Tuya API application

Generated by Doxygen 1.12.0

Chapter 1

Tuya API Application

A custom application to connect and manage IoT devices using the Tuya API. This project is a pilot project to understand the tuya API and it's Applications

---

1.1 Features

- Connect IoT devices to Tuya's cloud platform.
- Perform device configuration and control via API calls.
- User-friendly interface for managing devices.

1.2 Technology Stack

- **Language:** Python
- **API:** Tuya IoT Cloud API
- **Protocols:** HTTP
- **Platform:** ESP32 IoT devices, Tuya-enabled devices

1.3 Requirements

- Tuya platform supported IoT device
- Tuya Developer Account
- Python 3.8+
- Internet Connection

1.4 Installation

1. **Clone the repository:**
   ```
   git clone https://github.com/NishDananjaya/tuya_API_Application.git
   cd tuya_API_Application
   ```
   ```
   2. **Create a python virtual enviorenment**
   ```
   python python -m venv [your envioronment name]
2. **Install the requirements** ```bash pip install -r requirements.txt

3. **Crteat a .env file**

   Add following keys

   - TUYA_ACCESS_ID=
   - TUYA_ACCESS_KEY=
   - TUYA_BASE_URL=
   - DEVICE_ID =

   put the fields empty

1.5 Future Implementations

- [ ] **Automatically Update Device Status**
  Real-time updates for device status without manual refresh.
- [ ] **Enhanced User-Friendly GUI**
  - Automatically display device configurations upon selection.
- [ ] **Scene Management**
  Add support for creating and managing scenes.
- [ ] **4-Gang Switch Features**
  - Countdown timer
  - Schedule timer
  - Cycle settings
  - Switch inching
  - Backlight control

Chapter 2

Namespace Index

2.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 3

Hierarchical Index

3.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 4

Class Index

4.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 5

File Index

5.1 File List

Here is a list of all files with brief descriptions:

Chapter 6

Namespace Documentation

6.1 auth Namespace Reference

Classes

- class AuthManager

Variables

- auth_manager = AuthManager()

6.1.1 Detailed Description

```
@file auth_manager.py
@brief Authentication manager for the Tuya IoT API.
@details This module provides a comprehensive authentication management system for
         the Tuya IoT API, including automatic token refresh, signature generation,
         and environment variable management.

@author Nishan Dananjaya
@date 2025-02-13
@version 1.0
```

6.1.2 Variable Documentation

6.1.2.1 auth_manager

```
auth.auth_manager = AuthManager()
```

6.2 authtest Namespace Reference

Classes

- class AuthManager

Variables

- auth_manager = AuthManager()

6.2.1 Variable Documentation

6.2.1.1 auth_manager

```
authtest.auth_manager = AuthManager()
```

6.3 control Namespace Reference

Functions

- generate_signature (client_id, secret, access_token, t, nonce, url, method="POST", body="")
- control_device (url, client_id, secret, access_token, device_id, switch, command)

Variables

- dot_env_path = os.path.join(os.path.dirname(__file__), '..', '.env')
- API_URL = os.getenv("TUYA_BASE_URL")
- CLIENT_ID = os.getenv("TUYA_ACCESS_ID")
- SECRET = os.getenv("TUYA_ACCESS_KEY")
- ACCESS_TOKEN = os.getenv("ACCESS_TOKEN")
- DEVICE_ID = os.getenv("DEVICE_ID")
- switch = input("Enter what to control (e.g., switch_1): ")
- command = input("Enter the command (e.g., true/false): ")
- result = control_device(API_URL, CLIENT_ID, SECRET, ACCESS_TOKEN, DEVICE_ID, switch, command)

6.3.1 Detailed Description

```
@file device_control.py
@brief A Python script to control Tuya IoT devices through the API (for the 4 gang switch).
@details This script provides functionality to send control commands to Tuya devices
         using the Tuya IoT API with HMAC-SHA256 authentication. It supports
         controlling device switches and other boolean states.

@author Nishan Dananjaya
@date 2025-02-13
@version 1.0
```

6.3.2 Function Documentation

6.3.2.1 control_device()

```
control.control_device (
              url,
              client_id,
              secret,
              access_token,
              device_id,
              switch,
              command)
```

```
@brief Send a command to control a Tuya device.

@details Sends a control command to a specific Tuya device through the IoT API.
         The function handles authentication, request signing, and command execution.
         It supports boolean control commands (true/false) for device switches.

@param url str The base URL of the Tuya API
@param client_id str The client ID (Access ID) provided by Tuya
@param secret str The client secret (Access Key) provided by Tuya
@param access_token str The access token for API authentication
@param device_id str The ID of the device to control
@param switch str The switch identifier (e.g., "switch_1")
@param command str The command to send ("true" or "false")

@return dict|None Returns the JSON response from the API if successful,
                 None if the request fails

@exception requests.exceptions.RequestException Raised when the API request fails

@note Debug information including URL, timestamp, nonce, body, signature,
      and response details are printed to the console
```

6.3.2.2 generate_signature()

```
control.generate_signature (
              client_id,
              secret,
              access_token,
              t,
              nonce,
              url,
              method = "POST",
              body = "")
```

```
@brief Generate the HMAC-SHA256 signature for Tuya API requests.

@details Creates a signature for authenticating requests to the Tuya API using
         HMAC-SHA256. The signature is generated by combining various request
         parameters and creating a hash of the body content.

@param client_id str The client ID (Access ID) provided by Tuya
@param secret str The client secret (Access Key) provided by Tuya
@param access_token str The access token for API authentication
@param t str Timestamp in milliseconds
@param nonce str A unique identifier for the request
@param url str The API endpoint URL
@param method str The HTTP method (default: "POST")
@param body str The request body as a JSON string (default: "")

@return str The generated HMAC-SHA256 signature in uppercase hexadecimal format

@note The signature is crucial for API authentication and must be included
      in the request headers
```

6.3.3 Variable Documentation

6.3.3.1 ACCESS_TOKEN

```
control.ACCESS_TOKEN = os.getenv("ACCESS_TOKEN")
```

6.3.3.2 API_URL

```
control.API_URL = os.getenv("TUYA_BASE_URL")
```

6.3.3.3 CLIENT_ID

```
control.CLIENT_ID = os.getenv("TUYA_ACCESS_ID")
```

6.3.3.4 command

```
control.command = input("Enter the command (e.g., true/false):  ")
```

6.3.3.5 DEVICE_ID

```
control.DEVICE_ID = os.getenv("DEVICE_ID")
```

6.3.3.6 dot_env_path

```
control.dot_env_path = os.path.join(os.path.dirname(__file__), '..', '.env')
```

6.3.3.7 result

```
control.result = control_device(API_URL, CLIENT_ID, SECRET, ACCESS_TOKEN, DEVICE_ID, switch,
command)
```

6.3.3.8 SECRET

```
control.SECRET = os.getenv("TUYA_ACCESS_KEY")
```

6.3.3.9 switch

```
control.switch = input("Enter what to control (e.g., switch_1):  ")
```

6.4 get_devices_list Namespace Reference

Classes

- class Cloud

Variables

- dotenv_path = os.path.join(os.path.dirname(__file__), '..', '.env')
- CLIENT_ID = os.getenv("TUYA_ACCESS_ID")
- ACCESS_KEY = os.getenv("TUYA_ACCESS_KEY")
- devices = Cloud(apiKey=CLIENT_ID, apiSecret=ACCESS_KEY).getdevices()

6.4.1 Detailed Description

```
@file get_devices_list.py
@brief Returns the list of devices with only name and ID
@details This script retrieves the list of devices from the Tuya Cloud API and returns the device names and ID
         The script reads the Tuya Cloud API credentials from the .env file in the project root directory.
         The script can be run directly to print the device list to the console.
@author Nishan Dananjaya
@date 2025-02-13
@version 1.0
```

6.4.2 Variable Documentation

6.4.2.1 ACCESS_KEY

```
get_devices_list.ACCESS_KEY = os.getenv("TUYA_ACCESS_KEY")
```

6.4.2.2 CLIENT_ID

```
get_devices_list.CLIENT_ID = os.getenv("TUYA_ACCESS_ID")
```

6.4.2.3 devices

```
get_devices_list.devices = Cloud(apiKey=CLIENT_ID, apiSecret=ACCESS_KEY).getdevices()
```

6.4.2.4 dotenv_path

```
get_devices_list.dotenv_path = os.path.join(os.path.dirname(__file__), '..', '.env')
```

6.5 get_functions Namespace Reference

Functions

- generate_signature (client_id, secret, access_token, t, nonce, url, method="GET", body="")
- get_device_functions (url, client_id, secret, access_token, device_id)

Variables

- dotenv_path = os.path.join(os.path.dirname(__file__), '..', '.env')
- API_URL = os.getenv("TUYA_BASE_URL")
- CLIENT_ID = os.getenv("TUYA_ACCESS_ID")
- SECRET = os.getenv("TUYA_ACCESS_KEY")
- ACCESS_TOKEN = os.getenv("ACCESS_TOKEN")
- DEVICE_ID = os.getenv("DEVICE_ID")
- result = get_device_functions(API_URL, CLIENT_ID, SECRET, ACCESS_TOKEN, DEVICE_ID)

6.5.1 Detailed Description

```
@file get_functions.py
@brief A Python script to retrieve device functions from the Tuya API.
@details This script demonstrates how to retrieve device functions from the Tuya API using HMAC-SHA256 authent

@author Nishan Dananjaya
@date 2025-02-13
@version 1.0
```

6.5.2 Function Documentation

6.5.2.1 generate_signature()

```
get_functions.generate_signature (
              client_id,
              secret,
              access_token,
              t,
              nonce,
              url,
              method = "GET",
              body = "")
```

```
@brief Generate the HMAC-SHA256 signature for Tuya API requests.

@details This function creates a signature using HMAC-SHA256 algorithm required for
         authenticating requests to the Tuya API. The signature is generated based on
         the concatenation of various parameters and the request body hash.

@param client_id: str The client ID (Access ID) provided by Tuya
@param secret: str The client secret (Access Key) provided by Tuya
@param access_token: str The access token for API authentication
@param t: str Timestamp in milliseconds
@param nonce: str A unique identifier for the request
@param url: str The API endpoint URL
@param method: str The HTTP method (default: "GET")
@param body: str The request body (default: "")

@return str The generated HMAC-SHA256 signature in uppercase hexadecimal format
```

6.5.2.2 get_device_functions()

```
get_functions.get_device_functions (
              url,
              client_id,
              secret,
              access_token,
              device_id)
```

```
@brief Retrieve the device functions from Tuya API.

@details This function makes an authenticated GET request to the Tuya API
         to retrieve the functions available for a specific device. It handles
```

```
        the generation of required authentication parameters and headers.

@param url: str The base URL of the Tuya API
@param client_id: str The client ID (Access ID) provided by Tuya
@param secret: str The client secret (Access Key) provided by Tuya
@param access_token: str The access token for API authentication
@param device_id: str The ID of the device to query

@return dict|None Returns the JSON response from the API if successful, None if an error occurs

@exception requests.exceptions.RequestException Raised when the API request fails
```

6.5.3 Variable Documentation

6.5.3.1 ACCESS_TOKEN

```
get_functions.ACCESS_TOKEN = os.getenv("ACCESS_TOKEN")
```

6.5.3.2 API_URL

```
get_functions.API_URL = os.getenv("TUYA_BASE_URL")
```

6.5.3.3 CLIENT_ID

```
get_functions.CLIENT_ID = os.getenv("TUYA_ACCESS_ID")
```

6.5.3.4 DEVICE_ID

```
get_functions.DEVICE_ID = os.getenv("DEVICE_ID")
```

6.5.3.5 dotenv_path

```
get_functions.dotenv_path = os.path.join(os.path.dirname(__file__), '..', '.env')
```

6.5.3.6 result

```
get_functions.result = get_device_functions(API_URL, CLIENT_ID, SECRET, ACCESS_TOKEN, DEVICE_ID)
```

6.5.3.7 SECRET

```
get_functions.SECRET = os.getenv("TUYA_ACCESS_KEY")
```

6.6 gui Namespace Reference

Classes

- class ModernTuyaTokenGenerator

Functions

- main ()

6.6.1 Function Documentation

6.6.1.1 main()

```
gui.main ()
```

6.7 main Namespace Reference

Functions

- main ()

6.7.1 Detailed Description

```
@file main.py
@brief script for Tuya API authentication management.
@details This script demonstrates the usage of the AuthManager class for handling
         Tuya API authentication, including token management and signature generation.

@author Nishan Dananjaya
@date 2025-02-13
@version 1.0
```

6.7.2 Function Documentation

6.7.2.1 main()

```
main.main ()

@brief Main function to demonstrate AuthManager functionality.

@details This function demonstrates various authentication-related operations:
         - Retrieving access and refresh tokens
         - Checking token expiration
         - Getting authentication signatures and timestamps
         - Converting expiry times to human-readable format

The function creates an instance of AuthManager and showcases its core
authentication management capabilities.

@exception Exception Catches and prints any exceptions that occur during
                    authentication operations

Example usage:
@code
    if __name__ == "__main__":
        main()
@endcode
```

Chapter 7

Class Documentation

7.1 auth.AuthManager Class Reference

Public Member Functions

- __init__ (self)
- get_access_token (self)
- get_token (self)
- get_expiry_time (self)
- get_refresh_token (self)
- get_timestamp (self)
- get_signature (self)
- get_sign_method (self)
- get_nonce (self)
- is_token_expired (self)
- start_token_refresh_thread (self)
- stop_token_refresh_thread (self)

Public Attributes

- base_url = os.getenv("TUYA_BASE_URL")
- client_id = os.getenv("TUYA_ACCESS_ID")
- secret = os.getenv("TUYA_ACCESS_KEY")
- access_token = os.getenv("ACCESS_TOKEN")
- refresh_token = os.getenv("REFRESH_TOKEN")
- timestamp = os.getenv("TIMESTAMP")
- signature = os.getenv("SIGNATURE")
- sign_method = os.getenv("SIGN_METHOD")
- nonce = os.getenv("NONCE")
- token_expiry_time = float(os.getenv("TOKEN_EXPIRY_TIME", 0))
- str token_expiry_time = "/v1.0/token"
- bool _thread = True

Protected Member Functions

- _generate_signature (self, timestamp, nonce, string_to_sign)
- _save_to_env (self, key, value)
- _check_token_expiry (self)

Protected Attributes

- _stop_event = threading.Event()
- _thread = None

7.1.1 Detailed Description

```
@brief Authentication manager class for Tuya API.

@details Handles all authentication-related operations including token management,
         signature generation, and automatic token refresh in a background thread.
         The class manages environment variables and provides methods to access
         various authentication parameters.
```

7.1.2 Constructor & Destructor Documentation

7.1.2.1 __init__()

```
auth.AuthManager.__init__ (
              self)
```

```
@brief Initialize the AuthManager with environment variables.

@details Loads all required authentication parameters from environment variables
         and initializes threading components for automatic token refresh.
```

7.1.3 Member Function Documentation

7.1.3.1 _check_token_expiry()

```
auth.AuthManager._check_token_expiry (
              self)  [protected]
```

```
@brief Background task to check token expiration.

@details Runs in a separate thread to periodically check if the token
         is expired and automatically refreshes it when needed.
```

7.1.3.2 _generate_signature()

```
auth.AuthManager._generate_signature (
              self,
              timestamp,
              nonce,
              string_to_sign)  [protected]
```

```
@brief Generate HMAC-SHA256 signature for API authentication.

@param timestamp str Current timestamp in milliseconds
@param nonce str Unique identifier for the request
@param string_to_sign str The string to be signed

@return str The generated HMAC-SHA256 signature in uppercase hexadecimal format
```

7.1.3.3 _save_to_env()

```
auth.AuthManager._save_to_env (
              self,
              key,
              value)  [protected]
```

```
@brief Save a key-value pair to the .env file.

@param key str The environment variable key
@param value str The value to save
```

7.1.3.4 get_access_token()

```
auth.AuthManager.get_access_token (
              self)
```

```
@brief Get access token from Tuya API.

@details Retrieves a new access token if the current one is expired or
         returns the existing valid token. Handles the complete token
         acquisition process including signature generation and API call.

@return str The valid access token

@exception Exception Raised when token acquisition fails
@exception requests.exceptions.RequestException Raised on API request failure
```

7.1.3.5 get_expiry_time()

```
auth.AuthManager.get_expiry_time (
              self)
```

```
@brief Get the expiry time of the access token.

@return float Unix timestamp when the current token will expire
```

7.1.3.6 get_nonce()

```
auth.AuthManager.get_nonce (
              self)
```

```
@brief Get the nonce from the last API call.

@return str The nonce used in the most recent API call
```

7.1.3.7 get_refresh_token()

```
auth.AuthManager.get_refresh_token (
              self)
```

```
@brief Get the refresh token.

@return str The current refresh token
```

7.1.3.8 get_sign_method()

```
auth.AuthManager.get_sign_method (
              self)
```

```
@brief Get the sign method used in the last API call.

@return str The signing method (typically "HMAC-SHA256")
```

7.1.3.9 get_signature()

```
auth.AuthManager.get_signature (
              self)
```

```
@brief Get the signature from the last API call.

@return str The signature used in the most recent API call
```

7.1.3.10 get_timestamp()

```
auth.AuthManager.get_timestamp (
              self)
```

```
@brief Get the timestamp from the last API call.

@return str The timestamp used in the most recent API call
```

7.1.3.11 get_token()

```
auth.AuthManager.get_token (
              self)
```

```
@brief Public method to retrieve the access token.

@return str The current access token
```

7.1.3.12 is_token_expired()

```
auth.AuthManager.is_token_expired (
              self)
```

```
@brief Check if the current access token is expired.

@return bool True if the token is expired, False otherwise
```

7.1.3.13 start_token_refresh_thread()

```
auth.AuthManager.start_token_refresh_thread (
              self)
```

```
@brief Start the background token refresh thread.

@details Initializes and starts a daemon thread that monitors token
         expiration and handles automatic refresh.
```

7.1.3.14 stop_token_refresh_thread()

```
auth.AuthManager.stop_token_refresh_thread (
              self)
```

```
@brief Stop the background token refresh thread.

@details Signals the background thread to stop and waits for it to complete.
```

7.1.4 Member Data Documentation

7.1.4.1 _stop_event

```
auth.AuthManager._stop_event = threading.Event()  [protected]
```

7.1.4.2 _thread [1/2]

```
auth.AuthManager._thread = None  [protected]
```

7.1.4.3 _thread [2/2]

```
bool auth.AuthManager._thread = True
```

7.1.4.4 access_token

```
auth.AuthManager.access_token = os.getenv("ACCESS_TOKEN")
```

7.1.4.5 base_url

```
auth.AuthManager.base_url = os.getenv("TUYA_BASE_URL")
```

7.1.4.6 client_id

```
auth.AuthManager.client_id = os.getenv("TUYA_ACCESS_ID")
```

7.1.4.7 nonce

```
auth.AuthManager.nonce = os.getenv("NONCE")
```

7.1.4.8 refresh_token

```
auth.AuthManager.refresh_token = os.getenv("REFRESH_TOKEN")
```

7.1.4.9 secret

```
auth.AuthManager.secret = os.getenv("TUYA_ACCESS_KEY")
```

7.1.4.10 sign_method

```
auth.AuthManager.sign_method = os.getenv("SIGN_METHOD")
```

7.1.4.11 signature

```
auth.AuthManager.signature = os.getenv("SIGNATURE")
```

7.1.4.12 timestamp

```
auth.AuthManager.timestamp = os.getenv("TIMESTAMP")
```

7.1.4.13 token_expiry_time `[1/2]`

```
auth.AuthManager.token_expiry_time = float(os.getenv("TOKEN_EXPIRY_TIME", 0))
```

7.1.4.14 token_expiry_time `[2/2]`

```
str auth.AuthManager.token_expiry_time = "/v1.0/token"
```

The documentation for this class was generated from the following file:

- tuya_api/auth.py

7.2 authtest.AuthManager Class Reference

Public Member Functions

- __init__ (self)
- get_access_token (self)
- get_token (self)
- get_expiry_time (self)
- get_refresh_token (self)
- get_timestamp (self)
- get_signature (self)
- get_sign_method (self)
- get_nonce (self)
- is_token_expired (self)
- start_token_refresh_thread (self)
- stop_token_refresh_thread (self)

Public Attributes

- base_url = os.getenv("TUYA_BASE_URL")
- client_id = os.getenv("TUYA_ACCESS_ID")
- secret = os.getenv("TUYA_ACCESS_KEY")
- access_token = os.getenv("ACCESS_TOKEN")
- refresh_token = os.getenv("REFRESH_TOKEN")
- timestamp = os.getenv("TIMESTAMP")
- signature = os.getenv("SIGNATURE")
- sign_method = os.getenv("SIGN_METHOD")
- nonce = os.getenv("NONCE")
- token_expiry_time = float(os.getenv("TOKEN_EXPIRY_TIME", 0))
- str token_expiry_time = "/v1.0/token"
- bool _thread = True

Protected Member Functions

- _generate_signature (self, timestamp, nonce, string_to_sign)
- _save_to_env (self, key, value)
- _check_token_expiry (self)

Protected Attributes

- _stop_event = threading.Event()
- _thread = None

7.2.1 Constructor & Destructor Documentation

7.2.1.1 __init__()

```
authtest.AuthManager.__init__ (
              self)
```

7.2.2 Member Function Documentation

7.2.2.1 _check_token_expiry()

```
authtest.AuthManager._check_token_expiry (
              self)  [protected]
```

```
Periodically check if the token is expired and generate a new one if needed.
```

7.2.2.2 _generate_signature()

```
authtest.AuthManager._generate_signature (
              self,
              timestamp,
              nonce,
              string_to_sign)  [protected]
```

```
Generate HMAC-SHA256 signature.
```

7.2.2.3 _save_to_env()

```
authtest.AuthManager._save_to_env (
              self,
              key,
              value)  [protected]
```

```
Save a key-value pair to the .env file.
```

7.2.2.4 get_access_token()

```
authtest.AuthManager.get_access_token (
              self)
```

Get access token from Tuya API.

7.2.2.5 get_expiry_time()

```
authtest.AuthManager.get_expiry_time (
              self)
```

Returns the expiry time of the access token in seconds since epoch.

7.2.2.6 get_nonce()

```
authtest.AuthManager.get_nonce (
              self)
```

Returns the nonce used in the last API call.

7.2.2.7 get_refresh_token()

```
authtest.AuthManager.get_refresh_token (
              self)
```

Returns the refresh token.

7.2.2.8 get_sign_method()

```
authtest.AuthManager.get_sign_method (
              self)
```

Returns the sign method used in the last API call.

7.2.2.9 get_signature()

```
authtest.AuthManager.get_signature (
              self)
```

Returns the signature used in the last API call.

7.2.2.10 get_timestamp()

```
authtest.AuthManager.get_timestamp (
              self)
```

Returns the timestamp used in the last API call.

7.2.2.11 get_token()

```
authtest.AuthManager.get_token (
              self)
```

Public method to retrieve the access token.

7.2.2.12 is_token_expired()

```
authtest.AuthManager.is_token_expired (
              self)
```

```
Checks if the access token is expired.
```

7.2.2.13 start_token_refresh_thread()

```
authtest.AuthManager.start_token_refresh_thread (
              self)
```

```
Start the background thread to check and refresh the token.
```

7.2.2.14 stop_token_refresh_thread()

```
authtest.AuthManager.stop_token_refresh_thread (
              self)
```

```
Stop the background thread.
```

7.2.3 Member Data Documentation

7.2.3.1 _stop_event

```
authtest.AuthManager._stop_event = threading.Event()  [protected]
```

7.2.3.2 _thread [1/2]

```
authtest.AuthManager._thread = None  [protected]
```

7.2.3.3 _thread [2/2]

```
bool authtest.AuthManager._thread = True
```

7.2.3.4 access_token

```
authtest.AuthManager.access_token = os.getenv("ACCESS_TOKEN")
```

7.2.3.5 base_url

```
authtest.AuthManager.base_url = os.getenv("TUYA_BASE_URL")
```

7.2.3.6 client_id

```
authtest.AuthManager.client_id = os.getenv("TUYA_ACCESS_ID")
```

7.2.3.7 nonce

```
authtest.AuthManager.nonce = os.getenv("NONCE")
```

7.2.3.8 refresh_token

```
authtest.AuthManager.refresh_token = os.getenv("REFRESH_TOKEN")
```

7.2.3.9 secret

```
authtest.AuthManager.secret = os.getenv("TUYA_ACCESS_KEY")
```

7.2.3.10 sign_method

```
authtest.AuthManager.sign_method = os.getenv("SIGN_METHOD")
```

7.2.3.11 signature

```
authtest.AuthManager.signature = os.getenv("SIGNATURE")
```

7.2.3.12 timestamp

```
authtest.AuthManager.timestamp = os.getenv("TIMESTAMP")
```

7.2.3.13 token_expiry_time [1/2]

```
authtest.AuthManager.token_expiry_time = float(os.getenv("TOKEN_EXPIRY_TIME", 0))
```

7.2.3.14 token_expiry_time [2/2]

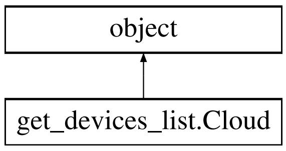
```
str authtest.AuthManager.token_expiry_time = "/v1.0/token"
```

The documentation for this class was generated from the following file:

- tuya_api/authtest.py

7.3 get_devices_list.Cloud Class Reference

Inheritance diagram for get_devices_list.Cloud:



Public Member Functions

- __init__ (self, apiKey=None, apiSecret=None, apiDeviceID=None, new_sign_algorithm=True, initial_↩ token=None, ∗∗extrakw)
- cloudrequest (self, url, action=None, post=None, query=None)
- getdevices (self)

Public Attributes

- str CONFIGFILE = 'tinytuya.json'
- apiKey = apiKey
- apiSecret = apiSecret
- apiDeviceID = apiDeviceID
- urlhost = os.getenv("TUYA_BASE_URL").replace('https://', '')
- uid = None
- token = os.getenv("ACCESS_TOKEN") if initial_token is None else initial_token
- error = None
- new_sign_algorithm = new_sign_algorithm
- int server_time_offset = 0
- bool use_old_device_list = True
- mappings = None
- tuple new_sign_algorithm
- str token = 'devices/%s' % deviceid

Protected Member Functions

- _tuyaplatform (self, uri, action='GET', post=None, ver='v1.0', recursive=False, query=None, content_↩ type=None)
- _getuid (self, deviceid=None)
- _update_device_list (self, result1, result2)
- _get_all_devices (self, uid=None, device_ids=None)

7.3.1 Detailed Description

```
A class to interact with the Tuya Cloud API.

This class provides methods to authenticate, send requests, and retrieve device information
from the Tuya Cloud platform.
```

7.3.2 Constructor & Destructor Documentation

7.3.2.1 __init__()

```
get_devices_list.Cloud.__init__ (
              self,
              apiKey = None,
              apiSecret = None,
              apiDeviceID = None,
              new_sign_algorithm = True,
              initial_token = None,
             ** extrakw)
```

```
Initialize the Cloud class with API credentials and configuration.

@param apiKey: The API key for Tuya Cloud.
@param apiSecret: The API secret for Tuya Cloud.
@param apiDeviceID: The device ID for Tuya Cloud.
@param new_sign_algorithm: Whether to use the new signing algorithm (default is True).
@param initial_token: An initial access token (optional).
@param extrakw: Additional keyword arguments.
```

7.3.3 Member Function Documentation

7.3.3.1 _get_all_devices()

```
get_devices_list.Cloud._get_all_devices (
              self,
              uid = None,
              device_ids = None)  [protected]
```

```
Retrieve all devices associated with a user or a list of device IDs.

@param uid: The user ID to retrieve devices for (optional).
@param device_ids: A list of device IDs to retrieve (optional).
@return: A dictionary containing the list of devices and metadata.
```

7.3.3.2 _getuid()

```
get_devices_list.Cloud._getuid (
              self,
              deviceid = None)  [protected]
```

```
Get user ID (UID) for a specific device.

@param deviceid: The device ID to get the UID for.
@return: The UID of the device.
```

7.3.3.3 _tuyaplatform()

```
get_devices_list.Cloud._tuyaplatform (
              self,
              uri,
              action = 'GET',
              post = None,
              ver = 'v1.0',
              recursive = False,
              query = None,
              content_type = None)  [protected]
```

```
Handle GET and POST requests to Tuya Cloud.

@param uri: The URI to request.
@param action: The HTTP action (GET, POST, PUT, DELETE).
@param post: The POST data (optional).
@param ver: The API version (default is ’v1.0’).
@param recursive: Whether to retry the request if the token is invalid (default is False).
@param query: The query parameters (optional).
@param content_type: The content type for the request (optional).
@return: The response from the Tuya Cloud API.
```

7.3.3.4 _update_device_list()

```
get_devices_list.Cloud._update_device_list (
              self,
              result1,
              result2)  [protected]
```

```
Merge two device lists.

@param result1: The first device list.
@param result2: The second device list.
```

7.3.3.5 cloudrequest()

```
get_devices_list.Cloud.cloudrequest (
              self,
              url,
              action = None,
              post = None,
              query = None)
```

```
Make a generic cloud request and return the results.

@param url: The URL to fetch, i.e. "/v1.0/devices/0011223344556677/logs".
@param action: The HTTP action (GET, POST, DELETE, or PUT). Defaults to GET, unless POST data is supplied.
@param post: The POST body data. Will be fed into json.dumps() before posting.
@param query: A dict containing query string key/value pairs.
@return: The response from the Tuya Cloud API.
```

7.3.3.6 getdevices()

```
get_devices_list.Cloud.getdevices (
              self)
```

```
Return a list of devices with only name and ID.

@return: A list of dictionaries containing device names and IDs.
```

7.3.4 Member Data Documentation

7.3.4.1 apiDeviceID

```
get_devices_list.Cloud.apiDeviceID = apiDeviceID
```

7.3.4.2 apiKey

```
get_devices_list.Cloud.apiKey = apiKey
```

7.3.4.3 apiSecret

```
get_devices_list.Cloud.apiSecret = apiSecret
```

7.3.4.4 CONFIGFILE

```
get_devices_list.Cloud.CONFIGFILE = 'tinytuya.json'
```

7.3.4.5 error

```
get_devices_list.Cloud.error = None
```

7.3.4.6 mappings

```
get_devices_list.Cloud.mappings = None
```

7.3.4.7 new_sign_algorithm [1/2]

```
get_devices_list.Cloud.new_sign_algorithm = new_sign_algorithm
```

7.3.4.8 new_sign_algorithm [2/2]

```
tuple get_devices_list.Cloud.new_sign_algorithm
```

Initial value:

```
=  ('%s\n' % action +                                          # HTTPMethod
                hashlib.sha256(bytes((body or "").encode('utf-8'))).hexdigest() + '\n' + # Content-SHA256
                ''.join(['%s:%s\n'%(key, headers[key])                          # Headers
                            for key in headers.get("Signature-Headers", "").split(":")
                            if key in headers]) + '\n' +
                '/' + sign_url.split('//', 1)[-1].split('/', 1)[-1])
```

7.3.4.9 server_time_offset

```
int get_devices_list.Cloud.server_time_offset = 0
```

7.3.4.10 token [1/2]

```
str get_devices_list.Cloud.token = os.getenv("ACCESS_TOKEN") if initial_token is None else
initial_token
```

7.3.4.11 token [2/2]

```
str get_devices_list.Cloud.token = 'devices/%s' % deviceid
```

7.3.4.12 uid

```
get_devices_list.Cloud.uid = None
```

7.3.4.13 urlhost

```
get_devices_list.Cloud.urlhost = os.getenv("TUYA_BASE_URL").replace('https://', '')
```

7.3.4.14 use_old_device_list

```
bool get_devices_list.Cloud.use_old_device_list = True
```

The documentation for this class was generated from the following file:

- config/get_devices_list.py

7.4 gui.ModernTuyaTokenGenerator Class Reference

Public Member Functions

- __init__ (self, root)
- automatically_generate_token (self)
- setup_main_layout (self)
- create_credentials_tab (self)
- fetch_and_select_device (self)
- get_device_list (self)
- device_selection_dialog (self, devices)
- save_selected_device (self, listbox, devices, dialog)
- toggle_entry_visibility (self, entry)
- create_config_tab (self)
- create_token_history_tab (self)
- create_help_tab (self)
- save_to_env (self)
- run_selected_file (self)
- display_control_buttons (self)
- show_control_panel (self)
- hide_control_panel (self)
- toggle_switch (self, switch)
- add_new_config (self)
- open_tuya_docs (self)

Public Attributes

- root = root
- dict entry_widgets = {}
- bool control_panel_visible = False
- control_panel = None
- list_frame = None
- tuple primary_font = ('Inter', 10)
- tuple header_font = ('Inter', 16, 'bold')
- auth_manager = AuthManager()
- main_container = ttk.Frame(self.root, padding=(30, 30, 30, 30))
- notebook = ttk.Notebook(self.main_container, style='primary.TNotebook')
- dict entry_vars = {}
- output_text
- bool list_frame = True)
- file_listbox
- config_output_text
- dict switch_states
- tuple control_panel_visible = (20, 0))

Static Public Attributes

- font
- padding
- background
- foreground
- button_text = widget.cget("text").lower()
- text
- style
- API_URL = os.getenv("TUYA_BASE_URL")
- CLIENT_ID = os.getenv("TUYA_ACCESS_ID")
- SECRET = os.getenv("TUYA_ACCESS_KEY")
- ACCESS_TOKEN = os.getenv("ACCESS_TOKEN")
- DEVICE_ID = os.getenv("DEVICE_ID")
- result = control_device(API_URL, CLIENT_ID, SECRET, ACCESS_TOKEN, DEVICE_ID, switch, command)

7.4.1 Constructor & Destructor Documentation

7.4.1.1 __init__()

```
gui.ModernTuyaTokenGenerator.__init__ (
              self,
              root)
```

7.4.2 Member Function Documentation

7.4.2.1 add_new_config()

```
gui.ModernTuyaTokenGenerator.add_new_config (
              self)
```

```
Add a new configuration file.
```

7.4.2.2 automatically_generate_token()

```
gui.ModernTuyaTokenGenerator.automatically_generate_token (
              self)
```

```
Automatically generate the access token when the GUI starts.
```

7.4.2.3 create_config_tab()

```
gui.ModernTuyaTokenGenerator.create_config_tab (
              self)
```

7.4.2.4 create_credentials_tab()

```
gui.ModernTuyaTokenGenerator.create_credentials_tab (
              self)
```

7.4.2.5 create_help_tab()

```
gui.ModernTuyaTokenGenerator.create_help_tab (
              self)
```

7.4.2.6 create_token_history_tab()

```
gui.ModernTuyaTokenGenerator.create_token_history_tab (
              self)
```

7.4.2.7 device_selection_dialog()

```
gui.ModernTuyaTokenGenerator.device_selection_dialog (
              self,
              devices)
```

Display a dialog for the user to select a device.

7.4.2.8 display_control_buttons()

```
gui.ModernTuyaTokenGenerator.display_control_buttons (
              self)
```

Display buttons to control the device in a sliding panel.

7.4.2.9 fetch_and_select_device()

```
gui.ModernTuyaTokenGenerator.fetch_and_select_device (
              self)
```

Fetch the list of devices and prompt the user to select one.

7.4.2.10 get_device_list()

```
gui.ModernTuyaTokenGenerator.get_device_list (
              self)
```

Fetch the list of devices using the get_devices_list.py script.

7.4.2.11 hide_control_panel()

```
gui.ModernTuyaTokenGenerator.hide_control_panel (
              self)
```

Hide the control panel.

7.4.2.12 open_tuya_docs()

```
gui.ModernTuyaTokenGenerator.open_tuya_docs (
              self)
```

Open Tuya developer documentation in the default web browser.

7.4.2.13 run_selected_file()

```
gui.ModernTuyaTokenGenerator.run_selected_file (
              self)
```

Run the selected configuration file or display control buttons for control.py.

7.4.2.14 save_selected_device()

```
gui.ModernTuyaTokenGenerator.save_selected_device (
              self,
              listbox,
              devices,
              dialog)
```

Save the selected device ID to the .env file and close the dialog.

7.4.2.15 save_to_env()

```
gui.ModernTuyaTokenGenerator.save_to_env (
              self)
```

```
Save input values to the .env file.
```

7.4.2.16 setup_main_layout()

```
gui.ModernTuyaTokenGenerator.setup_main_layout (
              self)
```

7.4.2.17 show_control_panel()

```
gui.ModernTuyaTokenGenerator.show_control_panel (
              self)
```

```
Show the control panel with a sliding effect.
```

7.4.2.18 toggle_entry_visibility()

```
gui.ModernTuyaTokenGenerator.toggle_entry_visibility (
              self,
              entry)
```

```
Toggle the visibility of the text in the entry widget.
```

7.4.2.19 toggle_switch()

```
gui.ModernTuyaTokenGenerator.toggle_switch (
              self,
              switch)
```

```
Toggle the state of a switch and send the command to the device.
```

7.4.3 Member Data Documentation

7.4.3.1 ACCESS_TOKEN

```
gui.ModernTuyaTokenGenerator.ACCESS_TOKEN = os.getenv("ACCESS_TOKEN")  [static]
```

7.4.3.2 API_URL

```
gui.ModernTuyaTokenGenerator.API_URL = os.getenv("TUYA_BASE_URL")  [static]
```

7.4.3.3 auth_manager

```
gui.ModernTuyaTokenGenerator.auth_manager = AuthManager()
```

7.4.3.4 background

```
gui.ModernTuyaTokenGenerator.background  [static]
```

7.4.3.5 button_text

```
gui.ModernTuyaTokenGenerator.button_text = widget.cget("text").lower()  [static]
```

7.4.3.6 CLIENT_ID

```
gui.ModernTuyaTokenGenerator.CLIENT_ID = os.getenv("TUYA_ACCESS_ID")  [static]
```

7.4.3.7 config_output_text

`gui.ModernTuyaTokenGenerator.config_output_text`

Initial value:

```
=  scrolledtext.ScrolledText(
            output_frame,
            wrap=tk.WORD,
            height=10,
            font=('Consolas', 10)
        )
```

7.4.3.8 control_panel

`gui.ModernTuyaTokenGenerator.control_panel = None`

7.4.3.9 control_panel_visible [1/2]

`bool gui.ModernTuyaTokenGenerator.control_panel_visible = False`

7.4.3.10 control_panel_visible [2/2]

`tuple gui.ModernTuyaTokenGenerator.control_panel_visible = (20, 0))`

7.4.3.11 DEVICE_ID

`gui.ModernTuyaTokenGenerator.DEVICE_ID = os.getenv("DEVICE_ID")  [static]`

7.4.3.12 entry_vars

`dict gui.ModernTuyaTokenGenerator.entry_vars = {}`

7.4.3.13 entry_widgets

`dict gui.ModernTuyaTokenGenerator.entry_widgets = {}`

7.4.3.14 file_listbox

`gui.ModernTuyaTokenGenerator.file_listbox`

Initial value:

```
=  tk.Listbox(
            self.list_frame,
            selectmode=tk.SINGLE,
            font=('Consolas', 10)
        )
```

7.4.3.15 font

`gui.ModernTuyaTokenGenerator.font  [static]`

7.4.3.16 foreground

`gui.ModernTuyaTokenGenerator.foreground  [static]`

7.4.3.17 header_font

`tuple gui.ModernTuyaTokenGenerator.header_font = ('Inter', 16, 'bold')`

7.4.3.18 list_frame [1/2]

`gui.ModernTuyaTokenGenerator.list_frame = None`

7.4.3.19 list_frame [2/2]

`bool gui.ModernTuyaTokenGenerator.list_frame = True)`

7.4.3.20 main_container

```
gui.ModernTuyaTokenGenerator.main_container = ttk.Frame(self.root, padding=(30, 30, 30, 30))
```

7.4.3.21 notebook

```
gui.ModernTuyaTokenGenerator.notebook = ttk.Notebook(self.main_container, style='primary.↩
TNotebook')
```

7.4.3.22 output_text

```
gui.ModernTuyaTokenGenerator.output_text
```

Initial value:

```
=  scrolledtext.ScrolledText(
            output_frame,
            wrap=tk.WORD,
            height=10,
            font=('Consolas', 10),
            background='#2c3e50',  # Dark background for code
            foreground='#ecf0f1'   # Light text for readability
        )
```

7.4.3.23 padding

```
gui.ModernTuyaTokenGenerator.padding  [static]
```

7.4.3.24 primary_font

```
tuple gui.ModernTuyaTokenGenerator.primary_font = ('Inter', 10)
```

7.4.3.25 result

```
gui.ModernTuyaTokenGenerator.result = control_device(API_URL, CLIENT_ID, SECRET, ACCESS_TOKEN,
DEVICE_ID, switch, command)  [static]
```

7.4.3.26 root

```
gui.ModernTuyaTokenGenerator.root = root
```

7.4.3.27 SECRET

```
gui.ModernTuyaTokenGenerator.SECRET = os.getenv("TUYA_ACCESS_KEY")  [static]
```

7.4.3.28 style

```
gui.ModernTuyaTokenGenerator.style  [static]
```

7.4.3.29 switch_states

```
dict gui.ModernTuyaTokenGenerator.switch_states
```

Initial value:

```
=  {
            "switch_1": False,
            "switch_2": False,
            "switch_3": False,
            "switch_4": False
        }
```

7.4.3.30 text

```
gui.ModernTuyaTokenGenerator.text  [static]
```

The documentation for this class was generated from the following file:

- gui.py

Chapter 8

File Documentation

8.1 config/control.py File Reference

Namespaces

- namespace control

Functions

- control.generate_signature (client_id, secret, access_token, t, nonce, url, method="POST", body="")
- control.control_device (url, client_id, secret, access_token, device_id, switch, command)

Variables

- control.dot_env_path = os.path.join(os.path.dirname(__file__), '..', '.env')
- control.API_URL = os.getenv("TUYA_BASE_URL")
- control.CLIENT_ID = os.getenv("TUYA_ACCESS_ID")
- control.SECRET = os.getenv("TUYA_ACCESS_KEY")
- control.ACCESS_TOKEN = os.getenv("ACCESS_TOKEN")
- control.DEVICE_ID = os.getenv("DEVICE_ID")
- control.switch = input("Enter what to control (e.g., switch_1): ")
- control.command = input("Enter the command (e.g., true/false): ")
- control.result = control_device(API_URL, CLIENT_ID, SECRET, ACCESS_TOKEN, DEVICE_ID, switch, command)

8.2 config/get_devices_list.py File Reference

Classes

- class get_devices_list.Cloud

Namespaces

- namespace get_devices_list

Variables

- get_devices_list.dotenv_path = os.path.join(os.path.dirname(__file__), '..', '.env')
- get_devices_list.CLIENT_ID = os.getenv("TUYA_ACCESS_ID")
- get_devices_list.ACCESS_KEY = os.getenv("TUYA_ACCESS_KEY")
- get_devices_list.devices = Cloud(apiKey=CLIENT_ID, apiSecret=ACCESS_KEY).getdevices()

8.3 config/get_functions.py File Reference

Namespaces

- namespace get_functions

Functions

- get_functions.generate_signature (client_id, secret, access_token, t, nonce, url, method="GET", body="")
- get_functions.get_device_functions (url, client_id, secret, access_token, device_id)

Variables

- get_functions.dotenv_path = os.path.join(os.path.dirname(__file__), '..', '.env')
- get_functions.API_URL = os.getenv("TUYA_BASE_URL")
- get_functions.CLIENT_ID = os.getenv("TUYA_ACCESS_ID")
- get_functions.SECRET = os.getenv("TUYA_ACCESS_KEY")
- get_functions.ACCESS_TOKEN = os.getenv("ACCESS_TOKEN")
- get_functions.DEVICE_ID = os.getenv("DEVICE_ID")
- get_functions.result = get_device_functions(API_URL, CLIENT_ID, SECRET, ACCESS_TOKEN, DEVICE_ID)

8.4 gui.py File Reference

Classes

- class gui.ModernTuyaTokenGenerator

Namespaces

- namespace gui

Functions

- gui.main ()

8.5 main.py File Reference

Namespaces

- namespace main

Functions

- main.main ()

8.6 README.md File Reference

8.7 tuya_api/auth.py File Reference

Classes

- class auth.AuthManager

Namespaces

- namespace auth

Variables

- auth.auth_manager = AuthManager()

8.8 tuya_api/authtest.py File Reference

Classes

- class authtest.AuthManager

Namespaces

- namespace authtest

Variables

- authtest.auth_manager = AuthManager()