API reference

V-Spark 4.4.1-1

Medallia Voci

Contents

API reference

Use V-Spark's REST API to

- Automate the upload of audio and optional metadata into V-Spark.
- Automate the download of fully annotated transcripts out of V-Spark.
- Examine or modify a V-Spark installation using API calls that
  - Retrieve, update, or delete configuration for companies, organization, folders, and applications.
  - Retrieve, update, or delete configuration for users.
  - List summary information for those entities.
  - Retrieve status information (in JSON or CSV format) about companies and folders.
  - Search transcription records by folder or organization to identify, count, and retrieve record matches.

The V-Spark API is identical for both on-premises and cloud-based deployments. To review all API functions, refer to Endpoints.

Endpoints

The values for `$host` and `$port`, and the protocol (`http` or `https`) used to access the V-Spark API varies by installation. The default protocol is `http`. The default port is `3000`. Every request requires a valid token.

V-Spark API endpoint	Usage	HTTP Methods
`http://$host:$port/`appedit	Retrieve and modify category configuration for analytics applications.	GET, POST
`http://$host:$port/`appmatches	Retrieve all of the phrases in a single transcript that matched application categories during analytics processing.	GET
`http://$host:$port/`appstats	Retrieve daily statistics for agent analytics application and category scores.	GET
`http://$host:$port/`config	Create, modify, and delete V-Spark system settings, companies, organizations, folders, applications, and user accounts.	GET, POST, DELETE
`http://$host:$port/`list	Retrieve name-level information for V-Spark company, organization, and folder entities, along with applications and user accounts.	GET
`http://$host:$port/`metadata	Modify and delete a transcript's metadata values.	PATCH, DELETE

V-Spark API endpoint	Usage	HTTP Methods
`http://$host:$port/`request	Retrieve transcription and analytics results.	GET
`http://$host:$port/`search	Search for transcript and audio records.	GET, POST, DELETE
`http://$host:$port/`stats	Retrieve daily statistics for V-Spark folders.	GET
`http://$host:$port/`status	Retrieve V-Spark folder and job status.	GET
`http://$host:$port/`sysinfo	Retrieve V-Spark system status, configuration, and software version.	GET
`http://$host:$port/`transcribe	Submit files to V-Spark for ASR transcription and text analytics.	POST

Refining requests by entity

Most endpoints accept the names of V-Spark company, organization, folder, application, or user account entities as path parameters. When refining requests with these entities, note that

- If no company short name (`$co_short`) is included with the request, the response includes all data for every company for which the request's token has read permissions.

- If a company short name is included with the request, the JSON response includes only that company's data.
- If any entity name is included with the request, the JSON response includes only information associated with the entity or entities in the request path.
- If any of these path parameters is invalid, the request returns an error.

Content types

Some request content varies by endpoint. Exceptions are noted in individual endpoint references.

Method	Expects	Returns
POST	application/json multipart/form	text/html
PATCH	application/json	text/html
GET	application/json	text/html application/json
DELETE		text/html
Errors		text/html

/appedit

Use `/appedit` to

- Retrieve the category configuration for an existing application.
- Modify the category configuration for an existing application.

> **NOTE:** The POST JSON contains the entire analytics application, including every category. `/appedit` cannot accept a configuration for only one category at a time.

`/appedit` changes application categories. To change application configuration, or to create a new application with the API, use the `/config` endpoint as described in Create and configure analytics applications.

Synopsis

```
GET /appedit/$co_short/$org_short/$app_name?token=$token > $json
POST /appedit/$co_short/$org_short/$app_name?token=$token --data @$json
```

`$co_short`

Company short name used to filter the request.

`$org_short`

Organization short name used to filter the request.

`$token`

Authorization token with the required read or write permissions for the request.

`$app_name`

Name of the application to be queried.

`$json`

Path to the request's JSON file.

Example cURL requests

The following example request retrieves the category configuration of an application named `AppEdit Test`, located under the `Technologies` company, in the `Technologies-RD` organization.

```
curl -s "http://example.company.com:3000/appedit/Technologies/Technologies-RD/AppEdit%20Test?token=TOKEN" > AppEdit-
Test.json
```

The following example request updates the category configuration of an application named `AppEdit Test`, located under the `Technologies` company, in the `Technologies-RD` organization, from the JSON file **`AppEdit-Upd.json`**:

```
curl -s -X POST -H "Content-Type:application/json" --data @AppEdit-Upd.json \
    "http://example.company.com:3000/appedit/Technologies/Technologies-RD/AppEdit%20Test?token=TOKEN"
```

Python example

Use api_post.py with these command-line arguments:

```
python api_post.py $host $token /appedit/$co_short/$org_short/$app_name $json
```

`$host`
Hostname or URL for the system running V-Spark.

`$token`

Authorization token with the required read or write permissions for the request.

`$co_short`

Company short name used to filter the request.

`$org_short`

Organization short name used to filter the request.

`$app_name`

Name of the application to be queried.

`$json`
Path to the request's JSON file.

Example application JSON

The following sample JSON defines an application named `New AppEdit Test`:

```
{
    "Technologies": {
        "Technologies-RD": {
            "New AppEdit Test": {
                "created": "2022-10-10",
                "defaultscoretype": "Hit/Miss",
                "enabled": "on",
                "folders": [
                    "AutoTests"
                ],
                "template": "custom"
            }
        }
    }
}
```

/appmatches

Use `/appmatches` to retrieve the full text of phrase matches for applications associated with a transcript.

`/appmatches` returns a JSON object with phrase match and score data for leaf categories defined in a given transcript's applications. By default, `/appmatches` returns match and score data for each leaf category and for every application associated with the specified transcript.

To limit results to a specific set of applications, categories, or subcategories, specify those entities in JSON data submitted with the request.

For applications with multiple categories, the API returns phrase match data for the categories specified in the request. The response also includes scores without phrase match data for any other categories in that application. A queried transcript with no application matches will return a JSON object with file information, but no `scorecard` object data.

Synopsis

```
GET /appmatches/$co_short/$org_short/$tId?token=$token
GET /appmatches/$co_short/$org_short/$tId?token=$token --data @$json
```

`$co_short`

Company short name used to filter the request.

`$org_short`

Organization short name used to filter the request.

`$tId`

UUID for the transcript to be queried.

> **`$token`**

> Authorization token with the required read or write permissions for the request.

> **`$json`**
> Path to the request's JSON file.

Example cURL requests

The following example specifies a `tId` of 3 in the company `Co-Short` and the organization `Org-Short`. The example does not include a JSON-formatted list of applications and categories, so the response includes all applications linked to the transcript's folder.

```
curl 'http://example.com:3000/appmatches/Co-Short/Org-Short/3?token=123'
```

The output of the preceding example is a JSON object with matched phrases and score data for all applications associated with that transcript.

The following example queries a `tId` of 6 in the company `Co-Short` and the organization `Org-Short`. The example also includes JSON data for specific applications and categories:

```
curl 'http://example.com:3000/appmatches/Co-Short/Org-Short/6?token=123' --header 'Content-Type: application/json' --
data-raw '{"Agent Scorecard":["Compliance.Recording"], "TopLevel":["NextLevel.LowestLevel"]}'
```

The output of the preceding example is a JSON object with matched phrases and score data for the following entities:

- The `Recording` subcategory of the `Compliance` category of the `Agent Scorecard` application.
- The `LowestLevel` subcategory of the `NextLevel` category in the `TopLevel` application.

To use a file instead of in-line JSON data, include the filename and path of the file with the request, as in the following example:

```
curl 'http://example.com:3000/appmatches/Co-Short/Org-Short/6?token=123' --header 'Content-Type: application/json' --
data @phrase-request.json
```

The preceding example references the file `phrase-request.json` instead of in-line JSON data. This file must be saved locally on the system that originates the request.

Example JSON input

The following example JSON specifies application, category, and category names to be submitted with a request to `/appmatches` :

```
{
  "APP1": [
    "Category1.Leaf1",
    "Category2.Leaf2"
  ],
  "APP2": [
    "Category3"
  ]
}
```

The preceding example specifies the following entities for phrase match data in request output:

- `Leaf1` , a subcategory of `Category1` , which is a category in the application `APP1` .
- `Leaf2` , a subcategory of `Category2` , which is a category in the application `APP1` .
- All subcategories defined for `Category3` , which is a category of the application `APP2` .

Because only two subcategories are specified for `APP1` , phrase matches will be returned for only those two subcategories, including any subcategories below them. Phrase matches will be returned for all subcategories under `APP2` 's `Category3` . The request returns only scores for other categories in `APP1` and `APP2`.

Any category below the ones specified will be included with results. Subcategories are referenced using the dot operator (for example, the `.` in `Category2.Leaf2`), and this operator may be used to reference as many subcategories as the application contains. For example, the entry `Category5.Subcategory5.Subcategory6.Subcategory7.LastLeaf` could be used to specify the deeply nested `LastLeaf` without including phrase matches for its parent subcategories.

Example JSON output

JSON data output from `/appmatches` includes the following information:

- Top-level fields for the transcript's audio `filename`, `tId`, and `organization`.
- The top-level `scorecard` object, which contains all application score, match, and phrase data.
- Fields below `scorecard` for each included application category and subcategory.
- A `score` field for each category's overall score.
- A `match_data` object that includes all `matches` objects for that application, along with the query phrase in the application that triggered the match.
- A `matches` object that includes the start and end times for the utterance containing the phrase match, along with the phrase's speaker (either agent or client).

> **NOTE:**
>
> Applications may specify phrases to be excluded from scoring. These exclude phrases can be identified in the application by the `-` prefix.
>
> As of V-Spark 4.2.0, when an application with leaf-level exclude phrases is requested in `/appmatches` output, that output contains all exclude phrases and their matches, even if those matches do not impact category scores.
>
> Exclude phrases only impact scoring when they are in the same speaker turn as include phrases, and speaker turns that contain only exclude phrases do not affect category scores. So, to identify exactly which exclude phrases affect scores in `/appmatches` output, look for speaker turns that contain both exclude and include phrases.

The following example JSON shows the data returned from an `/appmatches` request with relatively few matches. The basic structure for applications illustrated in the `you` section of the following example is repeated at the JSON object's lower subcategory levels in the `subcategories` section when applicable.

```
{
  "filename": "example.wav",
  "tId": 6,
  "organization": "Org-Short",
  "scorecard": {
      "you": {
              "subcategories": {},
              "score": 3,
              "match_data": [
                  {
                      "matches": [
                          [
                              1249.24,
                              1252.18,
                              "agent",
                              "you have a lock on you when you have a wonderful day. Okay,"
                          ],
                          [
                              1248.13,
                              1249.88,
                              "client",
                              "Okay. Thank you. Pamela for your help."
                          ]
                      ],
                      "phrase": "all: you ~s>1240"
                  }
              ]
          }
      }
}
```

The following table describes the hierarchy and contents of `/appmatches` JSON output:

Elements in **`/appmatches`** JSON output

Element	Type	Description
`filename`	string	Source audio's filename.
`tId`	value	Source transcript's UUID.
`organization`	string	Short name of the organization associated with the transcript.
`scorecard`	object	Stores one key-value pair for each application with score and match data. The key is the name of the application, and the value is a JSON object that contains the results from each matched application category.
`APPLICATION-NAME`	object	Stores the phrase and match data for the named application.
`subcategories`	object	Stores results from each matched application subcategory contained in its parent category object. Each subcategory with match data repeats the structure of the `APPLICATION-NAME` object at lower levels.

Element			Type	Description
	`score`		value	Overall category score.
	`match_data`		array	Stores one object for each phrase in the application category.
		`matches`	array	Phrase match's start time, end time, speaker, and utterance.
		`phrase`	string	Application phrase that triggered the match.

/appstats

Use `/appstats` to retrieve daily statistics for agent application and category scores. Responses include the number of calls the agent had and the overall app scores, along with any category scores specified using optional parameters.

Synopsis

```
GET /appstats/$co_short/$org_short/$app_name?token=$token&$options...
GET /appstats/$co_short/$org_short/$app_name/$folder?token=$token&$options...
```

`$co_short`

Company short name used to filter the request.

`$org_short`

Organization short name used to filter the request.

`$app_name`

Name of the application to be queried.

`$folder`
Folder used to filter the request.

`$token`

Authorization token with the required read or write permissions for the request.

> **`$options`**
> Optional parameters used to further refine the request and results.

Optional parameters

> daterange=**`$start`**-**`$end`**

> Filters the request by the dates specified in the range [`$start`, `$end`]. Values for `$start` and `$end` are optional, but the hyphen between them is required.
>
> The range specified by `$start` and `$end` may be expressed using any combination of years, months, days, hours, minutes, and seconds, expressed as YYYYMMDD[HHmmss]. Date ranges are always assumed to be positive (where `$start` is less than `$end`).
>
> `daterange` filters by values in the transcript record's `datetime` field. Data for the `datetime` field is stored using the organization's time zone, which may vary by organization.
>
> No verification is done to ensure that `daterange` values are correct; invalid date ranges will simply return no values. If `$start` is not specified, a default value of 01 January, 1900 is used. If `$end` is not specified, the current date is used.

> category=CATEGORY

> Enables you to return Agent Stats for a particular category. To specify a lower-level category, specify `CATEGORY` as a string that contains the full "path" to the lower-level category, as a period-separated list. For example, if querying an app that uses the Agent Scorecard template, a valid category name would be `Communication Skills.Client Informed`. Querying a particular category will always also return scores for the category's upper levels.

> depth=DEPTH

> Enables you to specify how many lower level categories you would like to return in results:

> 0
>
> Return only the level of the category specified. This is the default value if the category option is specified.
>
> n
>
> Where n is a positive non-zero integer, return the specified number of lower levels of the category
>
> -1
>
> Return all levels of category stats. This is the default value if no category option is specified.

agents=AGENTID

Enables you to specify the agent(s) for which to retrieve scores. If you want to retrieve scores for more than one agent, separate the `AGENTID`s with commas. For example:

```
agents=348,227,042
```

zeros

Whether or not to include in the JSON that is returned categories for which the agent did not receive any score.

> NOTE: This parameter refers to returning zero scores for `CATEGORIES`. If a date in the specified `daterange` does not contain any calls, no scores of any sort will be returned for that date.

> true
>
> Include categories in which the agent did not score, and report the score for that category as zero.

> > false
>
> > Exclude (from the JSON that is returned) categories in which the agent did not receive a score.

Example cURL requests

The following example retrieves scores from an application named `Agent ScoreCard` for records in the single-day `daterange` of 20210925 (September 25, 2021) in the `Docs` company and `Docs-Testing` organization.

```
curl -s 'http://example.company.com/appstats/Docs/Docs-Testing/Agent%20Scorecard?token=TOKEN&daterange=20210925'
```

The following example retrieves category scores for lower-level application categories. The request specifies a `depth` of 1, so it retrieves only the first level of category scores.

```
curl -s 'http://example.company.com/appstats/Docs/Docs-Testing/Agent%20Scorecard\
    ?token=TOKEN&daterange=20210925&category=Communication%20Skills.Client%20Informed&depth=1'
```

Python example

Use the api_get_test.py with these command-line arguments:

```
python api_get_stats.py $host $token /appstats/$co_short/$org_short/$app_name
'&daterange=20211005&category=Politeness'
```

This call uses the `daterange` parameter to limit results to those from files processed on a single date (October 5, 2021), and includes only the `Politeness` category.

> **`$host`**
> Hostname or URL for the system running V-Spark.

> **`$token`**

> Authorization token with the required read or write permissions for the request.

> **`$co_short`**

> Company short name used to filter the request.

> **`$org_short`**

> Organization short name used to filter the request.

> **`$app_name`**

> Name of the application to be queried.

Example JSON output

This example output shows application category scores for one call from an agent with identifier `001` . Categories without a score are not included in JSON output.

```
[
  {
    "date": "20170925",
    "agents": 3,
    "001": {
       "calls": 1,
        "overall": {
          "ncalls": 1,
          "hitmiss": "1.0000",
          "coverage": "0.2890",
          "duration": "0:11:55",
          "silence": "0:03:13"
         },
         "Communication Skills": {
            "ncalls": 1,
            "hitmiss": "1.0000",
            "coverage": "0.5667",
            "duration": "0:11:55",
            "silence": "0:03:13"
          },
          "Effectiveness": {
            "ncalls": 1,
            "hitmiss": "1.0000",
            "coverage": "0.4000",
            "duration": "0:11:55",
            "silence": "0:03:13"
          },...
      },...
   }
]
```

/config

Use `/config` to

- Retrieve, modify, or delete configuration information for V-Spark entities
    - Companies
    - Organizations
    - Folders
    - Analytics applications
- Retrieve and modify a more limited set of user account settings.
- Retrieve the system's `readonly` status.

`/config` POST requests must include entity information as top-level JSON data. Higher-level entity information does not need to be included in JSON POST data when entity names are included as path parameters.

Data written to V-Spark with `/config` is additive—that is, if objects in request JSON data exist, they are configured to use the request JSON. Objects that do not exist are created.

Synopsis

```
GET | POST | DELETE /config?token=$token
GET | POST | DELETE /config/users?token=$token
GET | POST | DELETE /config/orgs?token=$token
GET | POST | DELETE /config/folders?token=$token
GET | POST | DELETE /config/apps?token=$token
GET | POST | DELETE /config/$co_short?token=$token
GET | POST | DELETE /config/$co_short/orgs?token=$token
GET | POST | DELETE /config/$co_short/folders?token=$token
GET | POST | DELETE /config/$co_short/apps?token=$token
GET | POST | DELETE /config/$co_short/users/$user?token=$token

GET | POST | DELETE /config/$co_short/$org_short?token=$token
GET | POST | DELETE /config/$co_short/$org_short/folders?token=$token
GET | POST | DELETE /config/$co_short/$org_short/apps?token=$token
GET | POST | DELETE /config/$co_short/$org_short/$folder?token=$token
GET | POST | DELETE /config/$co_short/$org_short/apps/$app_name?token=$token

GET /config/system/readonly?token=$token
```

`$token`

Authorization token with the required read or write permissions for the request.

`$co_short`

Company short name used to filter the request.

`$org_short`

Organization short name used to filter the request.

`$folder`

Folder used to filter the request.

`$user`

The `username` of a V-Spark user account.

`$app_name`

Name of the application to be queried.

Endpoints

`/config` endpoints accept requests to retrieve, modify, or delete configurations for organizations, folders, applications, and user accounts, and to retrieve V-Spark's system configuration.

> NOTE: POSTing a configuration for an entity that does not exist creates a new entity with those attributes.

/config

Returns information about all companies in a V-Spark installation

/config/orgs

Returns information about all organizations that have been defined under companies in a V-Spark installation

/config/folders

Returns information about all folders that have been defined under organizations in a V-Spark installation

/config/apps

Returns information about all applications that have been defined in a V-Spark installation

/config/users

Use `/config/users` to retrieve, create, and configure V-Spark user accounts, to retrieve user account status, to retrieve the authorization method used for login, and to retrieve or configure the permissions that they have in V-Spark and within each company.

> NOTE: Users are associated with companies. Deleting companies in a V-Spark installation also deletes any users associated with those companies.

/config/system/readonly

Returns information about whether a V-Spark installation is or is not running in "read only" mode. You cannot call the `/config/system` API alone without calling `/config/system/readonly` API.

Example cURL requests

GET

This example request retrieves the configuration for all of the organizations the token has permission to read.

```
curl -s http://example.company.com:3000/config/orgs?token=TOKEN
```

The following example request retrieves the configuration for all of the organizations associated with the company `DocTestCo` the token has permission to read.

```
curl -s http://example.company.com:3000/config/DocTestCo/orgs?token=$token
```

The following example request retrieves the account configuration for the `testUser` user account. The JSON after the cURL request shows an example JSON user data response

```
curl -s 'http://example.company.com:3000/config/DocTestCo/users/testUser?token=$token'

 {
    "name": "bscott284@medallia.com",
    "email": "bscott284@medallia.com",
    "company": "MedalliaSSO",
    "auth": {
        "verified": true,
        "disabled": false,
        "method": "standard"
    },
    "permissions": {
        "Medallia": {
            "orgs": {
                "Medallia-Demo": [
                    "read",
                    "write"
                ]
            }
        }
    }
}
```

POST

The following example request creates or modifies the entity defined in the local file `post-config.json`.

```
curl -s -X POST -H "Content-Type:application/json" "http://example.company.com/config?token=TOKEN" --data @post-
```

```
config.json
```

The following example request creates or modifies the `testUser` user account to the configuration defined in the local file `user-config.json`.

```
curl -s -H "Content-Type:application/json" -X POST 'http://example.company.com:3000/config/DocTestCo/users/
testUser?token=$token' --data @user-config.json
```

DELETE

Use DELETE with `/config` to delete company, organization, folder, application, user, and system entities and configurations.

> WARNING: Entity deletion is permanent and cannot be undone.

The following example request deletes the `testUser` user account from the `Docs-Org` organization.

```
curl -s -X DELETE 'http://example.company.com:3000/config/Docs-Co/Docs-Org/users/testUser?token=$token'
```

The following example request deletes the `Test01` folder and all data below it.

```
curl -s -X DELETE 'http://example.company.com:3000/config/Docs-Co/Docs-Org/Test01?token=TOKEN'
```

The following example request deletes the `Docs-Org` organization and all folders and other data associated with it.

```
curl -s -X DELETE 'http://example.company.com:3000/config/Docs-Co/Docs-Org?token=TOKEN&multi=true&tree=true'
```

Use these parameters when deleting multiple entities with `/config`.

> **tree**
>
> Optional. Include `tree=true` with DELETE requests to delete non-empty entities like companies or organizations that contain lower-level folders or applications. By default, `tree=false` and entities containing lower-level entities cannot be deleted.
>
> **multi**
>
> Required to enable the deletion of multiple entities with a single request.

Python example

Use these sample Python scripts to test `/config` operations:

api_get_config.py

This example command saves system company configuration to an output file.

```
python api_get_config.py $host $token /config $http_code $output_file
```

get_deep_config.py

Uses the root token to retrieve the system's company token list and entity hierarchy.

```
python get_deep_config.py $host:$port $token
```

api_post_config.py

The first example command writes top-level company JSON to V-Spark. The specified company is created or updated to the supplied configuration.

```
python api_post.config.py $host $token /config 200 $input_file
```

The next example command writes lower-level organization JSON to V-Spark. The specified organization is created or updated to the supplied configuration.

```
python api_post_config.py $host $token /config/orgs 200 $input_file
```

`$host`
Hostname or URL for the system running V-Spark.

`$port`
Port configured for the host system. Default `3000` .

`$token`

Authorization token with the required read or write permissions for the request.

`$http_code`
The HTTP code expected to be returned with the response.

`$input_file`
File path for query input.

`$output_file`
File path for query output.

Entity JSON

`/config` operations use JSON data to represent V-Spark entities that resemble these examples.

/config

The sample JSON in this section shows sample output for a single company from the `/config` API.

Sample Company output from the /config API

```
"DocTestCo": {
    "allowedmodels": [
        "eng1:callcenter",
        "spa1:spa1_callcenter"
    ],
    "apptemplate": [
        "Agent Scorecard",
        "Call Categorization",
        "Call Drivers",
        "Customer Experience"
    ],
    "cloudtoken": "",
    "cloudmodels": [],
    "created": "2017-05-18",
    "limithours": -1,
    "name": "Doc Test Co",
    "retention": -1,
    "status": "OK",
    "servers": [
        "asrsrvr1"
    ],
    "uuid": "077d93ffd9b902b2cb7c6a0c521fd42c"
},...
```

NOTE: Sample JSON files in this document use *ellipses* (`...`) to indicate where more than one of a certain type of section can be present in a JSON file of that type.

This excerpt from the output of calling the `/config` API is very similar to the output that you would have received had you requested information about a single company by calling an API such as the `/config/DocTestCo` API on a V-Spark installation where the "Doc Test Co" company (with the company short name, "DocTestCo") had been defined. The latter call would have returned the following, which differs only in that it does not need to identify the short name of the company that it refers to because it was specified in the URL.

Sample Company output from the /config/CO_SHORT API

```
{
    "uuid": "077d93ffd9b902b2cb7c6a0c521fd42c",
    "name": "Doc Test Co",
    "created": "2018-06-07",
    "limithours": -1,
    "cloudtoken": "",
    "cloudmodels": [],
    "allowedmodels": [
        "eng1:callcenter",
        "spa1:callcenter"
    ],
    "servers": [
        "asr-wvh.office.company.com",
        "asrsrvr1",
        "asrsrvr8",
        "http://asrsrvr8:17171"
    ],
    "retention": -1,
    "apptemplate": [
        "Agent Scorecard",
        "Call Categorization",
        "Call Drivers",
        "Customer Experience"
    ],
    "status": "OK"
}
```

/config/CO_SHORT Fields

Name	Type	Values	Description
`uuid`	**Read-Only** String	33 character token	The authorization token for this company. Use the `uuid` to retrieve or modify data about any organization, folder, apps, or user that have been defined under this company. This field is added by V-Spark when the company is created. `"uuid": "077d93ffd9b902b2cb7c6a0c521fd42c"`
`name`	**REQUIRED** string when creating a new company		The full display name of the company. `"name": "Doc Test Co"`
`created`	**Read-Only** String	Date, in YYYY-MM-DD format	The year, month, and day that the company was created in V-Spark. This field is added by V-Spark when the company is created. `"created": "2018-06-07"`

Name	Type	Values	Description
`limithours`	**REQUIRED** Integer when creating a new company	"`-1`" = no limit	The maximum number of audio hours this company can process through V-Spark. Once that limit has been reached, the company can no longer process new audio, but users can still use V-Spark to examine existing calls. `"limithours": -1`
`servers`	**REQUIRED** list of strings if `cloudtoken` is not set	Hostnames or URLs	Networked computers this company uses as hosts for ASR. This field should only be set if `cloudtoken` is not set. `"servers": [ "asr-wvh.office.company.com", "asrsrvr1", "asrsrvr8", "http://asrsrvr8:17171" ]`
`allowedmodels`	List of strings	Limited to installed models	Transcription models that are available to this company when processing audio on `servers`. This field should only be set if `servers` is set.

Name	Type	Values	Description
			`"allowedmodels": [ "eng1:callcenter", "spa1:callcenter" ]`
`cloudtoken`	**REQUIRED** string if `servers` is not set	33 character token	The authorization token this company uses when connecting to V-Cloud servers. If this company is using `servers` no cloud token will be listed. If you are creating the company, and the company will be processing audio on V-Cloud this field must be defined. This field should only be set if `servers` is not set. `"cloudtoken": ""`
`cloudmodels`	List of strings	Limited to installed models	Transcription models this company can use when processing audio on V-Cloud servers. This field should only be set if `cloudtoken` is set and custom models will be used.

Name	Type	Values	Description
			`"cloudmodels": []`
`retention`	Integer **REQUIRED when creating a new company**	" `-1` " = no limit	The maximum number of days transcription data can be retained by V-Spark for organizations within this company before the data is deleted. A value of `-1` indicates that this company has no limit, and that data is retained indefinitely. `"retention": -1`

`apptemplate`	List of strings	Limited to names of installed templates	This **optional** list defines the application templates that are available for this company.
			`"apptemplate": [`
			`    "Agent Scorecard",`
			`    "Call Categorization",`
			`    "Call Drivers",`
			`    "Customer Experience"`
			`]`

Name	Type	Values	Description
`status`	**Read-Only** string	`OK`, `deleting`, `deleting ( PERCENTAGE )`	DELETE status information about the company, useful for long-running operations such as a DELETE. A status of "OK" indicates that any operations on the company have completed their work. `"status": "OK"`

/config/users

This excerpt is sample JSON output for a single user from the `/config/users` API.

Sample User Information from the /config/users API

```
"Testing": {
    "joe.user": {
        "auth": {
            "disabled": false,
            "verified": true,
            "method": "standard"
        },
        "company": "Testing",
        "email": "joeuser@example.com",
        "name": "Joe User",
        "permissions": {
            "DocTestCo": {
                "all": [
                    "read",
                    "write"
                ]
            },
            "Testing": {
                "all": [
                    "read"
                ],
                "orgs": {
                    "Testing-CallbackTest": [
                        "write"
                    ],
                    "Testing-ApplicationTesting": [
                        "write"
```

```
                              ]
                          }
                      }
                  }
              }
          }...
```

The identifier for each user account object is the username that identifies this account. When creating new user accounts, keep in mind that the username for each user in V-Spark must be unique to the V-Spark installation.

/config/users Fields

Name	Type	Values	Description
`auth`		NA	The authorization status of this account and authentication method used to verify the identity of the user when they log in. `"auth": { "disabled": false, "verified": true, "method": "standard", "password": "4s+7yaRf" },`
`disabled`	Boolean	true, false	Either "false," denoting an active account, or "true," denoting an account that has been disabled or has not yet been enabled after creation.
`verified`	Boolean	true, false	Either "false," denoting a requested account that has not yet been approved, or "true," denoting an account that has been verified and approved by a System or Company admin.
`method`		standard, oidc	How the user's identity will be authenticated when they log in.

```
"auth": {
    "disabled": false,
    "verified": true,
    "method": "standard",
    "password": "4s+7yaRf"
},
```

Name	Type	Values	Description
			A value of "standard" indicates that internal V-Spark authentication is used. Any other value indicates the integrated authentication method that should be used.
`password`	String		Only be provided when creating a new account with standard authentication. This value will serve as the account's initial password.
`company`	String		**REQUIRED when creating a new account**. The short name of the company within which this account exists. `"company": "Testing",`
`email`	String		**REQUIRED when creating a new account**. The fully qualified email address associated with this account. The email address for each user in V-Spark must be unique to the V-Spark installation. `"email": "joeuser@example.com",`

Name	Type	Values	Description
`name`	String		**REQUIRED when creating a new account**. The name of the person who uses this account. `"name": "Joe User",`

`permissions`			Permissions that this user account has for the companies and organizations in the V-Spark installation.
			
			In the following example, the user account has read and write permissions to all organizations within the "DocTestCo" company, read permissions to all organizations within the "Testing" company, and additional write permissions to the "CallbackTest" and "ApplicationTesting" organizations that are within the "Testing" company.
			`"permissions": {`
			`    "DocTestCo": {`
			`        "all": [`
			`            "read",`
			`            "write"`
			`        ]`
			`    },`

Name	Type	Values	Description

```
    "Testing": {
        "all": [
            "read"
        ],
        "orgs": {
            "Testing-CallbackTest": [
                "write"
            ],
            "Testing-ApplicationTesting": [
                "write"
            ]
        }
    }
}
```

The output shown in the previous excerpt is very similar to the first part of the output that you would have received had you called the `/config/TestCompany/users` API on a V-Spark installation where the "Test Company, Inc." company (with the company short name, "Testcompany") had been defined. The call would have returned JSON, which only differs from the previous excerpt in that it does not need to identify the short name of the company that it refers to, since you have specified that value in the URL.

The identifier for each user account object is the username that identifies this account. When creating new user accounts, keep in mind that the username for each user in V-Spark must be unique to the V-Spark installation.

Sample User output from the /config/CO_SHORT/users API

```
{
    "joe.user": {
        "company": "Testing",
        "email": "joe.user@company.com",
        "name": "Joe User",
        "auth": {
            "disabled": false,
            "verified": true,
            "method": "standard"
        },
        "permissions": {
            "DocTestCo": {
                "all": [
                   "read",
                   "write"
                ]
            },
        },...
    }
}
```

Example /config/users/ JSON

```
 {
   "test.user.07": {
       "auth": {
          "disabled": false,
          "verified": true,
          "method": "standard"
       },
       "company": "DocTestCo",
       "email": "test.user.07@example.com",
       "name": "Another Automated Test User",
       "permissions": {
           "DocTestCo": {
               "all": [
                   "read",
                   "write"
               ]
           }
       }
   }...
 }
```

/config/system/readonly

IMPORTANT: Because only the `/readonly` API exists under `/config/system`, there is no more general `/config/system` API. Attempting to GET, POST, or DELETE to the `/config/system` API directly will return HTTP error code 400.

Readonly mode enables administrators to perform maintenance or diagnose performance problems while a V-Spark installation is still running. While a V-Spark system is in readonly mode, no new data can be processed and no changes can be made to the V-Spark installation. V-Spark can still be used to examine existing data that has already been processed.

The `/config/system/readonly` API reports on the readonly status of the system, and displays the system-wide message that will be shown in V-Spark to notify users that the system has been put into readonly mode.

Sample output from the /config/system/readonly API

```
{
   "message": "Sample message about readonly mode",
   "status": false
}
```

/config/system/readonly Fields

Name	Type	Values	Description
`message`	String		When any user logs into this V-Spark installation while the system is in readonly mode, this message will be displayed. `"message": "Sample message about readonly mode",`
`status`			If set to "true," the system is in read-only mode. Putting a V-Spark installation into read-only mode only affects the V-Spark installation. The rest of the processes on the host where V-Spark is installed continue to operate normally. If set to "false," the system is not in read-only mode. `"status": false`

/config/orgs

The first JSON excerpt in this section shows sample output for a single organization from the `/config/orgs` API.

Sample Organization output from the /config/orgs API

```
"DocTestCo": {
    "DocTestCo-DocTesting": {
        "company": "DocTestCo",
        "created": "2017-05-18",
        "name": "Doc Testing",
        "retention": -1,
        "status": "OK",
        "timezone": "US/Eastern"
    },...
},...
```

When creating a new organization using the `/config/orgs` API, all fields that are not read-only are required.

/config/orgs Fields

Name	Type	Values	Description
`company`	**Required** String when creating		The "short name" of the company to which the organization belongs. `"company": "DocTestCo",`
`created`	**Read-Only** String		The date, in YYYY-MM-DD format, that the organization was created. This field is added by V-Spark when the organization is created. `"created": "2017-05-18",`
`name`	String		The full display name of the organization. `"name": "Doc Testing",`
`retention`			The maximum number of days transcription data is retained by

Name	Type	Values	Description
			V-Spark for this organization before the data is deleted. The retention period specified for the organization must be less than or equal to the retention period of the organization's owning company. A value of `-1` indicates that this organization has no limit, and that data is retained indefinitely. `"retention": -1,`
`status`			High-level status information about the organization, useful for long-running operations such as a DELETE. A status of "OK" indicates that any operations on the organization have completed their work. `"status": "OK",`
`timezone`			The time zone in which this organization should be considered to exist. This is usually the time zone of the main office of the organization. This does not just affect the time and date as displayed in V-Spark, but also affects the time at which certain actions (such as report generation) occur. This value must be a valid "TZ database name" for a time zone. Refer to the List

Name	Type	Values	Description
			of tz database time zones for more information. `"timezone": "US/Eastern"`

The excerpt from the output of calling the `/config/orgs` API shown previously is very similar to the output that you would have received had you requested information about a single organization by calling an API URL such as the `/config/DocTestCo/DocTestCo-DocTesting` API on a V-Spark installation where the "Doc Test Co" company and "Doc Testing" organization (with the company short name, "DocTestCo" and the Organization short name of "DocTestCo-DocTesting") had been defined.

The latter call would have returned JSON like the sample below, which differs only from the output shown previously in that it does not need to identify the short name of the company and organization that it refers to because it was specified in the URL.

Sample Organization output from the /config/CO_SHORT/ORG_SHORT API

```
{
          "company": "DocTestCo",
          "created": "2017-05-18",
          "name": "Doc Testing",
          "retention": -1,
          "timezone": "US/Eastern"
          }
```

/config/folders

The following JSON shows sample output for a single folder from the information retrieved via the `/config/folders` API.

Sample Folder output from the /config/folders API

```
"DocTestCo": {
    "DocTestCo-DocTesting": {
        "Test01": {
            "apps": [],
            "asroptions": {
                "billing": "customerX"
            },
            "audiotype": "Mono",
            "callback": {
                "aws_id": "12345678901234567890l",
                "aws_secret": "12345678901234567890l/12345678901234567890",
                "posturl": "S3:///joeuser/test",
                "sendaudio": "no",
                "sendtext": "no"
            },
            "created": "2017-05-18",
            "custom_meta": [],
            "mode": "active",
            "modelchan0": "eng1:callcenter",
            "nspeakers": 1,
            "purifyaudio": true,
            "purifytext": true,
            "status": "OK",
            "servers": [
                "asrsrvr1"
            ]
        },...
    },...
},...
```

When creating a new folder using the `/config/folders` API, all fields that are not read-only are required.

/config/folders Fields

Name	Type	Values	Description
`apps`			Applications that are linked to this folder that will analyze this folder's content. `"apps": [],`

`asroptions`			ASR stream tags that have been added to this folder. Tags are parameters that affect transcription requests.
			`"asroptions": {`
			`    "billing": "customerX"`
			`},`

`audiotype`	REQUIRED when creating a new folder		Whether the audio is two-channel ("Stereo") or single-channel ("Mono") audio. `"audiotype": "Mono",`
`callback`			Callback options for transcript delivery.

Name	Type	Values	Description
			```
			"callback": {
			    "aws_id": "12345678901234567890l",
			    "aws_secret": "12345678901234567890l/
			1234567890l234567890",
			    "posturl": "S3:///joeuser/test",
			    "sendaudio": "no",
			    "sendtext": "no"
			},
			```
`posturl`			The path that V-Spark will use to deliver transcripts and other information. This URL must start with a valid protocol, (such as "http://", "https://", "file://", "sftp://", or "s3://") and include any needed hostname, port number, and file system path.
`aws_id`	REQUIRED if **`posturl`** is set to an "s3://" URL.		Your AWS access key id.
`aws_secret`	REQUIRED if **`posturl`** is set to		Your AWS secret access key.

Name	Type	Values	Description
	an "s3://" URL.		
`username`	**REQUIRED if** `posturl` is set to an "sftp://" URL and `sshprivatekey` is not set.		The username on the remote system that V-Spark should use to log in.
`password`	**REQUIRED if** `posturl` is set to an "sftp://" URL, and `sshprivatekey` is not set.		The login password of the `username` account on the remote system,
`sshprivatekey`	**REQUIRED**		The `ssh` private key of the `username` account on the remote system This setting is **REQUIRED if** `posturl` is set to an "sftp://" URL, and `username` & `password` are not set.

Name	Type	Values	Description
`sendaudio`	REQUIRED if **`posturl`** is set.	yes, no	If set to "yes," V-Spark will send an MP3 version of the transcribed audio file to the callback server.
`sendtext`	REQUIRED if **`posturl`** is set.	yes, no	If set to "yes," V-Spark will send a plain text version of the transcribed audio file to the callback server.
`created`	READ-ONLY Date, in YYYY-MM-DD format		The date that the folder was created. This field is added by V-Spark when the folder is created, and is **read-only**. `"created": "2017-05-18",`
`custom_meta`			Custom metadata fields that are associated with this folder. `"custom_meta": [ "client name", "phone number" ],`
`mode`		active (default), paused	The `mode` field in the JSON output for a Folder indicates whether processing of that folder is "active" or "paused." Use

Name	Type	Values	Description
			the `/config/folders` API to pause and resume processing of the folder. Pause the processing of a Folder by POSTing a JSON configuration file for the Folder that has the `mode` property of the Folder set to the value `paused`. Resume processing by POSTing JSON for the Folder that has the `mode` property set to `active`. You will not be able to set Folder processing to `active` if the Folder has been paused due to company-level policies such as the processing hours limit being met. `"mode": "active",`
`modelchan0`		Acceptable values are limited to the names of the language models you are licensed to use. All language models work with all supported audio formats.	The language model to use when processing audio on Channel 0, which is the left channel if you are processing stereo audio. `"modelchan0": "eng1:callcenter",`

Name	Type	Values	Description
`modelchan1`		Acceptable values are limited to the names of the language models you are licensed to use. All language models work with all supported audio formats.	The language model to use when processing audio on Channel 1, which is the right channel if you are processing stereo audio. If this folder is not configured to process stereo audio, you will not have a `modelchan1`. `"modelchan1": "spa1:callcenter",`
`agentchan`			If this folder is configured to process stereo audio, the value of this field must be either "0" or "1," indicating the audio channel that contains agent speech. `"agentchan": "0",`
`nspeakers`	REQUIRED when creating a new folder.		The number of speakers in the audio files that are going to be placed into the folder. This option cannot be modified after the folder is created.

Name	Type	Values	Description
			`"nspeakers": 1,`
`purifyaudio`	Boolean	true, false	If set to "true," processing cleans the generated audio MP3 of any locations where numbers exist for Payment Card Information (PCI) or other sensitive numbers that are in the audio source so that these numbers cannot be heard. If set to "false," audio is not scrubbed. This option cannot be modified after the folder is created. `"purifyaudio": true,`
`purifytext`	Boolean	true, false	If set to "true," processing cleans the text transcript of any numbers for Payment Card Information (PCI) or other sensitive numbers that are in the audio source. If set to "false," text is not scrubbed. This option cannot be modified after the folder is created. `"purifytext": true,`

Name	Type	Values	Description
`status`			High-level status information about the folder, useful for long-running operations such as a DELETE. A status of "OK" indicates that any operations on the folder have completed their work. `"status": "OK",`
`servers`			The name(s) of the V-Spark servers that will be used for ASR. You must specify at least one server or `vcloud: cloudtoken` statement. `"servers": [ "asrsrvr1"`

The excerpt from the output of calling the `/config/folders` API shown previously is very similar to the output that you would have received had you requested information about a single folder by calling an API URL such as the `/config/DocTestCo/DocTestCo-DocTesting/Test01` API on a V-Spark installation where the "Doc Test Co" company, the "Doc Testing" organization, and the folder "Test01" (with the company short name, "DocTestCo," the Organization short name of "DocTestCo-DocTesting," and the folder name of "Test01") had been defined.

The latter call would have returned JSON like the sample below, which differs only from the output shown previously in that it does not need to identify the short name of the company, the short name of the organization, or the name of the folder that it refers to because they are specified in the URL.

Sample Folder output from the /config/CO_SHORT/ORG_SHORT/FOLDERNAME API

```
{
    "apps": [],
    "asroptions": {
        "billing": "DocTestCo-DocTesting-Test01"
    },
    "audiotype": "Mono",
    "callback": {
        "aws_id": "0SAMPLEVALUED0N0TUSE",
        "aws_secret": "ThisIsAlsoASample000/NotARealAWSSecret00",
        "posturl": "S3:///wvh/test",
        "sendaudio": "no",
        "sendtext": "no"
    },
    "created": "2017-05-18",
    "custom_meta": [],
    "mode": "active",
    "modelchan0": "eng1:callcenter",
    "nspeakers": 1,
    "purifyaudio": true,
    "purifytext": true,
    "servers": [
        "asrsrvr1"
    ]
}
```

/config/apps

The following sample JSON output is for a single application from the `/config/apps` API for the applications that have been defined for a single organization.

Sample application output from the /config/apps API

```
"DocTestCo": {
    "DocTestCo-DocTesting": {
        "Admin App": {
            "created": "2017-06-23",
            "defaultscoretype": "Hit/Miss",
            "enabled": "on",
            "folders": [
                "Test01"
            ],
            "template": "custom"
        },...
    },...
},...
```

/config/apps Fields

Name	Type	Values	Description
`created`	**READ-ONLY** date, in YYYY-MM-DD format		The date that the application was created. This field is added by V-Spark when the application is created. `"created": "2017-06-23",`
`defaultscoretype`			The default type of score to use for categories that are created within this application. Values for this field are "Coverage" or "Hit/Miss." For more information on the meanings of these score types, refer to the **V-Spark Application Development Guide**. `"defaultscoretype": "Hit/Miss",`
`enabled`			Whether or not this application is actively scoring new file uploads to the folders it scores. Values are "on" and "off." Disabled ("off") applications can still be edited and their existing results viewed, but no new results will be created until the application is re-enabled.

Name	Type	Values	Description
			`"enabled": "on",`

`folders`			The name(s) of the folder(s) this application will score.
			 `"folders": [`
			 `"Test01"`
			 `],`

`template`	REQUIRED when creating a new application.		The name of the application template on which this application is based, or "custom" indicating that this application is not based on a pre-defined template. The template option to **`Copy from existing organization`** is not supported in the API. `"template": "custom"`

The excerpt from the output of calling the `/config/apps` API shown previously is very similar to the output that you would have received had you made a request about the applications that are associated with an organization by calling an API URL such as `/config/DocTestCo/DocTestCo-DocTesting/apps/Admin%20App` API, as shown in the next sample JSON.

This sample output is from a V-Spark installation where the "Doc Test Co" company and the "Doc Testing" organization (with the company short name, "DocTestCo" and the Organization short name of "DocTestCo-DocTesting"), and the Application "Admin App" were previously defined. This output differs only in that it does not need to identify the short name of the company, organization, and application that it refers to, since test values were specified in the URL.

Sample Application output from the /config/CO_SHORT/ORG_SHORT/apps/APPNAME API

```
{
    "created": "2017-06-23",
    "defaultscoretype": "Hit/Miss",
    "enabled": "on",
    "folders": [
        "Test01"
    ],
    "template": "custom"
}
```

> NOTE: Application names can contain spaces, which must be URL-encoded by replacing them with `%20` when specifying the name of an application as part of a URL.

/list

Use `/list` to retrieve summary V-Spark system configuration and entity names. `/list` supports GET requests only. For detailed configuration output, use `/config` .

Synopsis

```
/list?token=$token
/list/users?token=$token
/list/orgs?token=$token
/list/folders?token=$token
/list/apps?token=$token

/list/$co_short/users?token=$token
/list/$co_short/orgs?token=$token || /list/$co_short?token=$token
/list/$co_short/folders?token=$token
/list/$co_short/apps?token=$token

/list/$co_short/$org_short/folders?token=$token || /list/$co_short/$org_short?token=$token
/list/$co_short/$org_short/apps?token=$token
```

`$co_short`

Company short name used to filter the request.

`$org_short`

Organization short name used to filter the request.

> **`$token`**

> Authorization token with the required read or write permissions for the request.

> **`$app_name`**

> Name of the application to be queried.

> **`$json`**
> Path to the request's JSON file.

Endpoints

> /list

> Returns all companies.

> /list/users

> Returns all users.

> /list/orgs

> Returns all organizations.

> /list/folders

> Returns all folders.

> /list/apps

> Returns all applications.

Example cURL request

The following example request retrieves a list of all organizations.

```
curl -s 'http://www.example.com:3000/list/orgs?token=$token'
```

Example JSON output

> /list
>
> ```
> [
>     "TestCompany",
>     "Testing",
>     "DocTestCo",
>     "WebAPITest",
>     "Limitedhours",
>     "CNCO",
>     "JWebAPITest"
> ]
> ```

> /list/users
>
> ```
> "DocTestCo": [
>     "joe.user",
>     "bill.generic"
> ```

```
],
```

/list/orgs

```
"DocTestCo": [
    "DocTestCo-DocTesting"
],...
```

/list/$co_short/orgs

```
[
    "DocTestCo-DocTesting"
]
```

/list/folders

```
"DocTestCo": {
    "DocTestCo-DocTesting": [
        "Test01"
    ]
},...
```

/list/$co_short/$org_short/folders

```
[
    "Test01"
]
```

/list/apps

```
"DocTestCo": {
    "DocTestCo-DocTesting": [
        "Testing CallbackTest",
        "Manager App",
        "Admin App"
    ]
},...
```

/list/$co_short/apps

```
[
    "Testing CallbackTest",
    "Manager App",
    "Admin App"
]
```

Python example

Use api_get_config.py to test `/list` as in this example request for a list of all organizations:

```
python api_get_config.py $host $token /list/orgs/ $http_code $output_file
```

`$host`

Hostname or URL for the system running V-Spark.

> **`$token`**

> Authorization token with the required read or write permissions for the request.

> **`$http_code`**
> The HTTP code expected to be returned with the response.

> **`$output_file`**
> File path for query output.

/metadata

Use `/metadata` to add, update, or delete transcript metadata values. `/metadata` supports PATCH and DELETE requests. PATCH adds or modifies metadata values. DELETE removes metadata values.

Synopsis

```
PATCH /metadata/$co_short/$org_short/$tId?token=$token --data @$fields.json
DELETE /metadata/$co_short/$org_short/$tId?token=$token --data @$fields.json
```

`$co_short`

Company short name used to filter the request.

`$org_short`

Organization short name used to filter the request.

`$tId`

UUID for the transcript to be queried.

`$token`

Authorization token with the required read or write permissions for the request.

`$fields.json`

JSON-formatted data that specifies the metadata to be changed. The required JSON data varies by HTTP method. `PATCH` requests must

> include a JSON-formatted object literal of metadata key-value pairs. `DELETE` requests must include a JSON-formatted array of metadata field names. This JSON data may be submitted in the body of the request or with a file. Due to the JSON's potential complexity, using a file is recommended.

Using `/metadata`

Metadata is a core component of processing files in V-Spark, and it's important to note the following behavior and potential ramifications when using the `/metadata` endpoint.

- **Metadata storage.** V-Spark writes metadata changes to three places: the database, Elasticsearch, and long-term storage. Metadata is saved to the database and to Elasticsearch only if that metadata field has been configured for the transcript's folder.
- **Metadata modification.** Any request to `/metadata` updates the `last_modified` field associated with the request's `tId` .
- **Adding new metadata fields.** Requests to `/metadata` do not change a folder's metadata field configuration. As a result, to add new custom metadata key-value pairs that are indexed and searchable, the new metadata fields must be added to the transcript's folder configuration before the request is made.
- **Values for non-configured fields.** Values for fields specified in the request but not configured for the folder are not stored in the database nor in Elasticsearch. However, when metadata fields and values for non-configured fields are submitted using `/metadata` , those fields and values are not lost; they are passed through and saved in the JSON transcript in long-term storage only.
- **Application processing.** Changing a transcript's metadata with `/metadata` does not trigger transcript-application reprocessing.
- **Reserved fields.** Requests with metadata field names in the reserved list are rejected.
- **Multiple requests.** Making multiple `/metadata` requests for the same `tId` at the exact same time may result in an error with HTTP code 400 and an `Error uploading metadata` message. The solution in this case is to retry the request.

Content types

Method	Expects	Returns
PATCH	application/json (object literal)	text/html application/json
DELETE	application/json (array litera)	text/html application/json
Errors		text/html

PATCH and DELETE requests return success messages with HTTP code 200 and the full updated metadata result for the transcript with the "application/json" MIME type.

Example requests

The following example `PATCH /metadata` request specifies a `tId` of `3` associated with the company `Co-Short` and the organization `Org-Short`. The request includes a JSON object with one key-value pair.

```
curl -H 'Content-type: application/json' -d '{"agentgroup":"2"}' -X PATCH 'http://example.com:3000/metadata/Co-Short/
Org-Short/3?token=123'
```

The following example `PATCH /metadata` request is similar to the preceding example, but specifies a file named `fields.json` for the request's JSON data.

```
curl -H 'Content-type: application/json' -d @fields.json -X PATCH 'http://example.com:3000/metadata/Co-Short/Org-
Short/3?token=123'
```

The following example `DELETE /metadata` request specifies a `tId` of `6` associated with the company `Co-Short` and the organization `Org-Short`. The request includes a JSON array with one field name, `transfers`, to be deleted from the transcript.

```
curl -H 'Content-type: application/json' -d '["transfers"]' -X DELETE 'http://example.com:3000/metadata/Co-Short/Org-
Short/6?token=123'
```

Upon success, the preceding examples return JSON data containing the full updated metadata for the specified transcript.

Example input

The following example JSON shows the format required for `PATCH` requests.

```
{
    "direction":"inbound",
    "hold time":"78",
    "transfers":"3"
}
```

On a successful `PATCH` request, the keys and values for the `direction`, `hold time`, and `transfers` fields are added to a `tId` associated with a folder that has these fields configured. If those metadata fields are already associated with that `tId`, the values for those fields are updated.

The following example JSON shows the format required for `DELETE` requests.

```
[
    "clientname",
    "agentname",
    "employeegroup"
]
```

On a successful `DELETE` request, the preceding example removes metadata fields and values for the `clientname`, `agentname`, and `employeegroup` elements from the database, Elasticsearch, and long-term storage for the specified `tId`.

/request

Use `/request` to retrieve information associated with a single `$requestID`, including

- Request processing status.
- Request results, including JSON transcript, metadata, and audio files.

To automatically send results to another host or system when transcription and analytics processing is complete, use a folder configured to send results using callbacks. `/request` supports GET requests only.

Synopsis

```
GET /request/$org_short/status/requestid=$requestID&token=$token
GET /request/$org_short/summary/requestid=$requestID&token=$token > $output_file.json
GET /request/$org_short/details/requestid=$requestID&token=$token > $output_file.json
GET /request/$org_short/results/requestid=$requestID&token=$token > $output_file.zip
```

`$org_short`

Organization short name used to filter the request.

`$requestID`

UUID for the query's target request.

`$token`

Authorization token with the required read or write permissions for the request.

> **`$output_file`**
> File path for query output.

Endpoints

`/request` endpoints accept GET requests for a single `$requestID`.

> /request/**`$org_short`**/status

> Returns a plain text message with these processing states:
>
> > analyzing
>
> > Request data has been transcribed, and its transcription data is being analyzed.
>
> > done
>
> > Transcription and analytics processing is complete. Results for the specified request are ready for callback or download.
>
> > error
>
> > Something went wrong with the request, and no results will ever be available. Validate the data in the original `/transcribe` request.
>
> > received
>
> > Initial `/transcribe` request file(s) have been received and were queued for ASR transcription.

> /request/**`$org_short`**/summary

> Returns JSON summary data for the specified `$requestID` with these fields.

> time_submit
>
> Timestamp generated when request is initially submitted.
>
> time_complete
>
> Timestamp generated when the request finished processing.
>
> status
>
> Overall request status; identical to the value returned by the `/request/$org_short/status` endpoint.
>
> processed
>
> Number of files in the request that were processed for transcription.
>
> analyzed
>
> Number of transcripts analyzed for the request.
>
> error
>
> Number of audio files that produced an error when processed or analyzed.
>
> submitted
>
> Number of audio files submitted with the request.
>
> NOTE: Count information is not provided for fields that are NULL.

/request/**`$org_short`**/details

Returns JSON data with the fields described for `/request/$org_short/summary` along with an additional `filedetails` array with advanced record details.

The `filedetails` array contains one object for each audio file in the request. Each object contains these fields.

fullfilename

The name of the file sent with the request.

status

Request processing status.

job_start

Timestamp generated when the request's transcription job began processing.

job_finish

Timestamp generated when the request's transcription job finished processing.

analyze

Timestamp generated when the request's analytics job finished processing.

size

Request's total file size.

/request/**`$org_short`**/result

Returns a zip with the results of the specified `$requestID`. Zip contents depend on these optional boolean query parameters.

Parameter	Examples	Description

json	json=0
	json=1 (default)
		Include the complete JSON transcript for each request audio file.

mp3	mp3=0 (default)
	mp3=1
		Include the request's transcoded mp3 file(s).

txt	txt=0 (default)
	txt=1
		Include a plain text version of request transcript(s).

Example cURL requests

/request/**`$org_short`**/status

```
curl 'http://example.company.com:3000/request/co1-orgA/status?requestid=$requestID&token=$token' > $output_file
```

/request/**`$org_short`**/summary

```
curl 'http://example.company.com:3000/request/co1-orgA/summary?requestid=$requestID&token=$token' >
$output_file.json
```

/request/**`$org_short`**/details

```
curl 'http://example.company.com:3000/request/co1-orgA/details?requestid=$requestID&token=$token' >
```

> ```
> $output_file.json
> ```

> /request/**`$org_short`**/result
>
> ```
> curl 'http://example.company.com:3000/request/co1-orgA/
> result?requestid=$requestID&token=$token&json=1&mp3=1&txt=true' > $output_file.zip
> ```

/search

Use `/search` to search transcript records with GET and POST requests, or to delete transcript records with DELETE requests.

Refine and filter `/search` requests with search term and output parameters.

Test GET and POST `/search` requests with get_search.py and post_search.py.

> GET

> Specify search terms as query parameters.

> POST

> Specify search terms as name-value pairs in JSON data sent with the request.

> DELETE

> Specify a transcript ID as a query parameter to queue its audio and JSON data for deletion.
>
> Deletion entails the removal of the audio file, its transcript and transcription results, and the rest of its system record.
>
> Summary charts and tables are not updated when an individual record is removed. Deleted record data is not available in the 💻 **`Dashboard Files View`**, and the record no longer appears in search results. A system under heavy load may take several minutes to fully delete a record, but this situation is unlikely.
>
> Files to be deleted with `/search` must be filtered by company and organization, and may also be filtered by folder. Each file to be deleted must be specified individually by `tId` .
>
> By default, the maximum number of audio files that can be deleted at once is 1000. This maximum can be changed by updating the `simultaneous_tid_deletion_max` system configuration option.

Synopsis

1. Search records

```
GET /search/$co_short/$org_short?token=$token&$options...
GET /search/$co_short/$org_short/$folder?token=$token&$options...
POST /search/$co_short/$org_short?token=$token&$options...
POST /search/$co_short/$org_short/$folder?token=$token&$options...
```

2. Delete individual records

```
DELETE /search/$co_short/$org_short?token=$token&terms.tid=$tId
DELETE /search/$co_short/$org_short?token=$token&tid=$tId
DELETE /search/$co_short/$org_short/$folder?token=$token&terms.tid=$tId
DELETE /search/$co_short/$org_short/$folder?token=$token&tid=$tId
```

3. Delete multiple records

```
DELETE /search/$co_short/$org_short?tid=$tId1,$tId2,...&token=$token&multi=true
DELETE /search/$co_short/$org_short/$folder?terms.tid=$tId1,$tId2,...&token=$token&multi=true
DELETE /search/$co_short/$org_short/$folder?terms.tid=$tId1,$tId2,...&token=$token&multi=true
DELETE /search/$co_short/$org_short/$folder?tid=$tId1,$tId2,...&token=$token&multi=true
```

NOTE: The parameters `tid` and `terms.tid` are equivalent.

`$co_short`

Company short name used to filter the request.

> **`$org_short`**

> Organization short name used to filter the request.

> **`$folder`**
> Folder used to filter the request.

> **`$token`**

> Authorization token with the required read or write permissions for the request.

> **`$options`**
> Optional parameters used to further refine the request and results.

> **`$tId`** or **`$tId1,$tId2,...`**

> UUID for the transcript(s) to be queried.

> multi

> Required to enable the deletion of multiple entities with a single request.

Content types

`/search` request content types depend on the request's method and parameters.

Method	Expects	Returns
DELETE	text/html query parameters	text/html
GET	application/json multipart/form	text/html JSON-format text/html CSV-format text/html application/zip
POST	application/json	text/html JSON-format text/html CSV-format text/html application/zip
`output=count`		text/html
`output=summary&type=json`		JSON-format text/html
`output=summary&type=csv`		CSV-format text/html
`output=details`		JSON-format text/html

Method	Expects	Returns
`output=zip`		application/zip
Errors		text/html

NOTE: When CSV reports are generated with the API, columns are always sorted alphabetically. Prior to V-Spark version 4.3.2, columns were sorted in the order specified for the `fields` parameter.

Example search requests

This example queries `/search` with GET. The request includes a `duration` search filter as a query parameter.

```
curl 'http://example.company.com:3000/search/Test/Test-Testing/Test01?token=$token&duration=4:30-5:30'
```

The next example queries `/search` with POST. The request includes a JSON-format `daterange` search filter.

```
curl -H 'Content-type: application/json' -d '{"daterange":"20190613-20190614"}' -X POST
'https://example.company.com:3000/search/Test/Test-Testing/CallCenterDemosTest01?token=$token'
```

Example DELETE requests

The following is an example cURL call using the `/search` endpoint's `DELETE` method to delete a single file:

```
curl -X DELETE 'http://example.com:3000/search/docs-co/docs-org?token=12345678&tid=999'
```

The preceding example deletes an audio file that is associated with the `docs-co` company and `docs-org` organization, using the token parameter `12345678` and a `transcriptID` of 999.

The following cURL example uses the `/search` endpoint's `DELETE` method to delete three files:

```
curl -X DELETE 'http://example.com:3000/search/docs-co/docs-org?token=12345678&tid=9991,9992,9993&multi=true'
```

The preceding example deletes three audio files associated with the `docs-co` company and `docs-org` organization, using the token parameter 12345678 and the `transcriptID` s 9991, 9992, and 9993.

Example JSON

This example output shows the JSON results of a `/search` request. This output was generated with default summary output fields.

```
[
    {
        "filename": "file1json.wav",
        "agentid": "105",
        "datetime": "2017-07-18 17:07:12",
        "duration": "0:05:25",
        "score": "1.0000",
        "tid": 5,
        "requestid": "11723359-d790-4a88-aff8-7925296e7df2",
        "agent_gender": "Female",
        "client_gender": "Male",
        "overall_emotion": "Improving",
        "client_emotion": "Negative",
        "agent_emotion": "Positive",
        "overtalk": "0.0000",
        "silence": "0.4269",
        "agent_clarity": "0.0000",
        "client_clarity": "0.8298",
        "diarization": "2.0000",
        "preview": {}
    },...
]
```

/search term parameters

Use these parameters to filter search queries to certain transcription and analytics metadata values.

agent_clarity=`$n`-`$m`

Search for audio records with agent voice clarity percentage within the range of `$n` to `$m`, which are floating point values. A score closer to 1 indicates higher clarity. Low clarity is the result of poor signal, background noise, accent, or poor enunciation. For example:

```
agent_clarity=0.6-1.0
```

agent_emotion=`$emotion`

Search for audio records where the emotional score of the agent's portion of the audio is `$emotion`. Possible values for `$emotion` are `Improving`, `Negative`, `Positive`, and `Worsening`. Only one `agent_emotion` value can be used within a single search. For example:

```
agent_emotion=Negative
```

agent_gender=`$gender`

Search for audio records where the gender of the agent has been identified as `$gender`. Possible values for `$gender` are `Female` and `Male`. Only one `agent_gender` value can be used within a single search. For example:

```
agent_gender=Female
```

app.name=`$appname` and app.`$category`=`$score`

Search for transcripts that have received the specified `$score` level from the analytics application `$appname` in the specified top- or lower-level category. To specify lower-level categories, use a full dot-separated path through the category tree. For example:

```
app.name=AgentScorecard
app.Politeness=All
app.$top_category.$lower_category.$lowest_category=Medium
```

`$score` may be one of the following values:

- `All` — Score greater than 0
- `High` — Score greater than 0.66
- `Medium` — Score greater than 0.33
- `Low` — Score between 0 and 0.33
- `None` — Score is 0

client_clarity=`$n-$m`

Search for audio records with client voice clarity percentage within the range of N to M, which are floating point values. A score closer to 1 indicates higher clarity. Low clarity is the result of poor signal, background noise, accent, or poor enunciation. For example:

```
client_clarity=0.6-1.0
```

client_emotion=`$emotion`

Search for audio records where the emotional score of the client's portion of the audio is `$emotion`. Possible values for `$emotion` are `Improving`, `Negative`, `Positive`, and `Worsening`. Only one `client_emotion` value can be used within a single search. For example:

```
client_emotion=Improving
```

client_gender=`$gender`

Search for audio records where the gender of the client has been identified as `$gender`. Possible values for `$gender` are `Female` and `Male`. Only one `client_gender` value can be used within a single search. For example:

```
client_gender=Male
```

daterange=`$start`-`$end`

Filters the request by the dates specified in the range [`$start`, `$end`]. Values for `$start` and `$end` are optional, but the hyphen between them is required.

The range specified by `$start` and `$end` may be expressed using any combination of years, months, days, hours, minutes, and seconds, expressed as YYYYMMDD[HHmmss]. Date ranges are always assumed to be positive (where `$start` is less than `$end`).

`daterange` filters by values in the transcript record's `datetime` field. Data for the `datetime` field is stored using the organization's time zone, which may vary by organization.

No verification is done to ensure that `daterange` values are correct; invalid date ranges will simply return no values. If `$start` is not specified, a default value of 01 January, 1900 is used. If `$end` is not specified, the current date is used.

diarization=`$score`

Search for audio records with a diarization score of at least `$score`. which is a floating point value between 0 and 1. When mono (single-channel) audio has multiple speakers, diarization can separate the speakers for analysis. The diarization score identifies how completely the call was divided into individual speakers. A score of 2 means the call was not diarized. Diarization technology is not perfect, and to find calls where diarization is done well, set `SCORE` closer to one than to zero. The tradeoff is that fewer calls will be retrieved.

duration=`$n`-`$m`

Search for audio records with duration in the range of `$n` to `$m`, where `$n` and `$m` are the minimum and maximum duration, respectively, of the audio records to be returned. Note the following:

- The durations `$n` and `$m` can be expressed in the colon-delimited format `hh:mm:ss` .
- Only the seconds (`ss`) value is required; hours (`hh`) and minutes (`mm`) are optional.
- The colon delimiter is required only when specifying hours or minutes.
- There is no limit to the numeric values of hours, minutes, or seconds.
- Only the first value, `$n`, is required, along with the hyphen. A request with no value for `$m` returns results for all audio records with duration greater than the minimum value `$n`.

For example, to search for audio records with duration 30 minutes to 60 minutes, use any of the following:

```
duration=30:00-1:00:00
duration=30:00-60:00
duration=1800-3600
```

Similarly, any of the following expressions would return audio records at least 10 minutes long:

```
duration=10:00-
duration=8:120-
duration=6000-
```

lastmodifiedrange=`$start`-`$end`

Filters the request by a record's `last_modified` value in the range of `$start` to `$end`. Data for the `last_modified` field is always stored using Coordinated Universal Time (UTC).

Values for `$start` and `$end` are optional. The range specified by `$start` and `$end` may be expressed using any combination of years, months, days, hours, minutes, and seconds, expressed as `YYYYMMDD[HHmmss]`. Date ranges for `lastmodifiedrange` are always assumed to be positive (where `$start` is less than `$end`) and expressed in Coordinated Universal Time (UTC).

> NOTE: Records without a date and time value for the `last_modified` field are not included in results.

overall_emotion=**`$emotion`**

Search for audio records where the overall emotional score of the audio is `$emotion`. Possible values for `$emotion` are `Improving`, `Negative`, `Positive`, and `Worsening`. Only one `overall_emotion` value can be used within a single search. For example:

```
overall_emotion=positive
```

overall_gender=**`$gender`**

Search for audio records where the overall gender of the speakers has been identified as `$gender`. Possible values for `$gender` are `Female` and `Male`. Only one `overall_gender` value can be used within a single search. For example:

```
overall_gender=Male
```

overtalk=**`$n`**-**`$m`**

Search for audio records with overtalk percentages within the range of `$n` to `$m`, which are floating point values. Overtalk occurs when speakers talk over one another. A recording's overtalk percentage is the count of agent-initiated overtalk turns as a percentage of the total number of agent speaking turns. In other words, out of all of the agent’s turns, overtalk measures how many agent turns interrupted a client’s turn. An overtalk value of 1 indicates the most overtalk. For example:

```
overtalk=0.2-0.6
```

silence=`$n`-`$m`

Search for audio records with silence percentages within the range of `$n` to `$m`, which are floating point values. Silence is equal to all non-speech time, as a percentage of the total audio duration. If music and noise are not decoded to word-events, they are counted as silence. Calls with a silence value of 1 contain the most silence. For example:

```
silence=0.2-0.5
```

terms.`$field`=`$value`

Acceptable terms for `$value` vary by `$field`, which may be any of the following:

- `agent`
- `agentid`
- `client`
- `file`
- `requestid`
- `speakers`
- `tag`
- `tid`
- `$client_data`

Acceptable terms for `$client_data` include any custom metdata field configured for the record's folder.

For example:

```
terms.agent=007
```

Include multiple terms in the value by using a comma-separated list. By default, multiple terms are combined with the `and` operator. To specify the `or` operator, use `terms.op`. For example:

```
terms.agent=007,006
```

terms.op=`$operator`

Define how multiple search terms should be logically combined. Acceptable values include `and` and `or`, where `and` is the default operator. If `and` is specified, the request returns only records that match both terms. If `or` is specified, the request returns any record that matches either term. For example:

```
terms.tid=1,2&terms.op=or
```

/search output parameters

Use these parameters to define the output returned by `/search` requests.

Format

Defines the output type for request results.

> format=**`$format`**

> Identifies the output format for results, where `$format` is `csv` or `json`. The default value is `json`. `format=csv` is valid only when `output=summary`.

> IMPORTANT: If custom metadata fields are associated with the folders in which search results are found, JSON output includes only fields with values. CSV output includes all fields, with an empty value for fields without an explicit value.

Output type

Specifies the level of detail included with `/search` responses, and determines the other options available for refining that detail.

> output=**`$type`**

> Specifies the type of results to be returned. Acceptable values are `count`, `details`, `summary`, and `zip`.
>
> > count
>
> > Returns the number of records that matched the query. Search term options may be specified after `output=count`.

> details
>
> Returns full JSON transcripts for each match. Search terms and output sorting options may be included after `output=details` .
>
> summary
>
> Returns a summary of the matching search results in JSON or CSV. `summary` is the default value, and is used when no other `output` value is specified. After specifying the `output=summary` option on the command line, in a JSON file, or by using it as a default value that you are not overriding, you can also specify search term, and output sorting, field, and format options.
>
> > NOTE: When CSV reports are generated with the API, columns are always sorted alphabetically. Prior to V-Spark version 4.3.2, columns were sorted in the order specified for the `fields` parameter.
>
> zip
>
> Returns all files that matched your query in a zip file. The zip file that is returned includes these audio files hierarchically. After specifying the `output=zip` option on the command line or in a JSON file, you can also specify search term and output sorting options. You can also pass the `zipfiles=` parameter to identify the files that you want to be included in the output zip file. This must be one or more of `json` , `mp3` , or `text` . Specifying multiple values is done as a comma-separated list.
>
> When explicitly specified in a JSON file as part of a `/search` POST call, the `output` value is specified as the following in your JSON file of search specifications:

```
"output" : "value"
```

Fields in summary output

When specifying `output=summary` in a `/search` request, use the `fields` parameter to define the data fields to be included with or excluded from search results.

Multiple fields may be provided in a comma-separated list. To exclude a field from search results, add a hyphen `-` before the name of the field.

The following section lists all output field option parameters and describes the fields they return when specified:

> voci
>
> Fields that are specific to audio data that has been processed by V-Spark. This is the default behavior if no value for the `fields` option is specified. `voci` fields include the following:
>
> > agent_clarity
> >
> > How clear the agent channel/speech is, expressed as a value between 0 and 1, where 1 is highest.
> >
> > agent_emotion
> >
> > Overall agent emotional intelligence assessment derived from both acoustic and linguistic information. Has one of the following values: `Positive`, `Worsening`, `Improving`, or `Negative`.
> >
> > agent_gender
> >
> > Agent gender, either `Male` or `Female`.
> >
> > agentid
> >
> > Identifier for a specific agent.
> >
> > client_clarity
> >
> > How clear the client channel/speech is, expressed as a value between 0 and 1, where 1 is highest.

> **client_emotion**
>
> Overall agent emotional intelligence assessment derived from both acoustic and linguistic information. Has one of the following values: `Positive`, `Worsening`, `Improving`, or `Negative`.
>
> **client_gender**
>
> Client gender, either `Male` or `Female`.
>
> **datetime**
>
> Transcript date and time. Data for the `datetime` field is stored using the organization's time zone, which may vary by organization.
>
> **diarization**
>
> Only provided in two-speaker, one-channel calls; a value between 0 and 1 identifies how completely the call was divided into individual speakers. A value of 1 is the best possible value for speaker separation. A value of 2 means the call was not diarized.
>
> **duration**
>
> Call duration.
>
> **es_doc_id**
>
> Unique identifier for a transcript in Elasticsearch index. *Not included when `voci` is specified as return field.*
>
> **filename**
>
> Name of the uploaded audio or JSON file that contains matches for the search request.

folder

Name of the folder where the file was uploaded. *Not included when `voci` is specified as return field.*

last_modified

The date and time at which an update to the `last_modified` field was last triggered in the Elasticsearch record associated with a transcript. If the `last_modified` field is not present or has no date and time value, its return value is `false`. Data for the `last_modified` field is stored in Coordinated Universal Time (UTC).

The following events trigger an update to the `last_modified` field:

- Creating a new transcript.
- Updating transcript scores by reprocessing an application.
- Deleting an application or application category associated with the transcript.
- Unlinking an application from the transcript's folder, if that application has previously been used to score the transcript.
- Updating transcript metadata using the API.

overall_emotion

Overall emotional intelligence assessment for an audio file, derived from both acoustic and linguistic information. Has one of the following values: `Positive`, `Worsening`, `Improving`, or `Negative`.

overtalk

Percentage of call when the agent talks over or interrupts the client. Equal to the number of turns where the agent initiated overtalk divided by the total number of agent turns.

preview

An excerpt of the transcribed call in which matched terms are highlighted.

requestid

Unique identifier for a transcription request. This value is assigned when an audio file is submitted for transcription.

score

Calculation of how well a transcript matches the terms specified in your search. This is represented as a value between 0 and 1, and can depend on the type of query that you submitted. For example, date range queries always provide a score value of 1 for any transcription that occurred in the specified date range.

silence

Percentage of call duration that is silence. Equal to all non-speech time, this value is calculated as call duration minus the sum of the duration of each word. If music and noise are not decoded to word-events, they are counted as silence.

tags

Tags added to files in the GUI. *Not included when `voci` is specified as return field.*

tid

Unique identifier for a transcript.

url

URL to visit the file details in the GUI. *Not included when `voci` is specified as return field.*

> **`$custom_metadata`**

> Custom metadata fields, specified by name, configured for the folder that processed the audio.

> all

> Combination of all `voci` and `$custom_metadata` fields.

> apps

> Scores for all applications. Requests may specify *either* `apps` or `app. $appname` , *not* both.

> app.**`$appname`**

> Scores for a specific application. Requests may specify *either* `apps` or `app. $appname` , *not* both.

> app.**`$appname.$top_category`**

> Scores for a specific category in an application. Note that the category's full name includes its parents' names, including any parent categories. For example, `app.$appname .$top_category.$lower_category` refers to an application category called `$lower_category`, which is a child of `$top_category` and `$appname`.

Sort

The V-Spark `/search` API provides multiple parameters that enable you to specify the way in which search results should be sorted. You can specify these parameters when using any `/search` API `output` type with the exception of the `count` output type.

Possible sorting output options are the following:

offset=**`$number`**

Specifies the number of the first search result that should be returned. The `offset` is used in conjunction with the `size` option to enable you to page through search results when their number exceeds the `size` value. For example, if a search matches 500 results and you are using a `size` value of 100, you would specify `&offset=100` to return results 101-200, and `&offset=200` to return results 201-300, and so on. The default `offset` value is `0` .

size=**`$number`**

Specifies the number of matching results that will be returned at one time. The default `size` value is `100` . The maximum value for `size` is 1000.

sort=**`$field`**

Specifies how matching search results should be ordered when returned. Regardless of the specified `$field`, `score` is *always* used as a secondary sort option. The default `sort` value is `datetime` .

`$field` accepts these values as sort options:

- `agent_clarity`
- `agent_emotion`
- `agent_gender`
- `client_clarity`
- `client_emotion`
- `client_gender`
- `datetime`

> - `diarization`
> - `duration`
> - `filename`
> - `last_modified`
> - `overall_emotion`
> - `overtalk`
> - `score`
> - `silence`

> sortdir=**`$direction`**

> Direction in which to sort output entries. `$direction` should be either `asc` for ascending order, or `desc` for descending order, based on data type. The default value is `desc`.

/stats

Use `/stats` to retrieve daily statistics for folders from a specified date range. `/stats` supports GET requests only.

To retrieve application and category scores, use /appstats.

Synopsis

```
GET /stats/$co_short/$org_short?token=$token&daterange=$daterange
GET /stats/$co_short/$org_short/$folder?token=$token&daterange=$daterange
```

`$co_short`

Company short name used to filter the request.

`$org_short`

Organization short name used to filter the request.

`$folder`

Folder used to filter the request.

`$token`

Authorization token with the required read or write permissions for the request.

daterange=**`$start`**-**`$end`**

Filters the request by the dates specified in the range [`$start`, `$end`]. Values for `$start` and `$end` are optional, but the hyphen between

> them is required.
>
> The range specified by `$start` and `$end` may be expressed using any combination of years, months, days, hours, minutes, and seconds, expressed as YYYYMMDD[HHmmss]. Date ranges are always assumed to be positive (where `$start` is less than `$end`).
>
> `daterange` filters by values in the transcript record's `datetime` field. Data for the `datetime` field is stored using the organization's time zone, which may vary by organization.
>
> No verification is done to ensure that `daterange` values are correct; invalid date ranges will simply return no values. If `$start` is not specified, a default value of 01 January, 1900 is used. If `$end` is not specified, the current date is used.

Example requests

The following example retrieves statistics for `folder2` on the current date.

```
curl -s 'http://example.company.com:3000/stats/Docs/Docs-Testing/folder2?token=$token'
```

The following example retrieves statistics for `folder2` on a single day.

```
curl -s 'http://example.company.com:3000/stats/Docs/Docs-Testing/folder2?token=$token&daterange=20211018-20211018'
```

Example Python implementation

Use the api_get_stats.py with these command-line arguments:

```
python api_get_stats.py $host $token /stats/$co_short/$org_short/$folder '&daterange=20211005'
```

This call uses the `daterange` parameter to limit results to those from files processed on a single date (October 5, 2021), and includes only the `Politeness` category.

> **`$host`**
>
> Hostname or URL for the system running V-Spark.

> **`$token`**

> Authorization token with the required read or write permissions for the request.

> **`$co_short`**

> Company short name used to filter the request.

> **`$org_short`**

> Organization short name used to filter the request.

> daterange=**`$start`**-**`$end`**

> Filters the request by the dates specified in the range [`$start`, `$end`]. Values for `$start` and `$end` are optional, but the hyphen between them is required.
>
> The range specified by `$start` and `$end` may be expressed using any combination of years, months, days, hours, minutes, and seconds, expressed as YYYYMMDD[HHmmss]. Date ranges are always assumed to be positive (where `$start` is less than `$end`).
>
> `daterange` filters by values in the transcript record's `datetime` field. Data for the `datetime` field is stored using the organization's time zone, which may vary by organization.
>
> No verification is done to ensure that `daterange` values are correct; invalid date ranges will simply return no values. If `$start` is not specified, a default value of 01 January, 1900 is used. If `$end` is not specified, the current date is used.

Example output

```
[
    {
        "date": "20170925",
        "calls": 5,
        "avgduration": "0:09:52",
        "avgsilence": "0:04:12",
        "avgwords": 1256.6,
        "agent": {
            "avgcalls": "1.7",
            "talk": {
                "avg": "0:02:57",
                "min": {
                    "id": "004",
                    "duration": "0:00:32"
                },
                "max": {
                    "id": "002",
                    "duration": "0:09:24"
                }
            },
            "emotion": {
                "positive": 3,
                "worsening": 0,
                "negative": 2,
                "improving": 0
            }
        },
            "client": {
                "talk": {
                    "avg": "0:02:43"
                },
                "emotion": {
                    "positive": 1,
                    "worsening": 0,
                    "negative": 3,
                    "improving": 1
                }
            }
        }
]
```

/status

Use `/status` to retrieve JSON data with processing status for

- A single folder.
- All folders in a single company or organization.
- All system folders.

`/status` supports GET requests only.

Synopsis

```
GET /status?token=$token
GET /status/$co_short?token=$token
GET /status/$co_short/$org_short?token=$token
GET /status/$co_short/$org_short/$folder?token=$token
```

`$co_short`

Company short name used to filter the request.

`$org_short`

Organization short name used to filter the request.

`$folder`

Folder used to filter the request.

`$token`

Authorization token with the required read or write permissions for the request.

`$format` (optional)
Specify `format=csv` to return comma-separated values. By default, `format=json` .

Example requests

The following request retrieves the status in JSON for a single folder named `Test01` .

```
curl -s 'http://example.company.com/status/co1/co1-orgA/Test01?token=$token'
```

The following request retrieves the status in CSV for a single folder named `Test01.`

```
curl -s 'http://example.company.com/status/co1/co1-orgA/Test01?format=csv&token=$token'
```

Example Python implementation

Use api_get_test.py with these command-line arguments to retrieve JSON with the status of a single folder.

```
python api_get_test.py $host $token /status/$co_short/$org_short/$folder ''
```

`$host`
Hostname or URL for the system running V-Spark.

Use api_config_status.py to retrieve processing status for all system folders.

Example folder output

JSON output for a single folder resembles this example.

```
{
    "mode": "active",
    "servers": {
        "srvr1": "OK"
    },
    "jobmgr": {
        "192.168.150.71": {
            "numerrs": 0,
            "timestamp": "2019-06-24_18:45:58",
            "hostname": "srvr2",
            "numtranscoding": 0,
            "version": "2.4.0-0",
            "starttime": "2019-06-21_20:25:07",
            "numdecoding": 0
        }
    },
    "queued": {
        "count": 0,
        "lastactive": "2019-06-24T18:36:15.000Z"
    },
    "analyzing": {
        "lastactive": "2019-06-24T18:45:58.000Z",
        "count": "2"
    },
    "error": {
        "lastactive": null,
        "count": "0"
    }
}
```

The same call, but with `format=csv`, returns this CSV output.

```
"company","organization","folder","mode","servers","queued.count",\
"queued.lastactive","decoding.count","decoding.lastactive","analyzing.count",\
"analyzing.lastactive","error.count","error.lastactive"
"DocTestCo","DocTestCo-DocTesting2","folder2","active","{""srvr1"":""OK""}",0\
,,,,"0",\
,"0",
```

Example company output

JSON output for a company resembles this example.

```
{
    "DocTestCo-DocTesting1": {},
    "DocTestCo-DocTesting2": {
        "folder2": {
            "mode": "active",
            "servers": {
                "srvr1": "OK"
            },
            "jobmgr": {
                "192.168.150.71": {
                    "numerrs": 0,
                    "timestamp": "2019-06-24_18:40:28",
                    "hostname": "srvr1",
                    "numtranscoding": 0,
                    "version": "2.4.0-0",
                    "starttime": "2019-06-21_20:25:07",
                    "numdecoding": 0
                }
            },
            "queued": {
                "count": 0,
                "lastactive": "2019-06-24T18:36:15.000Z"
            },
            "analyzing": {
                "lastactive": "2019-06-24T18:45:58.000Z",
                "count": "2"
            },
            "error": {
                "lastactive": null,
                "count": "0"
            }
        },...
    },...
}
```

The same call, but with `format=csv`, returns this CSV output.

```
"company","organization","folder","mode","servers","queued.count",\
"queued.lastactive","decoding.count","decoding.lastactive","analyzing.count",\
"analyzing.lastactive","error.count","error.lastactive"
"DocTestCo","DocTestCo-DocTesting2","folder2","active","{""srvr1"":""OK""}",0,\
,,,"0",,"0",
"DocTestCo","DocTestCo-DocTesting3","folder3","active","{""srvr1"":""OK""}",0,\
"2019-06-18T18:58:30.000Z",,,"0","2019-06-18T18:59:29.000Z","0",
"DocTestCo","DocTestCo-DocTesting3","folder33","active","{""srvr1"":""OK""}",0,\
"2019-06-18T19:01:12.000Z",,,"0","2019-06-18T19:14:44.000Z","2","2019-06-18T19:14:00.000Z"
```

Example organization output

JSON output for a single organization resembles this example.

```
{
    "DocTestCo-DocTesting2": {
        "folder2": {
            "mode": "active",
            "servers": {
                "srvr1": "OK"
            },
            "jobmgr": {
                "192.168.150.71": {
                    "numerrs": 0,
                    "timestamp": "2019-06-24_18:40:28",
                    "hostname": "srvr1",
                    "numtranscoding": 0,
                    "version": "2.4.0-0",
                    "starttime": "2019-06-21_20:25:07",
                    "numdecoding": 0
                }
            },
            "queued": {
                "count": 0,
                "lastactive": "2019-06-24T18:36:15.000Z"
            },
            "analyzing": {
                "lastactive": "2019-06-24T18:45:58.000Z",
                "count": "2"
            },
            "error": {
                "lastactive": null,
                "count": "0"
            }
        },...
    }
}
```

The same call, but with `format=csv` , returns this CSV output.

```
"company","organization","folder","mode","servers","queued.count",\
"queued.lastactive","decoding.count","decoding.lastactive","analyzing.count",
"analyzing.lastactive","error.count","error.lastactive"
"DocTestCo","DocTestCo-DocTesting","Test01","active","{""asrsrvr1"":""OK""}",0,\
"2017-06-16 09:06:45.584507060 -0400",0,"2017-06-16 09:05:45.451651713 -0400",0,\
"2017-06-16 09:15:24.235266461 -0400",1,"2017-05-18 12:46:55.780155897 -0400"
```

/sysinfo

Use `/sysinfo` to retrieve system status, configuration, and software version in JSON format. `/sysinfo` requires the root token and supports GET requests only.

Synopsis

```
GET /sysinfo?token=$root_token
GET /sysinfo?full&token=$root_token
```

`$root_token`
V-Spark's root authorization token. Only available to system administrators.

full
Returns additional system information details.

Example requests

The following example request uses `/sysinfo` to retrieve summary system information and writes output to the file **`Sysinfo-Test.json`**.

```
curl -s 'http://example.company.com:3000/sysinfo?token=$root_token' > Sysinfo-Test.json
```

The following example request uses `/sysinfo?full` to retrieve summary system information and writes output to the file **`Sysinfo-Full-Test.json`**.

```
curl -s 'http://example.company.com:3000/sysinfo?full&token=$root_token' > Sysinfo-Full-Test.json
```

System JSON

This topic lists `/sysinfo` example output and JSON field descriptions.

`/sysinfo` output

```
{
       "uname": "Linux analysis 2.6.32-696.30.1.el6.x86_64 #1 SMP Tue May 22 03:28:18 UTC 2018 x86_64 x86_64 x86_64
GNU/Linux",
       "uptime": "13:39:05 up 8 days,  3:14,  1 user,  load average: 0.02, 0.07, 0.16",
       "hostname": "analysis",
       "mysql": {
           "version": "5.1.73",
           "uptime": "702822"
       },
       "elasticsearch": {
           "uptime": [
               "name    uptime",
               "JtDxLLr     1d"
           ],
           "version": "5.3.0"
       },
       "vspark": {
           "started": "2018-07-12 12:22:19 -04:00",
           "version": "3.4.3-1"
       },
       "jobmgr": {
           "started": "2018-07-12 12:22:28 -04:00",
           "version": "2.3.1-1"
       },
       "product": "V-Spark"
    }
```

System JSON output contains these fields. The information level required for the field is also listed.

> cpuinfo (full)

> Information about the CPU architecture.
>
> ```
>          "cpuinfo": [
>          "Architecture:           x86_64",
>          "CPU op-mode(s):         32-bit, 64-bit",
>          "Byte Order:             Little Endian",
>          "CPU(s):                 12",
>          "On-line CPU(s) list:    0-11",
>          "Thread(s) per core:     1",
>          "Core(s) per socket:     12",
>          "Socket(s):              1",
>          "NUMA node(s):           1",
>          "Vendor ID:              GenuineIntel",
>          "CPU family:             6",
>          "Model:                  45",
>          "Model name:             Intel(R) Xeon(R) CPU E5-2670 0 @ 2.60GHz",
>          "Stepping:               7",
>          "CPU MHz:                2593.718",
>          "BogoMIPS:               5187.43",
>          "Hypervisor vendor:      KVM",
>          "Virtualization type:    full",
>          "L1d cache:              32K",
>          "L1i cache:              32K",
>          "L2 cache:               256K",
>          "L3 cache:               20480K",
>          "NUMA node0 CPU(s):      0-11"
> ```

```
        ],
```

meminfo (full)

A report of the free and used amounts of system memory, in mebibytes.

```
        "meminfo": [
           "total       used       free     shared    buffers     cached",
           "Mem:         19988      11184       8803          6        191       3925",
           "-/+ buffers/cache:       7067      12920",
           "Swap:         8039          0       8039"
        ],
```

storage (full)

A report of disk usage on the main system disk, in human-readable format.

```
        "storage": [
           "Filesystem            Size  Used Avail Use% Mounted on",
           "/dev/mapper/vg_analysis-lv_root",
           "                       50G   12G   36G  25% /",
           "tmpfs                 9.8G  6.0M  9.8G   1% /dev/shm",
           "/dev/sda1             477M   75M  377M  17% /boot",
           "/dev/mapper/vg_analysis-lv_home",
           "                       69G   36G   29G  56% /home"
        ],
```

elasticsearch (basic, full)

The release version and status information for the Elasticsearch process.

Sample basic `elasticsearch` output:

```
        "elasticsearch": {
           "uptime": [
              "name    uptime",
              "JtDxLLr     1d"
           ],
           "version": "5.3.0"
        },
```

Sample full `elasticsearch` output:

```
        "elasticsearch": {
           "info": {
              "name": "JtDxLLr",
              "cluster_name": "elasticsearch",
              "cluster_uuid": "J7jYjVfwQXWzyBDixJUDZw",
              "version": {
                  "build_hash": "3adb13b",
                  "build_date": "2017-03-23T03:31:50.652Z",
                  "build_snapshot": false,
                  "lucene_version": "6.4.1"
              },
              "tagline": "You Know, for Search"
           },
           "uptime": [
               "name    uptime",
               "JtDxLLr     1d"
            ],
            "health": {
               "cluster_name": "elasticsearch",
               "status": "yellow",
               "timed_out": false,
               "number_of_nodes": 1,
               "number_of_data_nodes": 1,
               "active_primary_shards": 5,
               "active_shards": 5,
               "relocating_shards": 0,
               "initializing_shards": 0,
               "unassigned_shards": 5,
               "delayed_unassigned_shards": 0,
               "number_of_pending_tasks": 0,
               "number_of_in_flight_fetch": 0,
               "task_max_waiting_in_queue_millis": 0,
               "active_shards_percent_as_number": 50
            },
            "indices": [
               "health status index                 uuid                   pri rep docs.count docs.deleted
store.size pri.store.size",
               "yellow open   product_20180525143400 unyWynDvQsWMdf8b_cqBGA   5   1    9519021      1692025
5gb           5gb"
            ],
            "count": 126213,
            "version": "5.3.0"
               },
```

vspark (basic, full)

The release version and start time of the V-Spark server process.

```
        "vspark": {
           "started": "2018-07-12 12:22:19 -04:00",
           "version": "3.4.3-1"
        },
```

uname (basic, full)

The contents of this field are the same as the output of the UNIX `uname -a` command:

- the operating system kernel name
- the network node hostname (in this case "analysis")
- the release level of the kernel
- the version number of the kernel
- the server's machine hardware name
- the processor type of the server
- the hardware platform of the server
- and the name of the operating system

.

```
        "uname": "Linux analysis 2.6.32-696.30.1.el6.x86_64 #1 SMP Tue May 22 03:28:18 UTC 2018 x86_64 x86_64
x86_64 GNU/Linux",
```

uptime (basic, full)

The contents of this field are the same as the output of the UNIX `uptime` command on the remote server:

- the current time
- how long the system has been running
- how many users are currently logged on
- and the system load averages for the past 1, 5, and 15 minutes

.

```
        "uptime": "13:34:30 up 8 days,  3:10,  1 user,  load average: 0.03, 0.11, 0.21",
```

hostname (basic, full)

The hostname of the server, in this case `sandbox`.

```
        "hostname": "sandbox",
```

jobmgr (basic, full)

The release version and start time of the job manager process.

```
        "jobmgr": {
           "started": "2018-07-12 12:22:28 -04:00",
           "version": "2.3.1-1"
        },
```

mysql (basic, full)

The release version and status information for the mySQL database process.

Sample basic `mysql` output:

```
            "mysql": {
                "version": "5.1.73",
                "uptime": "702822"
            },
```

Sample full `mysql` output:

```
          "mysql": {
             "version": "5.1.73",
             "status": {
                "queries": "38451876",
                "slow_queries": "3",
                "qps": "54.73",
                "rows_deleted": "146527",
                "rows_inserted": "122535",
                "rows_read": "256090844",
                "rows_updated": "11143",
                "bps_in": "26918.02",
                "bps_out": "77637.79",
                "threads_connected": "48",
                "threads_cached": "0",
                "threads_running": "1"
             },
             "uptime": "702547"
```

```
        },
```

vociprocs (full)

The number of V-Spark-related processes running on the system.

```
        "vociprocs": "416",
```

procinfo (full)

The uptime and load averages of the server host, and the task and CPU states of its primary CPU.

```
        "procinfo": [
           "Tasks: 708 total,   3 running, 705 sleeping,   0 stopped,   0 zombie",
           "Cpu(s):  1.2%us,  0.4%sy,  0.0%ni, 98.4%id,  0.0%wa,  0.0%hi,  0.1%si,  0.0%st"
        ],
```

vocirpms (full)

The results of an RPM package manager query for all installed Voci software packages.

```
        "vocirpms": [
           "voci-jobmanager-2.3.1-1.x86_64",
           "voci-server-client-5.4.5-1.x86_64",
           "voci-pyrequests-2.7.0-1.x86_64",
           "voci-user-1.0.0-1.x86_64",
           "voci-nltk-2.0.1rc1-1.x86_64",
           "voci-spark-cluster-3.4.3-1.x86_64",
           "voci-spark-sums-3.4.3-1.x86_64",
           "voci-webapi-2.0.0-1.x86_64",
           "voci-spark-all-3.4.3-1.x86_64",
           "voci-python-2.7.13-1.x86_64",
           "voci-rrdtool-1.4.7-1.x86_64",
           "voci-server-setup-min-1.0.2-1.x86_64",
           "voci-xmltodict-1.0.0-1.x86_64",
           "voci-server-sparklicense-1.2.2-2.x86_64",
           "voci-spark-search-3.4.3-1.x86_64",
           "voci-spark-backend-3.4.3-1.x86_64",
           "voci-spark-service-3.4.3-1.x86_64",
           "voci-python-scipy-1.1.0-1.x86_64",
           "voci-python-numpy-1.14.3-1.x86_64",
           "voci-server-server-5.4.5-1.x86_64",
           "voci-updater-2.0.0-1.x86_64",
           "voci-spark-common-3.4.3-1.x86_64",
           "voci-spark-report-3.4.3-1.x86_64",
           "voci-spark-doc-3.4.3-1.x86_64",
           "voci-pyinotify-0.9.2-1.x86_64",
           "voci-server-licensemgr-sw-5.4.3-1.x86_64",
           "voci-libshorttext-1.1-1.x86_64",
           "voci-admin-1.0.0-2.x86_64",
           "voci-pyyaml-3.10-1.x86_64",
           "voci-python-setuptools-4.0.1-1.x86_64",
           "voci-spark-am-3.4.3-1.x86_64",
           "voci-spark-server-3.4.3-1.x86_64",
           "voci-repo-internal-1.0.3-1.x86_64"
           ],
```

product (basic, full)

The name of the server software product. In this case, V-Spark.

```
        "product": "V-Spark"
```

/transcribe

Use `/transcribe` to submit audio and optional metadata to a folder for transcription and analytics.

Transcription and analytics JSON results include a `requestid` field set to the UUID returned by a successful `/transcribe` request.

> NOTE: All transcription parameters and options are configured for the V-Spark folder that receives the audio. It is therefore unnecessary to provide these parameters in a `/transcribe` request.

Synopsis

```
POST /transcribe/$org_short/$folder -F token=$token -F file=@$input_file
```

`/transcribe` does not accept token or file arguments as query parameters. The `-F` argument in the synopsis is used with cURL to send form parameters.

`$org_short`

Organization short name used to filter the request.

`$folder`

Folder used to filter the request.

`$token`

Authorization token with the required read or write permissions for the request.

> **`$input_file`**
> File path for query input.

Content Types

- POST requests require the "multipart/form-data" MIME type, and return data with the "text/html" MIME type.
- Errors are returned with the "text/html" MIME type.

Example request

This example request submits the local file `0706.mp3.zip` to `folder0` for transcription and analytics.

```
curl -X POST 'http://example.company.com:3000/transcribe/co1-orgA/folder0' -F token=$token -F file=@0706.mp3.zip
```

The previous example request returns a UUID for the request like the following.

```
d41814eb-b055-4169-83a8-e7f2d1f76b18
```

Example shell script implementation

Refer to the transcription and results shell script tutorial for an example of using `/transcribe` and `/request` together.

/transcribe with Amazon S3

To use `/transcribe` with data stored in Amazon Web Service Simple Storage System (AWS S3), include the relevant S3 credentials and bucket location as form parameters with the request.

The following example shows the general format of a cURL `/transcribe` request to transcribe a file or directory that is stored in AWS S3.

```
curl -F token=$token \
     -F aws_id=$aws_access_key \
     -F aws_secret=$aws_secret_key \
     -F s3key=s3://$bucket/$filepath \
     -F region=$s3_region \
     -X POST http://$host/transcribe/$org_short/$folder
```

> **`$host`**
> Hostname or URL for the system running V-Spark.

> **`$token`**

> Authorization token with the required read or write permissions for the request.

> **`$org_short`**

> Organization short name used to filter the request.

> **`$folder`**
> Folder used to filter the request.

> **`$aws_access_key`**

> The Amazon access key for the source data bucket.

> **`$aws_secret_key`**

> The secret Amazon key for the source data bucket.

> **`$bucket`**

> The name for the Amazon S3 source data bucket.

> **`$path`**

> The path to the file, zip, or hierarchy of files to be processed.
>
> Specifying a directory queues all of the files in that directory for transcription.
>
> Files that are not in a supported format are not processed. These files are noted in the V-Spark folder's processing log as UNSUPPORTED.

> **`$s3_region`**

> You **must** specify the Amazon S3 region of the S3 bucket. The region option on the request specifies which regional endpoint to use for the request.
>
> This option reduces request latency and is **required**.

The following example request calls `/transcribe` with a zip stored in S3.

```
curl -F token=0123456789abcde0123456789abcde01 \
     -F aws_id=01234567890123456789O \
     -F aws_secret=0123456789012345678901234567890123456789O \
     -F s3key=s3://example/documentation-TEST.zip \
     -F region=us-east-1 \
     -X POST http://example.company.com:3000/transcribe/Test-Testing/Test01
```

This example submits the file **`documentation-TEST.zip`** in the bucket `example` and puts the results of that transcription in the **`Test01`** folder of the organization `Test-Testing`. The example call returns the `requestid` for the job.

By default, the request only reads data from the specified S3 source and transcribed files remain stored in those buckets.

Include `delete=true` with the `/transcribe` request to delete files stored in S3 after they are processed, like this next example.

```
curl -F token=0123456789abcde0123456789abcde01 \
     -F aws_id=012345678901234567890 \
     -F aws_secret=01234567890123456789012345678901234567890 \
     -F s3key=s3://example/documentation-TEST.zip \
     -F region=us-east-1 \
     -F delete=true \
     -X POST http://example.company.com:3000/transcribe/Test-Testing/Test01
```

NOTE: These examples include escaped newline characters `\` for legibility. Do not include the `\` or newline characters with your requests.

Tokens

Every V-Spark API request requires an authorization token. There are two types of token: **company tokens**, and the system's **root token**.

The root token authorizes any operation for any system entity, and can be used for requests whose scope spans multiple entities with different company tokens.

As a best practice, the root token should not be used except for requests that require root access.

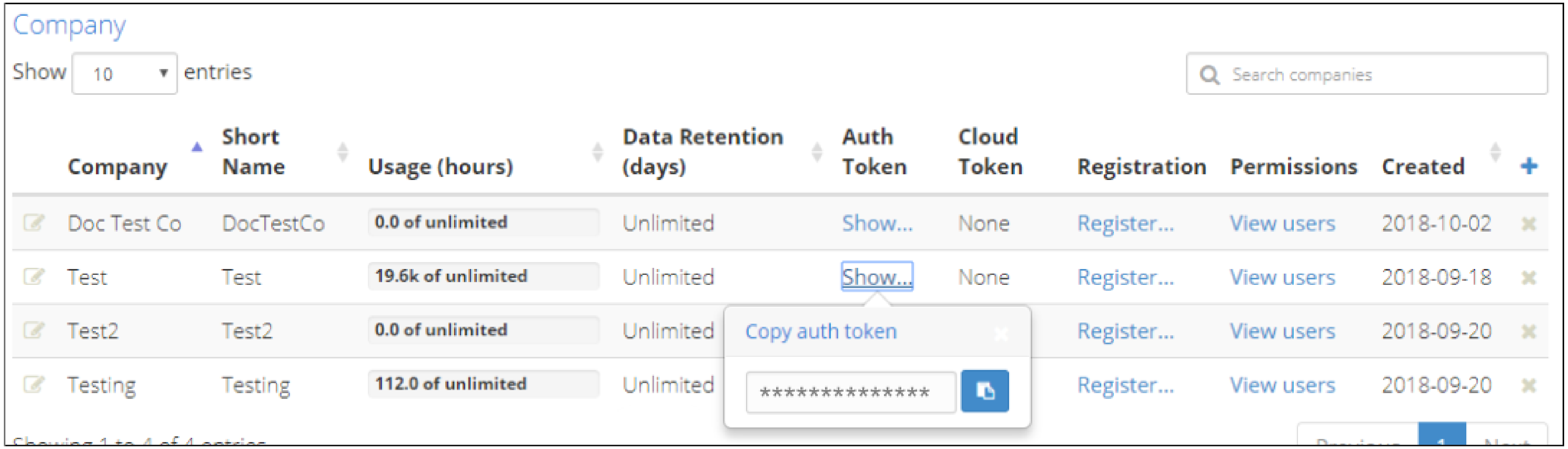
Only system administrators have access to the root token.

By default, V-Spark's root API token is stored in the file `/opt/voci/state/vspark/apitoken` on the host system.

Each company entity on the system has its own API token. Company tokens authorize company- and organization-specific requests. As a best practice, use company tokens for API requests whenever possible.

User accounts with company write permissions can view the company token on the ⚙ `Settings` > `Accounts` page.

Company token location



	Company	Short Name	Usage (hours)	Data Retention (days)	Auth Token	Cloud Token	Registration	Permissions	Created	
	Doc Test Co	DocTestCo	0.0 of unlimited	Unlimited	Show...	None	Register...	View users	2018-10-02	
	Test	Test	19.6k of unlimited	Unlimited	Show...	None	Register...	View users	2018-09-18	
	Test2	Test2	0.0 of unlimited	Unlimited			Register...	View users	2018-09-20	
	Testing	Testing	112.0 of unlimited	Unlimited			Register...	View users	2018-09-20	

```
**************
```

Error codes

V-Spark API endpoints return the errors below. Note that `/transcribe`, `/request`, and `/config` use some endpoint-specific errors.

General

> Can not delete multiple items

> A request to delete multiple objects was made to the `/config` API, but the `multi=true` parameter was not specified

> Can not delete non-empty object

> A request to delete an object that contains other objects was made to the `/config` API, but the `tree=true` parameter was not specified

> Error parsing search results

> The ElasticSearch server returned results based on your call to the `/search` API, but those results could not be converted into the output format used by the V-Spark API for this call

> Invalid JSON

> Any V-Spark API that accepts JSON content does a check to determine the validity of any JSON content that is POSTed, and returns this error if the JSON is not valid

> Invalid output option

> The `/search` was called with an output option other than `count`, `details`, `summary` or `zip`

> Invalid template option: template-name
>
> A request to upgrade an application was made, but the specified application is based on a template that no longer exists in V-Spark. You will have to delete and recreate the application.
>
> Search Error
>
> The command syntax used in a `/search` API call is incorrect, or the ElasticSearch server is not available.

/transcribe

> 400
>
> The `/request` API returns a 400 in cases when the authentication token that is required to access V-Spark is missing or invalid, or when S3 authentication information is invalid. The error text differs based on the cause of the error, and helps identify the cause of the problem:
>
> > Bad request
> >
> > Content is being submitted via the S3 protocol and the S3 key information that is required to access a given bucket was invalid or was not provided.
> >
> > Missing "token" field
> >
> > No authorization token was provided in your call to the `/request` API.
> >
> > Invalid token was provided
> >
> > The authorization token that was used in your call to the `/request` API is not correct.

> 402
>
> The `/transcribe` API returns a 402 in response to attempts to exceed the usage models that are associated with the V-Spark instance that you are running on:
>
> > Usage Limit is reached
> >
> > You have reached the size limit of the amount of audio data that you are licensed to process. To increase that limit, contact your Voci sales representative or send email to Voci product support (support@vocitec.com).

> 404
>
> The `/transcribe` API returns a 404 in response to multiple errors. The error text differs based on the cause of the error, and helps identify the cause of the problem:
>
> > Company not found
> >
> > V-Spark can look up the name of the company that you are using based on the value that you passed for the `ORG_SHORT` parameter. The value that you specified for this parameter is incorrect.
> >
> > Folder folder-name not found
> >
> > The folder that was passed as a parameter does not exist or cannot be accessed.
> >
> > Organization organization-name not found
> >
> > The organization whose short name was passed as a parameter does not exist or cannot be accessed.

> 406
>
> The `/transcribe` API returns a 406 in response to errors where the data that it is POSTing is not acceptable to its client, the V-Spark

server. The error text differs based on the cause of the error, and helps identify the cause of the problem:

> File size is too large. File should be smaller than LIMIT
>
> The size of the uploaded file is larger than the displayed `LIMIT`. This limit is configurable via the `transcribe_api_upload_limit` configuration variable, so the text of this error may vary.

500

The `/transcribe` API returns a 500 in response to serious internal errors that reflect a hardware, software, or configuration error on the system where V-Spark is running. Contact Voci product support (support@vocitec.com) for assistance in identifying and resolving the problem.

/request

400

The `/request` API usually returns a 400 in cases when the authentication token that is required to access V-Spark is missing or invalid. The error text differs based on the cause of the error, and helps identify the cause of the problem:

> Auth token doesn't match
>
> The authorization token that was used in your call to the `/request` API is not correct.
>
> Request not finished
>
> The results that you requested are not yet available.

404

The `/request` API returns a 404 in response to multiple errors. The error text differs based on the cause of the error, and helps identify the cause of the problem:

> **Company not found**
>
> V-Spark can look up the name of the company that you are using based on the value that you passed for the `ORG_SHORT` parameter. This error means that the value that you specified for this parameter was incorrect.
>
> **No file types were specified**
>
> The default JSON output format was disabled in your call to the `/request` API, but you did not enable another format (`mp3` or `text`)
>
> **Organization organization-name not found**
>
> The organization whose short name was passed as a parameter does not exist or cannot be accessed
>
> **RequestFile not found**
>
> This message indicates a problem communicating with the database that is used by V-Spark. Retrying the operation should succeed. If you continue to see this message, contact Voci product support (support@vocitec.com).
>
> **RequestId not found**
>
> The request ID that was passed as a parameter does not exist. Check for typographical errors if testing the `/request` API call from the command line, or check your application code to ensure that you are storing and using a valid request ID.

/config/folders

> 400

> The `/config/folders` API returns a 400 in cases when the configuration being set via the update is invalid. The error text differs based on the cause of the error, and helps identify the cause of the problem:
>
> > Folder [folder name] is disabled due to company-level policies
>
> > You are attempting to resume processing of a paused folder, but that folder cannot be resumed because it has been disabled. Usually, this is because the folder was paused due to company limit hours being met.

Tutorials

The tutorials in this section demonstrate some common V-Spark API use cases.

> Create and configure analytics applications

> Combine `/config/apps`, `/appedit`, and `/config/folders` requests to create an application, upload its configuration, and associate it with a folder.

> GET deletion job status with /config

> Use `/config` to retrieve deletion job status.

> Manage accounts with /config/users

> Use `/config/users` to retrieve, create, and configure V-Spark user accounts, to retrieve user account status, to retrieve the authorization method used for login, and to retrieve or configure the permissions that they have in V-Spark and within each company.

> Python scripts

> These example scripts were written in Python 2.7. They are included here for example purposes only and should not be used for production environments.

> Shell script for transcription and results

> This section shows a simple Linux Bash shell script that demonstrates using the `/transcribe` API to upload a file for transcription, and using the `/request` to monitor the status of processing that file and retrieve results once transcription has completed.

> Use callbacks to receive results

> This tutorial shows setting up a simple callback server, submitting audio for transcription with `/transcribe`, and examining the results, along with suggestions for troubleshooting common problems when setting up and using a callback server.

> Use **`fields`** to refine /search output

> When specifying `output=summary` in a `/search` request, use the `fields` parameter to define the data fields to be included with or excluded from search results.

> Using cURL

> Use cURL to test the V-Spark API from the command line. cURL is not required to use the V-Spark API, but it can be a helpful tool, and is freely available for most operating systems.

Create and configure analytics applications

Combine `/config/apps`, `/appedit`, and `/config/folders` requests to create an application, upload its configuration, and associate it with a folder.

The following sample JSON defines an application named `New AppEdit Test`.

Sample JSON that defines an Application

```
{
    "Technologies": {
        "Technologies-RD": {
            "New AppEdit Test": {
                "created": "2017-10-10",
                "defaultscoretype": "Hit/Miss",
                "enabled": "on",
                "folders": [
                    "AutoTests"
                ],
                "template": "custom"
            }
        }
    }
}
```

When programmatically creating an application, populating it, and binding it to a folder, you must create the application before you can do either of the other two tasks. The next sample shows the JSON that associates a folder with the application that was defined previously.

Sample JSON that Associates an Application with a Folder

```
{
    "Technologies": {
        "Technologies-RD": {
            "AutoTests": {
                "apps": [
                    "New AppEdit Test"
                ],
                "custom_meta": [],
                "modelchan0": "eng1:callcenter",
                "modelchan1": "spa1:callcenter",
                "nspeakers": 2,
                "servers": [
                    "asrsrvr1",
                    "asrsrvr8"
                ]
            }
        }
    }
}
```

The next sample shows an abbreviated version of the JSON configuration of an application.

Sample JSON for an Application

```
{
    "Sample Top Level Category": {
        "phrases": {
            "+": {
                "all": [
                    "phrase usually found in all calls of this category"
                ]
            },
            "-": {
                "all": [
                    "phrase that shouldn't occur in calls of this category"
                ]
            }
        },
        "subcategories": {
            "Sample 2nd Level Category": {
              ...
            },
            "Sample 2nd Level Leaf Category": {
            ...
            }
        }
    },
    "Sample Top Level Leaf Category": {
        "phrases": {
           ...
        },
        "subcategories": {}
    }
}
```

Once you have JSON that provides information about these three aspects of an application, you can create the application, bind it to a folder, and define its configuration with three cURL calls like the following:

1. Calls the `/config/apps` endpoint to create the application, using the contents of the file `app-definition.json`.

   ```
   curl -s -X POST -H "Content-Type:application/json" "http://example.company.com:3000/config/apps?token=$token" --
   data @app-definition.json
   ```

2. Calls the `/config/folders` endpoint to associate the new application with a specified folder, using the contents of the file `app-folder-binding.json`.

   ```
   curl -s -X POST -H "Content-Type:application/json" http:///example.company.com:3000/config/folders?token=$token"
   --data @app-folder-binding.json
   ```

3. Calls the `/appedit` endpoint to upload the configuration of the application, using the contents of the file `app-configuration-sample.json`.

   ```
   curl -s -X POST -H "Content-Type:application/json" "http:///example.company.com:3000/appedit/Technologies/
   Technologies-RD/AppEdit%20Test?token=$token" --data @app-configuration-sample.json
   ```

While cURL provides a quick way to use and test REST APIs, and shell scripts are a quick and convenient way of automating many tasks, it is typically faster in the long run to call APIs from applications. The next topic discusses how to use the `/appedit` API from within applications that are written in the Python programming language.

GET deletion job status with /config

Use `/config` to retrieve deletion job status.

The JSON returned by GET `/config` requests for companies, organizations, and folders includes a `status` field with one of the following values.

> OK

> No operations are in progress regarding the queried object

> deleting

> The queried object is in the process of being deleted

> deleting (XX%)

> In long-running deletion tasks for companies and organizations, the queried object is in the process of being deleted and shows the approximate percentage (as an integer value) of the delete operation that has completed

The sample below shows JSON that is returned by a call to the `/config /COSHORT/ORGSHORT/FOLDER` API when no delete operation is in progress.

Folder Status When No Delete Operation is In Progress

```
{
    "servers": [
        "asrsrvr1"
    ],
    "nspeakers": 2,
    "audiotype": "Stereo",
    "created": "2017-09-05",
    "purifyaudio": false,
    "purifytext": true,
    "modelchan0": "eng1:callcenter",
    "status": "OK",
    "modelchan1": "spa1:callcenter",
    "agentchan": 0,
    "mode": "active",
    "callback": {},
    "apps": [],
    "asroptions": {},
    "custom_meta": []
}
```

The next sample shows JSON that is returned by a call to the `/config/ COSHORT/ORGSHORT/FOLDER` API when a delete operation is in progress.

Folder Status When a Delete Operation is In Progress

```
{
    "servers": [
        "asrsrvr1"
    ],
    "nspeakers": 2,
    "audiotype": "Stereo",
    "created": "2017-09-05",
    "purifyaudio": false,
    "purifytext": true,
    "modelchan0": "eng1:callcenter",
    "status": "deleting",
    "modelchan1": "spa1:callcenter",
    "agentchan": 0,
    "mode": "active",
    "callback": {},
    "apps": [],
    "asroptions": {},
    "custom_meta": []
}
```

After using the `/config` API's DELETE method, it is a good idea to call the `/config` API's GET method for the object that you requested deletion of. If the GET call returns JSON like that shown in this section's second example, the folder is in the process of being deleted.

If the call returns " `Folder not found:` $folder", it means that the DELETE operation has completed. When calling the `/config` API to get DELETE status information, you will therefore need to test for both a successful return code (where JSON is returned, and the `status` field in that JSON is one of `OK`, `deleting` or `deleting(` NN `)` and a return message which indicates that the queried object does not exist, and has therefore already been deleted.

The return message `deleting( NN )` can be returned when deleting companies or organizations, where NN is an integer value that gives an estimate of the percentage of the delete operation that has completed.

Manage accounts with /config/users

Use `/config/users` to retrieve, create, and configure V-Spark user accounts, to retrieve user account status, to retrieve the authorization method used for login, and to retrieve or configure the permissions that they have in V-Spark and within each company.

NOTE: There are 3 user attributes the `/config/users` cannot update:

- `approved` — Account approval status cannot be set to 1.
- `auth.method` — Authentication method, which can only be specified when creating a user.
- `password`

`/config` POST requests must include entity information as top-level JSON data. Higher-level entity information does not need to be included in JSON POST data when entity names are included as path parameters.

Data written to V-Spark with `/config` is additive—that is, if objects in request JSON data exist, they are configured to use the request JSON. Objects that do not exist are created.

Set user permissions

Use `/config/users/` to set `View` (read) and `Create/Edit` (write) permissions for a user. Account permissions may be set for the system, a company, or an organization, depending on the request token's permissions scope.

IMPORTANT: When viewing or modifying user permissions via the API but verifying them in the GUI, you must be logged in to the GUI as a user who is authorized to see any changes that have been made. User accounts see only changes that have been made at a level that is equal to or lower than the authorization level of the account that made the change.

System permissions

System administrator access gives the user account read and write permissions to every aspect of the V-Spark host system. That means they can create, delete, and modify any company, organization, folder, application, or user account on the system.

Sysadmins can also approve user accounts, change user account passwords, create and delete system-wide announcements, and enable system-wide `readonly` mode.

The following example JSON describes a user account with sysadmin access.

```
"DocTestCo": {
    "test.user.01": {
        "name": "System Administrator",
        "email": "test.user.01@company.com",
        "company": "DocTestCo",
        "auth": {
            "verified": false,
            "disabled": false,
            "method": "standard"
            },
        "permissions": {
            "system": [
                "write"
            ]
        }
    }...
}...
```

Company permissions

Company-level access gives the user account read and write permissions for individual companies.

Company **`View`** (read) permission enables the user to view dashboards and transcripts for any organization under that company.

Company **`Create/Edit`** (write) permission enables the user to

- create and add user accounts to that company.
- modify permissions for company user accounts.
- create and edit company organizations, folders, and applications.

The following example JSON describes a user account with read and write access to the company `DocTestCo`.

```
"manual.user.03": {
    "auth": {
        "disabled": false,
        "verified": true,
        "method": "standard"
        },
    "company": "DocTestCo",
    "email": "manual.user.03@company.com",
    "name": "Manual User 03",
    "permissions": {
        "DocTestCo": {
            "all": [
                "read",
                "write"
            ]
        }
    }...
}...
```

Organization permissions

Organization-level access gives the user account read and write permissions for individual organizations.

Organization **`View`** (read) permission enables the user to view organization dashboards and transcripts.

Organization **`Create/Edit`** (write) permission enables the user to create and modify organization folders and applications.

The following example JSON describes a user account with read and write access to the organization `DocTesting`, which is under the company `DocTestCo`.

```
"manual.user.03": {
   "auth": {
      "disabled": false,
      "verified": true,
      "method": "standard"
   },
   "company": "DocTestCo",
   "email": "manual.user.03@company.com",
   "name": "Manual User 03",
   "permissions": {
      "DocTestCo": {
         "orgs": {
            "DocTestCo-DocTesting": [
               "read",
               "write"
```

```
                ]...
            }
        }...
    }...
}...
```

GET and POST JSON differences

The `/config/users` API supports additional name/value pairs that can be used as part of your JSON input when programmatically creating user accounts. These fields are in addition to those in sample JSON output.

/config/users/ Fields

Name	Type	Values	Description
`password`			Enables you to specify the password that will be assigned to a user account when it is created. An example of specifying a password using this field is the following: `"password": "changeme",` If the `password` field is not included in the `auth` section of the JSON for a user, a reset password link will be emailed to the new user. The sample JSON below is for a user who has system administration permissions for their home company (`DocTestCo` , in this case) and has a default password of `changeme` .

Name	Type	Values	Description
			Sample User JSON Including a Password Field

```
{
  "test.user.02": {
    "name": "Company-Level Administrator",
    "email": "test.user.02@example.com",
    "company": "DocTestCo",
    "auth": {
      "verified": true,
      "disabled": false,
      "method": "standard",
      "password": "changeme"
    },
    "permissions": {
      "DocTestCo": {
        "all": [
          "read",
          "write"
        ]
      }
    }
  }
},...
```

Name	Type	Values	Description
`method`		standard, oidc	Specify the user's login authorization method. `standard` specifies internal V-Spark authorization. `oidc` specifies OpenID Connect for single sign-on implementations. Once a user's authorization method has been set, it *cannot* be changed.

IMPORTANT: When creating a user account that will be integrated with an external authorization mechanism, the **Username** for the account that you are creating must be the same in V-Spark as it is in the external authorization mechanism. This username may be a simple username, an email address, or a "user principle name" (UPN), depending on the service.

Optional `/config/users/` fields can also be used in JSON sent to `/config/CO-SHORT/users/`. The main difference between those two endpoints is that `/config/users/` JSON must include the primary company for each user.

Python scripts

These example scripts were written in Python 2.7. They are included here for example purposes only and should not be used for production environments.

> GET request for config endpoints

> Use `api_get_config.py` to test GET requests.

> GET all company tokens and system entity hierarchy

> Use `get_deep_config.py` to print the system's company tokens and entity hierarchy to the console using multiple /config requests.

> GET request for multiple endpoints

> Use `api_get_test.py` to make a GET request to certain V-Spark endpoints.

> POST entity configuration

> Use `api_post_config.py` to test POST requests to the /config endpoint and to verify the expected HTTP return code.

> POST request with a JSON file

> Use this Python script to make a POST request to certain V-Spark endpoints.

> GET or POST **`/search`** requests

> Use `get_search.py` to test GET requests with `/search`. This script saves results and match count to a file.
>
> Use `post_search.py` to test POST requests with `/search`. This script saves results and match count to a file.

GET request for multiple endpoints

Use `api_get_test.py` to make a GET request to certain V-Spark endpoints.

api_get_test.py

```
#!/usr/bin/env python
# Unsupported example code - Not for production use.

import sys
import json
import urllib2
import string

# default values
PROTOCOL =  "http://"
PORT =  "3000"

if ( len(sys.argv) != 5 ):
    print "  Usage:", sys.argv[0], "HOST ROOT_TOKEN API_TO_CALL PARAMS"
    sys.exit(-1)
else:
    # get cmdline params
    HOST, ROOT_TOKEN, API_TO_CALL, PARAMS = sys.argv[1:]

# Define the URL in a single variable for JSON load
url =  "%s%s:%s%s?token=%s&%s" % (PROTOCOL, HOST, PORT, API_TO_CALL, ROOT_TOKEN, PARAMS)

# To get output data, return a Python object and dump it to a string that is a
#   JSON representation of that object. Complain and exit if the call fails.
try:
    data = json.load(urllib2.urlopen(url))
except urllib2.HTTPError, error:
    print '  Error: HTTP message: ', error.msg, ' HTTP return code: ', str(error.code)
    sys.exit(-1)

# Sanitize URL and params for use in creating output file name
tmp_str = string.replace(
    string.replace(
        string.replace(
            string.replace(API_TO_CALL+"_"+PARAMS+".json", '/', '_'), '&', '_'), '?', '_'), '%20', '_')
target = open(tmp_str[1:], 'w')

target.write(json.dumps(data, indent=4, sort_keys=True))
target.close()
print "  Output written to: "+tmp_str[1:]
```

Use `api_get_test.py` with the following command and arguments:

```
python api_get_test.py $host $token $endpoint '$options'
```

`$host`

Hostname or URL for the system running V-Spark.

`$token`

Authorization token with the required read or write permissions for the request.

`$endpoint`

Request endpoint, including any path parameters.

> **`$options`**
>
> Optional parameters used to further refine the request and results.

POST request with a JSON file

Use this Python script to make a POST request to certain V-Spark endpoints.

api_post.py

```
#!/usr/bin/env python
# Unsupported example code - Not for production use.

import sys
import json
import requests

# default values
PROTOCOL =  "http://"
PORT =  "3000"

if ( len(sys.argv) != 5 ):
    print "  Usage:", sys.argv[0], "HOST ROOT_TOKEN API_TO_CALL INPUT_JSON_FILE"
    sys.exit(-1)
else:
    HOST, ROOT_TOKEN, API_TO_CALL, INPUT_JSON_FILE = sys.argv[1:]

# Define the URL in a single variable for JSON load
url =  "%s%s:%s%s?token=%s" % (PROTOCOL, HOST, PORT, API_TO_CALL, ROOT_TOKEN)

print "POSTing data from " + INPUT_JSON_FILE + " to" + API_TO_CALL
with open(INPUT_JSON_FILE) as json_file:
    json_data = json.load(json_file)
headers = {'Content-type': 'application/json'}
response = requests.post(url, headers=headers, data=json.dumps(json_data))
if response.status_code != 200:
    print '  HTTP message: ', response.reason, ' HTTP return code: ', str(response.status_code)
```

Use the script with the following command and arguments:

```
python api_post.py $host $token $endpoint $json
```

`$host`
Hostname or URL for the system running V-Spark.

`$token`

Authorization token with the required read or write permissions for the request.

`$endpoint`
Request endpoint, including any path parameters.

`$json`
Path to the request's JSON file.

GET request for config endpoints

Use `api_get_config.py` to test GET requests.

api_get_config.py

```
#!/usr/bin/env python
# Unsupported example code - Not for production use.

import sys
import json
import urllib2
import requests

# default values
#
PROTOCOL =  "http://"
PORT =  "3000"

    if ( len(sys.argv) != 6 ):
        print "  Usage:", sys.argv[0], "HOST API_ROOT_TOKEN API_TO_CALL HTTP_CODE_TARGET OUTPOST_FILE"
        sys.exit(-1)
    else:
        # get cmdline params
        HOST, ROOT_TOKEN, API_TO_CALL, HTTP_CODE_TARGET, OUTPOST_FILE = sys.argv[1:]

# Define the URL in a single variable for JSON load
url =  "%s%s:%s%s?token=%s" % (PROTOCOL, HOST, PORT, API_TO_CALL, API_ROOT_TOKEN)

print "Checking " + API_TO_CALL + " on " + HOST + " and writing output to " + OUTPOST_FILE
print "  URL is " + url

response = requests.get(url)
```

```
print '  SUMMARY (GET): ' + API_TO_CALL, ': HTTP message: ', \
         response.reason, ' HTTP return code: ', \
         str(response.status_code), ' expected ' + HTTP_CODE_TARGET

target = open(OUTPOST_FILE, 'w')

# To get output data, return a python object and dump it to a string
#   that is a JSON representation of that object
data = json.load(urllib2.urlopen(url))

# pretty-print the result
target.write(json.dumps(data, indent=4, sort_keys=True))

target.close()
```

`api_get_config.py` makes a GET request, compares expected and actual HTTP return codes in a console output message, and saves the response of the GET request to a file.

Use this script with the following command and arguments:

```
python api_get_config.py $host $token $endpoint $http_code $output_file
```

> **`$host`**
> Hostname or URL for the system running V-Spark.

> **`$token`**

> Authorization token with the required read or write permissions for the request.

> **`$endpoint`**
> Request endpoint, including any path parameters.

> **`$http_code`**
> The HTTP code expected to be returned with the response.

> **`$output_file`**
> File path for query output.

GET all company tokens and system entity hierarchy

Use `get_deep_config.py` to print the system's company tokens and entity hierarchy to the console using multiple /config requests.

get_deep_config.py

```
#!/usr/bin/env python

# Copyright 2017 Voci Technologies All rights reserved.
# Unsupported example code - Not for production use.

# Uses root token to read company and folder information from the specified host
#
# Prints a hierarchical listing of available companies (each with its
# associated authorization token), the organizations within those
# companies, and the folders within those organizations.

import requests

def usage(argv):
    print "Usage:", argv[0], "<sparkhost:port> <root token>"
    exit(1)

def main(argv):
    if len(argv) != 3: usage(argv)
    host, token = argv[1:]
    tokens = gettokens(host,token)
    folderinfo = getfolderinfo(host,token)
    printfolders(host, folderinfo, tokens)

def gettokens(host, token):
    url = "http://%s/config?token=%s" % (host,token)
    cfg = requests.get(url).json()
    return dict([(comp,d['uuid']) for comp,d in cfg.iteritems()])

def getfolderinfo(host, token):
    url = "http://%s/config/folders?token=%s" % (host,token)
    return requests.get(url).json()

def printfolders(host, folder_info, tokens):
    for comp, comp_data in folder_info.iteritems():
        print comp+" (Token: "+tokens[comp]+")"
        for org, org_data in comp_data.iteritems():
            print "\t", org
            for folder, folder_data in org_data.iteritems():
                print "\t\t", folder

if __name__ == '__main__':
    from sys import argv
    main(argv)
```

`get_deep_config.py` prints a table of company tokens to the console, along with a hierarchy of the system's company, organization, and folder entities.

Use this script with the following command and arguments:

```
python get_deep_config.py $host:$port $root_token
```

`$host`

Hostname or URL for the system running V-Spark.

> **`$port`**
> Port configured for the host system. Default `3000` .

> **`$root_token`**
> V-Spark's root authorization token. Only available to system administrators.

POST entity configuration

Use `api_post_config.py` to test POST requests to the /config endpoint and to verify the expected HTTP return code.

api_post_config.py

```
#!/usr/bin/env python

# Copyright 2017 Voci Technologies All rights reserved.
# Unsupported example code - Not for production use.

import sys
import json
import requests

# default values
PROTOCOL =  "http://"
PORT = "3000"

if ( len(sys.argv) != 6 ):
    print "  Usage:", sys.argv[0], "HOST ROOT_TOKEN API_TO_CALL TARGET_HTTP_CODE INPOST_JSON_FILE"
    sys.exit(-1)
else:
    HOST, ROOT_TOKEN, API_TO_CALL, TARGET_HTTP_CODE, INPOST_JSON_FILE = sys.argv[1:]

# Define the URL in a single variable for JSON load
url =  "%s%s:%s%s&token=%s" % (PROTOCOL, HOST, PORT, API_TO_CALL, ROOT_TOKEN)

print "Checking " + API_TO_CALL + ", POSTing input from " + INPOST_JSON_FILE
print "  Whole URL: "+url

with open(INPOST_JSON_FILE) as json_file:
    json_data = json.load(json_file)
```

```
response = requests.put(url, data=json.dumps(json_data))

print '  SUMMARY (POST): ' + API_TO_CALL, ': HTTP message: ', response.reason, ' HTTP return code: ',
str(response.status_code), ' expected ' + TARGET_HTTP_CODE
```

`api_post_config.py` makes a POST request with the input JSON to create or configure the specified entity, and it compares expected and actual HTTP return codes in a console output message.

Use this script with the following command and arguments:

```
python api_post_config.py $host $token $endpoint $http_code $input_file
```

`$host`
Hostname or URL for the system running V-Spark.

`$token`

Authorization token with the required read or write permissions for the request.

`$endpoint`
Request endpoint, including any path parameters.

`$http_code`
The HTTP code expected to be returned with the response.

`$input_file`
File path for query input.

GET or POST `/search` requests

GET

Use `get_search.py` to test GET requests with `/search` . This script saves results and match count to a file.

get_search.py

```
#!/usr/bin/env python
#
# Copyright 2017 Voci Technologies  All rights reserved.
# Unsupported example code - Not for production use.
#

import requests
import json
import urllib2

def usage(argv):
    print "Usage:", argv[0], "<sparkhost:port> <root token> <company> <params>"
    exit(1)

def main(argv):
    if len(argv) != 5: usage(argv)
    host, token, company, searchparams = argv[1:]
    tokens = gettokens(host,token)
    folderinfo = getfolderinfo(host,token)
    findfolders(host, folderinfo, tokens, company, searchparams)

def gettokens(host, token):
    url = "http://%s/config?token=%s" % (host,token)
    cfg = requests.get(url).json()
    return dict([(comp,d['uuid']) for comp,d in cfg.iteritems()])

def getfolderinfo(host, token):
    url = "http://%s/config/folders?token=%s" % (host,token)
    return requests.get(url).json()

def findfolders(host, folder_info, tokens, company, searchparams):
    for comp, comp_data in folder_info.iteritems():
        if comp == company:
            print "Searching folders under "+company+" (Token: "+tokens[comp]+")"
            for org, org_data in comp_data.iteritems():
                for folder, folder_data in org_data.iteritems():
                    searchandprintresults(host, tokens[comp], comp, org, folder, \
                                          searchparams)

def searchandprintresults(host, token, comp, org, folder, searchparams):
    url = "http://%s/search/%s/%s/%s?token=%s%s" % (host, comp, org, folder, token, searchparams)
    response = requests.get(url)
    if response.status_code == 200:
        print "  URL is "+url
        counturl = url+"&output=count"
        countresponse = requests.get(counturl)
        OUTPUT_FILE = comp+"-"+org+"-"+folder+"-search.json"
        print "    Writing Matching JSON for "+countresponse.text+" matches to "+OUTPUT_FILE
        target = open(OUTPUT_FILE, 'w')
        data = json.load(urllib2.urlopen(url))
        target.write(json.dumps(data, indent=4, sort_keys=True))
        target.close()

if __name__ == '__main__':
    from sys import argv
    main(argv)
```

This example command uses `get_search.py` to search `DocTestCo` for all records with a positive client emotion value.

```
python get_search.py example.company.com $token DocTestCo '&client_emotion=positive'
```

`get_search.py` saves output to a file like the next example.

```
Searching folders under DocTestCo (Token: $token)
  URL is http://example.company.com/search/DocTestCo/DocTestCo-DocTesting/Test01?token=$token&client_emotion=positive
    Writing Matching JSON for 4 matches to DocTestCo-DocTestCo-DocTesting-Test01-log.json
```

POST

Use `post_search.py` to test POST requests with `/search`. This script saves results and match count to a file.

post_search.py

```
#!/usr/bin/env python
#
# Copyright 2017 Voci Technologies  All rights reserved.
# Unsupported example code - Not for production use.
#

import requests
import json
import urllib2

def usage(argv):
    print "Usage:", argv[0], "<sparkhost:port> <root token> <company> <JSON-params-file>"
    exit(1)

def main(argv):
    if len(argv) != 5: usage(argv)
    host, token, company, searchparamfile = argv[1:]
    tokens = gettokens(host,token)
    folderinfo = getfolderinfo(host,token)
    findfolders(host, folderinfo, tokens, company, searchparamfile)

def gettokens(host, token):
    url = "http://%s/config?token=%s" % (host,token)
    cfg = requests.get(url).json()
    return dict([(comp,d['uuid']) for comp,d in cfg.iteritems()])

def getfolderinfo(host, token):
    url = "http://%s/config/folders?token=%s" % (host,token)
    return requests.get(url).json()

def findfolders(host, folder_info, tokens, company, searchparamfile):
    for comp, comp_data in folder_info.iteritems():
        if comp == company:
            print "Searching folders under "+company+" (Token: "+tokens[comp]+")"
            for org, org_data in comp_data.iteritems():
                for folder, folder_data in org_data.iteritems():
                    searchandprintresults(host, tokens[comp], comp, org, folder,
                        searchparamfile)

def searchandprintresults(host, token, comp, org, folder, searchparamfile):
    url = "http://%s/search/%s/%s/%s?token=%s" % (host, comp, org, folder, token)
    with open(searchparamfile) as json_file:
        param_data = json.load(json_file)
    header = {'Content-type': 'application/json'}
    response = requests.post(url, data=json.dumps(param_data), headers=header)
    if response.status_code == 200:
        print "  URL is "+url
        param_data["output"] = "count".decode('utf-8')
        countresponse = requests.post(url, data=json.dumps(param_data), headers=header)
        OUTPUT_FILE = comp+"-"+org+"-"+folder+"-post-search.json"
        print "    Writing Matching JSON for "+countresponse.text+" matches to "+OUTPUT_FILE
        with open(OUTPUT_FILE, mode='wb') as localfile:
            localfile.write(response.content)
        localfile.close()

if __name__ == '__main__':
    from sys import argv
    main(argv)
```

This example command uses `post_search.py` to search `DocTestCo` for all records that match the search terms defined in `params.json`.

```
python post_search.py example.company.com $token DocTestCo params.json
```

`post_search.py` saves output to a file like the next example.

```
Searching folders under DocTestCo (Token: $token)
  URL is http://example.company.com/search/DocTestCo/DocTestCo-DocTesting/Test01?token=$token
    Writing Matching JSON for 4 matches to DocTestCo-DocTestCo-DocTesting-Test01-search.json
```

GET status of all system folders

Use `api_config_status.py` to retrieve status information for all system folders with a combination of /config and /status requests. This api_config_status.py

```
#!/usr/bin/env python
#
# Copyright 2017 Voci Technologies  All rights reserved.
# Unsupported example code - Not for production use.
#
# Reads company, org, and folder data from a specified host,
# uses it to call the /status endpoint for each folder,
# and saves output as JSON or CSV as specified.

import requests
import json

def usage(argv):
    print "Usage:", argv[0], "<sparkhost:port> <root token> <csv || json>"
    exit(1)

def main(argv):
    if len(argv) != 4: usage(argv)
    host, token, output_format = argv[1:]
    tokens = gettokens(host,token)
    folderinfo = getfolderinfo(host,token)
    getfolders(host, folderinfo, tokens, output_format)

def gettokens(host, token):
    url = "http://%s/config?token=%s" % (host,token)
    cfg = requests.get(url).json()
    return dict([(comp,d['uuid']) for comp,d in cfg.iteritems()])
```

```
def getfolderinfo(host, token):
    url = "http://%s/config/folders?token=%s" % (host,token)
    return requests.get(url).json()

def findfolders(host, folder_info, tokens, output_format):
    for comp, comp_data in folder_info.iteritems():
        print comp+" (Token: "+tokens[comp]+")"
        for org, org_data in comp_data.iteritems():
            for folder, folder_data in org_data.iteritems():
                writefolderstatus(host, tokens[comp], comp, org, folder, output_format)

def writefolderstatus(host, token, comp, org, folder, output_format):
    url = "http://%s/status/%s/%s/%s?token=%s&format=%s" % (host, comp, org, folder, token, output_format)
    print "  URL is "+url
    OUTPUT_FILE = comp+"-"+org+"-"+folder+"-status."+output_format
    print "    Writing JSON to "+OUTPUT_FILE
    target = open(OUTPUT_FILE, 'w')
    if output_format == "json":
        target.write(json.dumps(requests.get(url).json(), indent=4, sort_keys=True))
    if output_format == "csv":
        response = requests.get(url)
        target.write(response.content)
        target.write('\n')
    target.close()

if __name__ == '__main__':
    from sys import argv
    main(argv)
```

`api_config_status.py` uses the root API token to retrieve the processing status for all system folders.

Use this script with the following command and arguments:

```
python api_config_status.py $host:$port $root_token $format
```

> **`$host`**
> Hostname or URL for the system running V-Spark.

> **`$port`**
> Port configured for the host system. Default `3000` .

> **`$root_token`**
> V-Spark's root authorization token. Only available to system administrators.

> $format
> Specify `json` or `csv` to be used for the output file generated by the script.

`api_config_status.py` saves output to a file named with this format: co_short-org_short-$folder-status.$format

> **`$co_short`**

> Company short name used to filter the request.

> **`$org_short`**

> Organization short name used to filter the request.

> **`$folder`**
>
> Folder used to filter the request.

> **`$format`**
>
> The output format specified with the initial command-line argument.

Shell script for transcription and results

This section shows a simple Linux Bash shell script that demonstrates using the `/transcribe` API to upload a file for transcription, and using the `/request` to monitor the status of processing that file and retrieve results once transcription has completed.

> **NOTE:** When retrieving results using the `/request` API, note that the output is written to standard output.

```
#!/bin/bash
#
# Sample script for using the transcribe and request APIs together
#

echo "TEST: Running API tests for transcribe/result mode..."
echo ""

#
if [ $# != 2 ] ; then
    echo "Usage: $0 host token"
    exit
fi

SERVER=$1
TOKEN=$2

# Uncomment the value of FILE that you want to test with: the
# "standard" version with a file extension, the "guess what kind of
# file" version without a file extension, the standard audio file
# without an extension, or a 7z file. If you're trying something
# without an extension, you'll also have to uncomment the appropriate
# "cp" line.
#
# wvh 03-May-2017
#
# cp ../SAMPLES/CallTEST.mp3 ../SAMPLES/CallTEST
# cp ../SAMPLES/CallTEST.zip ../SAMPLES/CallTEST
# cp ../SAMPLES/CallTEST.7z ../SAMPLES/CallTEST
# SAMPLEFILE=../SAMPLES/CallTEST
# SAMPLEFILE=../SAMPLES/CallTEST.7z
# SAMPLEFILE=../SAMPLES/CallTEST.mp3
# SAMPLEFILE=../SAMPLES/audio-and-metadata.zip
# SAMPLEFILE=../SAMPLES/spanish.zip
# SAMPLEFILE=../SAMPLES/CallTEST.zip
# SAMPLEFILE=../SAMPLES/PERMANENT-FILE.ZIP
# SAMPLEFILE=../SAMPLES/FOO.zip


SAMPLEFILE=../SAMPLES/audio-and-metadata.zip

# SAMPLEFILE=DocTestCo-DocTesting-Test01-Fixed.zip

if [ ! -f $SAMPLEFILE ] ; then
    echo "  Specified upload ($SAMPLEFILE) does not exist!"
    exit
fi

SHORTORG=DocTestCo-DocTesting
FOLDER=Test01
```

```
DEBUG="--trace-ascii debug.out"

echo -n "Type of file to upload is: "
file $SAMPLEFILE

echo ""
echo "Submitting $SAMPLEFILE for transcription:"

CMD="curl -s -F token=$TOKEN -F \"file=@$SAMPLEFILE;type=application/zip\" -X POST $SERVER:3000/
transcribe/$SHORTORG/$FOLDER $DEBUG"

echo "CMD is $CMD"
echo ""

echo "  SUMMARY: Uploading $SAMPLEFILE for transcription at approximately $(date)"

REQUESTID=`curl -s -F token=$TOKEN \
     -F "file=@$SAMPLEFILE;type=application/zip" \
     -X POST $SERVER:3000/transcribe/$SHORTORG/$FOLDER $DEBUG`

# REQUESTID=`$CMD`

echo ""

if [[ ${REQUESTID} =~ ^[0-9a-zA-Z]{8}-[0-9a-zA-Z]{4}-[0-9a-zA-Z]{4}-[0-9a-zA-Z]{4}-[0-9a-zA-Z]{12} ]] ; then
    echo "  SUMMARY: Upload successful - requestID is \"${REQUESTID}\"..."
else
    echo "  SUMMARY: RequestID is \"${REQUESTID}\", which is not a valid request ID.."
    echo "  SUMMARY: Cannot transcribe..."
    exit
fi

STATUS=""
# number of second to sleep between retries of status into
SLEEP=10
TOTALWAIT=0

echo "Retrieving status for item submitted via transcribe API..."
echo "  Calling curl with \"$SERVER:3000/request/$SHORTORG/status?requestid=$REQUESTID&amp;token=$TOKEN\""

STATUS=`curl -s "$SERVER:3000/request/$SHORTORG/status?requestid=$REQUESTID&amp;token=$TOKEN"`

while [ "x$STATUS" != "xdone" ] ; do
    echo "  STATUS is \"$STATUS\" - trying again in $SLEEP seconds ($TOTALWAIT so far)..."
    sleep $SLEEP
    TOTALWAIT=$(($TOTALWAIT + $SLEEP))
    STATUS=`curl -s "$SERVER:3000/request/$SHORTORG/status?requestid=$REQUESTID&amp;token=$TOKEN"`
#    ((RETRY+=1))
#    if [ "x$RETRY" = "x$MAXTRIES" ] ; then
#        echo "  Quitting due to limit of $MAXTRIES retries..."
#        exit 1
#    fi
done

echo $REQUESTID &gt; .REQUESTID

echo "  SUMMARY: Transcription completed successfully at $(date)"
```

Use callbacks to receive results

In REST applications, a callback is the address and (optionally) the method name and parameters of a web application that can receive data via HTTP. Callbacks are typically used to enable another application to receive and directly interact with the transcripts produced by V-Spark.

This tutorial shows setting up a simple callback server, submitting audio for transcription with `/transcribe`, and examining the results, along with suggestions for troubleshooting common problems when setting up and using a callback server.

Setting up a sample callback server

To follow this example, you must have a callback server running on a given host and port. If you do not already have a callback server, the easiest way to simulate a callback server is to use the `netcat` application to listen on a specified port and display the information that it receives. The `netcat` application is a computer networking utility for reading from and writing to network connections using the TCP or UDP protocols. The name of its executable version is typically `nc` or `nc.exe`, depending on the operating system that you are using. The `netcat` utility is included in most Linux distributions and is freely available for most modern operating systems.

The sample output shown later in this section was produced by `netcat` that was started using the following Linux command-line command:

```
while true ; do nc -l 5555 -k ; done
```

The trivial callback server that we are implementing here with the `netcat` command continually executes the `netcat` command, listening on port 5555 (the `-l` option), and keeps its connections alive by listening for another connection after its current connection is completed (the `-k` option). It does not have to return any values to applications that talk to that port because V-Spark does not expect a return code and therefore does not retry until a return code is received.

```
while true ; do nc -l 5555 -k ; done
```

The trivial callback server that we are implementing here with the `netcat` command continually executes the `netcat` command, listening

on port 5555 (the `-l` option), and keeps its connections alive by listening for another connection after its current connection is completed (the `-k` option). It does not have to generically return any values to applications that talk to that callback server because V-Spark either expects an HTTP return code of success or only retries a limited number of times (100, by default) before canceling the callback.

```
while true ; do echo -e "HTTP/1.1 200 OK\r\n" | nc -l 5555; done
```

V-Spark retries submissions to a callback until the callback returns success (HTTP code 200). For this reason, the trivial callback server that we're implementing here with the `netcat` command, which is listening on port 5555, echoes that success code to the `netcat` command inside a loop, so that it always sends that success code with anything that is calling it.

Receiving transcription results

A successful call to the V-Spark API returns the transcript in the default (JSON) format or whatever other format you specified with the `output` stream tag in your call to V-Spark's `/transcribe` API.

A callback server is generally used to collect output and forward it to some other application, process the transcript itself, or perhaps simply to preserve the output for subsequent use. Using the sample callback server that was introduced earlier, transcripts are written to the standard output for the shell in which you executed the `netcat` command.

```
curl -F "file=@sample7.wav;type=audio/wav" -F output=text \
     -F token=0123456789ABCDEFGHIJ0123456789ABS \
     -F callback=http://196.168.6.64:5555 \
     https://vcloud.vocitec.com/transcribe
```

This call would return a message like the following:

```
{"requestid":"3b1c30e0-e62e-4da0-9487-c2f2c76310c7"}
```

The following example shows the output that the `netcat` callback server displays after a call to that server when text output was requested:

```
POST / HTTP/1.1
Host: 73.174.3.131:5555
Accept-Encoding: identity
Content-Length: 701
Content-Type: text/plain

Thank you for calling Center point energy technical support. I understand you need to report a gas leak and I have
your name please
my name is Joe and I thank you Mr. Know what is your address or account number
my address and then one Martin Houston, Texas is there. Anyone inside the house? I know everyone is out of the
house. I notice the strange smell when I got home and I called you I am sending and gas technician to your home to
fix the problem. Could you give me a good number to reach you at
you can call 28195345 zero's.
Thank you, please be safe and wait for the technician to arrive call us back if anything changes.
Thank you, bye. Good bye and thank you for calling Center point energy.
POST / HTTP/1.1
Content-Type: multipart/form-data;boundary=x7UiTsbnoupKk6ndj9DxpOvyt6NtDFjnn3K0OC
User-Agent: Java/1.7.0_161
Host: 73.174.3.131:5555
Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2
Connection: keep-alive
Content-Length: 1100

--x7UiTsbnoupKk6ndj9DxpOvyt6NtDFjnn3K0OC
Content-Disposition: form-data; name="requestid"
Content-Type: text/plain;charset=UTF-8
Content-Length: 36

3b1c30e0-e62e-4da0-9487-c2f2c76310c7
```

```
--x7UiTsbnoupKk6ndj9DxpOvyt6NtDFjnn3K0OC
Content-Disposition: form-data; name="file"; filename="sample7.txt"
Content-Type: text/plain
Content-Length: 702

Thank you for calling Center point energy technical support. I understand you need to report a gas leak and I have
your name please
my name is Joe and I thank you Mr. Know what is your address or account number
my address and then one Martin Houston, Texas is there. Anyone inside the house? I know everyone is out of the
house. I notice the strange smell when I got home and I called you I am sending and gas technician to your home to
fix the problem. Could you give me a good number to reach you at
you can call 28195345 zero's.
Thank you, please be safe and wait for the technician to arrive call us back if anything changes.
Thank you, bye. Good bye and thank you for calling Center point energy.

--x7UiTsbnoupKk6ndj9DxpOvyt6NtDFjnn3K0OC--
```

As discussed earlier, the goal of a callback server is to enable another application to receive and directly interact with the transcriptions produced by V-Spark. However, a simple callback server such as the one used in this section can also be convenient when testing the effects of trying different options with calls to V-Spark's `/transcribe` method.

For example, the following is the callback server's output after transcribing the same sample audio file using the `output=text` option and adding the `diarize=true` option:

```
POST / HTTP/1.1
Content-Type: multipart/form-data;boundary=PX0PI61Stzs4xNw-G7SyqnxXPcstL3PEmbF
User-Agent: Java/1.7.0_161
Host: 73.174.3.131:5555
Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2
```

```
Connection: keep-alive
Content-Length: 1096

--PX0PI61Stzs4xNw-G7SyqnxXPcstL3PEmbF
Content-Disposition: form-data; name="requestid"
Content-Type: text/plain;charset=UTF-8
Content-Length: 36

9ca9674c-65b7-46a7-aa06-1ceaeae9d8af
--PX0PI61Stzs4xNw-G7SyqnxXPcstL3PEmbF
Content-Disposition: form-data; name="file"; filename="sample7.txt"
Content-Type: text/plain
Content-Length: 707

Thank you for calling Center point energy to technical support.
I understand you need to report a gas leak and I have your name please.
My name is John.
Thank you, Mr. Darrow.
What is your address or account number?
My address is and and Walmart in Houston Texas.
Is there anyone inside the house?
Know everyone is out of the house so I noticed the strange now when I
got home and I called you.
Hi, I'm sending a gas technician to your home to fix the problem.
Could you give me a good number to reach you at.
You can call 281-953-4507.
Thank you, please be safe and wait for the technician to arrive call us back if anything changes.
Thank you bye.
Good, bye and thank you for calling Center point energy.
```

```
--PX0PI61Stzs4xNw-G7SyqnxXPcstL3PEmbF--
```

In the example output, you can see that enabling V-Spark's `diarize` option has improved the identification of the different speakers on the call, even though the audio file is still in mono.

Troubleshooting callbacks

If files are being uploaded successfully to V-Spark, you received a success code (HTTP code 200) and a requestid in response to uploading to V-Spark. If your callback server is not receiving results, check the items in the following list:

- **Verify that external hosts can reach your callback server** - Receiving a success code and requestid in response to POSTing a request to V-Spark shows that the system that is POSTing the request can reach V-Spark. This does not mean that V-Spark can reach your callback host. This lack of reachability is usually due to firewall or network connectivity restrictions.

  Verifying connectivity can most easily be done using a simple callback server like the one in the previous section.

  To test connectivity between V-Spark and your callback server, log in on a host that is not on your local network and can be reached directly from the Internet. Once you are logged in there, attempt to reach the host on which your callback server is running. The following is a sample `curl` command that simply probes the URL at which a callback server is listening:

  ```
  curl -i http://host:5555
  ```

```

```

The `host` and `port` that you specify are the host and port on which your callback server is listening.

The -i option tells the `curl` command to display the HTTP header that it receives. For example, if you are using the sample callback server that was discussed earlier, you will receive a result that is something like the following:

```
HTTP/1.1 200 OK
```

> NOTE: If your network administration policies restrict inbound connectivity from external hosts, contact support@vocitec.com for the list of V-Spark IP addresses from which access needs to be allowed.

- **Identify problems in your callback server** - If you are able to reach the host and port on which your callback server is running from some other host on the Internet, connectivity is not the problem. Try the following steps to identify problems with your callback server:
  - **Verify that you can POST directly to your callback server** - use a command like the following to simulate the data that would be sent by V-Spark to your callback server:

    ```
    curl -F "file=@test.json;type=application/json" \
         -F requestid=700e7496-4fce-4963-aa7b-b3b26600f813 \
         https://HOST:PORT/endpoint
    ```

    This command provides the two fields of the multipart POST that your callback server needs to be able to handle. Ensure that your callback server correctly returns success (HTTP code 200) when these two fields are received.
  - **Verify correct error handling** - it is possible for V-Spark transcription to encounter an error. In such cases, an error message will be POSTed in an error field to your callback server. Your callback server must be able to handle receiving error messages from V-Spark. The following example command sends the error message `This is a sample error` to your callback server:

```
curl -F "error=This is a sample error" \
     -F requestid=700e7496-4fce-4963-aa7b-b3b26600f813 \
     http://HOST:PORT/endpoint
```

This sample command should trigger error handling in your callback server, such as logging a message.

If you still cannot identify or resolve the problem with your callback server, contact support@vocitec.com for assistance in diagnosing the problem that you are experiencing.

Testing callbacks

The following command calls the V-Spark API, specifies the address of the callback server, specifies that you want text format output, and identifies the audio file that you want to transcribe:

```
curl -F callback=http://www.example.com:5555 \
      -F token=123e4567e89b12d3a456426655440000 \
      -F output=text -F "file=@sample7.wav;type=audio/wav" \
      http://asr_server:17171/transcribe
curl -F callback=http://www.example.com:5555 \
      -F token=123e4567e89b12d3a456426655440000 \
      -F output=text -F "file=@sample7.wav;type=audio/wav" \
      http://example_host/transcribe
curl -F callback=http://www.example.com:5555 \
     -F token=123e4567e89b12d3a456426655440000 \
     -F output=text -F "file=@sample7.wav;type=audio/wav" \
     http://vcloud.vocitec.com/transcribe
```

This sample command sends a text transcript of the audio file **`sample7.wav`** to the callback server. The text that was transcribed via the cURL command that was shown previously is the following:

```
Thank you for calling Center point energy technical support. I understand you need to report a gas leak and I have
your name please
my name is Joe and I thank you Mr. Know what is your address or account number
my address and then one Martin Houston, Texas is there. Anyone inside the house? I know everyone is out of the
house. I notice the strange smell when I got home and I called you I am sending and gas technician to your home to
fix the problem. Could you give me a good number to reach you at
you can call 28195345 zero's.
Thank you, please be safe and wait for the technician to arrive call us back if anything changes.
Thank you, bye. Good bye and thank you for calling Center point energy.
POST / HTTP/1.1
Host: 73.174.3.131:5555
Accept-Encoding: identity
Content-Length: 701
Content-Type: text/plain

Thank you for calling Center point energy technical support. I understand you need to report a gas leak and I have
your name please
my name is Joe and I thank you Mr. Know what is your address or account number
my address and then one Martin Houston, Texas is there. Anyone inside the house? I know everyone is out of the
house. I notice the strange smell when I got home and I called you I am sending and gas technician to your home to
fix the problem. Could you give me a good number to reach you at
you can call 28195345 zero's.
Thank you, please be safe and wait for the technician to arrive call us back if anything changes.
Thank you, bye. Good bye and thank you for calling Center point energy.
```

Note that the sample audio file used in this example is a mono audio file, so the different portions of the audio in which voices are active (known as utterances) are separated by newlines.

Use `fields` to refine /search output

When specifying `output=summary` in a `/search` request, use the `fields` parameter to define the data fields to be included with or excluded from search results.

The following example GET request shows a `/search` call with multiple options specified for the `fields` parameter:

```
curl -s "http://example.company.com/search/ExampleCo/ExampleOrg/
ExampleFolder?token=1234567890&daterange=20210701-20210702&format=csv&output=summary&fields=filename,duration"
```

In the preceding example, the `output=summary&fields=filename,duration` parameters include only the `filename` and `duration` fields in search results.

The following example GET request shows how to exclude fields:

```
curl -s "http://example.company.com/search/ExampleCo/ExampleOrg/
ExampleFolder?token=1234567890&daterange=20210701-20210702&format=csv&output=summary&fields=all,-agentid"
```

In the preceding example, the `output=summary&fields=all,-agentid` parameters include `all` (both `voci` and `CLIENT-DATA`) fields except for `agentid` in search results.

When specifying values for the `fields` parameter using a POST request, the submitted JSON data must list fields as a JSON-formatted list, even if there is only one value, as in the following example POST request:

```
curl -H 'Content-type: application/json' -d '{"output":"summary","fields":["all","-datetime","-requestid","-
diarization"]}' -X POST 'http://example.company.com/search/ExampleCo/ExampleOrg/ExampleFolder?token=1234567890'
```

In the preceding example, `{"output":"summary","fields":["all","-datetime","-requestid","-diarization"]}` is a JSON-formatted set of parameters that includes `all` fields, except for the excluded fields `datetime`, `requestid`, and `diarization`. These parameters may also be submitted in a JSON file, as in the following example:

```
curl -H 'Content-type: application/json' -d @params.json -X POST 'http://example.company.com/search/ExampleCo/
ExampleOrg/ExampleFolder?token=1234567890'
```

In the preceding example, JSON-formatted parameters are replaced with a reference to the file containing them, which in this case is named `params.json`. Note that using a file with cURL requires inserting the `@` symbol before the filename.

Using cURL

Use cURL to test the V-Spark API from the command line. cURL is not required to use the V-Spark API, but it can be a helpful tool, and is freely available for most operating systems.

> IMPORTANT: When using cURL to make API calls, it is important to remember that because URLs typically include the `?` and `&` symbols to identify HTTP/HTTPS parameters, you must enclose the URL portion of your cURL command within quotation marks to prevent a Linux shell from intercepting and interpreting these characters.

Debugging cURL calls

These are some of the strategies Voci engineers use to debug cURL errors.

1. **Obtain verbose debugging information.** Append `--trace-asii $filename` to your cURL command to log every interaction between cURL and the host to the file `$filename`. The contents of the saved file may help identify the problem.
2. **Add a timeout to the command.** cURL requests don't timeout by default, and depending on network and host conditions, there may be a significant delay between your request and the host's response. To specify a timeout, append `--max-time $seconds` to your cURL command, where `$seconds` is the timeout threshold expressed in seconds.
3. **Add a conditional speed timeout.** To timeout the request when its data transfer speed is slower than a certain threshold `$speed` for a certain amount of time `$seconds`, append `--speed-limit $speed --speed-time $seconds` to your command. Daniel Stenberg, cURL's maintainer, provides several useful example commands using this tag on his blog.
4. **Limit output to the response's HTTP code.** Append `-LI -w '%{http_code}'` to your command to include only the response's HTTP code, then compare that code to your expected results.

Output readability

The JSON output that is produced by V-Spark APIs is not easily readable by humans. When using cURL to make API calls, the output of the cURL command can be piped to Python in order to print that output in a more human-legible format, as in the following example:

```
$cURL_command | python -m json.tool
```

The `json.tool` module that is provided by Python sorts keys in JSON output alphabetically. To reformat JSON output without reorganizing key values, consider using the Python command `jsonlint`. This command includes JSON reformatting along with JSON validation and other capabilities, and is provided as part of the `python-demjson-2.2.2-1.el7.noarch` package on CentOS 7 systems.

Common cURL flags

These cURL tags are used frequently in examples. Refer to the cURL documentation for a full list.

-d

Identifies the data that you are sending to the HTTP server. File names must be preceded by an `@` symbol. You can also use the `-` symbol after an `@` symbol to indicate that the data to send to the HTTP server will be coming from standard input on your system, such as when a cURL POST command uses a pipe to receive data from another application.

-H

Identifies the type of content that you are sending (`"Content-Type:application/json"`).

-I

Only returns the HTTP header, which is where the HTTP response code is located

> -L

> Tells cURL to follow the URL if it is marked as having been moved

> -o

> Tells cURL where to write the general output of the command. In this case, `/dev/null`, the Linux and macOS bit bucket is used. On Windows systems, you can write to a file named `nul`, for which Windows provides built-in device driver support.

> -s

> Causes cURL to run in silent mode. Ordinarily, cURL displays a progress meter as it executes.

> -w

> Specifies what cURL should write to standard output, and how to format that information. The string `'%{http_code}'` simply writes the contents of the variable `http_code`

> -X

> Identifies the request method to use when communicating with the target HTTP server. If no method is specified, the request defaults to GET.