License Manager

License Manager System for all Doubango products (ANPR, ALPR, MRZ, MICR, NSFW, SceneText, ICT, OCR)

https://github.com/DoubangoTelecom/LicenseManager-SDK

https://www.doubango.org/LicenseManager/

Table of Contents

1 Intro....................4
2 Jargon....................5
2.1 Runtime Key....................5
2.2 Activation....................5
2.3 Token....................5
2.4 Master Key....................5
2.5 Slave Key....................5
3 Web application....................7
4 REST API....................8
4.1 Generate Slave Key....................8
4.1.1 Request....................8
4.1.2 Response....................8
4.2 Activate Runtime Key....................9
4.2.1 Request....................9
4.2.2 Response....................9
5 Activation use cases....................10
5.1 Manual activation....................10
5.1.1 You have access to the device....................10
5.1.2 You don't have access to the device....................10
5.2 Automatic activation....................11

This is a short technical guide to help developers and integrators take the best from our license manager system. You don't need to be a developer to understand and follow the recommendations defined in this guide.

1 Intro

Our license manager service is a secure cloud-based system to help our customers manage their licenses with minimal effort. The system was designed with simplicity in mind to allow easy integration in your products.

The service has two parts :

1. A web application : https://www.doubango.org/LicenseManager/
2. A multi-language SDK : https://github.com/DoubangoTelecom/LicenseManager-SDK

The web application is the easiest way to manage your licenses but sometimes you'll need to automate the process and this is when the SDK is needed. The automation allows managing hundreds of licenses with little effort.

The first step is to obtain your login and password from us.

2 Jargon

2.1 Runtime Key

A **Runtime Key** is a unique base64 string identifier representing a device (PC, mobile phone, camera...). Every SDK developed by us have a function named "`requestRuntimeLicenseKey()`" to generate this unique string.

This function doesn't require network connection.

This key can be publicly shared.

2.2 Activation

Once the **Runtime Key** is generated you have to activate it in order to get a **Token**.

The activation can be done using the web application, the REST API, or the C++ SDK.

The activation function requires network connection.

The activation function requires a **Master Key** or a **Slave Key.**

2.3 Token

A **Token** is an activated **Runtime Key**. This is actually the license key for the device uniquely identified by the **Runtime Key**.

We recommend saving the **Token** in a file (for example in "assets" folder on Android) to avoid repeating the activation process.

The **Token** can be retrieved using the web application, the REST API, or the C++ SDK.

The **Token** is valid forever.

The **Token** can be publicly shared (e.g. hard-coded in the code or in a file).

2.4 Master Key

The **Master key** is available via the web application. You have to login first to get your **Master Key**.
For every product (e.g. ANPR or NSFW) you'll have as many master keys as operating systems (e.g. Android or Raspberry Pi). For example, if you've subscribed to the ANPR SDK for Android and Raspberry Pi then, you'll have 2 master keys. A **Master Key** can only activate a **Runtime Key** targeting the same operating system and product.

The Master Key is confidential and must NEVER EVER be shared in any form. For example, you must not include the **Master Key** in your code or send it to the end user.

2.5 Slave Key

As explained above the **Master Key** must never ever be shared with the end user. A **Slave Key** is generated using a **Master Key** and it's like a bearer bond with 1-unit credit. From Wikipedia, about bearer bond: *"Whoever physically holds the paper on which the bond is issued is the presumptive*

owner of the instrument”.
You can include a **Slave Key** in the application or send it to the end user via secure channel (e.g. email or usb key).

A slave can be generated using the web application, the REST API, or the C++ SDK.

A **Slave Key** can only be used once and will decrease your balance by 1-unit at maximum. Once it's used to activate a **Runtime Key** it'll be frozen. If you try to use the same **Slave** and **Runtime** keys for another activation then, it'll succeed without changing your balance. But, if you try to use the **Slave** with a different **Runtime** key then, it will fail and your balance won't change.

3 Web application

The web application is hosted at https://www.doubango.org/LicenseManager/.

The interface is pretty clear and easy to manage.

4 REST API

The web application uses REST API to communicate with our license server. You can use the same REST API to automatically manage your assets. This could be useful if you have a large amount of licenses to manage or you want to automatically activate the Runtime Keys.

The REST API contains more than 18 operations but only 2 are worth automating:

1. Generate **Slave Key**
2. Activate **Runtime Key**

For the next sections the activation address is obfuscated and replaced with *“activation-host-and-port”* to avoid spamming. You should have received the real address along with your connection information.

4.1 Generate Slave Key

See previous sections to understand why a **Slave Key** is needed.

4.1.1 Request

Request-Line		
Method		POST
URL		https://activation-host-and-port/slaves
JSON body		
Property	Type	Description
masterKey	String	Your base64 secret **Master Key**. You must be the one executing this request because the **Master Key** must be kept secret. You must not execute this request on the end user computer.

4.1.2 Response

The next body is about success responses (code within [200, 299]).

JSON body		
Property	Type	Description
organisation	String	The organisation identifier associated to the newly created slave. This is your organisation id.
master	String	The master identifier associated to the newly created slave. This is a more friendly string than Master Key to track the slaves.
used	Bool	Whether the slave is already used. Must be false.
createDate	Date	Date defining when the slave was created.
key	String	The newly generated **Slave Key** to share with the end user or include in the application.

4.2 Activate Runtime Key

To activate a **Runtime Key** you'll need a **Master** or **Slave Key**. If you're doing the operation by yourself then, you can use your **Master Key.** Otherwise, a **Slave Key** must be used to avoid sharing your secret **Master Key**. See previous section on how to generate a **Slave Key**.

4.2.1 Request

Request-Line		
Method	POST	
URL	https://activation-host-and-port/activate	
JSON body		
Property	Type	Description
masterOrSlaveKey	String	Your base64 secret **Master** or **Slave Key** generated as previously explained.
runtimeKey	String	Base64 string uniquely identifying a device.

4.2.2 Response

The next body is about success responses (code within [200, 299]).

JSON body		
Property	Type	Description
token	String	This is the actual license key (base64) to use in the application to unlock all the features.
totalCredit	Integer	Total credit units on your balance.
usedCredit	Integer	Used credit units on your balance after the activation operation. Multiple activation operations on the same **Slave** or **Runtime Key** will only decrease this entry by 1-unit.

5 Activation use cases

We highly recommend using the manual activation for your first device to understand the process. You can activate the device several times to make yourself familiar with the process. Activating the same device several times won't change your balance.

5.1 Manual activation

This section is about manual activation.

5.1.1 You have access to the device

The steps for manual activation when you have access to the device are:

1. Generate a **Runtime Key** as explained in the previous sections.
2. Login to the web application.
3. Select the activation icon next to the master you want to use to open the activation page.
4. Enter the Runtime Key. The Master Key should be already automatically filled.
5. Press "Send" to activate.
6. A new box with the token will appear at the bottom if the operation succeed. Later, you can retrieve the token using the web application (Runtimes page) without repeating the activation process.
7. Copy the token and put it on a file on the target device.
8. When you initialize the engine use JSON entry "`license_token_data`" or "`license_token_file`" to set the token. The first option accepts a base64 string while the second requires a path to a file.

This method is time consuming but could be useful when the target device doesn't have access to internet.

5.1.2 You don't have access to the device

When you don't have access to the device their are 2 solutions:

- Solution A:
  1. Ask the end user to send you the Runtime Key.
  2. Use steps in the previous section to activate the Runtime Key.
  3. Send back a file containing the **Token** to the end user.
- Solution B:
  1. Use the web application to create a Slave Key.
  2. Copy the Slave Key.
  3. When the end user runs the application for the first time, redirect him to

https://www.doubango.org/LicenseManager/activation.html?slaveKey=Base64Encoded&runtimeKey=Base64Encoded.

4. The end user will have to follow steps [5, 6 and 7] described in the previous section.

Please note that for solution B you can also redirect the user to your own website and use the REST API to activate the Runtime Key.

5.2 Automatic activation

The manual activation is time consuming with poor user experience. You can use the manual activation if your company is the end user with low number of **Runtime Keys** to activate. If you're sub-licensing the product to your customers or have a large number of licenses to activate then, it's definitely not for you.

The automatic activation uses the REST API defined in the previous sections which means the end user must have internet connection. This is only required at the activation time, the first time the application is opened.

The activation server is hosted on Doubango cloud but sometimes you need the end user to connect to your own servers instead of ours. What we recommend in such cases is to host a node.js proxy-like REST API server and let the end user connect to the proxy instead of our cloud.

Here are the steps for automatic activation:

1. Using the REST API (see previous sections), generate as many **Slave Keys** as end users. You can also use the web application to generate the **Slave Key**s.
2. Copy every Slave Key (Base64) to a file and package it with the application binaries to be sent to the end user. For example, on Android, put the file in "assets" folder.
3. When the end user opens the application for the first time then, use the REST API to activate the Runtime Key. The **Runtime Key** is generated on the device using API function **`requestRuntimeLicenseKey`** as explained in the previous sections. The activation requires a **Slave Key** and the **Runtime Key**.
4. The activation will return a **Token** (Base64). Save the **Token** to a file. Next time the user opens the application then, just read the **Token** from the file instead of repeating the activation process.

Our sample applications contains the code needed for automatic activation.