

SDL Server Guides

Document current as of 12/19/2019 12:07 PM.

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

Prerequisites

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

PROJECT	VERSION
Postgres	9.6+
Node.js	4.0.0+
NPM	3.0.0+

You must also acquire a set of SHAIID API keys. These are made available to level 4 OEM members through the [developer portal](#).

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 new version of the Policy Server requires a SQL database. Currently the only supported implementation is PostgreSQL. This guide will not cover how to get one running.

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.

Here are the environment variables that will most likely be used:

- `POLICY_SERVER_HOST`: String. The hostname or public IP address which the server runs on.
- `POLICY_SERVER_PORT`: Integer. The port which the server runs on. It is optional and the default is 3000.
- `POLICY_SERVER_PORT_SSL`: Integer. 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. 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. The filename of the SSL certificate's private key located in `./customizable/ssl`. Required if a value is set for `POLICY_SERVER_PORT_SSL`.
- `SHAIID_PUBLIC_KEY`: String. A public key given to you through the [developer portal](#) that allows access to SHAIID endpoints.
- `SHAIID_SECRET_KEY`: String. A secret key given to you through the [developer portal](#) that allows access to SHAIID endpoints.
- `DB_USER`: String. The name of the user to allow the server to access the database
- `DB_DATABASE`: String. The name of the database where policy and app data is stored
- `DB_PASSWORD`: String. The password used to log into the database
- `DB_HOST`: String. The host name or IP address of the database
- `DB_PORT`: Integer. The port number of the database
- `CACHE_MODULE`: String. The name of the caching module to use. Currently supports `null` (no caching, default) or `"redis"`.
- `CACHE_HOST`: String. The host name or IP address of the cache. Default `null`.
- `CACHE_PORT`: Integer. The port number of the cache. Default `null`.
- `CACHE_PASSWORD`: String. The password used to log into the cache. Default `null`.

- `SMTP_HOST`: String. The host name or IP address of an SMTP server to use for email notifications. A null value implies that outgoing emails are disabled. Default null.
- `SMTP_PORT`: Integer. The port number of the SMTP server. Default 25.
- `SMTP_USERNAME`: String. The username of the optional SMTP user. Default null.
- `SMTP_PASSWORD`: String. The password of the optional SMTP user. Default null.
- `SMTP_FROM`: String. The email address which emails are sent from. A null value implies that outgoing emails are disabled. Default null.
- `NOTIFY_APP_REVIEW_FREQUENCY`: String enum (DISABLED, REALTIME). The frequency of which outgoing emails should be sent to notify the OEM of new apps ready for review. Default DISABLED.
- `NOTIFY_APP_REVIEW_EMAILS`: String. A comma-separated list of email addresses to send an email to when new apps are ready for review. Default empty string.
- `AUTO_APPROVE_ALL_APPS`: String boolean ("true" or "false"). Whether or not to auto-approve all app versions received by SHAID (except for blacklisted apps). Default "false".
- `CA_PRIVATE_KEY_FILENAME`: String. Mandatory. The filename of your .key file generated, to be placed in `customizable/ca/`
- `CA_CERTIFICATE_FILENAME`: String. Mandatory. The filename of your .pem file generated, to be placed in `customizable/ca/`
- `CERTIFICATE_PASSPHRASE`: String. Mandatory. A secret password used for every certificate generated.
- `CERTIFICATE_COMMON_NAME`: String. Mandatory. Default information of the issuer's fully qualified domain name to secure.
- `PRIVATE_KEY_BITSIZ`: Integer. The size of the private keys generated. Default 2048.
- `PRIVATE_KEY_CIPHER`: String. The type of cipher to use for encryption/decryption. Defaults to "des3".
- `CERTIFICATE_COUNTRY`: String. Default information of the issuer's country (two-letter ISO code).
- `CERTIFICATE_STATE`: String. Default information of the issuer's state.
- `CERTIFICATE_LOCALITY`: String. Default information of the issuer's city.
- `CERTIFICATE_ORGANIZATION`: String. Default information of the issuer's legal company name.
- `CERTIFICATE_ORGANIZATION_UNIT`: String. Default information of the issuer's company's branch.
- `CERTIFICATE_EMAIL_ADDRESS`: String. Default information of the issuer's email address.

- `CERTIFICATE_HASH`: String. The cryptographic hash function to use. Default 'sha256'.
- `CERTIFICATE_DAYS`: Integer. The number of days until the certificate expires. Default 7.
- `ENCRYPTION_REQUIRED`: String boolean ("true" or "false"). Whether or not to require RPC encryption for auto-approved app versions. Default "false".
- `STAGING_PG_USER` **DEPRECATED**: String. The name of the user to allow the server access the database (staging mode)
- `STAGING_PG_DATABASE` **DEPRECATED**: String. The name of the database where policy and app data is stored (staging mode)
- `STAGING_PG_PASSWORD` **DEPRECATED**: String. The password used to log into the database (staging mode)
- `STAGING_PG_HOST` **DEPRECATED**: String. The host name or IP address of the database (staging mode)
- `STAGING_PG_PORT` **DEPRECATED**: Integer. The port number of the database (staging mode)
- `PRODUCTION_PG_USER` **DEPRECATED**: String. The name of the user to allow the server access the database (production mode)
- `PRODUCTION_PG_DATABASE` **DEPRECATED**: String. The name of the database where policy and app data is stored (production mode)
- `PRODUCTION_PG_PASSWORD` **DEPRECATED**: String. The password used to log into the database (production mode)
- `PRODUCTION_PG_HOST` **DEPRECATED**: String. The host name or IP address of the database (production mode)
- `PRODUCTION_PG_PORT` **DEPRECATED**: Integer. The port number of the database (production mode)

Production/Staging environment variables for the database are now deprecated. Please use the corresponding `DB_` values in place of them (ex. `DB_USER` instead of `PRODUCTION_PG_USER` or `STAGING_PG_USER`).

Once these environment variables are set, initialize the database. The database should be given the same name set in `DB_DATABASE`.

Using the `createdb` program that comes with the installation of PostgreSQL, for example:
`createdb policy_server`

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 server with hot reloading so any changes made to the UI are instantly updated in the browser
- `build`: Generates a new staging/production build using webpack. This command should only be run if you made front-end modifications to the UI.
- `start-pg-staging` **DEPRECATED**: Runs the migration up script which initializes data in the database, sets the environment to `staging` and starts the Policy Server
- `start-pg-production` **DEPRECATED**: Runs the migration up script which initializes data in the database, sets the environment to `production` and starts the Policy Server
- `db-migrate-reset-pg-staging` **DEPRECATED**: Runs the migration down script which drops all the data and tables in the staging database

Production/Staging scripts are now deprecated. Please use `start-server` instead of `start-pg-staging` or `start-pg-production` .

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

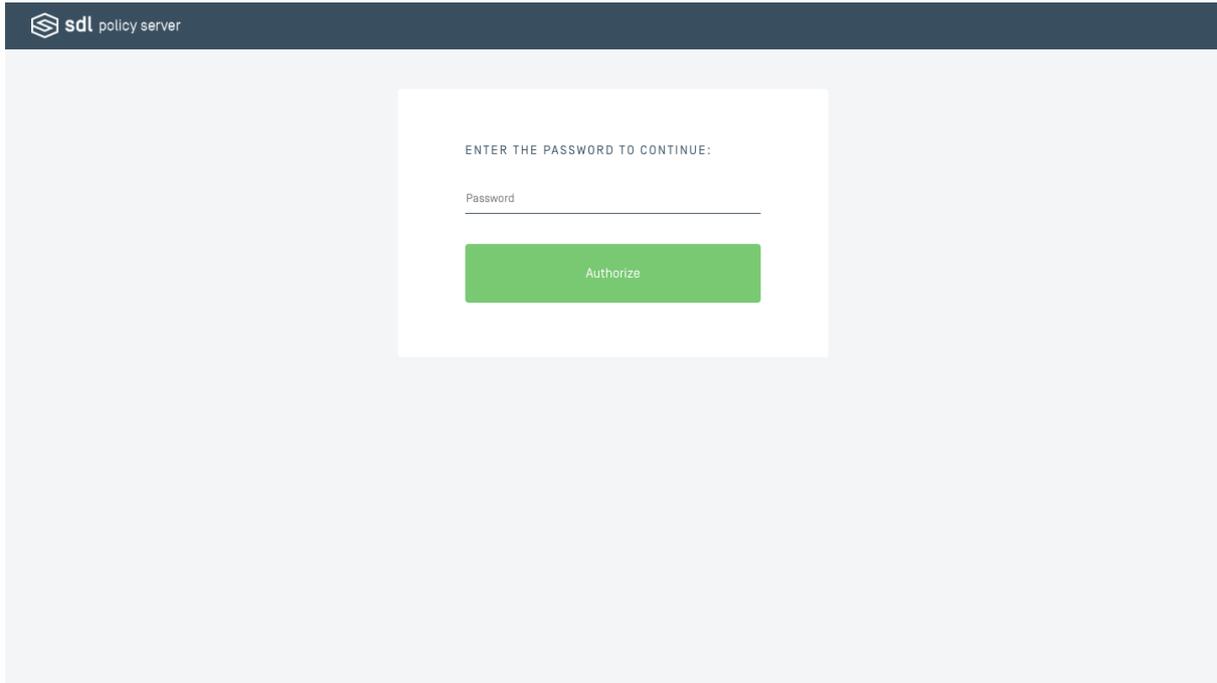
If you are unable to modify your environment variables, you may define these settings in the `./settings.js` configuration file

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: `./customizable/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
<code>openssl genrsa -out CA.key 2048</code>	This creates a 2048 bit RSA private key and saves it in the file "CA.key". It will later be used for signing certificates.
<code>openssl req -x509 -new -nodes -key CA.key -sha256 -days 3650 -out CA.pem</code>	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_BITSIZ	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.

VARIABLE	IS MANDATORY	DESCRIPTION
CERTIFICATE_ORGANIZATION	false	Default information of the issuer's legal company name.
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.

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 `/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.

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

About

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. Currently, if any changes are made to the files then before restarting the server the build command, found in the package.json, must be run in the terminal so as to rebuild the project with the changes. The /build folder contains all files associated with **Webpack**.

Applications

sdl policy server

Applications 1

- View Policy Table
- Functional Groups
- Consumer Messages
- Custom Vehicle Data
- Module Config
- About

Pending Applications

Application Name	Last Update	Platform	State
Livio Music Player	2019-03-05T19:45:34.285Z	IOS	PENDING Review

Accepted Applications

No applications in this approval state.

Limited Applications

No applications in this approval state.

Blacklisted Applications

No applications in this approval state.

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
- CalculateRoute

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 with regards to 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.

Consumer Friendly Messages

AppPermissions 20 languages	AppPermissionsHelp 20 languages	AppPermissionsRevoked 20 languages
AppUnauthorized 20 languages	AppUnsupported 20 languages	DataConsent 4 languages
DataConsentHelp 3 languages	DisableApps 20 languages	DrivingCharacteristics 20 languages
Location 20 languages	LockScreenDismissalWarning 1 language	Notifications 20 languages

Functional Groups

AdministratorGroup 6 permissions	AppServiceConsumerGroup DEFAULT 5 permissions	AppServiceProviderGroup 7 permissions
BackgroundAPT 3 permissions	Base-4 DEFAULT 46 permissions	Base-6 43 permissions
BaseBeforeDataConsent 19 permissions	DataConsent-2 0 permissions	DiagnosticMessageOnly 1 permission
DialNumberOnlyGroup DEFAULT 1 permission	DrivingCharacteristics-3 4 permissions	Emergency-1 4 permissions

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.

The screenshot shows the 'sdI policy server' interface with a dark sidebar on the left containing navigation items: Applications (18), View Policy Table, Functional Groups, Consumer Messages, Custom Vehicle Data, Module Config, and About. The main content area displays two message cards for 'zh-cn' and 'zh-tw' languages. Each card has a close button (x) in the top right corner. The 'zh-cn' card contains the following text in its fields: TTS: %appName% 正在请求使用下列车辆信息和权限: %functionalGr; LINE 1: 是否允许请求的; LINE 2: 权限?; TEXT BODY: (empty); LABEL: (empty). The 'zh-tw' card contains: TTS: %appName% 正请求使用 %functionalGroupLabels% 的车辆资讯和; LINE 1: 允许; LINE 2: 授权请求?; TEXT BODY: (empty); LABEL: (empty). Below the cards is a plus sign (+) and a green button labeled 'Save consumer message'.

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

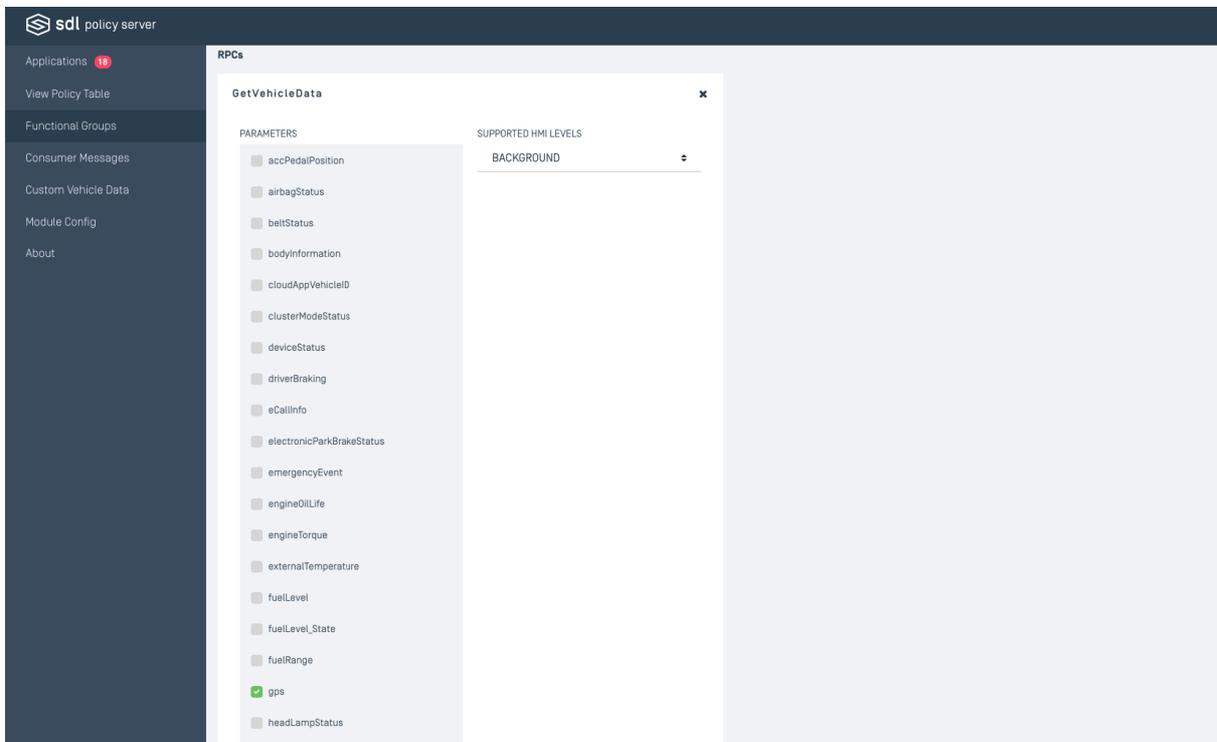
CHECKBOX	NOTES
Grant this functional group to all applications by default	If set to true, all staging and accepted applications will have access to this functional group and its permissions.
Grant this functional group to all applications prior to the user accepting SDL data consent	
Grant this functional 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

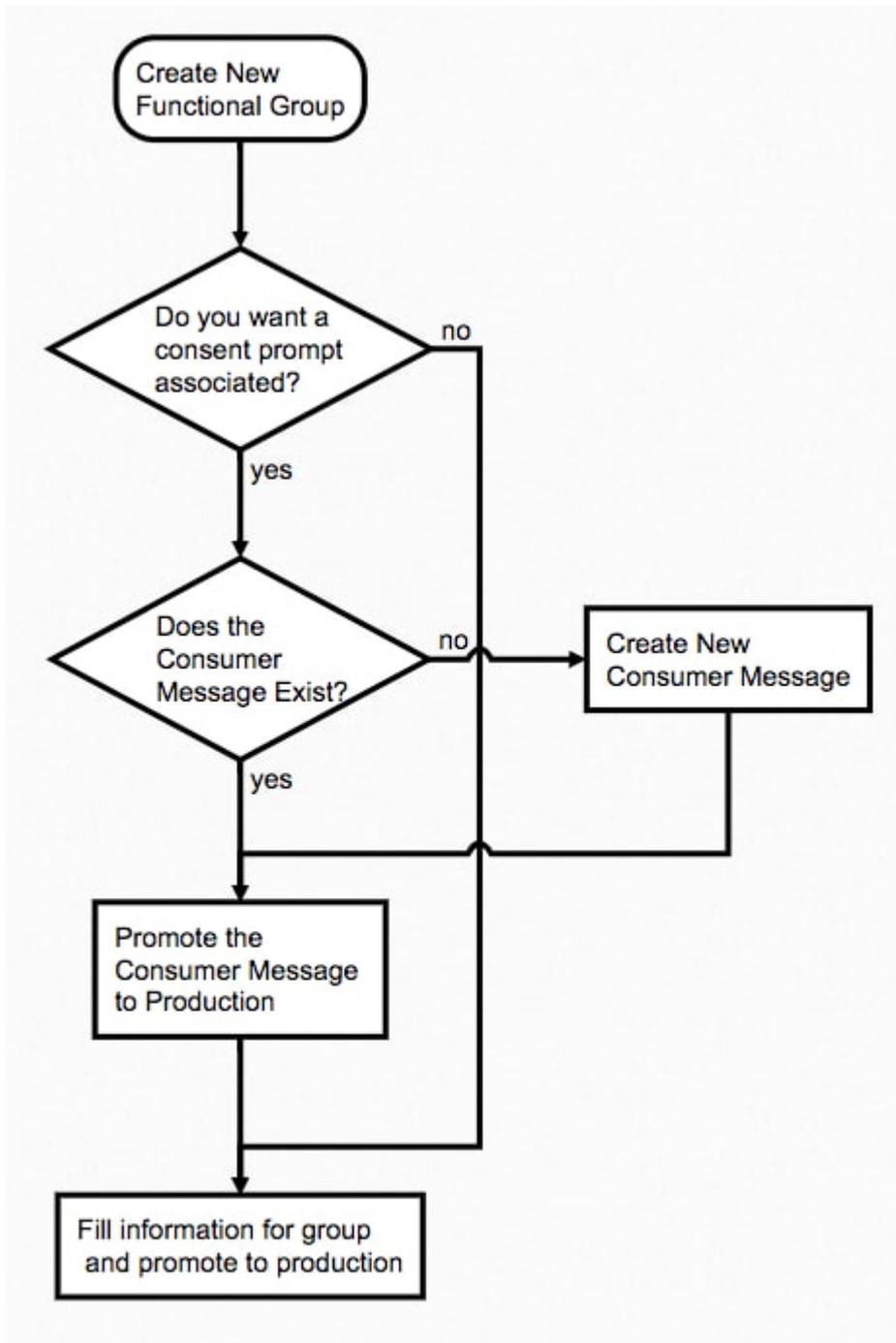
PROPERTY	DEFINITION
Parameters	References possible vehicle information that can be retrieved. This is only applicable to vehicle data RPCs. eg. GetVehicleData, SubscribeVehicleData
Supported HMI Levels	SDL Core interface display levels allowed by the app



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.

Consumer Message ? Delete

Name

AppPermissions

Languages

de-de x

TTS: %appName% benötigt die folgenden Fahrzeuginformationen und Z

LINE 1: Zugriffsanfrage(n)

LINE 2: erlauben?

TEXT BODY:

LABEL:

en-au x

TTS: %appName% is requesting the use of the following vehicle informe

LINE 1: Grant requested

LINE 2: permission(s)?

TEXT BODY:

LABEL:

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.

The screenshot shows the 'Module Config' page in the SDL Policy Server. The interface is divided into a dark sidebar on the left and a main content area on the right. The sidebar contains navigation links: 'Applications' (with a red '44' badge), 'View Policy Table', 'Functional Groups', 'Consumer Messages', and 'Module Config'. The main content area has a top navigation bar with 'Staging' (selected) and 'Production' tabs. Below this is the 'Module Config' title and a help icon. The configuration is organized into sections:

- Refresh the Policy Table after every:**
 - Input: 100, Unit: Ignition Cycles
 - Input: 1800, Unit: Kilometers Traveled
 - Input: 30, Unit: Days
- Policy Table Refresh Timeout:**
 - Input: 60, Unit: Seconds
- When a Policy Table Refresh Fails:**

Retry after	1	second	✘
Retry after	5	seconds	✘
Retry after	25	seconds	✘

Editing and Saving

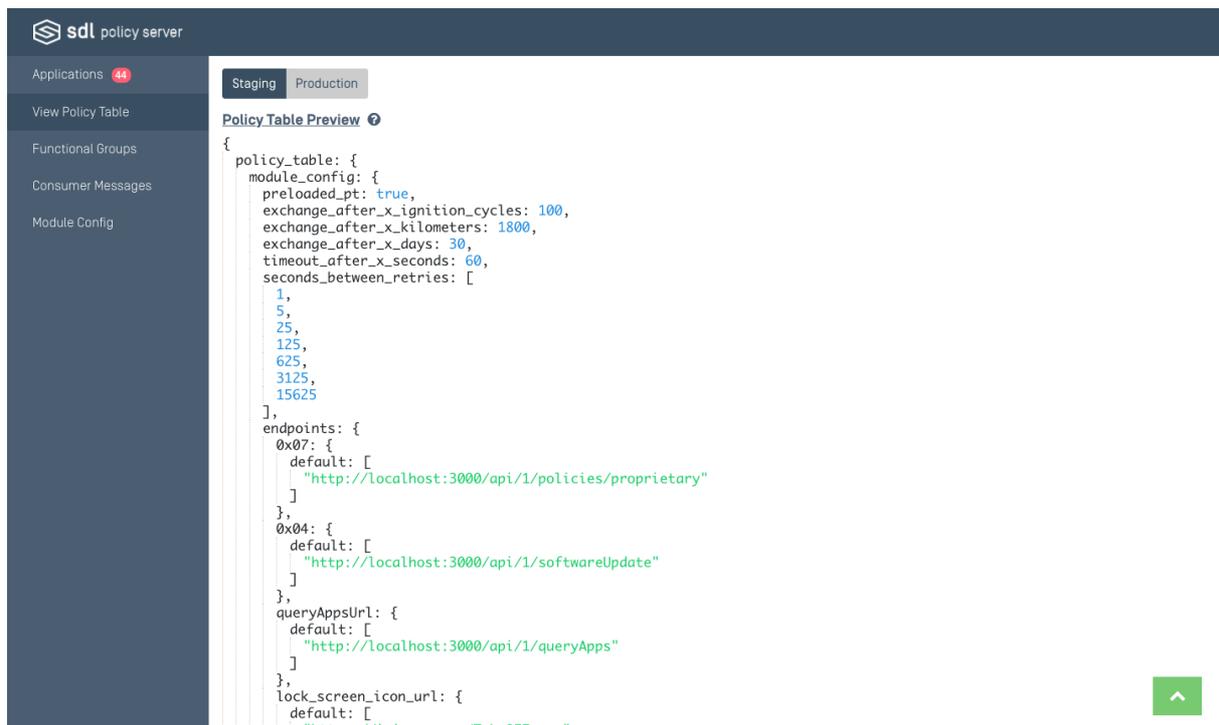
The process of editing and saving is very similar to that of functional groups and consumer messages. It is simpler here because the entire object is either in staging or production. Production versions cannot be edited, but can be overwritten by promoting a staging module config. There is no creating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



The screenshot shows the 'sdl policy server' interface. On the left is a dark sidebar with navigation options: Applications (44), View Policy Table, Functional Groups, Consumer Messages, and Module Config. The main area is titled 'Policy Table Preview' and shows a JSON configuration for a policy table. The configuration includes module settings, endpoints for different vehicle types, and query URLs. A green arrow icon is visible in the bottom right corner of the main area.

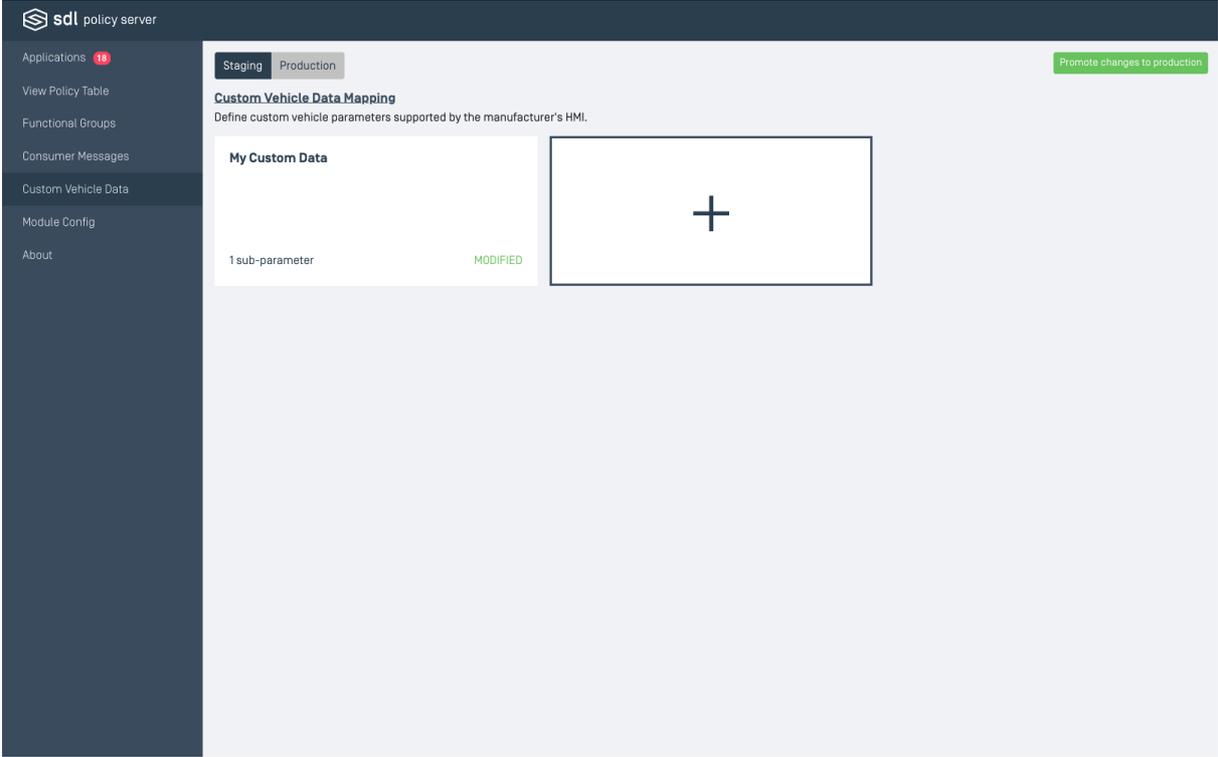
```
{
  policy_table: {
    module_config: {
      preloaded_pt: true,
      exchange_after_x_ignition_cycles: 100,
      exchange_after_x_kilometers: 1800,
      exchange_after_x_days: 30,
      timeout_after_x_seconds: 60,
      seconds_between_retries: [
        1,
        5,
        25,
        125,
        625,
        3125,
        15625
      ],
    },
    endpoints: {
      0x07: {
        default: [
          "http://localhost:3000/api/1/policies/proprietary"
        ]
      },
      0x04: {
        default: [
          "http://localhost:3000/api/1/softwareUpdate"
        ]
      },
    },
    queryAppsUrl: {
      default: [
        "http://localhost:3000/api/1/queryApps"
      ]
    },
    lock_screen_icon_url: {
      default: [
        "https://i.imgur.com/Taku0T7.png"
      ]
    }
  }
}
```

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.

The screenshot shows the 'sdI policy server' interface. On the left is a dark sidebar with navigation links: Applications (18), View Policy Table, Functional Groups, Consumer Messages, Custom Vehicle Data (highlighted), Module Config, and About. The main content area is a form for creating a custom vehicle data item. The form fields are: * NAME (Sub Param), * TYPE (Float), * KEY (sub-param), IS MANDATORY, MIN LENGTH, MAX LENGTH, MIN SIZE, MAX SIZE, MIN VALUE, MAX VALUE, and IS ARRAY. A plus sign (+) is centered below the form. At the bottom of the form is a green button labeled 'Save custom vehicle data item'.

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

The screenshot shows the 'Custom Vehicle Data Item' configuration page. The left sidebar contains navigation links: Applications (18), View Policy Table, Functional Groups, Consumer Messages, Custom Vehicle Data (selected), Module Config, and About. The main content area is titled 'Custom Vehicle Data Item' and includes the instruction 'Define custom vehicle data params supported by the HMI.' The form contains the following fields and options:

- * NAME: Text input field
- * TYPE: Dropdown menu
- * KEY: Text input field
- IS MANDATORY: Must be false for the root level
- MIN LENGTH: Text input field
- MAX LENGTH: Text input field
- MIN SIZE: Text input field
- MAX SIZE: Text input field
- MIN VALUE: Text input field
- MAX VALUE: Text input field
- IS ARRAY

A red error message at the bottom of the form states: 'All name, type, and key fields must be defined.'

Staging

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

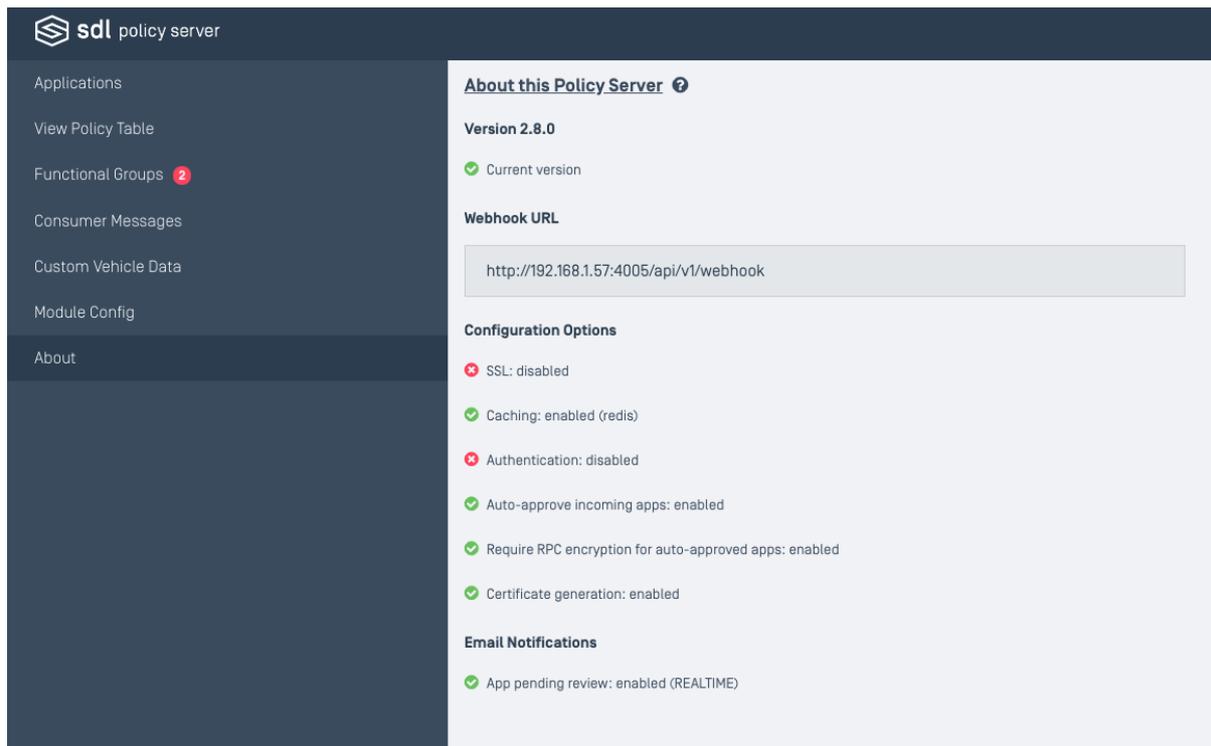
Production

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)



sdl policy server

- Applications
- View Policy Table
- Functional Groups 2
- Consumer Messages
- Custom Vehicle Data
- Module Config
- About**

About this Policy Server [?](#)

Version 2.8.0

- ✔ Current version

Webhook URL

http://192.168.1.57:4005/api/v1/webhook

Configuration Options

- ✘ SSL: disabled
- ✔ Caching: enabled (redis)
- ✘ Authentication: disabled
- ✔ Auto-approve incoming apps: enabled
- ✔ Require RPC encryption for auto-approved apps: enabled
- ✔ Certificate generation: enabled

Email Notifications

- ✔ App pending review: enabled (REALTIME)

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 appHMI Types 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	

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" ],
    "AppHMType": [ "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 a 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. 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 <code>fullAppID</code> will be used in the <code>app_policies</code> section as it's key. If false or omitted, the short-form <code>appID</code> 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.

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": {
  "endpoints": {
    "0x07": {
      "default": [ "http://localhost:3000/api/1/policies/proprietary" ],
    }
  },
  "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 recorded. 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 statistics 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_rpc_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_rfc_com_limit_reached	Number	Number of times the maximum number of rfc com 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.

PROPERTY	TYPE	DESCRIPTION
minutes_in_hmi_full	Number	Number of minutes the application is in the HMI_FULL state.
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 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 appHMI Types 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	

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 a 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. 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 <code>fullAppID</code> will be used in the <code>app_policies</code> section as it's key. If false or omitted, the short-form <code>appID</code> 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 recorded. 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 statistics 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_rpc_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_rfc_com_limit_reached	Number	Number of times the maximum number of rfc com 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.

PROPERTY	TYPE	DESCRIPTION
minutes_in_hmi_full	Number	Number of minutes the application is in the HMI_FULL state.
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
    }
  }
}
```

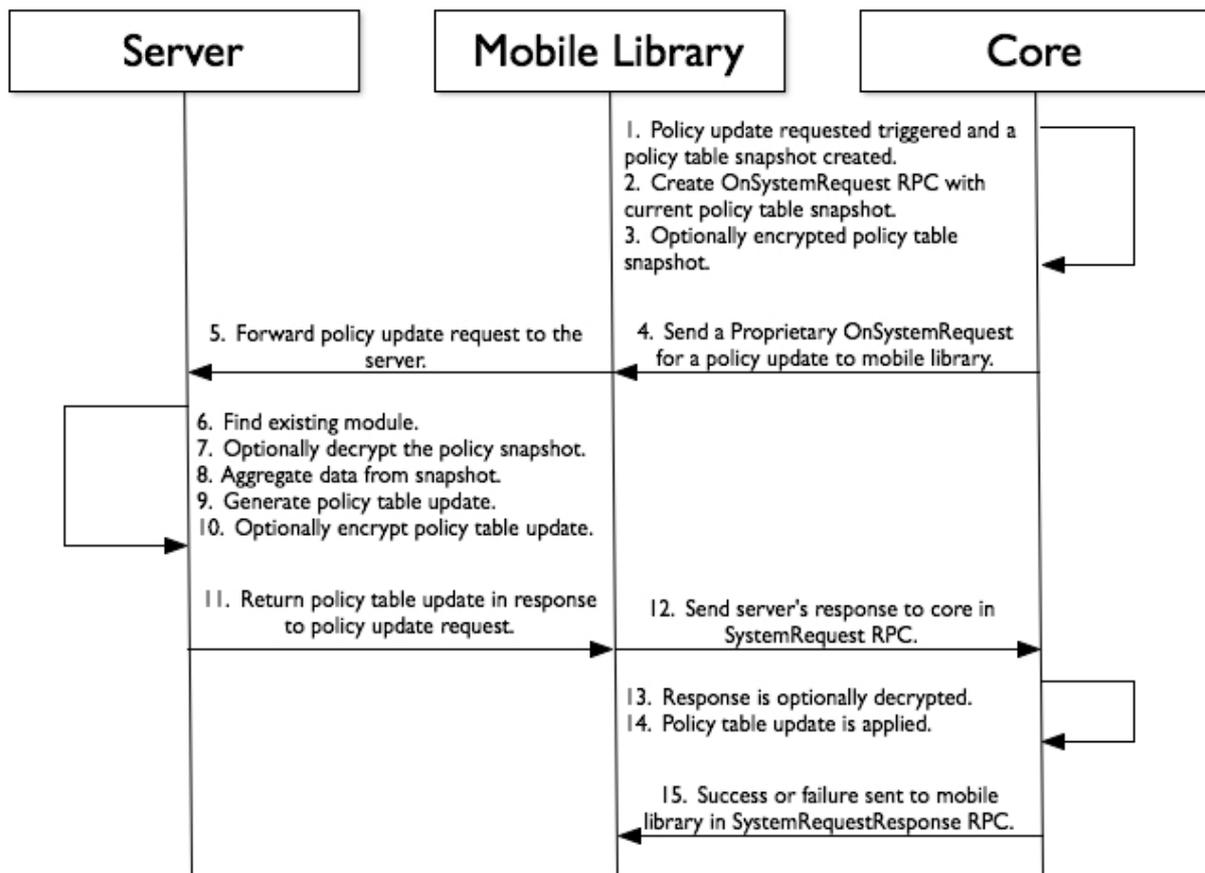
Policy Table Update

Periodically changes will be made to a Policy Table, either by an 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 lookup 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 by Core 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`.