TRAUMA SDK VALIDATOR DEVELOPERS GUIDE

Version 1.1.3

TABLE OF CONTENTS

REVISION HISTORY

1.1.3: Formatting and updating of text. Add Revision History

1.1.2: Add Validation Modes model. Move to semantic versioning

1.1: Fix yaml and documentation for num_records_succeed to be consistent. Update api_key to be specified as a body rather than a header requirement.

OVERVIEW

The Trauma Vendor Alliance (TVA) has developed a set of new Cloud based validators to ensure a way to consistently validate high quality records against multiple mechanisms. Using a cloud based mechanism will provide for easy deployments, updates and consistency without any impact to integration or deployment efforts.

ABOUT THE TVA

The TVA has been established by the nation's three leading hospital trauma registry vendors to help ensure support of industry needs crossing vendor platforms that will ultimately benefit the entire trauma community. DI, along with the other members of the TVA, will be donating time, expertise, and technical contributions as a community service to help better serve our customers with our shared mission of allowing trauma registry systems to better receive and share information with a greater variety of systems and products. This will ultimately help improve patient outcomes and save lives.

USAGE

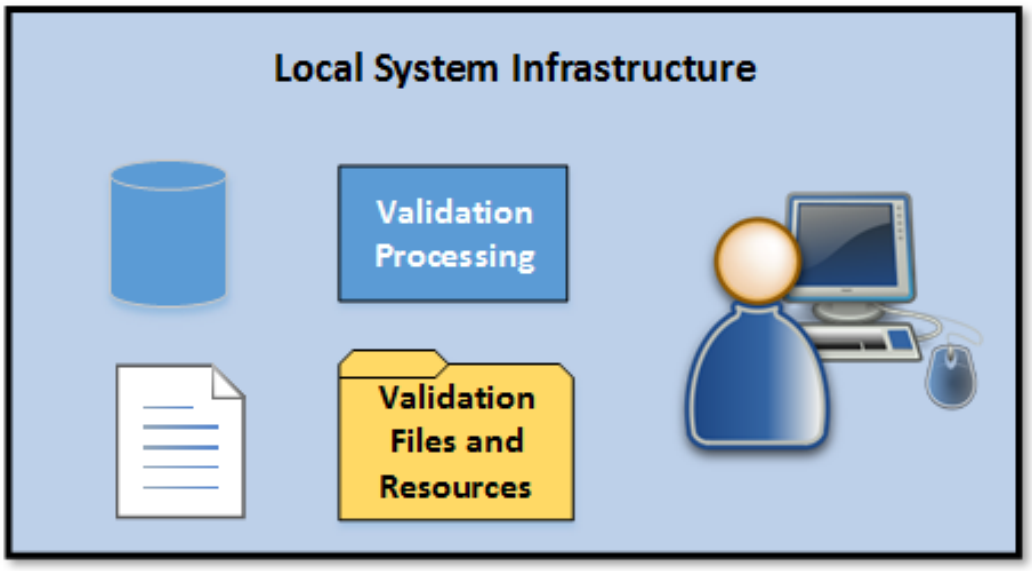
There are 3 primary users anticipated for this validator.

1. Clients wishing to validate an entire file for local validation or for later submission
2. Clients wishing to do a record check as part of a data entry or other activity. It is recommended that validation be tied as closely as possible to the data entry process as possible to improve data consistency.
3. Existing clients using a legacy SDK or EXE for validating files that want to seamlessly move to the new cloud based solutions

VALIDATION MODES

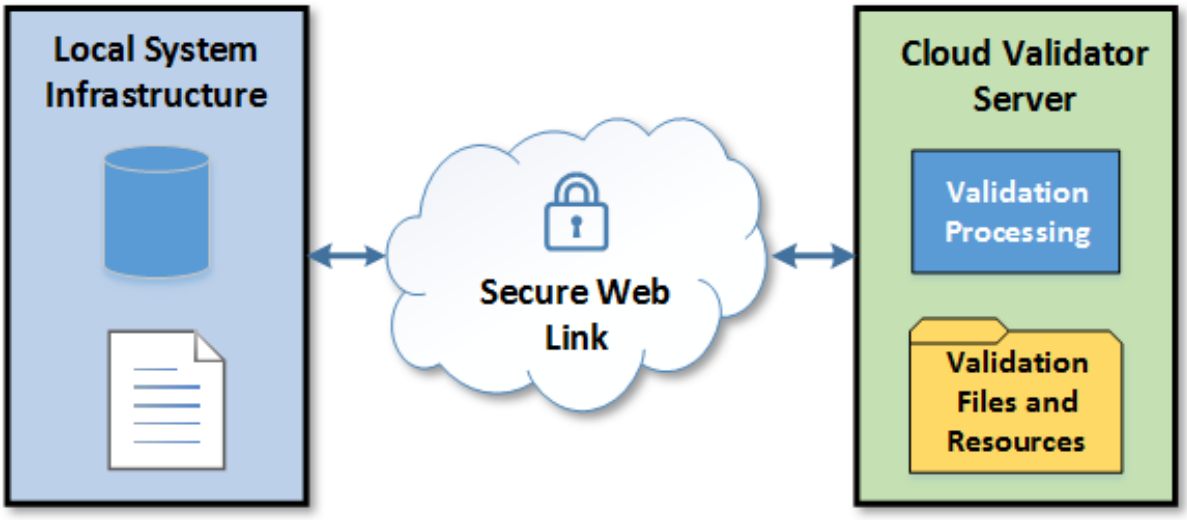
MANUAL UPDATING VALIDATION MODEL

In the manual updating validation model, a submission file is prepared and validated all within the local infrastructure. The validation processing files and configuration can become out of date and need periodic updating or a false validation result could be presented to the user. The system developer must prepare an update and update method for users to run.



CLOUD VALIDATION MODEL

In the cloud validation model, export files are prepared within the local infrastructure. The file is submitted securely to a centralized validation web service, 'in the cloud'. The centralized validation processing allows for a 'change once for many' solution, favorable in large distributive markets. Once the file is processed, the validation results are accessible to the end user via the local system.



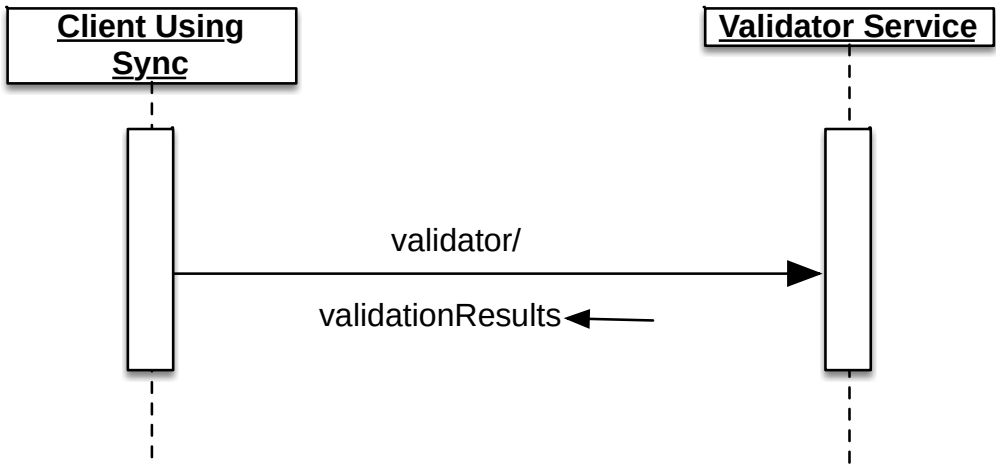
VALIDATOR API

The new API uses a simple interface to allow for the validation of a set of records. The approach requires the client to have an api_key which is unique identifier for the account to gain access to the cloud.

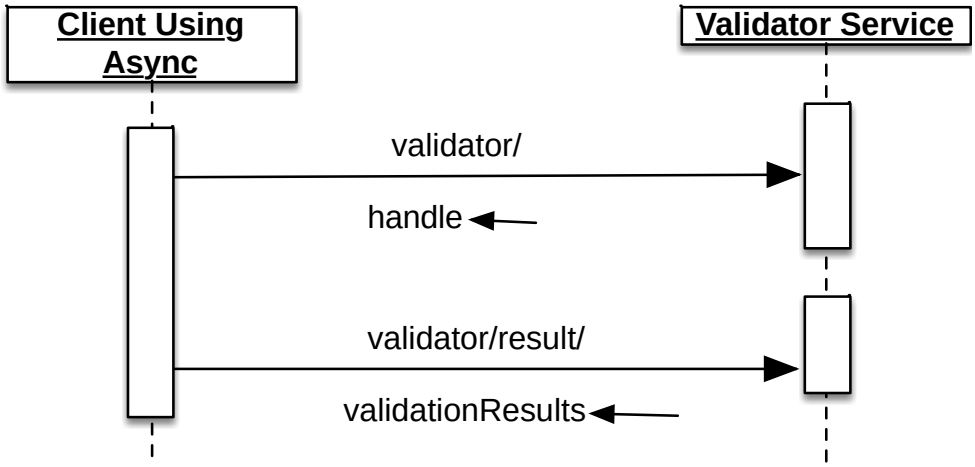
The validator call uses a `multipart/form-data` content message. A validator will either return the results directly or a handle for a subsequent /validate/response retrieval.

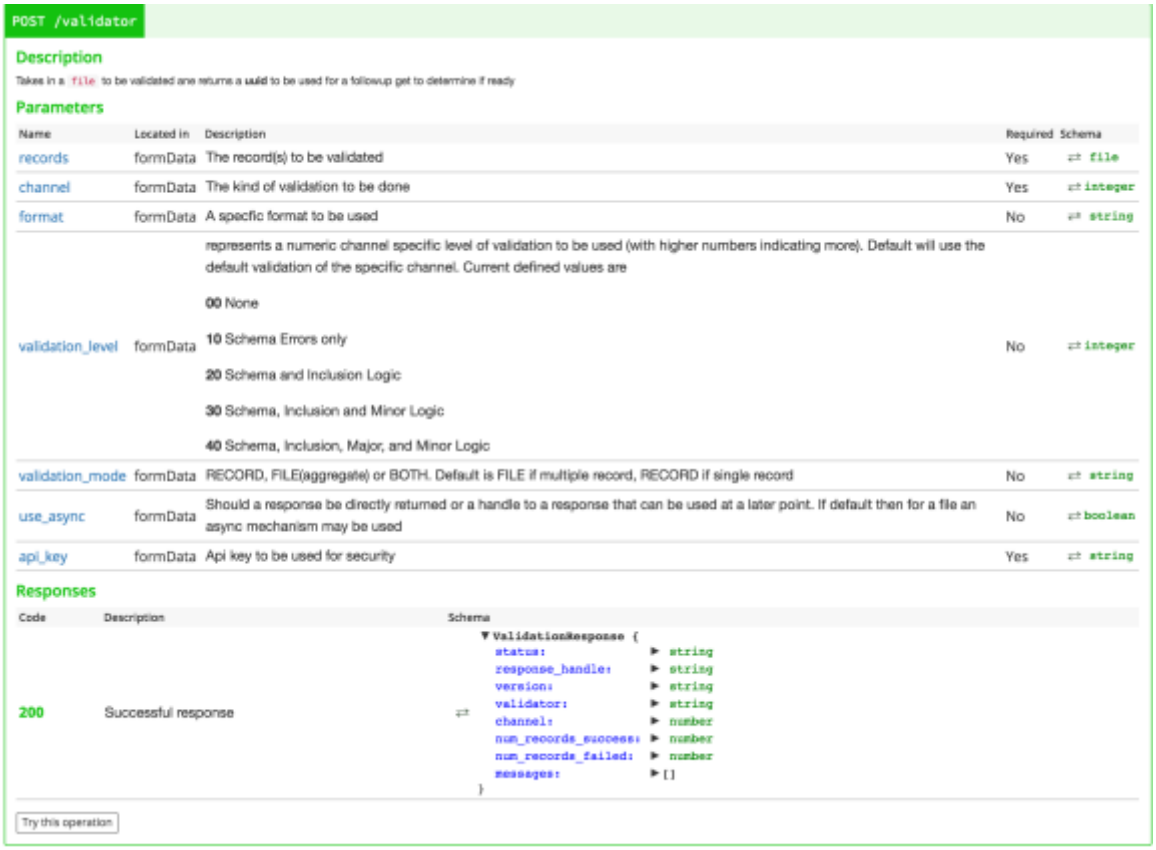
There are two ways in which a client may interact with the service.

First there is synchronously where the client sends the record(s) and gets a ValidationResult as the response of the request (best intended for single record at a time runs)



Secondly, asynchronously where a request to validate (normally multiple records) responds with a handle that is later called to get the validation results. This is intended for larger submissions or cases of high volume load.





POST /validator

Description

Takes in a file to be validated and returns a uuid to be used for a followup get to determine if ready

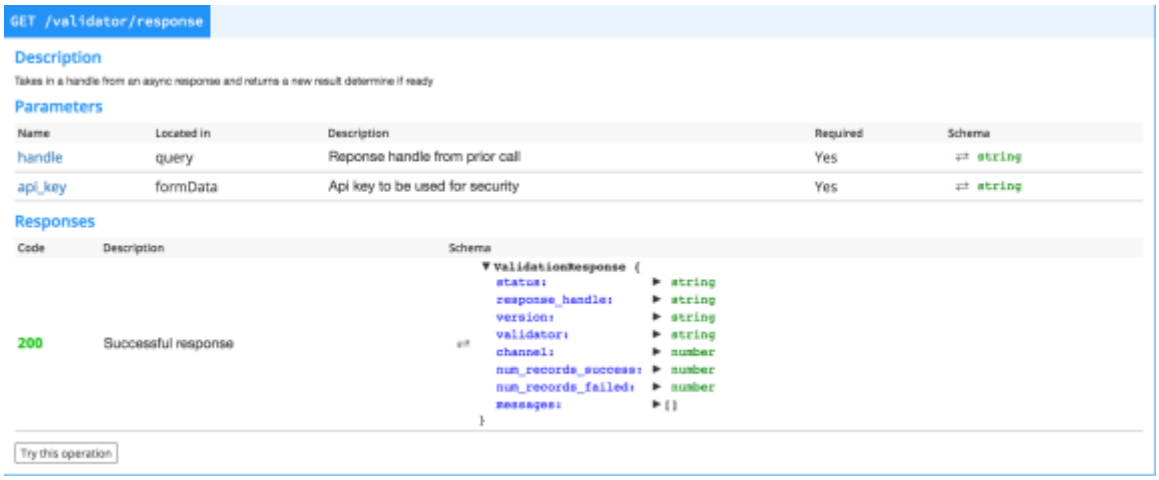
Parameters

Name	Located in	Description	Required	Schema
records	formData	The record(s) to be validated	Yes	file
channel	formData	The kind of validation to be done	Yes	integer
format	formData	A specific format to be used	No	string
validation_level	formData	represents a numeric channel specific level of validation to be used (with higher numbers indicating more). Default will use the default validation of the specific channel. Current defined values are **00** None **10** Schema Errors only **20** Schema and Inclusion Logic **30** Schema, Inclusion and Minor Logic **40** Schema, Inclusion, Major, and Minor Logic	No	integer
validation_mode	formData	RECORD, FILE(aggregate) or BOTH. Default is FILE if multiple record, RECORD if single record	No	string
use_async	formData	Should a response be directly returned or a handle to a response that can be used at a later point. If default then for a file an async mechanism may be used	No	boolean
api_key	formData	Api key to be used for security	Yes	string

Responses

Code	Description	Schema
200	Successful response	ValidationResponse { status: string, response_handle: string, version: string, validator: string, channel: number, num_records_success: number, num_records_failed: number, messages: [] }

Try this operation



GET /validator/response

Description

Takes in a handle from an async response and returns a new result determine if ready

Parameters

Name	Located in	Description	Required	Schema
handle	query	Reponse handle from prior call	Yes	string
api_key	formData	Api key to be used for security	Yes	string

Responses

Code	Description	Schema
200	Successful response	ValidationResponse { status: string, response_handle: string, version: string, validator: string, channel: number, num_records_success: number, num_records_failed: number, messages: [] }

Try this operation

SAMPLE CLIENT USAGE

For the below assume we have a sample file (example_108.xml) using channel 108 and a api_key={APIKEY}

CURL USAGE

curl --form "records=@example_108.xml" --form channel=108 --form validation_mode=file --form api_key={APIKEY} *https://tva-validator-test.dicorp.com/validator*

PYTHON SAMPLE USAGE

Sync

python validate_file.py https://tva-validator-test.dicorp.com {APIKEY} *--file tools/tests/data/ntds_example_108.xml*

Async

Submit

python tools/validate_file.py https://tva-validator-test.dicorp.com {APIKEY} *--file ntds_example_108.xml --use_async*

returns response handle {HANDLE} (for example)

Response

python tools/validate_file.py https://tva-validator-test.dicorp.com {APIKEY} *--handle* {HANDLE}

PYTHON SAMPLE CLIENT CODE

```
#!/usr/bin/env python
#
# Copyright (c) 2017 DI Solutions, Inc.  All rights reserved.
#
#   The information and source code contained herein is the exclusive property of
#   DI Solutions, Inc. and may not be disclosed, examined, or
#   reproduced in whole or in part without the explicit written authorization from DI Solutions, Inc.
#
#
import argparse
import logging

import requests

log = logging.getLogger(__name__)


def validator(url, api_key, records_file, channel = 108, validation_level=40, validation_mode="file",
use_async = False):
    result = requests.post(url+"/validator", files={'records': records_file},
                           data=dict(channel=channel, validation_level=validation_level, use_async=use_async,
validation_mode = validation_mode, api_key = api_key))
    if result.status_code != 200:
        print("Failure submitting file : %s", result.text)
    else:
        print(result.json())

def validator_response(url, api_key, handle):
    result = requests.get(url+"/validator/response", data = dict(handle = handle, api_key = api_key))
    if result.status_code != 200:
        print("Failure submitting handle_request : %s", result.text)
    else:
        print(result.json())

if __name__ == '__main__':
    parser = argparse.ArgumentParser(description="Load tailoring")
    parser.add_argument("url", help="A remote url to connect to")
    parser.add_argument("api_key", help="The api_key to use")
    parser.add_argument("--file", default = None, help="The filename to load", type=argparse.FileType('rb'))
    parser.add_argument("--handle", default = None, help="The handle to use")
    parser.add_argument("--channel", default=108, help="Channel to use")
    parser.add_argument("--validation_level", default=108, help="Validation Level")
    parser.add_argument("--validation_mode", default='file', help="Validation Mode", choices = ('record','file'))
    parser.add_argument("--use_async", dest="use_async", action="store_true", help="Use async")

    parser.add_argument("--logging", "--level", default=logging.INFO,
                        help="Logging level of current information. This is usually DEBUG, INFO or ERROR")  # dest =
"level",  # parser.add_argument("username", help="The username to log into for the cram service")
    parser.add_argument("--requests_logging", default=logging.ERROR,
                        help="Logging level for communication. This is usually DEBUG, INFO or ERROR")  # dest =
"level",  # parser.add_argument("password", help="The user's password to log into for the cram service")
```

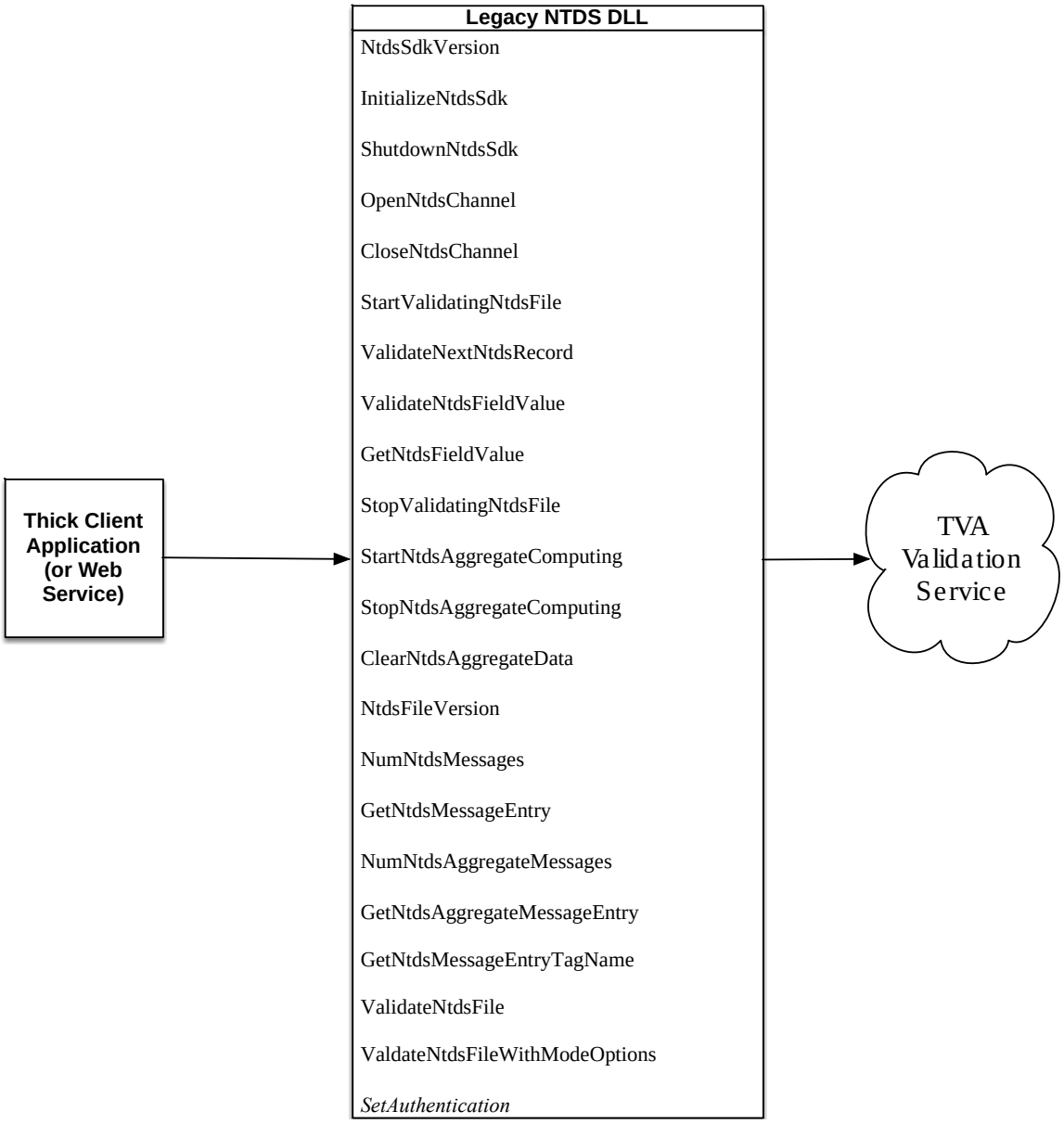


```
    args = parser.parse_args()
    parser.set_defaults(use_async=False)

    if args.file is None and args.handle is None:
        print("ERROR: You must specify a file or a handle")
    else:
        requests_log = logging.getLogger("requests.packages.urllib3")
        requests_log.setLevel(args.requests_logging)
        requests_log.propagate = True
        logging.basicConfig(level=args.logging, format='%(asctime)s %(name)s %(levelname)s %(message)s')
        log.debug(args)

        if args.file:
            validator(url=args.url, api_key=args.api_key, records_file=args.file, channel=args.channel,
validation_level=args.validation_level, validation_mode = args.validation_mode, use_async=args.use_async)
        else: # handle
            validator_response(url=args.url, api_key=args.api_key, handle=args.handle)
```


LEGACY USAGE

As part of the overall approach, the TVA provides a nearly effortless approach to allow existing standard SDK based validators to utilize the new cloud based validation solution.

At the highest level, a vendor can replace the existing DLL with the new ‘pin compatible’ one provided by the TVA and it will integrate. An optional configuration file or a simple call to SetAuthentication will allow any web based credentials to be set for the specific solution.



APPENDIX: SWAGGER API YAML DEFINITION

```
YAML Swagger definiton of Trauma Vendor Appliance Validator API

# Brandon Goldfedder


swagger: '2.0'


# This is your document metadata

info:

  description: |

    The new API uses a simple interface to allow for the validation of
a set of records.

    The approach requires the client to have an api_key which is a
unique identifier for the account to gain access to the cloud.

    The calls assumes a multipart-encoded file is passed


  version: "0.0.0"

  title: Trauma Vendor Appliance Validator API



# Describe your paths here

paths:

  # This is a path endpoint. Change it.

  /validator:

    # This is a HTTP operation

    post:

      # Describe this verb here. Note: you can use markdown
```



```
      description: |

        Takes in a `file` to be validated ane returns a **uuid** to be
used for a followup get to determine if ready

      # This is array of GET operation parameters:

      parameters:

        - name: records

          in: formData

          required: true

          type: file

          description: The record(s) to be validated

        - name:  channel

          in: formData

          type: integer

          required: true

          description: The kind of validation to be done

        - name: format

          in: formData

          type: string

          required: false

          description: A specfic format to be used

          default: The default validation of the specific channel

        - name: validation_level

          in: formData

          type: integer

          required: false

          description: represents a numeric channel specific level of
validation to be used (with higher numbers indicating more). Default
```



```
will use the default validation of the specific channel. Current
defined values are

            **00** None

            **10** Schema Errors only

            **20** Schema and Inclusion Logic

            **30** Schema, Inclusion and Minor Logic

            **40** Schema, Inclusion, Major, and Minor Logic

        - name: validation_mode
          in: formData
          description: RECORD, FILE(aggregate) or BOTH. Default is FILE
if multiple record, RECORD if single record
          type: string
          required: false
          enum:
```



```
            - RECORD
            - FILE
            - BOTH
        - name: use_async
          in: formData
          required: false
          type: boolean
          description: Should a response be directly returned or a
handle to a response that can be used at a later point. If default then
for a file an async mechanism may be used
        - name: api_key
          in: formData
          required: true
          type: string
          description: Api key to be used for security
      consumes:
        - multipart/form-data
      produces:
        - application/json
      # Expected responses for this operation:



      responses:
        # Response code
        200:
          description: Successful response
          schema:
```



```
            $ref: "#/definitions/ValidationResponse"
  # This is a HTTP operation


  /validator/response:
    get:
      description: |
        Takes in a handle from an async response and returns a new
result determine if ready
      consumes:
        - application/json
      produces:
        - application/json
      parameters:
        - name: handle
          in: query
          description: Reponse handle from prior call
          required: true
          type: string
        - name: api_key
          in: formData
          required: true
          type: string
          description: Api key to be used for security


      # Expected responses for this operation:
      responses:
```



```
        200:
          description: Successful response
          # A schema describing your response object.
          # Use JSON Schema format
          schema:
            $ref: "#/definitions/ValidationResponse"



definitions:
  MessageStructure:
    type: object
    properties:
      message_type:
        type: string
        description: The type of message. Validator specific but
usually one of All, Schema, Inclusion, Major, Minor
        enum:
          - ALL
          - SCHEMA
          - INCLUSION
          - MAJOR
          - MINOR
      id:
        type: integer
        description: An numeric code representing the message
      message:
```



```
        type: string
        description: The string representation of the message
      tag:
        type: string
        description: Optional xml tag associated with the message
      record_number:
        type: integer
        description: Which record (if any) is this message associated
with


  ValidationResponse:
    type: object
    properties:
      status:
        type: string
        description: The status of the request. Either SUCCESS,
PENDING, PROCESSING, SUCCESS, ERROR_GENERAL, ERROR_INVALID, FAILURE.
          If status is PENDING then the response handle will be
populated with a handle to use for retrieval
        enum:
          - SUCCESS
          - PENDING
          - PROCESSING
          - ERROR_GENERAL
          - ERROR_INVALID
          - FAILURE
      response_handle:
```



```
        type: string
        description: If an async request is done, then this will
represent the uuid of the request to later retrieve
      version:
        type: string
        description: The internal cloud version used to validate
      validator:
        type: string
        description: The specific validator used for this request
      channel:
        type: number
        description: The channel number used for this request
      num_records_success:
        type: number
        description: Number of records successfully processed
      num_records_failed:
        type: number
        description: Number of records which failed processing
      messages:
        type: array
        items:
          $ref: "#/definitions/MessageStructure"
```

