**TABLE OF CONTENTS**

[1. Face Validation API (TNI) 2](#_Toc145029890)

[2. Face Registration API (TNI) 4](#_Toc145029891)

[3. Face Registration API with multiple shapes (TNI) 6](#_Toc145029892)

[4. Face Authentication API (TNI) 9](#_Toc145029893)

[5. Registered Face Check API (TNI) 11](#_Toc145029894)

[6. Face Lookup API (ITD) 12](#_Toc145029895)

[7. Valid Face ID List API (TNI) 13](#_Toc145029896)

[8. Procursion Detection API (ITD) 14](#_Toc145029897)

[9. Intrusion Detection API (ITD) 14](#_Toc145029898)

[10. System Health API (ITD) 15](#_Toc145029899)

[11. API Read System Configuration (TNI) 15](#_Toc145029900)

[12. System Configuration Recording API (TNI) 16](#_Toc145029901)

[13. Camera Number Configuration API (TNI) 17](#_Toc145029902)

[14. Camera List API (TNI) 18](#_Toc145029903)

[15. Camera Configuration API (TNI) 19](#_Toc145029904)

[16. System Restart API (TNI) 21](#_Toc145029905)

[17. Trace List API (TNI) 22](#_Toc145029906)

[18. Tracking by ID (TNI) API 24](#_Toc145029907)

[19. Image Tracing API (TNI) 27](#_Toc145029908)

[20. Person Detection API (TNI) 30](#_Toc145029909)

[21. Intruder Tracking API (ITD) 32](#_Toc145029910)

# Face Validation API (TNI)

* **Purpose:**The client calls the API to check the validity of a person's face photo.
* Support: 1 photo for 1 person.

**Request content:**

* Timesend:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No** | **Key** | **Datatype** | **Descriptions** | **Example** |
| first | TimeSend | string | Time to send data to the server | yyyy-MM-  ddTHH:mm:ss.fff |

* Item:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No** | **Key** | **Datatype** | **Descriptions** | **Example** |
| first | ImageData | string/null | The image contains a base64 encoded face from a .jpg . file |  |

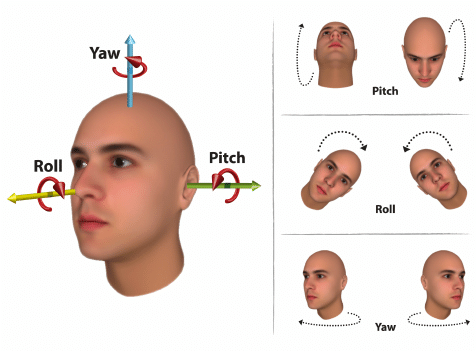
* Request example:
* URL:<http://localhost:8004/face_validation>
* method: POST
* json:

|  |
| --- |
| {  "TimeSend": "2023-01-06T09:10:49,182",  "Item": {  "ImageData": "a91h3jyjksi1"  }  } |

**Response content: Only respond when the test is done.**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No** | **Key** | **Datatype** | **Descriptions** | **Example** |
| first | StatusCode | int | Return code:  0: Failed  1: Success |  |
| 2 | Message | string | Detailed description of the status code |  |
| 3 | IsValid | int | Determine if the snapshot is valid: 0/1, 0 is invalid, 1 is valid. |  |

* Enum definition:
* STATUS\_CODE:
  + FAIL = 0
  + SUCCESS = 1
* MESSAGE:
  + WRONG\_FORMAT = "Wrong format request fields"
  + CAPTURE\_INCORRECT = "Image is captured incorrectly"
  + CAPTURE\_CORRECT = "Image is captured correctly"
  + INTERNAL\_SERVER\_ERROR = "Internal Server Error"
* Check the presence (existence), data type (type), value binding (value) of the defined fields, if not satisfied, return:
* StatusCode = STATUS\_CODE.FAIL
* Message = MESSAGE.WRONG\_FORMAT
* IsValid = -1
* If the image is not properly captured, return:
* StatusCode = STATUS\_CODE.FAIL
* Message = MESSAGE.CAPTURE\_INCORRECT
* IsValid = 0
* If the image is properly captured, return:
* StatusCode = STATUS\_CODE.SUCCESS
* Message = MESSAGE.CAPTURE\_CORRECT
* IsValid = 1
* If an unknown error occurs during processing, return:
* StatusCode = STATUS\_CODE.FAIL
* Message = MESSAGE.INTERNAL\_SERVER\_ERROR
* IsValid = -1
* **Criteria of face photo for registration:**
* The face must be the main subject in the frame, with a minimum horizontal and vertical dimension of 32 pixels. The software automatically checks and informs if not satisfied.
* Head rotation constraints: roll +-20 degrees, pitch +-30 degrees, yaw +-20 degrees. The software automatically checks and informs if not satisfied.



* The face is not wearing a mask, not obscured by any objects. The software does not automatically check.
* Clear glasses are allowed, dark glasses are not allowed. The software does not automatically check.
* Lighting conditions are sufficient, and clear, with no dimming, no blurring, or no glare. The software does not automatically check.

# Face Registration API (TNI)

* **Purpose:** Client calls API to register/update/delete a person's face.
* Support: 1 photo for 1 person.

**Request content:**

* Timesend:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No** | **Key** | **Datatype** | **Descriptions** | **Example** |
| first | TimeSend | string | Time to send data to the server | yyyy-MM-  ddTHH:mm:ss.fff |

* Item:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No** | **Key** | **Datatype** | **Descriptions** | **Example** |
| first | ImageData | string/null | The image contains a base64 encoded face from a .jpg . file |  |
| 2 | FaceID | int  -1/>=0 | Face ID in DB (unique) | BILLION |
| 3 | OpCode | int  [0, 1, 2] | Operation code:  0: Create new  1: Update the registered face (overwrite)  2: Delete |  |

* Combined Constraints:
* In the case of OpCode=0, FaceID=-1, ImageData has a value.
* In the case of OpCode=1, FaceID has a value, ImageData has a value.
* In the case of OpCode=2, FaceID has a value, ImageData =null.
* Request example:
* URL:<http://localhost:8004/face_registration>
* method: POST
* json:

|  |
| --- |
| {  "TimeSend": "2023-01-06T09:10:49,182",  "Item": {  "ImageData": "a91h3jyjksi1",  "FaceID": -first,  "OpCode": 0  }  } |

**Response content: Only respond when the data has been saved.**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No** | **Key** | **Datatype** | **Descriptions** | **Example** |
| first | StatusCode | int | Return code:  0: Failed  1: Success |  |
| 2 | Message | string | Detailed description of the status code |  |
| 3 | FaceID | int | Face ID in DB (unique) |  |

* Enum definition:
* STATUS\_CODE:
  + FAIL = 0
  + SUCCESS = 1
* MESSAGE:
  + WRONG\_FORMAT = "Wrong format request fields"
  + CAPTURE\_INCORRECT = "Image is captured incorrectly"
  + REMOVE\_SUCCESS = "Remove face successfully"
  + REMOVE\_FAIL = "Remove face failed, FaceID may not exist"
  + ADD\_NEW\_SUCCESS = "Add new face successfully"
  + UPDATE\_SUCCESS = "Update face successfully"
  + UPDATE\_FAIL = "Update face failed, FaceID may not exist"
  + INTERNAL\_SERVER\_ERROR = "Internal Server Error"
* Check the presence (existence), data type (type), value binding (value) of the defined fields, and[association constraints](#gpotq2ss1m9d), if not satisfied, returns:
* StatusCode = STATUS\_CODE.FAIL
* Message = MESSAGE.WRONG\_FORMAT
* FaceID = -1
* If OpCode=2 and FaceID does not exist, return:
* StatusCode = STATUS\_CODE.FAIL
* Message = MESSAGE.REMOVE\_FAIL
* FaceID = -1
* If OpCode=2 and FaceID exist, delete the record that has the same FaceID as in the request, returning:
* StatusCode = STATUS\_CODE.SUCCESS
* Message = MESSAGE.REMOVE\_SUCCESS
* FaceID = FaceID
* If OpCode=0 or 1, checks if the image was taken correctly, if not, returns:
* StatusCode = STATUS\_CODE.FAIL
* Message = MESSAGE.CAPTURE\_INCORRECT
* FaceID = -1
* If OpCode=0 and the image is correct, allocate FaceID, write a new record to the DB, return:
* StatusCode = STATUS\_CODE.SUCCESS
* Message = MESSAGE.ADD\_NEW\_SUCCESS
* FaceID = FaceID

**Note**, the new FaceID always counts up and does not reuse the deleted FaceIDs, and has the formula: FaceID=max(FaceIDs) + 1.

* If OpCode=1 and the image is correct and FaceID does not exist, return:
* StatusCode = STATUS\_CODE.FAIL
* Message = MESSAGE.UPDATE\_FAIL
* FaceID = -1
* If OpCode=1 and the image is correct and FaceID exists, update the record, returning:
* StatusCode = STATUS\_CODE.SUCCESS
* Message = MESSAGE.UPDATE\_SUCCESS
* FaceID = FaceID
* If an unknown error occurs during processing, return:
* StatusCode = STATUS\_CODE.FAIL
* Message = MESSAGE.INTERNAL\_SERVER\_ERROR
* FaceID = -1

# Face Registration API with multiple shapes (TNI)

* **Purpose:**Client calls API to register/update/delete a person's face.
* Support: Multiple photos for 1 person, up to 5 photos.

**Request content:**

* Timesend:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No** | **Key** | **Datatype** | **Descriptions** | **Example** |
| first | TimeSend | string | Time to send data to the server | yyyy-MM-  ddTHH:mm:ss.fff |

* Item:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No** | **Key** | **Datatype** | **Descriptions** | **Example** |
| first | ImageData | list of strings | - List contains many strings, each string corresponds to 1 image  - Each image contains a base64 encoded face from the .jpg . file | [  “a91h3jyjksi1”,  “u2n1b3ud91k”,  “u2n13j1d91k”,  “02k1b3ud91k”,  “u2n1b3u29k1”,  ] |
| 2 | FaceID | int  -1/>=0 | Face ID in DB (unique) |  |
| 3 | OpCode | int  [0, 1, 2] | Operation code:  0: Create new  1: Update the registered face (overwrite)  2: Delete |  |

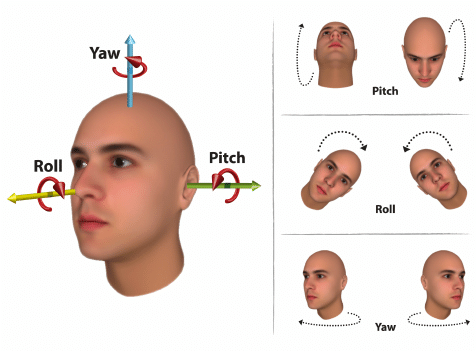
* **Combined Constraints:**
* In the case of OpCode=0, FaceID=-1, ImageData has a value.
* In the case of OpCode=1, FaceID has a value, ImageData has a value.
* In the case of OpCode=2, FaceID has a value, ImageData =null.
* Request example:
* URL:<http://localhost:8004/face_list_registration>
* method: POST
* json:

|  |
| --- |
| {  "TimeSend": "2023-01-06T09:10:49,182",  "Item": {  "ImageData": [  "a91h3jyjksi1",  "u2n1b3ud91k",  "u2n13j1d91k",  "02k1b3ud91k",  "u2n1b3u29k1"  ],  "FaceID": -first,  "OpCode": 0  }  } |

**Response content: Only respond when the data has been saved.**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No** | **Key** | **Datatype** | **Descriptions** | **Example** |
| first | StatusCode | int | Return code:  0: Failed  1: Success |  |
| 2 | Message | string | Detailed description of the status code |  |
| 3 | FaceID | int | Face ID in DB (unique) |  |

* Define Enum and Check Constraints: Similar to Face Registration API
* Criteria of face photo for registration:
* Size of the whole captured frame: from 720p (the shorter side has a length of 720 pixels) - 1920p (the longer side has a length of 1920 pixels) → the software can check
* The face must be the main object in the frame, the ratio of the face area in the frame: 4% - 36%, scale interpretation: → the software can check
* The horizontal and vertical dimensions of the face are 1/5 of the horizontal and vertical dimensions of the frame, so the area ratio will be 1/5\*1/5~4%.
* The horizontal and vertical dimensions of the face are 3/5 of the horizontal and vertical dimensions of the frame, so the area ratio will be 3/5\*3/5~36%.
* Head rotation constraints: roll +-35 degrees, pitch +-35 degrees, yaw +-35 degrees. → checkable software



* The face is not wearing a mask, not obscured by any object → the software cannot check
* Allow wearing transparent glasses, do not wear dark glasses → the software cannot check
* Lighting conditions are sufficient, and clear, not dark, not blurry, or not bright → software cannot check

# Face Authentication API (TNI)

* **Purpose:** Verify a pair of photos (pre-registered image, and current photo taken from the camera), whether it is the same person or not.

**Request content:**

* Timesend:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No** | **Key** | **Datatype** | **Descriptions** | **Example** |
| first | TimeSend | string | Time to send data to the server | yyyy-MM-  ddTHH:mm:ss.fff |

* Item:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No** | **Key** | **Datatype** | **Descriptions** | **Example** |
| first | ImageData | string | Face image (taken from camera) has been base64 encoded from .jpg . file | "A91h3jyjksi1…." |
| 2 | FaceID | int  >=0 | Face ID in DB (unique) | 0 |

* Request example:
* URL:<http://localhost:8004/face_verification>
* method: POST
* json:

|  |
| --- |
| {  "TimeSend": "2023-01-06T09:10:49,182",  "Item": {  "ImageData": "A91h3jyjksi1",  "FaceID": 0,  }  } |

**Response content: Only respond when the comparison has been processed.**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No** | **Key** | **Datatype** | **Descriptions** | **Example** |
| first | StatusCode | int | Return code:  0: Failed  1: Success  2: Wrong format |  |
| 2 | Message | string | Detailed description of the status code |  |
| 3 | Similarity | float | Similarity between the previously registered image and the current image taken from the camera |  |

* Enum definition:
* STATUS\_CODE:
  + FAIL = 0
  + SUCCESS = 1
* MESSAGE:
  + WRONG\_FORMAT = "Wrong format request fields"
  + CAPTURE\_INCORRECT = "Image is captured incorrectly"
  + FACE\_NOT\_EXIST = "FaceID is not exist"
  + VERIFY\_SUCCESS = "Verification successfully"
  + INTERNAL\_SERVER\_ERROR = "Internal Server Error"
* Checks for the presence (existence), type, and value binding of the defined fields, if not satisfied, returns:
* StatusCode = STATUS\_CODE.FAIL
* Message = MESSAGE.WRONG\_FORMAT
* Similarity = -1
* If FaceID does not exist, return:
* StatusCode = STATUS\_CODE.FAIL
* Message = MESSAGE.FACE\_NOT\_EXIST
* Similarity = -1
* If FaceID exists, checks if the image was taken correctly, if not, returns:
* StatusCode = STATUS\_CODE.FAIL
* Message = MESSAGE.CAPTURE\_INCORRECT
* Similarity = -1
* If FaceID exists and the image was taken correctly, calculate the similarity and return:
* StatusCode = STATUS\_CODE.SUCCESS
* Message = MESSAGE.VERIFY\_SUCCESS
* Similarity = Similarity
* If an unknown error occurs during processing, return:
* StatusCode = STATUS\_CODE.FAIL
* Message = MESSAGE.INTERNAL\_SERVER\_ERROR
* Similarity = -1

# Registered Face Check API (TNI)

* **Purpose:**Check to see if a person has registered a face or not.

**Request content:**

* Timesend:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No** | **Key** | **Datatype** | **Descriptions** | **Example** |
| first | TimeSend | string | Time to send data to the server | yyyy-MM-  ddTHH:mm:ss.fff |

* Item:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No** | **Key** | **Datatype** | **Descriptions** | **Example** |
| first | ImageData | string | Face image (taken from camera) has been base64 encoded from .jpg . file |  |
| 2 | Threshold | float >= 0 | Threshold for identifying acquaintances, range [0,1] |  |

* Request example:
* URL:<http://localhost:8004/face_existence>
* method: POST
* json:

|  |
| --- |
| {  "TimeSend": "2023-01-06T09:10:49,182",  "Item": {  "ImageData": [  "A91h3jyjksi1",  ],  "Threshold": 0.4,  }  } |

**Response content: Only respond when processing is complete.**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No** | **Key** | **Datatype** | **Descriptions** | **Example** |
| first | StatusCode | int | Return code:  0: Failed  1: Success  2: Wrong format |  |
| 2 | Message | string | Detailed description of the status code |  |
| 3 | IsRegistered | int | Determine registered or not: 0/1, 0 is unregistered, 1 is registered |  |

* Enum definition:
* STATUS\_CODE:
  + FAIL = 0
  + SUCCESS = 1
* MESSAGE:
  + WRONG\_FORMAT = "Wrong format request fields"
  + CAPTURE\_INCORRECT = "Image is captured incorrectly"
  + VERIFY\_SUCCESS = "Verification successfully"
  + INTERNAL\_SERVER\_ERROR = "Internal Server Error"
* Checks for the presence (existence), type, and value binding of the defined fields, if not satisfied, returns:
* StatusCode = STATUS\_CODE.FAIL
* Message = MESSAGE.WRONG\_FORMAT
* IsRegistered = -1
* Check if the image is taken correctly, if not satisfied, return:
* StatusCode = STATUS\_CODE.FAIL
* Message = MESSAGE.CAPTURE\_INCORRECT
* IsRegistered = -1
* If the image is properly captured, computes the highest similarity of the face in question to the faces in the DB, compares it with Threshold, and returns:
* StatusCode = STATUS\_CODE.SUCCESS
* Message = MESSAGE.VERIFY\_SUCCESS
* IsRegistered = IsRegistered
* If an unknown error occurs during processing, return:
* StatusCode = STATUS\_CODE.FAIL
* Message = MESSAGE.INTERNAL\_SERVER\_ERROR
* IsRegistered = -1

# Face Lookup API (ITD)

* **Purpose:**Look up whether the face information exists in the Face Registration System.
* Input: An image of a person's face.
* Output: List of N (up to 3) FaceId with the closest similarity to the input image in the Face Registration System if available.

**Request content:**

* Timesend:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No** | **Key** | **Datatype** | **Descriptions** | **Example** |
| first | TimeSend | string | Time to send data to the server | yyyy-MM-  ddTHH:mm:ss.fff |

* Item:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No** | **Key** | **Datatype** | **Descriptions** | **Example** |
| first | ImageData | string (base64) | Face image (taken from camera) has been base64 encoded from .jpg . file |  |

# Valid Face ID List API (TNI)

**Purpose:**Look up the list of valid Face IDs in the system (ID has not been removed)

**Response content: Only respond when processing is complete.**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No** | **Key** | **Datatype** | **Descriptions** | **Example** |
| first | StatusCode | int | Return code:  0: Failed  1: Success |  |
| 2 | Message | string | Detailed description of the status code |  |
| 3 | Data | list | [  // face 1  {  “ID”: Face\_ID\_1,  “image”: image\_url\_1  },  // face 2  {  “ID”: Face\_ID\_2,  “image”: image\_url\_2  }, …  ] |  |

* Request example:
* URL:[http://localhost:8004/valid\_face\_ids](http://localhost:8004/face_list_registration)
* method: GET
* params:

|  |
| --- |
| "TimeSend": "2023-01-06T09:10:49.182" |

* Enum definition:
* STATUS\_CODE:
  + FAIL = 0
  + SUCCESS = 1
* MESSAGE:
  + WRONG\_FORMAT = "Wrong format request fields"
  + FACE\_BANK\_EMPTY = "Face bank is empty"
  + GET\_VALID\_FACES\_SUCCESS="Get valid FaceIDs successfully”
  + INTERNAL\_SERVER\_ERROR = "Internal Server Error
* If a valid Face ID does not exist, return:
  + - StatusCode = STATUS\_CODE.SUCCESS
    - Message = MESSAGE.FACE\_BANK\_EMPTY
    - Data = []
* If a valid Face ID exists, return:
  + - StatusCode = STATUS\_CODE.SUCCESS
    - Message = MESSAGE.GET\_VALID\_FACES\_SUCCESS
    - Data = [{ID:ID\_1, image:URL\_1}, {ID:ID2, image:URL\_2…]
* If an unknown error occurs during processing, return:
* StatusCode = STATUS\_CODE.FAIL
* Message = MESSAGE.INTERNAL\_SERVER\_ERROR
* Data = []

# Procursion Detection API (ITD)

* Definition of loitering (sorted from top to bottom, in order of preference):
* Standing or moving in the detection area (ROI) for longer than a certain time, eg 5s.
* The movement is not straight, winding, repetitive.

**Request content:**

<https://drive.google.com/file/d/1HijnRtsLQEy9ie2O-xqCBxC5W68sBicU/view?usp=share_link>

# Intrusion Detection API (ITD)

* Intrusion definition: Detects the object in one of the following two cases:
* Enter the area of ​​identification (ROI)
* Crossing the line with the specified direction (line crossing).
* Also recognizes the face (if the face is visible), and indicates whether this person is an employee (checked in the DB).

**Request content:**

<https://drive.google.com/file/d/1HijnRtsLQEy9ie2O-xqCBxC5W68sBicU/view?usp=share_link>

# System Health API (ITD)

* The purpose is for the server side to know if the AI ​​services are working, and the system's parameters, including:
* CPU Usage (%)
* GPU Usage (%)
* RAM usage(%)
* Storage hard disk usage (%)
* CPU temperature (degrees Celsius)
* GPU temperature (degrees Celsius)
* Time of computer life from boot to present (Date May)

**Request content:**

* <https://drive.google.com/file/d/1HijnRtsLQEy9ie2O-xqCBxC5W68sBicU/view?usp=share_link>

# 

# API Read System Configuration (TNI)

* Read system configuration file (json)
* URL:<http://localhost:8000/GetSystemConfig>
* **Request content:**
* Timesend:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No** | **Key** | **Datatype** | **Descriptions** | **Example** |
| first | TimeSend | string | Time to send data to the server | yyyy-MM-  ddTHH:mm:ss.fff |

* Item:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No** | **Key** | **Datatype** | **Descriptions** | **Example** |
| Hollow | | | | |

* **Response content: Only respond when processing is complete.**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No** | **Key** | **Datatype** | **Descriptions** | **Example** |
| first | StatusCode | int | Return code:  0: Failed  1: Success  2: Wrong format |  |
| 2 | Message | string | Detailed description of the status code |  |
| 3 | Data | dict | Contents of the system configuration file (json) | |

* If there is no error, return (StatusCode=1, Message=“Success”).
* If the request does not have enough required fields, return (StatusCode=2, Message=“Wrong request fields”, Data={}).
* Request example:
* URL:<http://localhost:8000/GetSystemConfig>
* method: POST
* json:

|  |
| --- |
| {  "TimeSend": "2023-01-06T09:10:49,182",  "Item": {}  } |

# 

# System Configuration Recording API (TNI)

* Write the content to the system configuration file (json).
* URL:<http://localhost:8000/SetSystemConfig>
* **Request content:**
* Timesend:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No** | **Key** | **Datatype** | **Descriptions** | **Example** |
| first | TimeSend | string | Time to send data to the server | yyyy-MM-  ddTHH:mm:ss.fff |

* Item:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No** | **Key** | **Datatype** | **Descriptions** | **Example** |
| first | Data | dict | Content to write to the system configuration file (json) |  |

* **Response content: Only respond when processing is complete.**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No** | **Key** | **Datatype** | **Descriptions** | **Example** |
| first | StatusCode | int | Return code:  0: Failed  1: Success  2: Wrong format |  |
| 2 | Message | string | Detailed description of the status code |  |

* If there is no error, return (StatusCode=1, Message=“Success”).
* If the request does not have enough required fields, return (StatusCode=2, Message=“Wrong request fields”).
* Request example:
* URL:[http://localhost:8000/](http://localhost:8000/GetSystemConfig)[SetSystemConfig](http://localhost:8000/SetSystemConfig)
* method: POST
* json:

|  |
| --- |
| {  "TimeSend": "2023-01-06T09:10:49,182",  "Item": {  "Data": {  "ais\_api": {},  "ais\_deepstream": {},  ……………………  }  }  } |

# Camera Number Configuration API (TNI)

* Purpose: Configure the maximum number of cameras used for the system.
* When using this API, the list of cameras (if exists) will be deleted to create a new list. At this point, the user needs to use the Camera Configuration API to add information for each camera.

**Request content:**

* Timesend:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No** | **Key** | **Datatype** | **Descriptions** | **Example** |
| first | TimeSend | string | Time to send data to the server | yyyy-MM-  ddTHH:mm:ss.fff |

* Item:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No** | **Key** | **Datatype** | **Descriptions** | **Example** |
| first | MaxCam | int | Maximum number of cameras used for the system | twelfth |

**Response content: Only respond when processing is complete.**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No** | **Key** | **Datatype** | **Descriptions** | **Example** |
| first | StatusCode | int | Return code:  0: Failed  1: Success  2: Wrong format |  |
| 2 | Message | string | Detailed description of the status code |  |

* Constraint MaxCam (int) > 0, if not satisfied, return (StatusCode=0, Message=“Invalid MaxCam”).
* If there is no error, return (StatusCode=1, Message=“Success”).
* If the request does not have enough required fields, return (StatusCode=2, Message=“Wrong request fields”, Data=[]).

# 

# Camera List API (TNI)

* Returns a list of cameras.
* **Request content:**
* Timesend:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No** | **Key** | **Datatype** | **Descriptions** | **Example** |
| first | TimeSend | string | Time to send data to the server | yyyy-MM-  ddTHH:mm:ss.fff |

* Item:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No** | **Key** | **Datatype** | **Descriptions** | **Example** |
| Hollow | | | | |

* **Response content: Only respond when processing is complete.**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No** | **Key** | **Datatype** | **Descriptions** | **Example** |
| first | StatusCode | int | Return code:  0: Failed  1: Success  2: Wrong format |  |
| 2 | Message | string | Detailed description of the status code |  |
| 3 | Data | list | [  // cam1  {  “CamID”: STT\_1\_to\_MaxCam,  “DetectStreamURL”: url\_string,  “TopicID”: topic\_string,  “ROI”: [ // list or null  x1, y1,  x2, y2,  …  xn, yn,  ],  “Line”: [ // list or null  line\_x1, line\_y1,  line\_x2, line\_y2,  dir\_x1, dir\_y1,  dir\_x2, dir\_y2,  ],  “Floor”: floor\_int,  “EnableIntrusion”: True/False  “EnableLoitering”: True/False  “EnableReID”: True/False  },  // cam2  // cam3  …  ] | |

* If there is no error, return (StatusCode=1, Message=“Success”).
* If the request does not have enough required fields, return (StatusCode=2, Message=“Wrong request fields”, Data=[]).

# Camera Configuration API (TNI)

* Purpose: Configure the parameters of a camera.
* Note, after the configuration is complete for all cameras, it is necessary to call the API Restart the system for the system to update the new configuration.
* **Request content:**
* Timesend:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No** | **Key** | **Datatype** | **Descriptions** | **Example** |
| first | TimeSend | string | Time to send data to the server | yyyy-MM-  ddTHH:mm:ss.fff |

* Item:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No** | **Key** | **Datatype** | **Descriptions** | **Example** |
| first | CamID | int | Camera sequence number, with a continuous value in the range [1; MaxCam] |  |
| 2 | DetectStreamURL | string | Camera stream URL | rtsp://10.0.8.115:554/0/  profile1/media.smp |
| 3 | TopicID | string | Camera identifier | cam1\_tang1 |
| 4 | ROD | list | Zones are represented by polygons. List or null | [  x1, y1,  x2, y2,  …  xn, yn,  ] |
| 5 | Line | list | Lines represent lines and directions. Direction from 1 to 2. List or null | [  line\_x1, line\_y1,  line\_x2, line\_y2,  dir\_x1, dir\_y1,  dir\_x2, dir\_y2,  ] |
| 6 | Floor | int | Number of floor floor | first |
| 7 | EnableIntrusion | bool | Enable Intrusion Detection | True/False |
| 8 | EnableLoitering | bool | Enable loitering detection | True/False |
| 9 | EnableReID | bool | Activate the tracing function | True/False |

**Response content: Only respond when processing is complete.**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No** | **Key** | **Datatype** | **Descriptions** | **Example** |
| first | StatusCode | int | Return code:  0: Failed  1: Success  2: Wrong format |  |
| 2 | Message | string | Detailed description of the status code |  |
| 3 | OverlayImage | string (base64) | Image has overlayed camera information such as ROI, Line, Floor, EnableIntrusion, EnableLoitering, and EnableReID |  |

* Constraints CamID (int) > 0 and <= MaxCam, if not satisfied, returns (StatusCode=0, Message=“Invalid CamID”).
* If the frame cannot be read from the camera URL, return (StatusCode=0, Message=“Invalid camera URL”).
* The TopicID constraint must not coincide with the TopicIDs of other cameras, if not satisfied, return (StatusCode=0, Message=“Existed TopicID”).
* Constraint coordinates of ROI points must be inside the image, not outside, if not satisfied, return (StatusCode=0, Message=“Invalid ROI”).
* Constraint coordinates of Line points must be inside the image, not outside, if not satisfied, return (StatusCode=0, Message=“Invalid Line”).
* Floor constraint (int) > 0, if not satisfied, return (StatusCode=0, Message=“Invalid Floor”).
* If the frame is read from the camera URL, return (StatusCode=1, Message=“Success”).
* If the request does not have enough required fields, return (StatusCode=2, Message=“Wrong request fields”, Data=[]).

# System Restart API (TNI)

* Purpose: After the configuration is complete, it is necessary to restart the system to update the new configuration

**Request content:**

* Timesend:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No** | **Key** | **Datatype** | **Descriptions** | **Example** |
| first | TimeSend | string | Time to send data to the server | yyyy-MM-  ddTHH:mm:ss.fff |

* Item:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No** | **Key** | **Datatype** | **Descriptions** | **Example** |
| Hollow | | | | |

**Response content: Only respond when processing is complete.**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No** | **Key** | **Datatype** | **Descriptions** | **Example** |
| first | StatusCode | int | Return code:  0: Failed  1: Success  2: Wrong format |  |
| 2 | Message | string | Detailed description of the status code |  |

* If there are no errors, return (StatusCode=1, Message=“Success”), and restart the system.
* If the request does not have enough required fields, return (StatusCode=2, Message=“Wrong request fields”, Data=[]).
* Request example:
* URL:<http://localhost:8000/GetSystemConfig>
* method: POST
* json:

|  |
| --- |
| {  "TimeSend": "2023-01-06T09:10:49,182",  "Item": {}  } |

# 

# Trace List API (TNI)

* Returns a list of people for the specified time period.
* **Request content:**
* Timesend:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No** | **Key** | **Datatype** | **Descriptions** | **Example** |
| first | TimeSend | string | Time to send data to the server | yyyy-MM-  ddTHH:mm:ss.fff |

* Item:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No** | **Key** | **Datatype** | **Descriptions** | **Example** |
| first | TimeStart | string | Start time | yyyy-MM-  ddTHH:mm:ss.fff |
| 2 | TimeEnd | string | End time | yyyy-MM-  ddTHH:mm:ss.fff |
| 3 | TopicIDs | list | List of TopicIDs, default is null corresponding to get all TopicIDs | [“cam1\_tang2”, “cam1\_tang3”] |

* **Request example:**
* URL:<http://localhost:8005/reid_list>
* method: POST
* json:

|  |
| --- |
| {  "TimeSend": "2023-03-29T10:00:00.000",  "Item":  {  "TimeStart": "2023-03-29T00:00:00.000",  "TimeEnd": "2023-03-29T10:00:00.000",  "TopicIDs": null  }  } |

**Response content: Only respond when processing is complete.**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No** | **Key** | **Datatype** | **Descriptions** | **Example** |
| first | StatusCode | int | Return code:  0: Failed  1: Success  2: Wrong format |  |
| 2 | Message | string | Detailed description of the status code |  |
| 3 | Data | list | [  // person1  {  “ID”: UUID,  “box\_trajectory”: [  (x1\_1, y1\_1, x2\_1, y2\_1),  (x1\_2, y1\_2, x2\_2, y2\_2),  …,  (x1\_n, y1\_n, x2\_n, y2\_n),  ],  “time\_trajectory”: [t1, t2, …, tn],  “camera”: cam\_id,  “image”: image\_url, // cropped  “video”: video\_url, // from t1 to tn  },  // person2  // person3  …  ] | |

* The TimeStart < TimeEnd constraint, if not satisfied, returns (StatusCode=0, Message=“Invalid time, require TimeStart < TimeEnd”, Data=[]).
* The TimeEnd < now constraint, if not satisfied, returns (StatusCode=0, Message=“Invalid time, require TimeEnd < now”, Data=[]).
* If TopicIDs are not null, constraints the elements in TopicIDs to exist in the config, otherwise return (StatusCode=0, Message=“Invalid TopicIDs, <TopicID> does not exist”, Data=[]).
* Conditions to add 1 person in the DB to the returned list:
* Have the occurrence time (variable t1 in the time\_trajectory field) within the time frame (start-end) of the request.
* If TopicIDs is not null and is a list, only people who appear in the camera have a TopicID listed in that list of TopicIDs.
* If the return list is empty (eg: Time period when no one appeared in front of the camera), return (StatusCode=0, Message=“No people”, Data=[]).
* If the return list is not empty, return (StatusCode=1, Message=“Success”, Data=[...]).
* The returned list is sorted chronologically by the variable t1 (the time when the person appeared) in the time\_trajectory field.
* If the request does not have enough required fields, return (StatusCode=2, Message=“Wrong request fields”, Data=[]).

# Tracking by ID (TNI) API

* Select an ID from the list returned from the Tracking List API, which returns information about people who closely match the person with that ID.
* The list of returnees is sorted by similarity from highest to lowest.

**Request content:**

* Timesend:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No** | **Key** | **Datatype** | **Descriptions** | **Example** |
| first | TimeSend | string | Time to send data to the server | yyyy-MM-  ddTHH:mm:ss.fff |

* Item:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No** | **Key** | **Datatype** | **Descriptions** | **Example** |
| first | TimeStart | string | Start time | yyyy-MM-  ddTHH:mm:ss.fff |
| 2 | TimeEnd | string | End time | yyyy-MM-  ddTHH:mm:ss.fff |
| 3 | TopicIDs | list | List of TopicIDs, default is null corresponding to get all TopicIDs | [“cam1\_tang2”, “cam1\_tang3”] |
| 4 | ID | string | The person's ID, retrieved from the list returned by the Track List API | uwj12k2909jfn |
| 5 | MaxPeople | int | The maximum number of similar people, the returned result may be lower than this number | 8 |
| 6 | Threshold | float | Similarity threshold to limit the number of people in the returned list | 0.4 |

* **Request example:**
* URL:[http://localhost:8005/reid\_list\_by\_id](http://localhost:8005/reid_list)
* method: POST
* json:

|  |
| --- |
| {  "TimeSend": "2023-03-29T10:00:00.000",  "Item":  {  "TimeStart": "2023-03-30T00:00:00.000",  "TimeEnd": "2023-03-30T10:00:00.000",  "TopicIDs": null,  "ID": "632ca474-0edc-46bb-bb9a-6524ad19a2fe",  "MaxPeople": 8,  "Threshold": 0.5  }  } |

**Response content: Only respond when processing is complete.**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No** | **Key** | **Datatype** | **Descriptions** | **Example** |
| first | StatusCode | int | Return code:  0: Failed  1: Success  2: Wrong format |  |
| 2 | Message | string | Detailed description of the status code |  |
| 3 | Data | list | [  // person1  {  “ID”: UUID,  “box\_trajectory”: [  (x1\_1, y1\_1, x2\_1, y2\_1),  (x1\_2, y1\_2, x2\_2, y2\_2),  …,  (x1\_n, y1\_n, x2\_n, y2\_n),  ],  “time\_trajectory”: [t1, t2, …, tn],  “camera”: cam\_id,  “image”: image\_url,  “video”: video\_url,  “similarity”: float\_range\_0\_1,  },  // person2  // person3  …  ] | |

* The TimeStart < TimeEnd constraint, if not satisfied, returns (StatusCode=0, Message=“Invalid time, require TimeStart < TimeEnd”, Data=[]).
* The TimeEnd < now constraint, if not satisfied, returns (StatusCode=0, Message=“Invalid time, require TimeEnd < now”, Data=[]).
* If TopicIDs are not null, constraints the elements in TopicIDs to exist in the config, otherwise return (StatusCode=0, Message=“Invalid TopicIDs, <TopicID> does not exist”, Data=[]).
* Constraint MaxPeople(int) > 0, if not satisfied, return (StatusCode=0, Message=“Invalid MaxPeople”, Data=[]).
* Threshold (float) constraint, range [0; 1], if not satisfied, return (StatusCode=0, Message=“Invalid Threshold”, Data=[]).
* Conditions to add 1 person in the DB to the returned list:
* Have the occurrence time (variable t1 in the time\_trajectory field) within the time frame (start-end) of the request.
* If TopicIDs is not null and is a list, only people who appear in the camera have a TopicID listed in that list of TopicIDs.
* Similarity to the person requested: similarity > Threshold.
* The returned list is sorted by similarity (similarity field) from high to low.
* The list is limited in number not to exceed MaxPeople.
* If the return list is empty, return (StatusCode=0, Message=“No people”, Data=[]).
* If the return list is not empty, return (StatusCode=1, Message=“Success”, Data=[...]).
* If the request does not have enough required fields, return (StatusCode=2, Message=“Wrong request fields”, Data=[]).

# Image Tracing API (TNI)

* Give a photo (cropped) of a person, return the information of people who are similar to that person.
* The list of returnees is sorted by similarity from highest to lowest.

**Request content:**

* Timesend:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No** | **Key** | **Datatype** | **Descriptions** | **Example** |
| first | TimeSend | string | Time to send data to the server | yyyy-MM-  ddTHH:mm:ss.fff |

* Item:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No** | **Key** | **Datatype** | **Descriptions** | **Example** |
| first | TimeStart | string | Start time | yyyy-MM-  ddTHH:mm:ss.fff |
| 2 | TimeEnd | string | End time | yyyy-MM-  ddTHH:mm:ss.fff |
| 3 | TopicIDs | list | List of TopicIDs, default is null corresponding to get all TopicIDs | [“cam1\_tang2”, “cam1\_tang3”] |
| 4 | ImageData | string (base64) | Face image (taken from camera) has been base64 encoded from .jpg . file |  |
| 5 | MaxPeople | int | The maximum number of similar people, the returned result may be lower than this number | 8 |
| 6 | Threshold | float | Similarity threshold to limit the number of people in the returned list (range [0, 1] | 0.4 |

**Request example:**

* URL:<http://localhost:8005/reid_list_by_image>
* method: POST
* json:

|  |
| --- |
| {  "TimeSend": "2023-03-29T10:00:00.000",  "Item":  {  "TimeStart": "2023-03-30T00:00:00.000",  "TimeEnd": "2023-03-30T10:00:00.000",  "TopicIDs": null,  "ImageData": "/9j/4QAYRXhpZgAASUkqA",  "MaxPeople": 8,  "Threshold": 0.5  }  } |

**Response content: Only respond when processing is complete.**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No** | **Key** | **Datatype** | **Descriptions** | **Example** |
| first | StatusCode | int | Return code:  0: Failed  1: Success  2: Wrong format |  |
| 2 | Message | string | Detailed description of the status code |  |
| 3 | Data | list | [  // person1  {  “ID”: UUID,  “box\_trajectory”: [  (x1\_1, y1\_1, x2\_1, y2\_1),  (x1\_2, y1\_2, x2\_2, y2\_2),  …,  (x1\_n, y1\_n, x2\_n, y2\_n),  ],  “time\_trajectory”: [t1, t2, …, tn],  “camera”: cam\_id,  “image”: image\_url,  “video”: video\_url,  “similarity”: float\_range\_0\_1,  },  // person2  // person3  …  ] | |

* The TimeStart < TimeEnd constraint, if not satisfied, returns (StatusCode=0, Message=“Invalid time, require TimeStart < TimeEnd”, Data=[]).
* The TimeEnd < now constraint, if not satisfied, returns (StatusCode=0, Message=“Invalid time, require TimeEnd < now”, Data=[]).
* If TopicIDs are not null, constraints the elements in TopicIDs to exist in the config, otherwise return (StatusCode=0, Message=“Invalid TopicIDs, <TopicID> does not exist”, Data=[]).
* Constraint MaxPeople(int) > 0, if not satisfied, return (StatusCode=0, Message=“Invalid MaxPeople”, Data=[]).
* Threshold (float) constraint, range [0; 1], if not satisfied, return (StatusCode=0, Message=“Invalid Threshold”, Data=[]).
* Conditions to add 1 person in the DB to the returned list:
* Have the occurrence time (variable t1 in the time\_trajectory field) within the time frame (start-end) of the request.
* If TopicIDs is not null and is a list, only people who appear in the camera have a TopicID listed in that list of TopicIDs.
* Similarity to the person requested: similarity > Threshold.
* The returned list is sorted by similarity (similarity field) from high to low.
* The list is limited in number not to exceed MaxPeople.
* If the return list is empty, return (StatusCode=0, Message=“No people”, Data=[]).
* If the return list is not empty, return (StatusCode=1, Message=“Success”, Data=[...]).
* If the request does not have enough required fields, return (StatusCode=2, Message=“Wrong request fields”, Data=[]).

# Person Detection API (TNI)

* Give a panoramic picture, return the information of the people who found it.
* The list of returnees is sorted by reliability from high to low.

**Request content:**

* Timesend:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No** | **Key** | **Datatype** | **Descriptions** | **Example** |
| first | TimeSend | string | Time to send data to the server | yyyy-MM-  ddTHH:mm:ss.fff |

* Item:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No** | **Key** | **Datatype** | **Descriptions** | **Example** |
| first | ImageData | string (base64) | Base64 encoded image from .jpg . file |  |

**Request example:**

* URL:<http://localhost:8006/PersonDetection>
* method: POST
* json:

|  |
| --- |
| {  "TimeSend": "2023-01-06T09:10:49,182",  "Item": {  "ImageData": "a91h3jyjksi1"  }  } |

**Response content: Only respond when processing is complete.**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No** | **Key** | **Datatype** | **Descriptions** | **Example** |
| first | StatusCode | int | Return code:  0: Failed  1: Success  2: Wrong format |  |
| 2 | Message | string | Detailed description of the status code |  |
| 3 | Data | list | [  // person1  {  “box”: [int\_x1, int\_y1, int\_x2, int\_y2],  “score”: float\_range\_0\_1,  },  // person2  // person3  …  ] | |

**Check Enum and union constraints:**

* If the return list is not empty, return (StatusCode=1, Message=“Success”, Data=[...]).
  + Define:
    - box: Draw the object area in the form of a box
    - score: Human-likeness

|  |
| --- |
| {  "Data": [  {  "box": [  60,  11,  218,  153  ],  "score":0.5144849419593811  }  ],  "Message":"Detect successfully",  "StatusCode":first  } |

* If the request does not have all the required fields, return
  + Data = []
  + StatusCode = 0
  + Message = MESSAGE.WRONG\_FORMAT

* If ImageData = null:
  + Data = []
  + StatusCode = 0
  + Message = MESSAGE.WRONG\_FORMAT
* If TimeSend = null:
  + Data = []
  + StatusCode = 0
  + Message = MESSAGE.WRONG\_FORMAT