Server will be granted the approval state of pending. Pending applications are treated like limited applications in that they will not be given any changes requested, but will be given permissions in default functional groups. Pending applications require action performed on them in order for the application to be officially approved or limited.

STAGING

Applications in the staging state will have their permissions granted when using the staging policy table, but not the production policy table. This mode is useful for testing purposes.

ACCEPTED

Applications in the accepted state will have their permissions granted when using both the staging and the production policy table. This state is for applications that are allowed to be used in a production environment.

LIMITED

Limited applications will not receive their requested changes. However, permissions received from the previously accepted version and from default functional groups will still be given. Additional options include providing a reasoning for limiting the application for your future reference. While in the limited state, you also have the option to blacklist the application.

BLACKLISTED

A blacklisted application will not receive any permissions, including permissions from default functional groups. All future update requests will also be blacklisted. This action is reversible.

New Application Versions

Each time an app is updated on the SDL Developer Portal at smartdevicelink.com, the app's changes will appear in your Policy Server pending re-approval. If an app is from a trusted developer and you would like to always approve future revisions of it, you can choose to "Automatically approve updates" under "General App Info" of the app's review page.

Newer versions of applications that come in will have a state of pending, but that will not affect the statuses granted to its previously approved versions. The latest permitted application will have their changes used for the policy table until a new version's changes are also permitted.

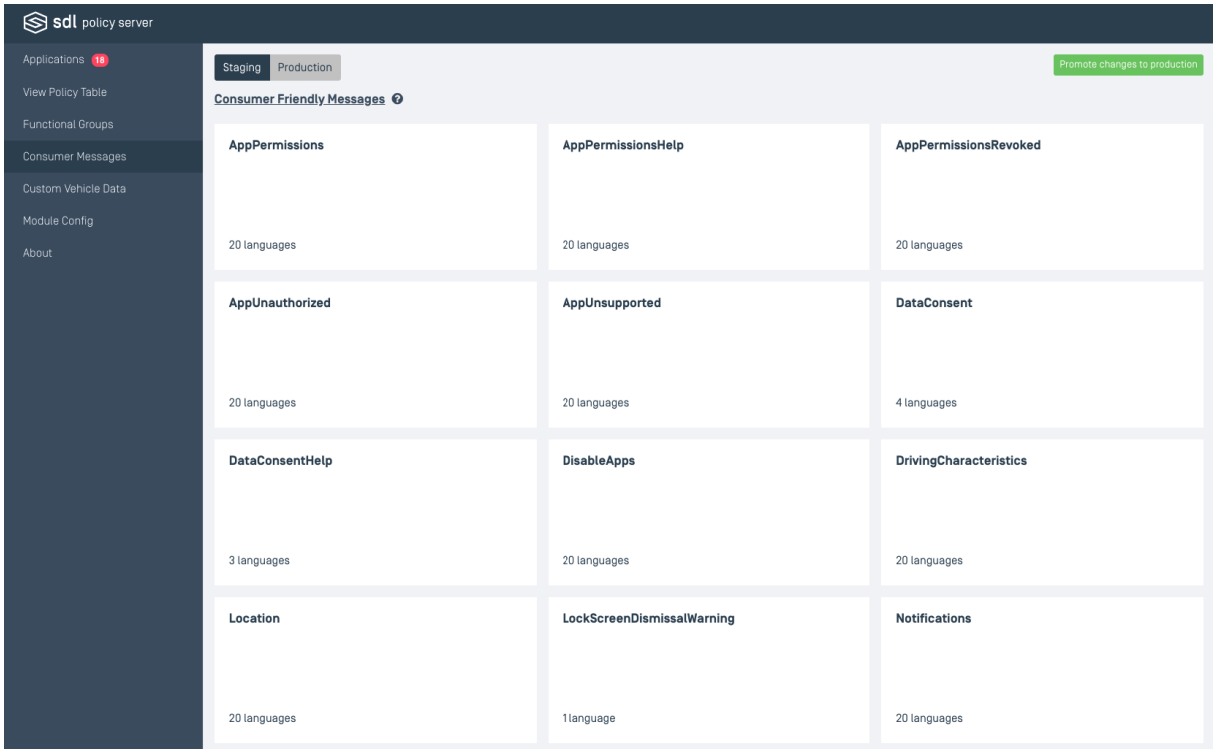
Consumer Messages & Functional Groups

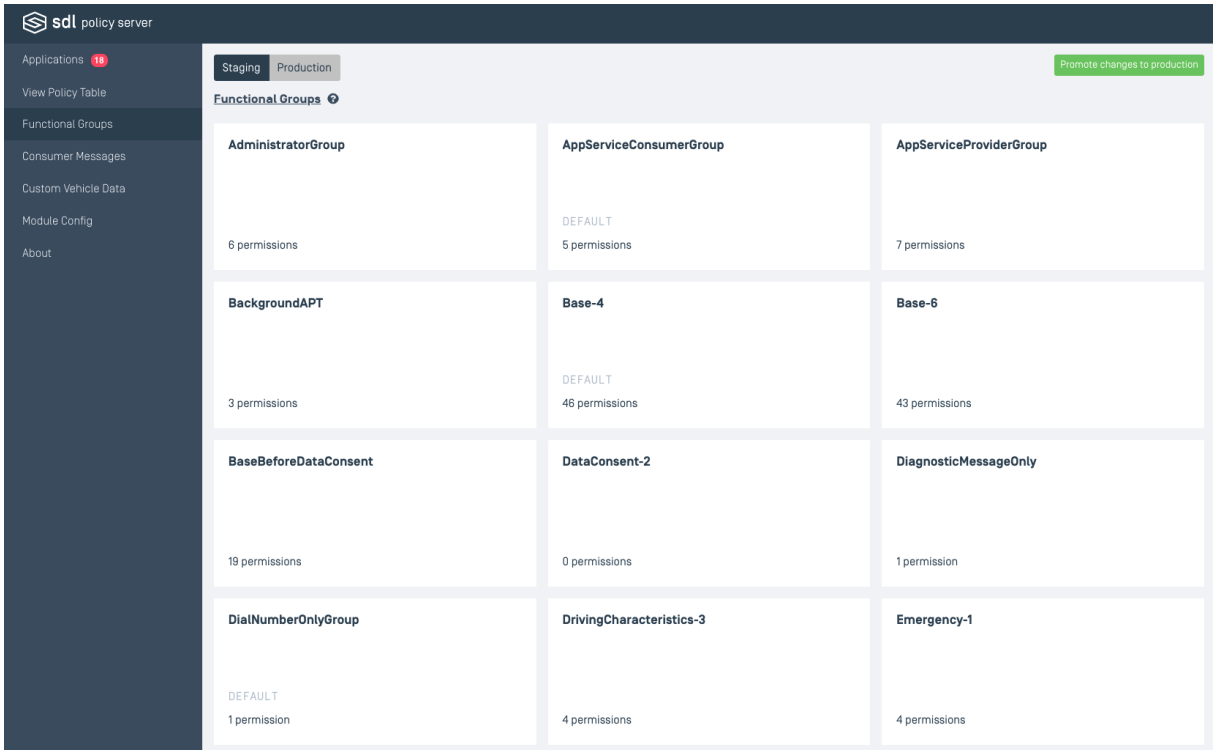
The pages for displaying lists of consumer messages and functional groups are structured in the same way, using similar Vue.js components. For information on the properties of the consumer messages and functional groups, refer back to the earlier documentation regarding the Policy Table.

Cards

Each functional group or consumer message card will have identifying information displayed on a card. This information includes the name, and the number of permissions

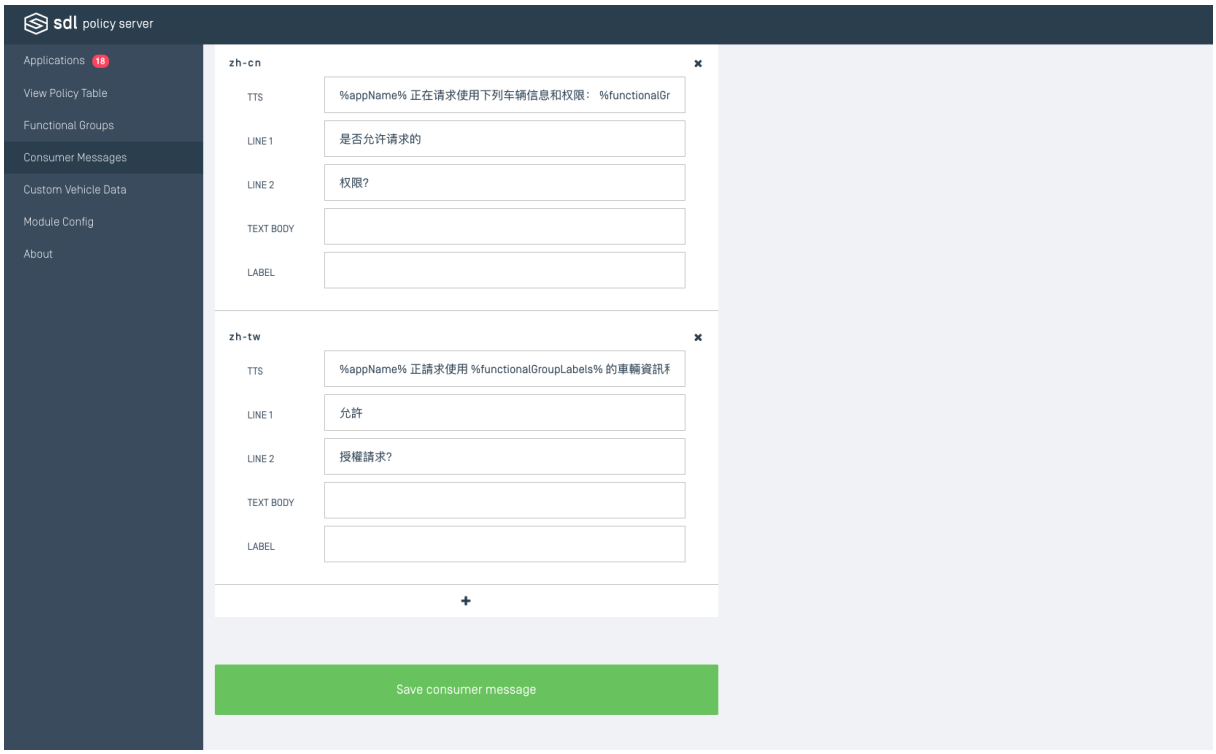
or languages. If the information in the card has been altered since the time of creation then it will have a "MODIFIED" tag. All cards are listed in alphabetical order by name.





Editing

It should be noted that the cards under "Production" cannot be edited. If you wish to edit an existing functional group that has been set to "Production" then you must edit the staging version of that group. Remember to hit the save button at the bottom of the page to keep any changes.



Functional Groups

PROPERTY	DEFINITION
Name	The String for which to identify the functional groups.
Description	A body of text to outline the permissions associated with this functional group.
User Consent Prompt	The consumer friendly message to be displayed when requesting input from the user.

Special Grants

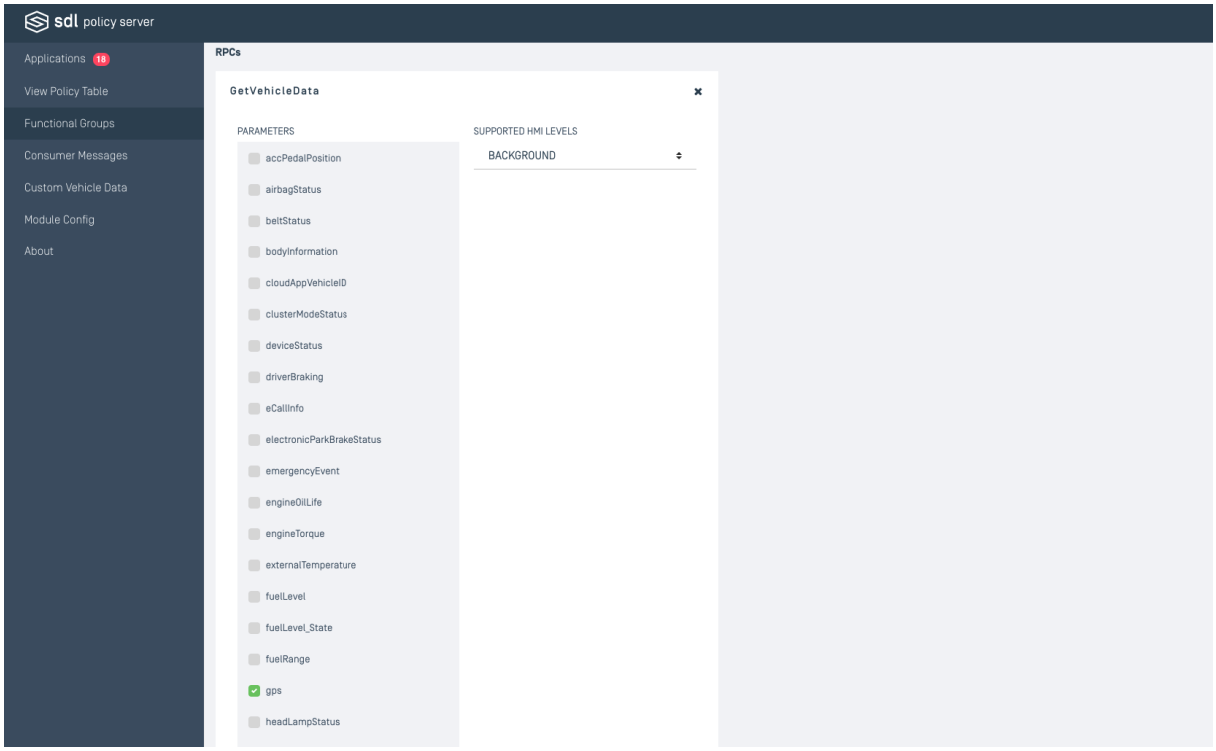
CHECKBOX	NOTES
Grant this functional group to all applications by default	If set to true, all staging and accepted applications will have access to this functional group and its permissions.
Grant this functional group to all applications prior to the user accepting SDL data consent	
Grant this functional group to all applications after the user has accepted SDL data consent	
Grant this functional group to all applications with at least one service provider type	
Grant this functional group to applications with "Administrator" privileges	
Grant this functional group to applications with widget management privileges	
This is a proprietary functional group	

Encryption

CHECKBOX	NOTES
Require RPCs in this functional group to be encrypted	

RPCs

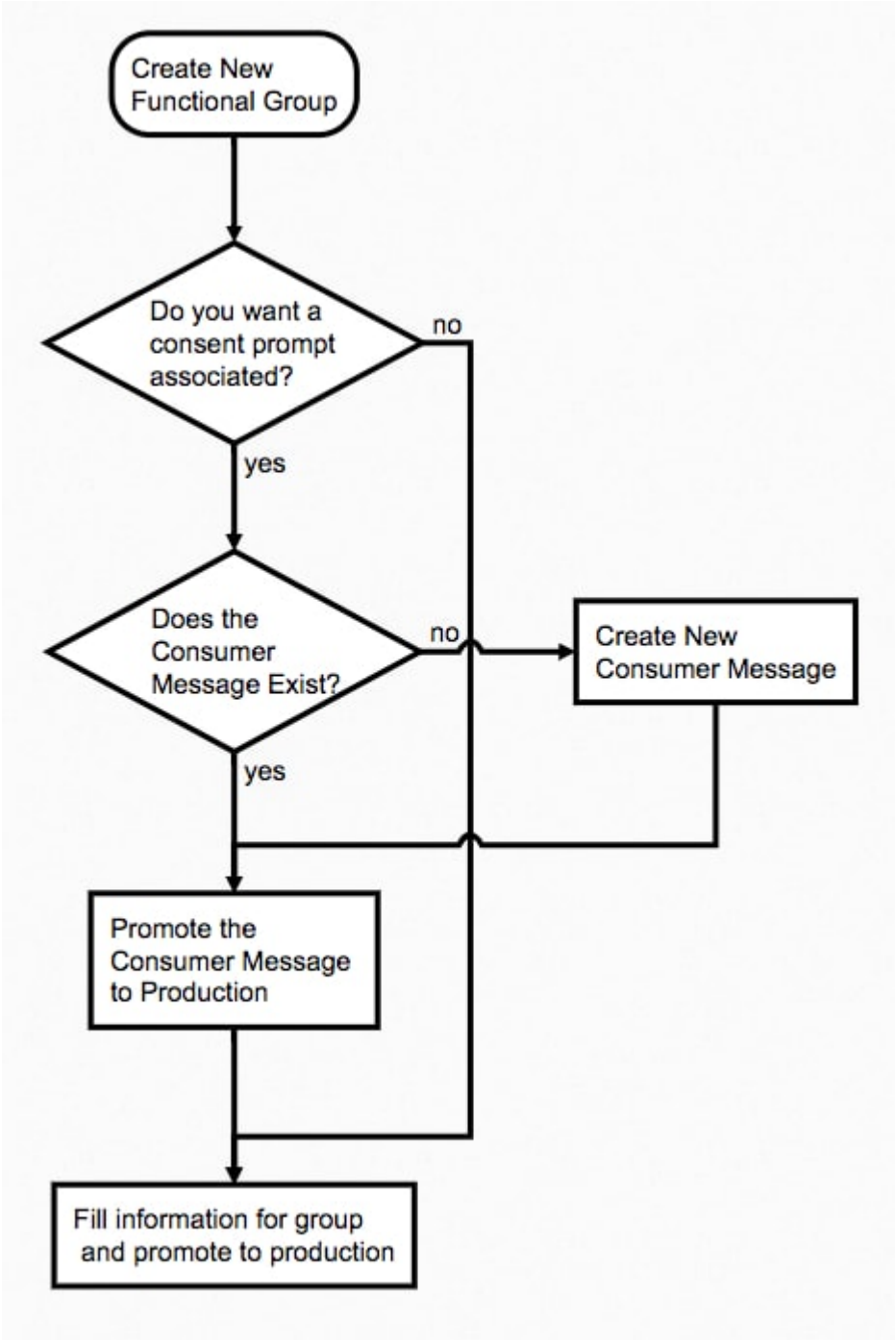
PROPERTY	DEFINITION
Parameters	References possible vehicle information that can retrieved. This is only applicable to vehicle data RPCs. eg. GetVehicleData, SubscribeVehicleData
Supported HMI Levels	SDL Core interface display levels allowed by the app



Creating a New Functional Group

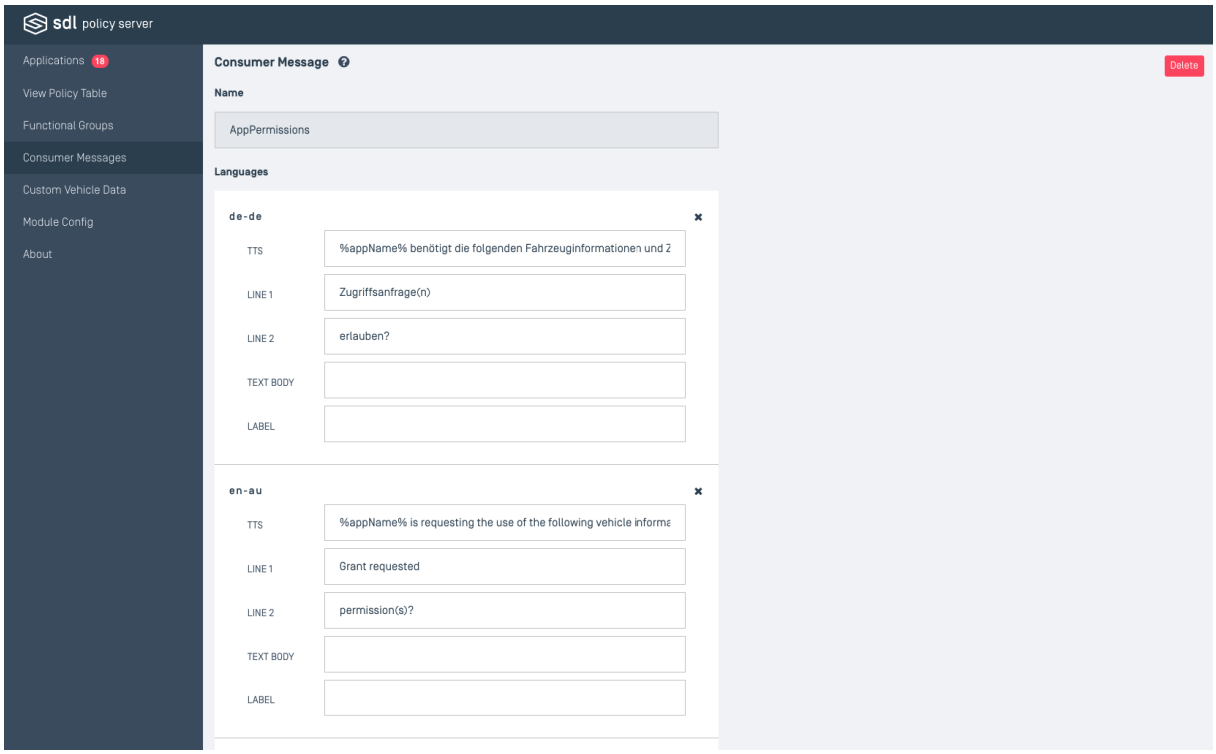
When creating a new functional group, first consider if there should be a user consent prompt associated with the group. If yes, the following diagram will walk through the

correct steps.



Consumer Messages

For information on the language object properties, refer back to the documentation on the consumer messages object.



Staging

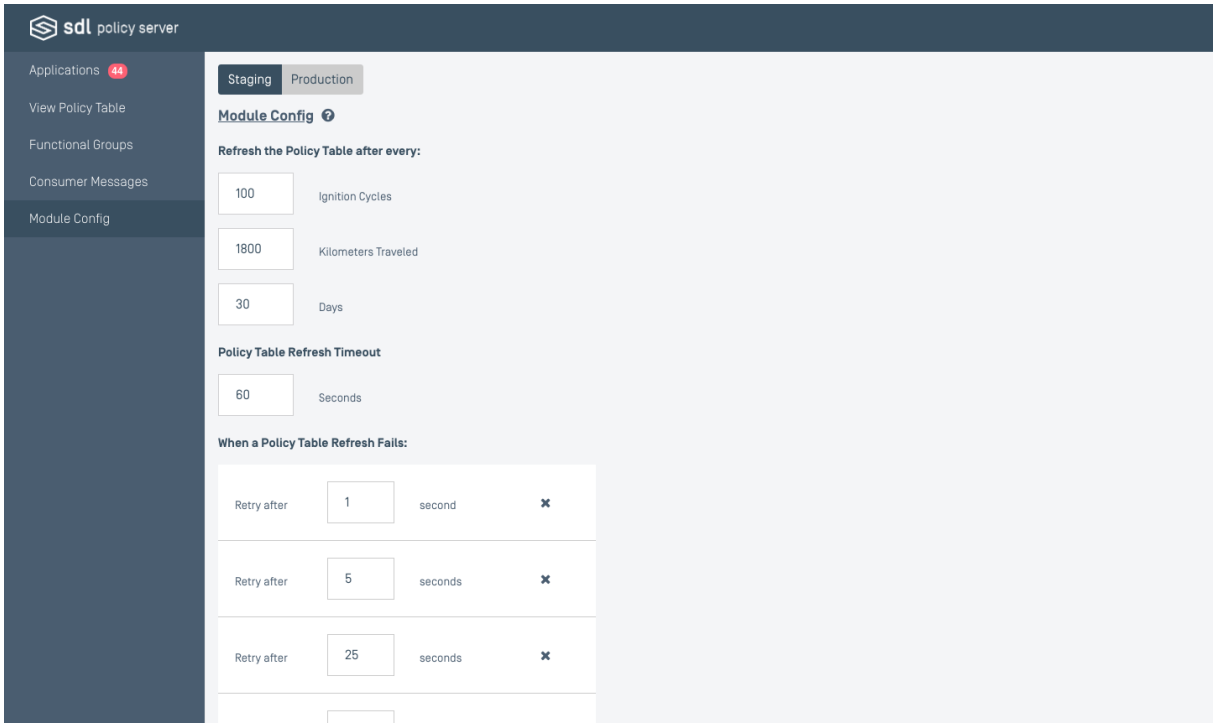
This environment is where temporary or unfinished entries reside. They can be edited and reworked.

Production

Production entries are not directly editable and may only be created/edited/deleted by promoting them from the staging entries. Only promote staging entries to production if you are certain that all information associated is correct.

Module Config

The `module_config` object of the Policy Table is represented here. For information on the properties of the module config, refer back to the earlier documentation regarding the Policy Table.



Editing and Saving

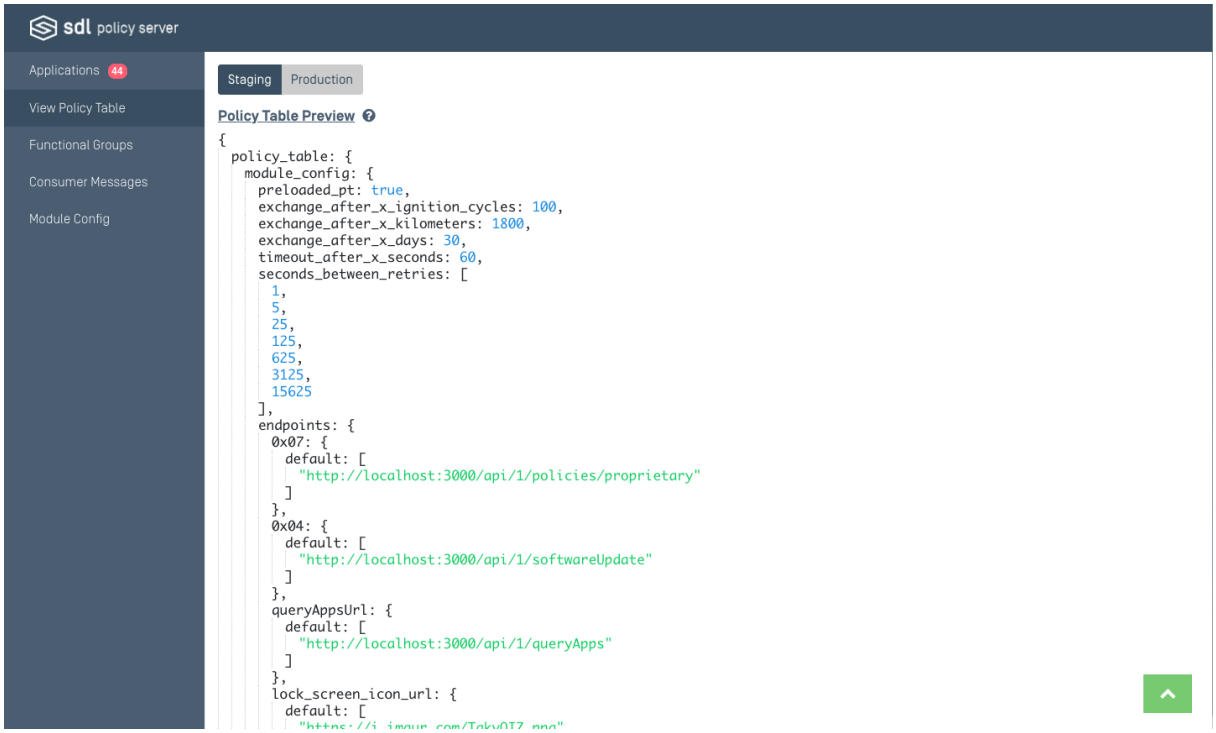
The process of editing and saving is very similar to that of functional groups and consumer messages. It is simpler here because the entire object is either in staging or production. Production versions cannot be edited, but can be overwritten by promoting a staging module config. There is no creating or deleting module configs.

Policy Table

For information on the different properties that make up the Policy Table object, refer back to the Policy Table documentation.

Staging & Production

This page is for viewing an example Policy Table with functional groups and consumer messages available to the server. Staging is where any changes should be made and where any temporary entries should exist. Production is for finalized groups and messages that should no longer be changed. This example table will use the most recent version for the environment chosen. You can minimize certain properties by clicking anywhere there is "[]" or "{}".

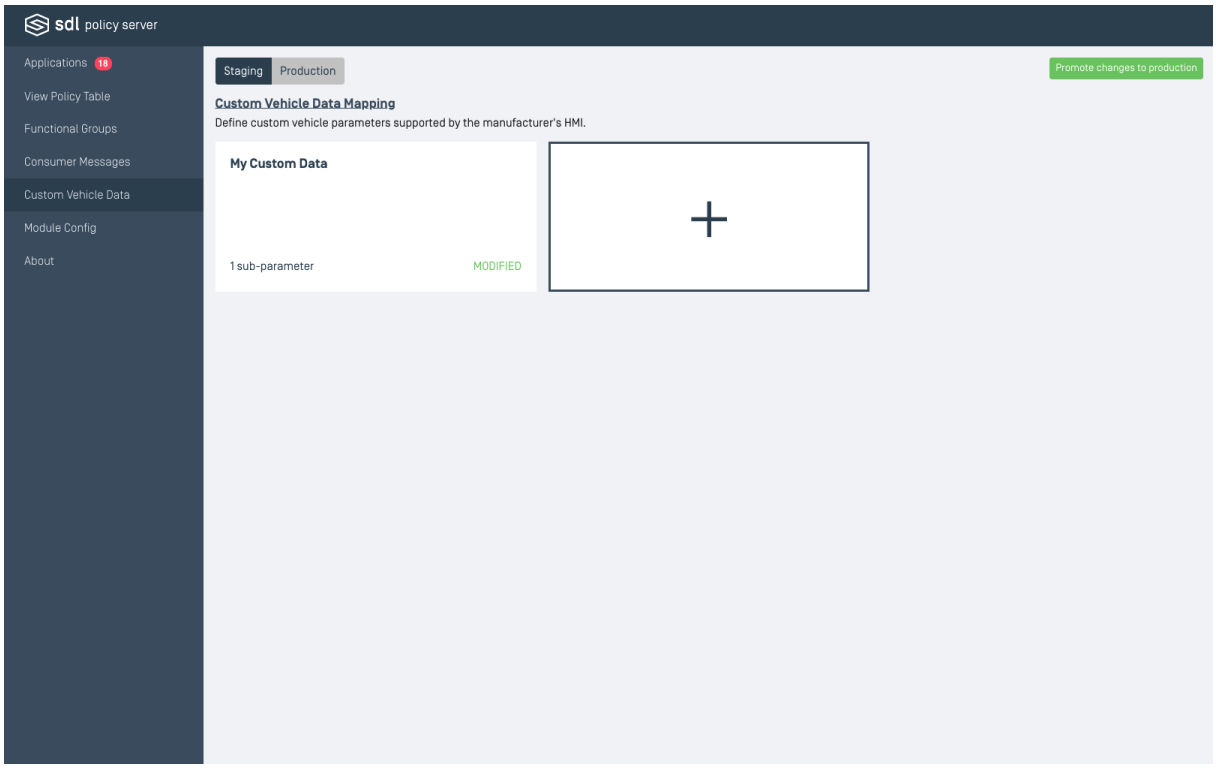


Custom Vehicle Data

This is where OEM-specific custom vehicle data definitions can be defined and managed.

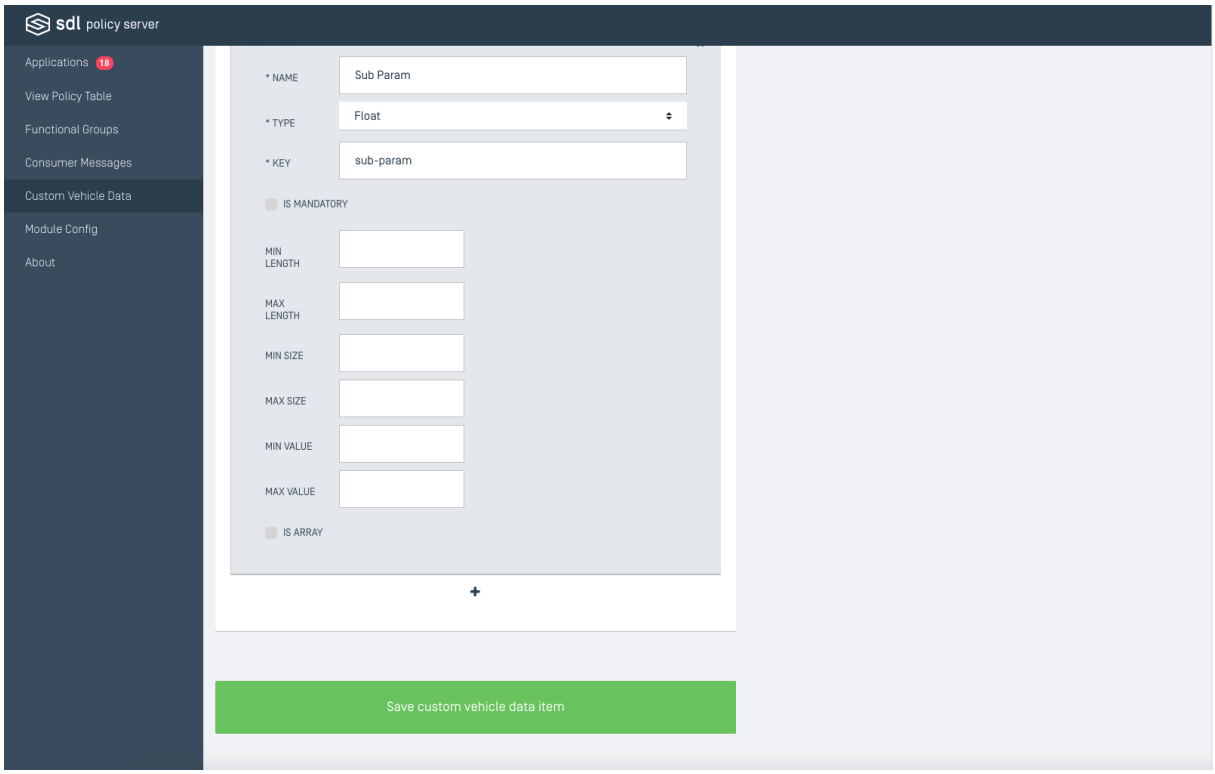
Cards

Each card will have identifying information, which includes the name of the top level vehicle data, and the number of nested parameters it contains. If the information in the card has been altered since the time of creation then it will have a "MODIFIED" tag. All cards are listed in alphabetical order by name.



Editing

It should be noted that the cards under the "Production" view cannot be edited. If you wish to edit existing cards then you need to be in the "Staging" view and then click on the card. When editing, remember to hit the save button at the bottom of the page to keep any changes.



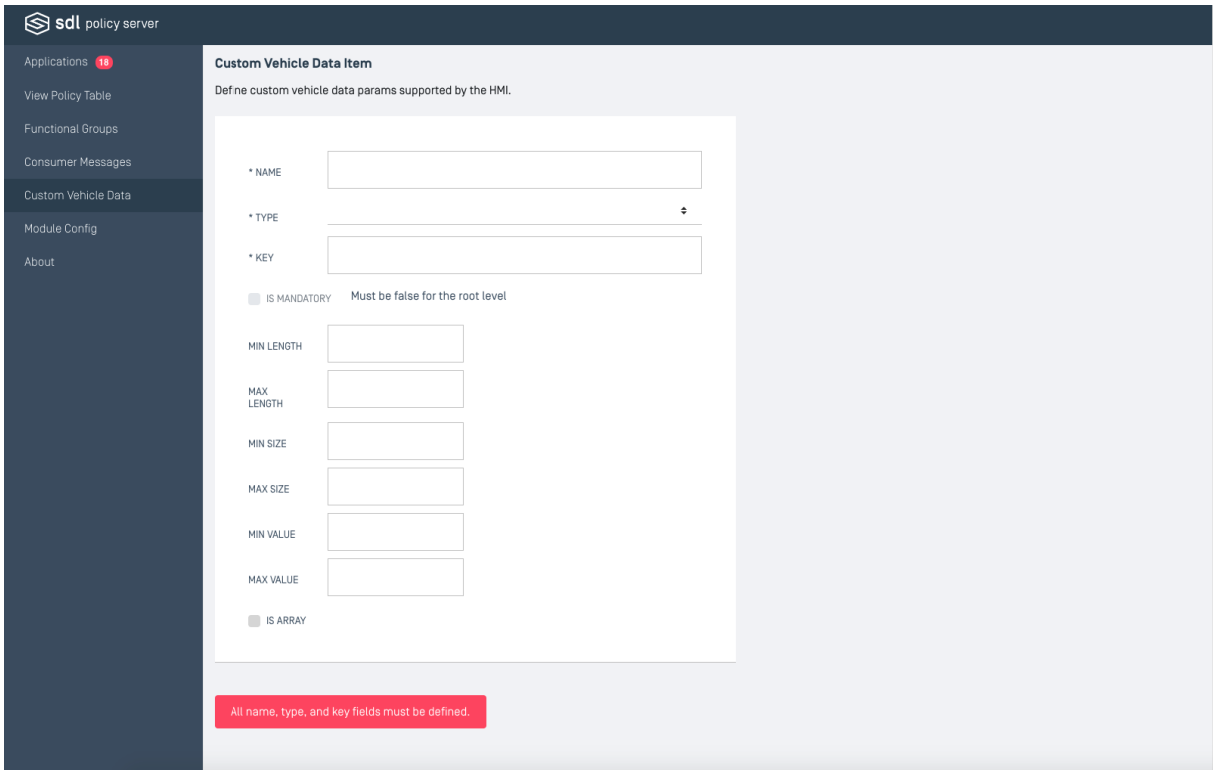
Once a new custom vehicle data item is created, it will be available as an option to assign to vehicle data RPCs in functional groups.

Properties

PROPERTY	DEFINITION
Name	The vehicle data item (ex. gps, speed). This is the parameter SDL Core uses for requests.
Type	The data type of the vehicle data item. It can be a generic type like Integer or String, or an enumeration defined in the API XML file. For a vehicle data item that has sub parameters, this value would be Struct.
Key	A reference to the OEM Network Mapping table which defines the attributes for this vehicle data item.
Is Mandatory	Whether this parameter is required to be included for the vehicle data item.
Min Length	The minimum length of the value if it is a string.
Max Length	The maximum length of the value if it is a string.
Min Size	The minimum number of items for the value if it is an array.
Max Size	The maximum number of items for the value if it is an array.
Min Value	The minimum value for the value if it is a number.
Max Value	The maximum value for the value if it is a number.

PROPERTY	DEFINITION
Is Array	Whether this parameter is an array of the specified type.

Creating a New Vehicle Data Item



Staging

This environment is where temporary or unfinished entries reside. They can be edited and reworked.

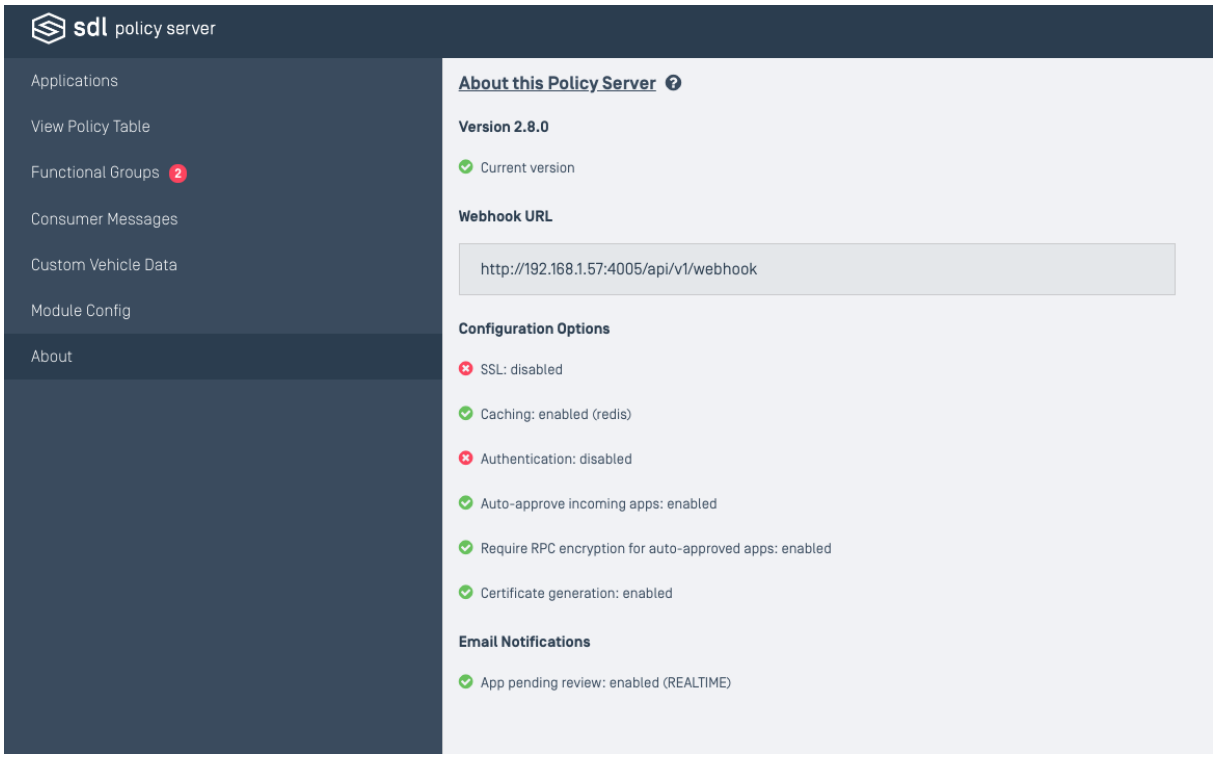
Production

Production entries are not directly editable and may only be created/edited/deleted by promoting them from the staging entries. Only promote staging entries to production if you are certain that all information associated is correct.

About

This section provides basic information about your SDL Policy Server's configuration settings, including:

- Currently installed version (and if a new version is available)
- Webhook URL (to be entered on smartdevicelink.com)
- SSL port (if enabled)
- Caching service (if enabled)
- Authentication type (if enabled)
- Auto-approve incoming apps (if enabled)
- Require RPC Encryption for auto-approved apps (if enabled)
- Certificate generation (if enabled)
- Email notifications (if enabled)



Policy Tables Overview

Policies are rules enforced by SDL core that configure how the system can and/or will behave. For example, a policy could prohibit the use of an application (e.g. Flappy Bird) in a specific type of vehicle. In general, policies are configured by an OEM (e.g. Ford, Toyota, Suzuki) and stored in their SDL Policy Server. Once configured, all policies for a specific vehicle can be requested in the form a JSON document called a Policy Table. Policy Tables are downloaded to a vehicle's head unit where it can be enforced by SDL Core.

Example Policy Table

An example Policy Table is available in the SDL Core repository.

Application Policies

An application's permissions and settings are stored in the **app_policies** property in a Policy Table. The application policies are used to grant applications access to a specific set of features, such as vehicle data and/or running in the background. Any other application related data, such as user-consents, can also be stored in application policies as well.

Application ID

Settings for a specific application are stored in the **app_policies** object as a property named after the application's unique ID (e.g. "663645645" or any string of at most 100 characters). The value of this property can be either an object containing properties listed below or a reference to another sibling property (e.g. "default" or "device"). In addition, a special value of "null" can be used to indicate that the application has been revoked.

APPLICATION PROPERTY	TYPE	DESCRIPTION
keep_context	Boolean	When true, allows the application to display messages even if another app enters the foreground (HMI level FULL).
steal_focus	Boolean	When true, allows the application to steal the foreground from another application at will.
priority	String	Priority level assigned to the application.
default_hmi	String	HMI level given to the application following a successful registration with SDL Core.
groups	Array of Strings	A list of functional groupings the application has access to.
preconsented_groups	Array of Strings	List of functional groupings that do not require a user consent because the consent has already been given in another place. (e.g. an application EULA)

APPLICATION PROPERTY	TYPE	DESCRIPTION
RequestType	Array of Strings	List of Request Types that an app is allowed to use in a SystemRequest RPC. If omitted, all requestTypes are disallowed. If an empty array is provided, all requestTypes are allowed.
RequestSubType	Array of Strings	List of Request SubTypes (defined by individual OEMs) that an app is allowed to use in a SystemRequest RPC. If omitted, all requestSubTypes are disallowed. If an empty array is provided, all requestSubTypes are allowed.
AppHMIType	Array of Strings	List of HMI Types used to group the application into different containers in an HMI system. If omitted, all appHMITypes are allowed.
heart_beat_timeout_ms	String	A streaming/projection app will be automatically disconnected if no app communication occurs over this period of time (in milliseconds).
certificate	String	The app's encryption certificate for video streaming/projection (if applicable)

APPLICATION PROPERTY	TYPE	DESCRIPTION
nicknames	Array of Strings	A list of names the application goes by. Some OEMs may require the app's name to match a value in this array in order to run.

Application HMI Types

An application can be categorized by an HMI type allowing the SDL-enabled head unit to understand how to appropriately handle the application. There are several HMI types listed below.

APPLICATION HMI TYPE	DESCRIPTION
BACKGROUND_PROCESS	
COMMUNICATION	
DEFAULT	
INFORMATION	
MEDIA	
MESSAGING	
NAVIGATION	
SOCIAL	
SYSTEM	
TESTING	
WEB_VIEW	

Application HMI Levels

An HMI Level describes the state of an application. Resources are granted to an application based on its current state. While some resources are granted automatically to an application in a specific HMI Level, many can be controlled by the Policy Table.

LEVEL	VALUE	SHORT DESCRIPTION
Full	0	An application is typically in `Full` when it is displayed in the HMI. In `Full` an application has access to the HMI supported resources, e.g. UI, VR, TTS, audio system, and etc.
Limited	1	An application is typically placed in `Limited` when a message or menu is displayed `Limited` to restrict its permissions.
Background	2	An application is typically in `Background` when it is not being displayed by the HMI. When in `Background` an application can send RPCs according to the Policy Table rules.
None	3	When placed in `None` an application has no access to HMI supported resources.

Request Types

REQUEST TYPE	DESCRIPTION
HTTP	
FILE_RESUME	
AUTH_REQUEST	
AUTH_CHALLENGE	
AUTH_ACK	
PROPRIETARY	
QUERY_APPS	
LAUNCH_APP	
LOCK_SCREEN_ICON_URL	
TRAFFIC_MESSAGE_CHANNEL	
DRIVER_PROFILE	
VOICE_SEARCH	
NAVIGATION	
PHONE	
CLIMATE	
SETTINGS	

REQUEST TYPE	DESCRIPTION
VEHICLE_DIAGNOSTICS	
EMERGENCY	
MEDIA	
FOTA	
OEM_SPECIFIC	Used for OEM defined requests, requestSubType should be used to determine how to handle this type of request.

Default

A default application configuration can be stored in the **app_policies** object as a property named **default**. This property's value is an object containing any valid application property excluding **certificate** and **nicknames**.

Device

Permissions granted to the user's device post-DataConsent.

Example

An example of how the Application Policy portion of a Policy Table might look.

```
"app_policies": {
    "default": {
        "keep_context": true,
        "steal_focus": true,
        "priority": "NONE",
        "default_hmi": "NONE",
        "groups": [ "Base-1" ],
        "preconsented_groups": [],
        "RequestType": [],
        "memory_kb": 5,
        "watchdog_timer_ms": 55
    },
    "device": {
        "keep_context": true,
        "steal_focus": true,
        "priority": "NONE",
        "default_hmi": "NONE",
        "groups": [ "Base-2" ],
        "preconsented_groups": []
    },
    "pre_DataConsent": {
        "keep_context": true,
        "steal_focus": true,
        "priority": "NONE",
        "default_hmi": "NONE",
        "groups": [ "BaseBeforeDataConsent" ],
        "preconsented_groups": [],
        "memory_kb": 5,
        "watchdog_timer_ms": 55
    },
    "[App ID 1]": "null",
    "[App ID 2]": "default",
    "[App ID 3]": {
        "nicknames": [ "Awesome Music App" ],
        "keep_context": true,
        "steal_focus": true,
        "priority": "NONE",
        "default_hmi": "NONE",
        "groups": [ "Base-1", "VehicleInfo-1" ],
        "preconsented_groups": [],
        "RequestType": [],
        "RequestSubType": [ "Sub Type" ],
        "AppHMIType": [ "MEDIA" ],
        "memory_kb": 5,
        "watchdog_timer_ms": 55,
        "certificate": "[Your Certificate]"
    }
}
```

Consumer Friendly Messages

There are certain scenarios when SDL Core needs to display a message to the user. Some examples are when an error occurs or an application is unauthorized. These messages can include spoken text and text displayed to a user in multiple languages. All of this information is stored in the **consumer_friendly_messages** property.

Messages

All messages are given a unique name (e.g. "AppUnauthorized" or "DataConsent") and stored as an object in the **consumer_friendly_messages** object's **messages** property.

Language

Since each message should support multiple languages, each message object will contain a property named **languages**. Language properties are named by combining the ISO 639-1 language code and the ISO 3166 alpha-2 country code. For example, messages for **English** speaking citizens of the **United States** would be under the key **en-us**.

Message Text

Inside each language object is the data to be displayed or spoken by the module. The data is organized in the following properties.

MESSAGE TEXT PROPERTY	TYPE	DESCRIPTION
tts	String	Text that can be read aloud by the vehicle module.
line1	String	First line of text to be displayed on the head unit.
line2	String	Second line of text to be displayed on the head unit.
text-body	String	Body of text to be displayed on the head unit.
label	String	

Version

The version property in the **consumer_friendly_messages** object defines the current version of all the messages. It is used during a Policy Table update to determine whether or not the consumer friendly messages need to be updated. The version must be in the format `###.###.###`.

Example

An example of how the Consumer Friendly Messages portion of a Policy Table might look.

```
"consumer_friendly_messages": {
    "version": "001.001.015",
    "messages": {
        "AppUnauthorized": {
            "languages": {
                "de-de": {
                    "tts": "Diese Version von %appName% ist nicht autorisiert und wird nicht
mit SDL funktionieren.",
                    "line1": "nicht autorisiert"
                },
                "en-ie": {
                    "tts": "This version of %appName% is not authorized and will not work
with SDL.",
                    "line1": "not authorized"
                },
                "en-us": {
                    "tts": "This version of %appName% is not authorized and will not work
with SDL.",
                    "line1": "Not Authorized"
                }
            }
        },
        "DataConsent": {
            "languages": {
                "en-us": {
                    "tts": "To use mobile apps with SDL, SDL may use your mobile device's
data plan....",
                    "line1": "Enable Mobile Apps",
                    "line2": "on SDL? (Uses Data)"
                }
            }
        }
    }
}
```

Device Data

Information about each device that connects to SDL Core is recorded in the Policy Table. This information is used to persist configurations for the head unit based on the device connected.

Device Specific Information

Devices are identified in the Policy Table using a unique identifier. Device unique identifier(s) are either a bluetooth mac address or USB serial address irreversibly encrypted/hashed using SHA-256. Information about a specific device is stored using its unique identifier as a key. The following properties describe the information stored.

PROPERTY	TYPE	DESCRIPTION
hardware	String	Type and/or name of the hardware. (e.g. iPhone 7)
max_number_rfcom_ports	Number	Number of RFCOM ports supported by the device.
firmware_rev	String	Device's firmware version
os	String	Operating system. (e.g. iOS or Android)
os_version	String	Device's operating system version.
carrier	String	The mobile phone's carrier. (e.g. Verizon or AT&T)

User Consents

Whether or not an SDL user has given permission for a feature can be stored for each device and application connected to a vehicle's head unit. For example, a user may consent to allowing SDL to use their phone's cellular data to download Policy Table updates. These consent records are stored in the **user_consent_records** property.

Device

User consent(s) for a device are stored in a property named **device** in the **user_consent_records** object. The value of this property is an object with the following properties:

USER CONSENT RECORD PROPERTY	TYPE	DESCRIPTION
consent_groups	Object	A listing of SDL features that are accepted or declined.
input	String	Accepted values are "GUI" or "VUI"
time_stamp	String	A timestamp in ISO 8601 format.

Application

User consent(s) can also be saved per application on a device under a property named after its Application ID. The value of this property is an object with the same user consent record properties as device above.

Example

An example of how the Device Data portion of a Policy Table might look.

```
"device_data": {
    "[ID VALUE HERE]": {
        "hardware": "iPhone 4S",
        "max_number_rfcom_ports": 25,
        "firmware_rev": null,
        "os": "iOS",
        "os_version": "5",
        "carrier": "AT&T",
        "user_consent_records": {
            "device": {
                "consent_groups": {
                    "DataConsent-1": true
                },
                "input": "VUI",
                "time_stamp": "4/24/2012 12:30:00 PM"
            },
            "[APP ID HERE]": {
                "consent_groups": {
                    "Location-1": true,
                    "DrivingData-1": false
                },
                "input": "VUI",
                "time_stamp": "3/26/2012 10:41:00 AM "
            }
        }
    }
}
```

Functional Groupings

Before an application can use each feature offered by SDL it must first be granted permission to do so in the Policy Table. Each feature may require several RPCs with specific HMI level permission, as well as allowed parameters and other information. In order to avoid duplicating this data for each application, SDL instead uses functional groupings. A functional grouping is simply a group of RPC messages and parameters with specific HMI permissions and allowed parameters. So for example, if an application named Torque wanted access to vehicle data you would simply add the **VehicleData** functional group to Torque's allowed policies.

Functional Group

Each functional group is given a unique name (e.g. BasicVehicleData) that is used to reference that group from anywhere within the Policy Table. Each functional group may contain the following properties.

FUNCTIONAL GROUP PROPERTY	TYPE	DESCRIPTION
rpcs	Object	A list of Remote Procedure Calls and their configurations for the current functional grouping.
user_consent_prompt	String	References a consumer friendly message prompt that is required to use the RPC. If this field is not present, then a consumer friendly message prompt is **not** required.

RPCS

Each RPC in the **rpcs** property has a unique name that represents an existing RPC (e.g. AddSubMenu). In each RPC object there may be the following properties.

PROPERTY	TYPE	DESCRIPTION
hmi_levels	Array	An ordered list of HMI levels that an application is allowed to use the RPC command in.
parameters	Array	A list of allowed parameters that the application can use with the RPC command.

Example

An example of how the Functional Groupings portion of a Policy Table might look.

```
"functional_groupings": {
    "Base-1": {
        "rpcs": {
            "AddCommand": {
                "hmi_levels": [
                    "BACKGROUND",
                    "FULL",
                    "LIMITED"
                ]
            },
            "AddSubMenu": {
                "hmi_levels": [
                    "BACKGROUND",
                    "FULL",
                    "LIMITED"
                ]
            },
            "Alert": {
                "hmi_levels": [
                    "FULL",
                    "LIMITED"
                ]
            },
        }
    },
    "VehicleInfo-1": {
        "user_consent_prompt": "VehicleInfo",
        "rpcs": {
            "GetVehicleData": {
                "hmi_levels": [
                    "BACKGROUND",
                    "FULL",
                    "LIMITED"
                ],
                "parameters": [
                    "engineTorque",
                    "externalTemperature",
                    "fuelLevel",
                    "fuelLevel_State",
                    "headLampStatus",
                    "instantFuelConsumption",
                    "odometer",
                    "tirePressure",
                    "vin",
                    "wiperStatus"
                ]
            },
        }
    }
}
```

Module Config

The module configuration property contains information used to configure SDL Core for use on the current vehicle.

Notifications

There is a limit for the number of notifications that can be displayed per priority level. The limit is instead based on notifications per minute. You can configure these in the **notifications_per_minute_by_priority** property which has a max array size of 5. The following are the available priority levels.

PROPERTY	TYPE	DESCRIPTION
EMERGENCY	Number	Number of emergency notifications that can be displayed per minute.
COMMUNICATION	Number	Number of communication notifications that can be displayed per minute.
NAVIGATION	Number	Number of navigation notifications that can be displayed per minute.
NONE	Number	Number of notifications without a priority that can be displayed per minute.
NORMAL	Number	Number of notifications with a normal priority that can be displayed per minute.
voiceCommunication	Number	Number of voice communication notifications that can be displayed per minute.

Policy Table Update Configurations

Periodically changes will be made to a Policy Table, either by the Policy Server or SDL Core. This means SDL Core should check for and perform a Policy Table update, which synchronizes the local and Policy Server Policy Tables. You can configure when SDL Core will check using the following configurations.

PROPERTY	TYPE	DESCRIPTION
exchange_after_x_ignition_cycles	Number	Update Policy Table after a number of ignitions.
exchange_after_x_kilometers	Number	Update Policy Table after a number of kilometers traveled.
exchange_after_x_days	Number	Update Policy Table after a number of days.

Preloaded Policy Tables

SDL Core can use a predefined Policy Table located locally on the vehicle's head unit. This is present to initially configure SDL Core as well as to enable the storage of vehicle data before a Policy Table update has occurred.

PROPERTY	TYPE	DESCRIPTION
preloaded_pt	Boolean	When true, SDL Core will use the local copy of the Policy Table.

Policy Table Structure Configurations

The policy table's structure is determined by the following configurations.

PROPERTY	TYPE	DESCRIPTION
full_app_id_supported	Boolean	When true, an app's `fullAppID` will be used in the `app_policies` section as it's key. If false or omitted, the short-form `appID` will be used.

Server Requests

All requests made directly by SDL Core or by proxy can be configured using the following attributes.

PROPERTY	TYPE	DESCRIPTION
timeout_after_x_seconds	Number	Elapsed seconds until a Policy Table update request will timeout.
endpoints	Object	Contains a list of endpoints (see below) that may contain a default or app-specific array of server endpoints.
seconds_between_retries	Array	A list of seconds to wait before each retry.

Endpoints

This section is a list of URLs that are used throughout the SDL lifecycle, such as Policy Table updates, module software updates, and lock screen imagery.

PROPERTY	TYPE	DESCRIPTION
0X07	Array	A list of URLs that can be used for Policy Table updates.
0X04	Array	A list of URLs that can be used to retrieve module software updates.
queryAppsUrl	Array	A list of URLs that can be used to receive valid apps for querying on iOS devices.
lock_screen_icon_url	Array	A list of URLs to image files which can be displayed by the application on the driver's device during lockout.
custom_vehicle_data_mapping_url	Array	A list of URLs that can be used for the OEM Network Mapping table.

Endpoint Properties

This section stores additional properties related to endpoints.

PROPERTY	TYPE	DESCRIPTION
custom_vehicle_data_mapping_url.version	String	The current OEM Network Mapping table version.

Vehicle Information

Vehicle identification information is stored in the module configuration portion of the Policy Table.

PROPERTY	TYPE	DESCRIPTION
vehicle_make	String	Manufacturer of the vehicle.
vehicle_model	String	Model of a vehicle.
vehicle_year	String	Year the vehicle was made.

Example

An example of how the Module Config portion of a Policy Table might look.

```
"module_config": {
    "lock_screen_dismissal_enabled": true,
    "endpoints": {
        "0x07": {
            "default": [ "http://localhost:3000/api/1/policies/proprietary" ]
        },
        "lock_screen_icon_url": {
            "default":[  "https://i.imgur.com/TgkvOIZ.png" ]
        },
        "custom_vehicle_data_mapping_url":{
          "default":[ "http://localhost:3000/api/1/vehicleDataMap" ]
        }
      }
    },
    "endpoint_properties": {
        "custom_vehicle_data_mapping_url": {
            "version":"0.1.2"
        }
    },
    "exchange_after_x_ignition_cycles": 100,
    "exchange_after_x_kilometers": 1800,
    "exchange_after_x_days": 30,
    "full_app_id_supported": true,
    "notifications_per_minute_by_priority": {
        "EMERGENCY": 60,
        "NAVIGATION": 15,
        "voiceCommunication": 10,
        "COMMUNICATION": 6,
        "NORMAL": 4,
        "NONE": 0
    },
    "seconds_between_retries": [ 1, 5, 25, 125, 625 ],
    "timeout_after_x_seconds": 60,
    "vehicle_make": "Ford",
    "vehicle_model": "F-150",
    "vehicle_year": "2015"
}
```


Module Meta

Language and Country

The current language and regional settings can be configured using the following properties.

PROPERTY	TYPE	DESCRIPTION
language	String	Current system language. ISO 639-1 combined with ISO 3166 alpha-2 country code.

Module Version

The current version of the vehicle's module should be stored in the following property.

PROPERTY	TYPE	DESCRIPTION
ccpu_version	String	Software version for the module running SDL Core.

Policy Table Update

Information about when a Policy Table update has last taken place is stored in the following properties.

PROPERTY	TYPE	DESCRIPTION
pt_exchanged_at_odometer_x	Number	Marks the odometer reading in kilometers at the time of the last successful Policy Table update.
pt_exchanged_x_days_after_epoch	Number	Marks the time of the last successful Policy Table update.
ignition_cycles_since_last_exchange	Number	Number of ignition cycles since the last Policy Table update.

Vehicle Data

Additional vehicle information is stored in the module meta property.

PROPERTY	TYPE	DESCRIPTION
vin	String	The vehicle's unique identification number.

Example

An example of how the Module Meta portion of a Policy Table might look.

```
"module_meta": {
    "ccpu_version": "4.1.2.B_EB355B",
    "language": "en-us",
    "pt_exchanged_at_odometer_x": 1903,
    "pt_exchanged_x_days_after_epoch": 46684,
    "ignition_cycles_since_last_exchange": 50,
    "vin": "1FAPP4442VH100001"
}
```


Usage and Errors

Errors and usage statistics that occur while an application is in use or are related to an application are record. The information does not contain user information and is very small as to use as little mobile data as possible. This data is sent to the Policy Server when performing a Policy Table update.

Application Errors

Errors and usage statistic that occur while an application is in use or are related to an application are recorded. The following properties are tracked in a property named after the application's ID.

PROPERTY	TYPE	DESCRIPTION
app_registration_language_gui	String	Language used to register the application using GUI.
app_registration_language_vui	String	Language used to register the application using VUI.
count_of_rejected_rpcs_calls	Number	Count of RPC calls that were rejected because access was not allowed due to a policy.
count_of_rejections_duplicate_name	Number	Number of times an application registration uses a name which is already registered in the current ignition cycle.
count_of_rejections_nickname_mismatch	Number	Number of times an app is not allowed to register because its registration does not match one of the app-specific policy nicknames.
count_of_removals_for_bad_behavior	Number	The module has criteria for identifying unacceptably bad application behavior. This tracks the number of times that distinction leads the module to unregister an application.
count_of_rfcom_limit_reached	Number	Number of times the maximum number of rfcom channels are used on a device by the application.

PROPERTY	TYPE	DESCRIPTION
count_of_rpcs_sent_in_hmi_none	Number	Number of times an application tried to use an RPC (not unregisterAppInterface) in the HMI_NONE state. Counts the number of conflicts with the built-in/hardcoded restriction for HMI_STATE=NONE.
count_of_run_attempts_while_revoked	Number	Incremented when the user selects a revoked application from the HMI menu.
count_of_user_selections	Number	Number of times a user selected to run the app. Increment one when app starts via Mobile Apps Menu or VR. Increment one the first time the app leaves its default_hmi for HMI_FULL, as in the resuming app scenario. Do not increment anytime an app comes into HMI_FULL. Do not increment when cycling sources. For all 3 scenarios, both successful and unsuccessful app starts shall be counted.
minutes_in_hmi_background	Number	Number of minutes the application is in the HMI_BACKGROUND state.
minutes_in_hmi_full	Number	Number of minutes the application is in the HMI_FULL state.

PROPERTY	TYPE	DESCRIPTION
minutes_in_hmi_limited	Number	Number of minutes the application is in the HMI_LIMITED state.
minutes_in_hmi_none	Number	Number of minutes the application is in the HMI_NONE state.

General Errors

Some basic usage and error counts are stored in the following properties.

PROPERTY	TYPE	DESCRIPTION
count_of_iap_buffer_full	Number	Number of times the iOS accessory protocol buffer is full.

Example

An example of how the Usage and Error portion of a Policy Table might look.

```
"usage_and_error_counts": {
    "count_of_iap_buffer_full": 1,
    "app_level": {
        "[App ID Here]": {
          "app_registration_language_gui": "en-us",
          "app_registration_language_vui": "en-us",
          "count_of_rejected_rpcs_calls": 9,
          "count_of_rejections_duplicate_name": 2,
          "count_of_rejections_nickname_mismatch": 1,
          "count_of_removals_for_bad_behavior": 6,
          "count_of_rfcom_limit_reached": 1,
          "count_of_rpcs_sent_in_hmi_none": 7,
          "count_of_run_attempts_while_revoked": 0,
          "count_of_user_selections": 7,
          "minutes_in_hmi_background": 123,
          "minutes_in_hmi_full": 123,
          "minutes_in_hmi_limited": 456,
          "minutes_in_hmi_none": 456
        }
    }
}
```

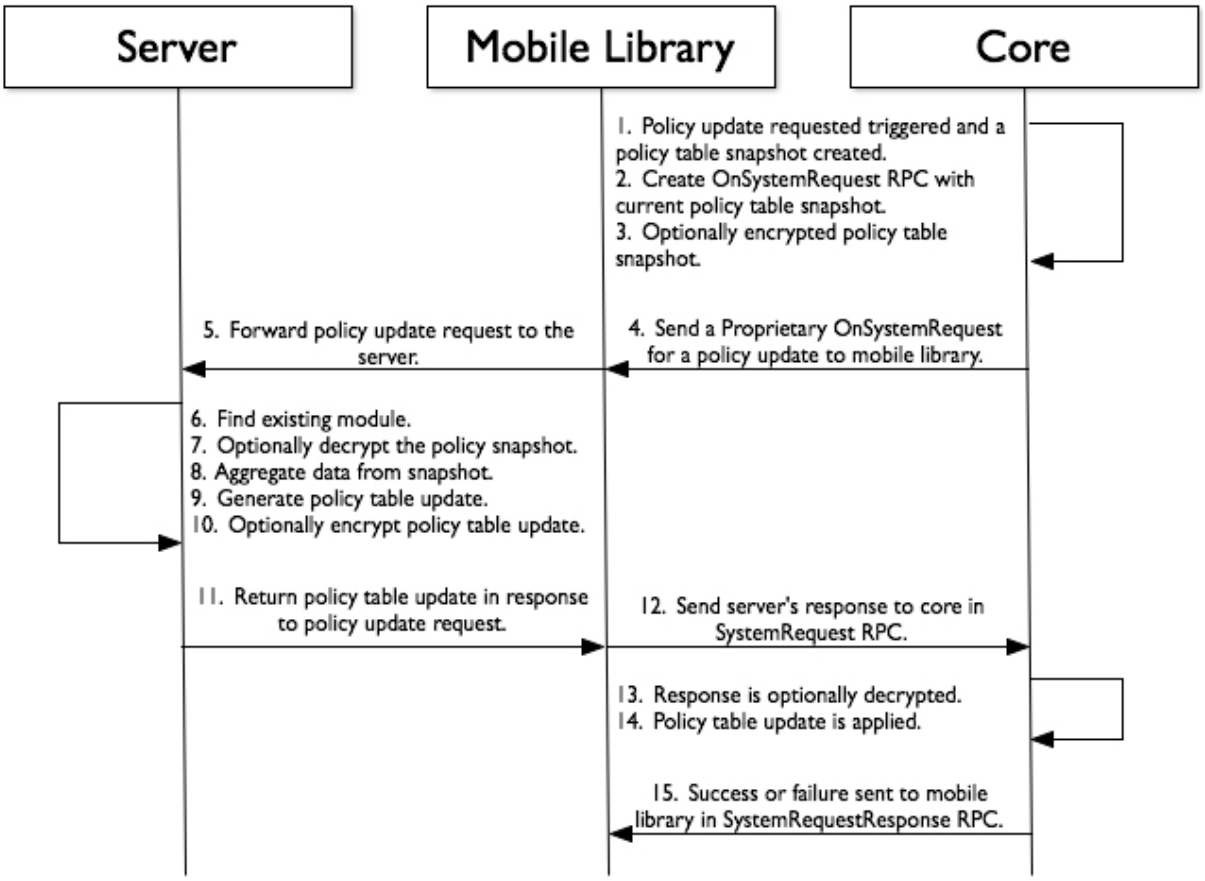

Policy Table Update

Periodically changes will be made to a Policy Table, either by the Policy Server or SDL Core. In order to synchronize the two tables, a Policy Table update must be performed. An update is triggered by Core by either an application connecting for the first time, or by one of the Policy Table update configurations, or by a user's request. When requesting a Policy Table update, SDL Core sends its current Policy Table, called a Policy Table snapshot, to the server. The server records any aggregate usage data as needed or designed, then responds to the request with a Policy Table update that contains the latest module config, functional groupings, application policies, and consumer friendly messages. The application policies section will only contain information for the current list of applications in the received Policy Table snapshot. In addition, the consumer friendly messages will only be included if an update is required, meaning the received Policy Table snapshot has an older version than the server.

SEQUENCE DIAGRAM

Policy Table Update Sequence Diagram

View Diagram



Policy Table Update Sequence Diagram Steps

1. A Policy Table update is triggered by SDL Core and a snapshot of the current Policy Table is created. The snapshot includes the entire local Policy Table with one exception. Only the version number property of the consumer friendly messages section is included in the snapshot.

2. An OnSystemRequest RPC is created with a request type of proprietary. The RPC contains a Policy Table snapshot in binary and a URL from one of the endpoints defined in the module config. In addition, HTML request headers can be present to be used when making the request.
3. The RPC's data is, optionally, encrypted using an asynchronous key that only the Policy Server can decrypt. The URL and headers are not encrypted since they are required by the mobile library to forward the request to the Policy Server.
4. The RPC is then sent to the mobile library.
5. The mobile library will ignore the request body containing the Policy Table snapshot, because it is marked as proprietary, and will forward the request to the URL included in the OnSystemRequest RPC. If the request fails to send then the mobile library will attempt to retry using the configuration specified in the module config.
6. When the server receives the Policy Table update request it will first look up the module in the server's database using a unique identifier. If the module is not found an error will be returned in the server's response.
7. If the Policy Table snapshot is encrypted, then the server will use the symmetric key found in the module's database record, the one we just looked up, to decrypt the Policy Table snapshot. If the data cannot be decrypted, then the data is not from a trusted source and an error is returned in the server's response.
8. The aggregate usage data and vehicle data in the received Policy Table snapshot is recorded to the server's database. Typically Usage and Error Counts, Device Data, and Module Meta contain data to be recorded.
9. A Policy Table update is created based on the received Policy Table snapshot. Note that only applications listed in the policy snapshot will be included in the update. In addition, if the consumer friendly messages version number is lower than the version available on the server, then the updated consumer friendly messages will also be included in the policy update.
10. Then the Policy Table update is, optionally, encrypted using an asynchronous key from the module record we previously looked up.
11. Finally the Policy Table update is returned in the response to the policy update request.
12. The mobile library then forwards the server's response to SDL Core using a SystemRequest RPC message.
13. After being received byCore the response body, if encrypted, is decrypted using an asymmetric key. If the body cannot be decrypted, then the data is not from a trusted source and an error is returned to the mobile library using a SystemRequestResponse RPC.

14. The Policy Table update is applied by replacing the following fields in the local Policy Table with the fields from the Policy Table update: module config, functional groupings, and application policies. In addition, if the consumer friendly messages section of the Policy Table update contains a **messages** subsection, then the entire consumer friendly messages portion of the local Policy Table will be replaced with the values from the Policy Table update.
15. If the response is valid and everything updates ok, then success is returned to the mobile library using a SystemRequestResponse RPC.

About

The SDL Policy Server helps manage functional groups for the user. Using the UI, groups of permissions can be easily created and tested. Each functional group represents a collection of permissions that should be granted together when incoming application requests sets of permissions. How these apps get the correct functional groups is another part of the problem, and the SDL Policy Server automatically handles that for the user.

Factors

An application must be granted its permissions in order for functional groups to be assigned to it. An application is granted permissions if that application version's approval state is in STAGING or in ACCEPTED, and the difference between the states is whether that application's permissions are granted when using only the staging policy table or when using both staging and production policy tables.

Incoming applications will request specific permissions (ex. Alert, Show, speed, gps) in a certain HMI level. The permission requested and the HMI level requested must both be present in a functional group for that functional group to be eligible for being granted to the user. For every permission that is granted by an application, the server will search through all functional groups to find ones matching that permission and HMI level. If there is a match found, that functional group and all other permissions found in that group will be granted to the user.

Any functional group that is checked to be granted to all applications by default will automatically be given to all applications that are not blacklisted.

Any functional group that is checked to be granted to all applications prior to the user accepting SDL data consent will automatically be given to all applications that are not blacklisted.

Proprietary functional group are to be manually assigned to applications in review.

Applications requesting widget management privileges will be given functional groups that have the corresponding checkbox checked.

Applications requesting administrator privileges will be given functional groups that have the corresponding checkbox checked.

Applications requesting at least one service provider type will be given functional groups that have the corresponding checkbox checked.

When using the staging policy table, the functional groups that are available for assignment will be the same functional groups seen in the Functional Groups UI menu in STAGING mode. Similarly, the production policy table uses the functional groups seen in PRODUCTION mode.

Example

An application comes in requesting permissions for the vehicle data `gps` in `HMI_BACKGROUND`. The application's approval state is in ACCEPTED.

The functional groups in STAGING mode include the following:
1. Contains gps in HMI levels FULL, LIMITED, BACKGROUND. Contains speed in HMI level FULL
2. Contains gps in HMI levels FULL, LIMITED, BACKGROUND. Contains rpm in HMI level FULL

The functional groups in PRODUCTION mode include the following:
1. Contains gps in HMI levels FULL, LIMITED. Contains speed in HMI level FULL
2. Contains gps in HMI levels FULL, LIMITED, BACKGROUND. Contains rpm in HMI level FULL

If the STAGING policy table is requested, the application is allowed permissions because the approval state is ACCEPTED. It will potentially receive functional groups in STAGING mode. It gets functional group #1 and #2 because both contain the requested `gps` permission in `HMI_BACKGROUND`. It also gets `speed` in HMI level FULL and `rpm` in HMI level FULL.

If the PRODUCTION policy table is requested, the application is allowed permissions because the approval state is ACCEPTED. It will potentially receive functional groups in PRODUCTION mode. It gets functional group #2 because only #2 contains the requested `gps` permission in `HMI_BACKGROUND`. It also gets `rpm` in HMI level FULL. If the approval state was STAGING, it would only get the default functional groups, and there are none in this case.

PostgreSQL

The Policy Server uses a PostgreSql database to store, retrieve, and update information.

Migrations

All scripts for the initial data migration are located in the migrations folder. The scripts necessary to build or reset the database are found there. Ensure that your policy server has been updated to have the latest migrations. If new migrations exist, they will be run on startup.

Database Alterations

Any action that generates newly created or updated data, such as modifying a consumer message, will first generate a SQL statement to execute the desired query. The Policy Server generates these statements with the npm module sql-bricks-postgres.