

Using the V-Spark 3.4.3 API



Using the V-Spark 3.4.3 API

© Copyright 2019 Voci All Rights Reserved.

The information contained in this document is the proprietary and confidential information of Voci Technologies, Inc. incorporated. you may not disclose, provide or make available this document, or any information contained in this document, to any third party, without the prior written consent of Voci.

The information in this document is provided for use with V-Spark Voice Analytics. No license, express or implied, to any intellectual property associated with this document or such products is granted by this document.

All Voci Technologies, Inc. products described in this document, including V-Spark and others prefaced by Voci are owned by Voci (or those companies that have licensed technology to Voci) and are protected by patents, trade secrets, copyrights or other industrial property rights. The Voci products described in this document may still be in development. The final form of each product and release date thereof is at the sole and absolute discretion of Voci. Your purchase, license and/or use of Voci products shall be subject to Voci's then current sales terms and conditions.

Trademarks

The following terms used in this document are trademarks of Voci Technologies, Inc. in the United States and other countries:

- Voci
- V-Blaze
- V-Cloud
- V-Discovery
- V-Ferno
- V-Purify
- V-Spark
- Voci

Other third party disclaimers or notices may be set forth in Voci's online or printed documentation. All other product and service names, and trademarks not owned by Voci are the property of their respective owners.

Table of Contents

1. V-Spark API Overview	1
1.1. Overview of V-Spark Organization	1
1.2. V-Spark API Permission Requirements	2
1.3. Using cURL for REST API Testing	2
1.3.1. Obtaining and Using the cURL program	2
1.3.2. Tips for Debugging and Managing cURL Calls	3
1.4. Using Python with REST APIs	4
2. Pre-Requisites	5
3. Retrieving and Updating V-Spark Information	7
3.1. Retrieving and Updating V-Spark Installation Configuration	7
3.1.1. Reference for the /config API	7
3.1.1.1. Refining by Companies, Organizations, Folders, and Apps	9
3.1.2. Sample JSON Output from the /config API	9
3.1.2.1. Sample /config JSON Output for a Company	10
3.1.2.2. Sample /config/orgs JSON Output for an Organization	12
3.1.2.3. Sample /config/folders JSON Output for a Folder	13
3.1.2.4. Sample /config/apps JSON Output for an Application	16
3.1.2.5. Sample /config/users JSON Output for a User	18
3.1.2.6. Sample /config/system/readonly JSON Output for System Status	20
3.2. Permissions and Capabilities in the /config/users API	20
3.2.1. V-Spark Permissions and the /config/users API	21
3.2.2. Differences between GET and POST JSON for the /config/users API	23
3.3. Using the /config API with cURL	24
3.3.1. GET'ing Information Using cURL and the /config API	24
3.3.2. POST'ing Information Using cURL and the /config API	26
3.3.3. DELETE'ing Information Using cURL and the /config API	26
3.3.3.1. Getting DELETE Status Information	27
3.4. Using the /config API with Python	28
3.4.1. GET'ing Information Using Python and the /config API	29
3.4.2. Integrating Multiple GET Results Using Python	30
3.4.3. POST'ing Information Using Python and the /config API	32
3.4.4. DELETE'ing Information Using Python and the /config API	34
3.5. Listing Configuration Information	35
3.5.1. Reference for the /list API	35
3.5.2. Sample JSON Output from the /list API	36
3.5.2.1. Sample /list JSON Output for Companies	36
3.5.2.2. Sample /list/orgs JSON Output for a Company	36
3.5.2.3. Sample /list/folders JSON Output for a Company	37
3.5.2.4. Sample /list/apps JSON Output	38
3.5.2.5. Sample /list/users JSON Output for an Installation	38
3.5.3. Using the /list API with cURL	38
3.5.4. Using the /list API with Python	39
4. Submitting audio and metadata for processing	41
4.1. Reference for the transcribe API	42
4.1.1. Examples of calling the transcribe API	43
4.2. Using the transcribe API with AWS S3	43
5. Receiving transcripts and status information	45
5.1. Using Callbacks in V-Spark	45
5.1.1. Configuring Callbacks in V-Spark	46
5.1.2. Example Callback Server	51
5.1.2.1. Setting up a Sample Callback Server	52

5.1.2.2. Submitting a Sample File for Text Transcription	52
5.1.2.3. Receiving Transcription Results	52
5.1.2.4. Troubleshooting a Callback Server	53
5.2. Reference for the request API	54
5.2.1. Examples of calling the request API	56
6. Retrieving Log and Status Information	59
6.1. Retrieving Log Information	59
6.1.1. Reference for the /log API	59
6.1.2. Sample JSON Output for a folder from the /log API	60
6.1.3. Using the /log API with cURL	60
6.1.4. Using the /log API with python	61
6.2. Retrieving Status Information	63
6.2.1. Reference for the /status API	63
6.2.2. Sample JSON and CSV Output from the /status API	64
6.2.2.1. Sample /status JSON and CSV Output for a Company	64
6.2.2.2. Sample /status JSON and CSV Output for an Organization	65
6.2.2.3. Sample /status JSON and CSV Output for a Folder	65
6.2.3. Using the /status API with cURL	66
6.2.4. Using the /status API with Python	67
7. Searching V-Spark Data	71
7.1. Reference for the /search API	71
7.1.1. Output Type Options	72
7.1.2. Search Term Options	72
7.1.3. Output Format Options	74
7.1.4. Output Field Options	74
7.1.5. Output Sorting Options	75
7.2. Sample JSON Output for a query from the /search API	76
7.3. Using the /search API with cURL	76
7.4. Using the /search API with Python	77
7.4.1. Using the /search API via GET with Python	77
7.4.2. Using the /search API via POST with Python	79
8. Retrieving Folder and Application Statistics Information	83
8.1. Retrieving Folder Statistics	83
8.1.1. Reference for the /stats API	83
8.1.2. Sample JSON from the /stats API	84
8.1.3. Using the /stats API with cURL	84
8.1.4. Using the /stats API with Python	86
8.2. Retrieving Agent Application Statistics and Category Scores	88
8.2.1. Reference for the /appstats API	88
8.2.2. Sample JSON from the /appstats API	90
8.2.3. Using the /appstats API with cURL	90
8.2.4. Using the /appstats API with Python	92
9. Configuring V-Spark Applications	95
9.1. Reference for the /appedit API	95
9.2. Using the /appedit API with cURL	95
9.2.1. Creating and Populating an Application Using cURL	96
9.3. Using the /appedit API with Python	99
10. Retrieving System Information	101
10.1. Reference for the /sysinfo API	101
10.2. Sample JSON from the /sysinfo API	101
10.3. Using the /sysinfo API with cURL	105
A. Sample transcribe/request API Shell Script	107
B. Possible Error Codes from the V-Spark API	109
B.1. Possible Error Codes from the /transcribe API	109

B.2. Possible Error Codes from the /request API	110
B.3. Possible Error Codes from the /config/folders API	110
B.4. General Error Codes from the V-Spark APIs	110

List of Figures

1.1. Location of a Company Authorization Token	2
3.1. Sample Company output from the /config API	10
3.2. Creating a Company in the V-Spark GUI	11
3.3. Sample Company output from the /config/CO_SHORT API	12
3.4. Sample Organization output from the /config/orgs API	12
3.5. Creating an Organization in the V-Spark GUI	13
3.6. Sample Organization output from the /config/CO_SHORT/ORG_SHORT API	13
3.7. Sample Folder output from the /config/folders API	14
3.8. Creating a Folder in the V-Spark GUI	15
3.9. Sample Folder output from the /config/CO_SHORT/ORG_SHORT/FOLDERNAME API	16
3.10. Sample Application output from the /config/apps API	16
3.11. Creating an Application in the V-Spark GUI	17
3.12. Sample Application output from the /config/CO_SHORT/ORG_SHORT/apps/APPNAME API	18
3.13. Sample User Information from the /config/users API	18
3.14. Registering for a V-Spark Account	19
3.15. Sample User output from the /config/CO_SHORT/users API	20
3.16. Sample User JSON for a System Administrator	21
3.17. Sample User JSON with Company-Level Permissions	22
3.18. Sample User JSON with Organization-Level Permissions	22
3.19. Sample User JSON Including a Password Field	23
3.20. Sample User JSON for use with the /config/users API	25
3.21. Sample User JSON for use with the /config/CO_SHORT/users API	25
3.22. Folder Status When No Delete Operation is In Progress	28
3.23. Folder Status When a Delete Operation is In Progress	28
3.24. Sample Python code to query the /config API	30
3.25. Sample Python code to combine data read from the /config API	31
3.26. Sample Python code to write (POST) data with the /config API	33
3.27. Sample Python code to delete data using the /config API	34
3.28. Invoking the code in Figure 3.24 to use the /list API	40
4.1. Location of the V-Spark Organization Short Name	42
5.1. V-Spark Folder Settings	46
5.2. Editing V-Spark Folder Settings	47
5.3. Further Customization of V-Spark Folder Settings	48
5.4. Specifying custom metadata fields to include in output	49
5.5. Specifying ASR options	49
5.6. Configuring an HTTP Callback Server and related options	50
5.7. Configuring an SFTP Callback Server and related options	51
6.1. Sample Folder Log Output from the /log API	60
6.2. Sample Folder Log Output from the /log API	60
6.3. Sample Python Code to Search for Audio Processed on a Given Day	62
6.4. Previewing /status information in a spreadsheet	66
6.5. Sample Python Code to Retrieve /status Information	68
7.1. Sample Python Code to search for Audio using GET, Part 1	78
7.2. Sample Python Code to search for Audio using GET, Part 2	79
7.3. Sample Python Code to Search for Audio using POST, Part 1	80
7.4. Sample Python Code to Search for Audio using POST, Part 2	81
8.1. Sample Folder Statistics Output from the /stats API	84
8.2. Sample Python Code for Retrieving Folder Statistics from the /stats API	87
8.3. Sample Application Statistics and Category Score Output from the /appstats API	90
9.1. Sample JSON that defines an Application	97

9.2. Sample JSON that Associates an Application with a Folder	97
9.3. Sample JSON for an Application	98
9.4. Sample Python Application for POST'ing JSON to APIs	99
10.1. Sample Basic System Output from the /sysinfo API	101

Chapter 1. V-Spark API Overview

V-Spark is an all-inclusive speech analysis application that enables you to visualize audio using state-of-the-art speech recognition, transcription, and text analysis technologies. V-Spark automatically transcribes audio into searchable text, then organizes and archives the data, and finally provides an intuitive web interface through which you can examine and explore that data. The information is stored in a database where the audio can be searched and analyzed for compliance, customer insights, and agent performance. V-Spark has the most complete set of speech technologies in a single solution on the market today.

This guide explains how to use V-Spark's Representational State Transfer (REST) Application Programming Interface (API) to:

- automate the flow of audio and optional metadata into V-Spark
- automate the flow of fully annotated transcripts out of V-Spark
- examine or modify a V-Spark installation using APIs that enable you to:
 - retrieve, update, delete, or list information about companies, organization, folders, and applications
 - retrieve information about users
 - retrieve log information about folders
 - retrieve status information about companies and folders. This information can be retrieved in JSON and CSV format.
 - search the files in a folder or under an organization to identify and retrieve matching search results. This information can be retrieved in JSON and CSV format, and as a ZIP-format archive files that contains the results.

Note that the V-Spark API is identical for both on-premise and cloud-based deployments. The V-Spark REST API uses the HyperText Transfer Protocol (HTTP) for data transfers. Every Voci solution includes a REST API to make integration with our products quick and easy in any computer language. A basic level of programming skill is required to use this API effectively.

1.1. Overview of V-Spark Organization

Understanding how to use the V-Spark API requires that you understand how the different components that make up V-Spark are organized, which can be thought of as the V-Spark hierarchy:

1. **company** - the highest structural entity within the hierarchy of a V-Spark installation. Multiple companies can be defined within a V-Spark installation, but only the administrators of that installation or users with company-level administrative privileges can view and modify the data that is associated with a company. Each **user** in a V-Spark installation is associated with a single company.
2. **organizations** - a logical unit (group) within a company
3. **folders** - a repository for files and transcriptions that are associated with an organization. Multiple folders can be associated with a single organization.
4. **applications** - customized analytics tools that are associated with one or more folders
5. **users** - individual accounts that are defined within a company and can belong to one or more organizations

6. **system** - administrative configuration information that is associated with a single V-Spark installation.



Tip

For information about the users of a V-Spark installation and the roles and permissions that they have within a V-Spark installation, see the "*Companies, Organizations, and Accounts*" and "*User Account Types*" sections of the "*V-Spark 3.4.3 Management Guide*".

1.2. V-Spark API Permission Requirements

When calling any V-Spark REST API function, you must provide an authorization token that shows that you have the right to perform the operation that you are requesting. V-Spark provides two different types of authorization tokens:

- **root token** - authorizes you to call any V-Spark API and perform any V-Spark API operation. This includes API functions that apply to your entire V-Spark installation and span multiple companies, such as `/config`, `/config/users`, `/config/orgs`, `/config/folders`, `/config/apps`, and `/config/system/readonly`. The root token also authorizes you to call any company-specific, organization-specific, folder-specific, or application-specific API function. The root token for a V-Spark installation is found in the file `/opt/voci/state/vspark/apitoken` on the system on which V-Spark is installed. This token is therefore only available to users who can access the machine on which V-Spark is installed, and who have sufficient privileges to access the V-Spark installation.
- **company token** - authorizes you to call any V-Spark API within the scope of that company and perform any V-Spark API operation that is within the scope of that company. This includes general API calls that require one or more company-related arguments such `CO_SHORT` identifies the company that is associated with that token, `ORG_SHORT` is the name of an organization within that company, and `FOLDER` identifies a folder within an organization. Examples of such API calls are `/transcribe/ORG_SHORTFOLDER`, `/request/ORG_SHORT`, `/config/CO_SHORT`, `/config/CO_SHORT/ORG_SHORT`, and so on. A company's authorization token is found on the V-Spark **Settings** page in the **Company** section, as shown in [Figure 1.1, "Location of a Company Authorization Token"](#).

Company	Short Name	Usage (hours)	Data Retention (days)	Auth Token	Cloud Token	Registration	Created
<input checked="" type="checkbox"/> Doc Test Co	DocTestCo	161.6 of unlimited	180	Show...	None	Register...	2018-02-16
<input checked="" type="checkbox"/> New Company	NewCompany	0.0 of 125k	0%	180	Show...	None	Register...

Figure 1.1. Location of a Company Authorization Token

1.3. Using cURL for REST API Testing

1.3.1. Obtaining and Using the cURL program

The cURL utility makes it easy to test using the V-Spark API by providing a simple command-line mechanism for invoking API methods. cURL is not required to use the V-Spark API, but it can be a

helpful tool, even while developing a more programmatic implementation. The next few sections provide examples of using the V-Spark API from the command-line via the cURL command.

The cURL utility is freely available for operating systems including [Linux, Windows, and Mac OS](https://curl.haxx.se/download.html) [https://curl.haxx.se/download.html]. The command-line command for invoking the cURL utility is `curl` on Linux or Apple Mac OS systems. The executable command for invoking the cURL utility is `curl.exe` on Microsoft Windows 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.

In cURL examples and associated output that is provided in this document:

- Escaped newlines (that is, lines in cURL commands or example output that end with a backslash) are added for readability. They must not be present in cURL commands, and are also not present in the output of those commands.
- Example commands are shown in normal monospaced text. When example output from those commands is very short, it is combined into the same monospaced example block as the cURL call itself and is shown in **bold** monospaced text. When example output from examples is verbose, that output is provided as a separate monospace example block.

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

```
cURL command | python -m json.tool
```

One item to remember when using this model of pretty-printing JSON output is that Python's `json.tool` module sorts keys in JSON output alphabetically. If you want to pretty-print JSON output without reorganizing key values, you may want to use a Python command such as **jsonlint**, which includes pretty-printing along with other capabilities and is provided as part of the `python-demjson-1.6-1.el6.noarch` package on CentOS systems.

1.3.2. Tips for Debugging and Managing cURL Calls

If you are having problems troubleshooting a **curl** command, you can obtain verbose debugging information by appending the `--trace-ascii FILENAME` to your curl command. This will save a record of every interaction between the **curl** command and the host that you are trying to contact into the file `FILENAME`. Examining the content of this file can often help you identify the cause of a problem.

The cURL command does not have a built-in timeout. Waiting forever for a cURL command to return can be irritating because it depends on network congestion and the responsiveness of the host on which cURL is running, both of which are often out of your control. To specify a timeout for the entire span of your cURL command, you can add cURL's `--max-time SECONDS` option to your cURL commands. This option terminates your cURL command if it exceeds the number of seconds that you specified as an argument. The cURL command also provides options (`--speed-limit` and `--speed-time`) that enable you to set requirements for transfer speed and which provide other ways to automatically terminate poorly-performing cURL commands.

The `cURL` command also provides options that enable you to retrieve the status of a `cURL` call. You can use a `cURL` call like the following to retrieve the HTTP return code from an API call:

```
curl -s -LI URL -o /dev/null -w '%{http_code}'
```

The arguments to this `cURL` call are the following:

- `-L` - tells `cURL` to follow the URL if it is marked as having been moved
- `-I` - only returns the HTTP header, which is where the HTTP response code is located
- `-s` - causes `cURL` to run in silent mode. Ordinarily, `cURL` displays a progress meter as it executes.
- `-o` - tells `cURL` where to write the general output of the command. In this case, `/dev/null`, the Linux and Mac OS X bit bucket is used. On Windows systems, you can write to a file named `null`, for which Windows provides built-in device driver support.
- `-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`

1.4. Using Python with REST APIs

Python is a very popular programming language is freely available for operating systems including [Linux](#), [Windows](#), and [Mac OS X](#) [<https://www.python.org>], and is often included as part of a Linux distribution.

As a standard and popular programming language, many books and online articles are available to help you use Python,

One common problem related to using open source programming languages and public-contributed libraries is support across different operating system or language versions. For example, if you are running the sample Python application in this document on a CentOS 6.x system and are using that distribution's standard Python installation, you may see deprecation warnings for Python functions that are changing for Python 2.7 and later. To suppress these warnings, you can add the following code to the beginning of the Python file:

```
import warnings

with warnings.catch_warnings():
    warnings.simplefilter("ignore", category=DeprecationWarning)
    import cryptography
```

If you see deprecation warnings for modules other than `cryptography`, you can add import statements for those to suppress the warnings when loading them.

Chapter 2. Pre-Requisites

Before you can begin interacting with V-Spark by using all functions within the API, you must create an **Organization**, **Company**, and **Folder** using the V-Spark GUI and API. A **Folder** has associated configuration information such as the language model that is used during transcription and also provides the names of metadata fields that are available for filtering. See the "*V-Spark 3.4.3 Quickstart Guide*" and "*V-Spark 3.4.3 Management Guide*" or the online documentation for more information about using V-Spark. PDF versions of these documents are available under the **Help** pull-down immediately after logging into V-Spark.

This document focuses on providing information about the V-Spark API. Using the upload and transcribe (`transcribe`) and download (`request`) APIs requires less configuration than using V-Spark, because V-Spark handles the transcription of uploaded audio, and is therefore where most of the transcription options are specified.

Chapter 3. Retrieving and Updating V-Spark Information

V-Spark provides several REST APIs that enable you to read and, when possible, update V-Spark configuration, search, status, and log information about a V-Spark installation. All of these APIs return and use information in JSON format. These APIs are the following:

- `/config` - enables you to retrieve, update, and delete configuration information about the companies, organizations, folders, and applications in a V-Spark installation. You can also retrieve information about the users in a V-Spark installation, but you can only update or modify some user information using the `/config` API.
- `/list` - enables you to list high-level configuration information about a V-Spark installation. The `/list` API provides the names of items within the configuration of a V-Spark installation and does not include the detailed configuration information that the `/config` API's GET verb provides. The `/list` API does not support any REST verb other than GET.
- `/log` - enables you to retrieve information about the status of processing folders (and therefore the files in those folders) in a V-Spark installation. The `/log` API is a read-only API. You can only retrieve log information using this API - you cannot update or delete log information using the `/log` API.
- `/search` - enables you to search a V-Spark installation for matching text, tags, and so on
- `/status` - enables you to retrieve status information about processing the files in a folder in a V-Spark installation. The `/status` API is a read-only API. You can only retrieve status information using this API - you cannot update or delete status information using the `/status` API.

When calling any of these APIs, you must provide an *authorization token* which proves that you are authorized to perform that operation. For information about the authorization tokens that you can provide for use with the V-Spark API, see [Section 1.2, “V-Spark API Permission Requirements”](#).

As REST APIs, these APIs can be used in any programming language or with any application that supports both REST calls and which provides or can invoke a JSON parser that enables you to work with the output of these calls.

The next few sections explain the capabilities of the `/config` and `/list` APIs. Information about the `/log` and `/status` APIs is provided in [Chapter 6, *Retrieving Log and Status Information*](#), which contains both reference information and examples of using those APIs from both the command-line (via cURL) and in Python application. Information about the `/search` API is provided in [Chapter 7, *Searching V-Spark Data*](#).

3.1. Retrieving and Updating V-Spark Installation Configuration

Voci provides the `/config` API to enable reading, writing, and deleting V-Spark configuration information within an application or script. The `/config` API is the only API in this section that enables you to write V-Spark information programmatically.

3.1.1. Reference for the `/config` API

The `/config` API enables you to read (GET), write (POST), and delete (DELETE) V-Spark configuration information.

Synopsis

```

/config
/config/users
/config/orgs
/config/folders
/config/apps
/config/system/readonly
/config/CO_SHORT
/config/CO_SHORT/orgs
/config/CO_SHORT/folders
/config/CO_SHORT/apps
/config/CO_SHORT/users
/config/CO_SHORT/users/USERNAME
/config/CO_SHORT/ORG_SHORT
/config/CO_SHORT/ORG_SHORT/folders
/config/CO_SHORT/ORG_SHORT/apps
/config/CO_SHORT/ORG_SHORT/FOLDERNAME
/config/CO_SHORT/ORG_SHORT/apps/APPNAME

```

As shown in the synopsis, calls to the V-Spark `/config` API can include optional entries that enable you to specify the short name of a company (`CO_SHORT`), the short name of an organization within a company (`ORG_SHORT`), the name of a specific folder (`FOLDERNAME`), the name of a specific user (`USERNAME`), or the name of a specific application (`APPNAME`) to refine and limit the amount of information that you are retrieving, updating, or adding. See [Section 3.1.1.1, “Refining by Companies, Organizations, Folders, and Apps”](#) for more information.

Calls to the V-Spark API without specific company, organization, folder, user, or application values return all information for the API that you are calling. In other words, a call to `/config/DocTestCo` only returns information about a company whose short name is "DocTestCo". A call to `/config` returns information about all of the companies that have been defined in the V-Spark installation that you are querying.

Description

The preceding calls can GET, POST, or DELETE the following types of information about a V-Spark installation:

- `/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` - returns information about all users that have been defined in a V-Spark installation, the status of their user accounts, the authorization method used for login, and the permissions that they have in V-Spark and within each company.



Note

Users are associated with companies, so DELETE'ing companies in a V-Spark installation also deletes any users that were associated with those companies.

- `/config/system/readonly` - returns information about whether a V-Spark installation is or is not running in readonly mode. You cannot call the `/config/system` API.

Options

In addition to being able to optionally specify the short name of a company or organization to refine the data that is being returned, calls to the `/config` method take the following parameters to further control their behavior:

- `token` (*mandatory*): the root or a company-specific token for the V-Spark installation about which you want to retrieve information. You cannot successfully call any function in the `/config` API without an authentication token. The root token for a V-Spark installation can always be used. A company-specific token can be used for any calls within the scope of that company. The root token for a V-Spark installation is located in the file `/opt/voci/state/vspark/apitoken`. The company-specific token is part of the configuration information for the associated company. For an example of programmatically retrieving company tokens and using them with organizations and folders, see [Section 3.4.2, “Integrating Multiple GET Results Using Python”](#).
- `tree=true/false` (*optional*): only used with the DELETE verb, specifying `true` enables you to delete non-empty companies and organizations in a V-Spark installation. By default, you cannot delete non-empty elements in a V-Spark installation.
- `multi=true/false` (*optional*): only used with DELETE verb, specifying `true` enables you to delete multiple items at once. By default, you can only delete a single item at one time in a V-Spark installation.

As REST API functions, `/config` API functions return both an HTTP message and a return code. The success and failure messages associated with various calls to the `/config` API are listed in the sections that describe those calls.

3.1.1.1. Refining by Companies, Organizations, Folders, and Apps

As shown in the synopsis section of the [Section 3.1.1, “Reference for the /config API”](#), API calls can optionally specify the short name of a company (`CO_SHORT`), the short name of an organization (`ORG_SHORT`), the name of a folder (`FOLDERNAME`), or the name of an application (`APPNAME`) to limit the data that is returned by an API call to only that which is specific to those names:

- If the short name of a company (`CO_SHORT`) is not specified, the JSON that is returned contains information about all relevant data for any company in a V-Spark installation.
- If the short name of a company is specified, only information that is related to that company is returned.
- If the short name of a company, that of an organization (`ORG_SHORT`), the name of a folder (`FOLDERNAME`) or application (`APPNAME`) is specified, only information that is associated with that entry ids returned. If any of these entries are invalid, the API call returns an error message. You cannot request or try to update information about an entry if any higher-level value is incorrect.

3.1.2. Sample JSON Output from the /config API

The `/config` API returns a JSON representation of the V-Spark installation data that is being requested. The following sections describe the JSON output for each aspect of a V-Spark installation:

- [Section 3.1.2.1, “Sample /config JSON Output for a Company”](#)
- [Section 3.1.2.2, “Sample /config/orgs JSON Output for an Organization”](#)

- [Section 3.1.2.3, “Sample /config/folders JSON Output for a Folder ”](#)
- [Section 3.1.2.4, “Sample /config/apps JSON Output for an Application ”](#)
- [Section 3.1.2.5, “Sample /config/users JSON Output for a User ”](#)
- [Section 3.1.2.6, “Sample /config/system/readonly JSON Output for System Status ”](#)

3.1.2.1. Sample /config JSON Output for a Company

Figure 3.1, “Sample Company output from the /config API” shows sample output for a single company from the /config API.

```

"DocTestCo": {
  "allowedmodels": [
    "devel:callcenter",
    "devel:spal_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"
},...

```

Figure 3.1. Sample Company output from the /config API

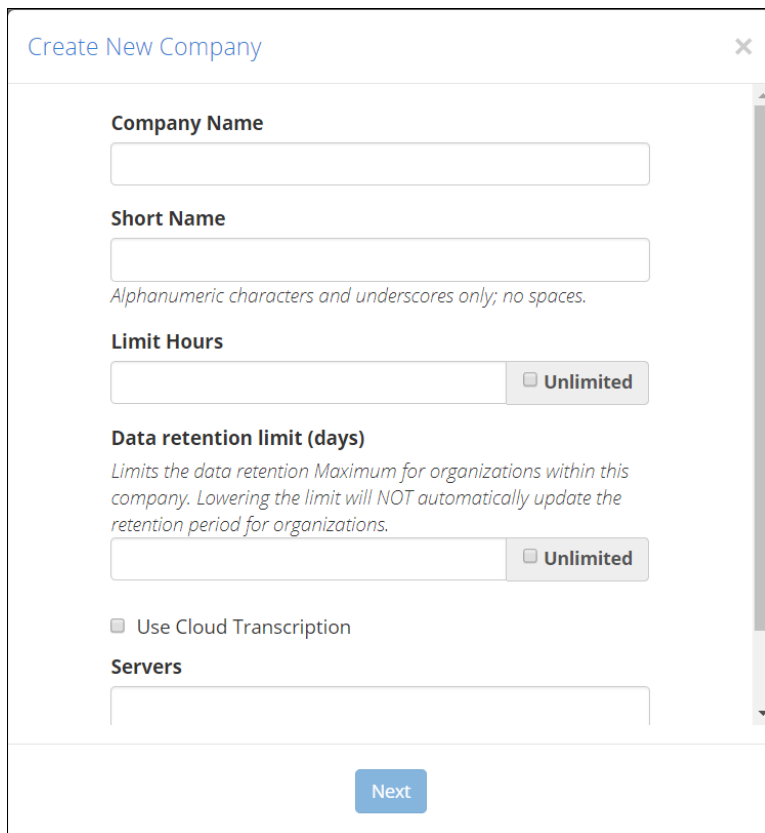


Note

Figure 3.1 and the other 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.

The fields in per-company JSON output contain but are not limited to the information that you provide when creating a company using the V-Spark graphical user interface. [Figure 3.2, “Creating a Company in the V-Spark GUI”](#) shows the first dialog used when creating a company in the V-Spark user interface.

When providing JSON to the API in order to create a company, the `created` field need not be specified because it defaults to the current time. Other fields depend on the type of system on which you are creating the company and whether field are automatically populated by V-Spark when they are being created. For example, the `uuid` field does not need to be specified while creating a company on a server system because V-Spark automatically supplies a value for it when it is not present. Similarly, the `cloudtoken` and `cloudmodels` can be (but do not need to be) specified when creating a folder to use cloud servers.



The screenshot shows a web form titled "Create New Company" with a close button (X) in the top right corner. The form contains the following sections:

- Company Name**: A text input field.
- Short Name**: A text input field with a note below it: "Alphanumeric characters and underscores only; no spaces."
- Limit Hours**: A text input field with a checkbox labeled "Unlimited" to its right.
- Data retention limit (days)**: A text input field with a note below it: "Limits the data retention Maximum for organizations within this company. Lowering the limit will NOT automatically update the retention period for organizations." and a checkbox labeled "Unlimited" to its right.
- Use Cloud Transcription**: A checkbox.
- Servers**: A text input field.

A blue "Next" button is located at the bottom center of the form.

Figure 3.2. Creating a Company in the V-Spark GUI

For detailed information about the information that is part of the definition of a company, see the section entitled "Create a Company" in the *V-Spark 3.4.3 Management Guide*.

This excerpt from the output of calling the `/config` API shown in [Figure 3.1](#), "Sample Company output from 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:

```

{
  "allowedmodels": [
    "devel:callcenter",
    "devel:spal_callcenter"
  ],
  "apptemplate": [
    "Agent Scorecard",
    "Call Categorization",
    "Call Drivers",
    "Customer Experience"
  ],
  "cloudmodels": [],
  "cloudtoken": "",
  "created": "2017-05-18",
  "limithours": -1,
  "name": "Doc Test Co",
  "retention": -1,
  "servers": [
    "asrsrvr1"
  ],
  "uuid": "077d93ffd9b902b2cb7c6a0c521fd42c"
}

```

Figure 3.3. Sample Company output from the /config/CO_SHORT API

Much like the root token is the key to retrieving information about any general aspect of a V-Spark installation, the authorization token for a company (contained in the `uuid` field of the information that can be retrieved about a company) is the key to directly querying data about any organization, folder, apps, or user that have been defined under that that company. See [Section 3.4.2, “Integrating Multiple GET Results Using Python”](#) for an example of using the `uuid` field from company information for companies stored on a V-Spark server to drill down into per-company information.

3.1.2.2. Sample /config/orgs JSON Output for an Organization

[Figure 3.4, “Sample Organization output from the /config/orgs API”](#) shows sample output for a single organization from the `/config/orgs` API.

```

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

```

Figure 3.4. Sample Organization output from the /config/orgs API

The fields in per-organization JSON output, as shown in [Figure 3.4, “Sample Organization output from the /config/orgs API”](#), include but are not limited to the information that you provide when creating an organization using the V-Spark graphical user interface. When providing JSON to the API in order to create an organization, the `created` field need not be specified because it defaults to the current time. [Figure 3.5, “Creating an Organization in the V-Spark GUI”](#) shows the first dialog used when creating an organization in the V-Spark user interface.

Create New Organization [X]

Organization Name

Company

Short Name

Can only contain alphanumeric characters and underscores; no spaces. Prefixed with the company's short name.

Retain data for # days
 Unlimited

Time Zone

Figure 3.5. Creating an Organization in the V-Spark GUI

For detailed information about the information that is part of the definition of an organization, see the section entitled "Create an Organization" in the *V-Spark 3.4.3 Management Guide*.

The JSON excerpt shown in [Figure 3.4, "Sample Organization output from the /config/orgs API"](#), from the output of calling the `/config/orgs` API, 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. This call would have returned the output shown in [Figure 3.6, "Sample Organization output from the /config/CO_SHORT/ORG_SHORT API"](#), which differs only 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.

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

Figure 3.6. Sample Organization output from the /config/CO_SHORT/ORG_SHORT API

3.1.2.3. Sample /config/folders JSON Output for a Folder

[Figure 3.7, "Sample Folder output from the /config/folders API"](#) shows sample output for a single folder from the information retrieved via the `/config/folders` API.

```
"DocTestCo": {
  "DocTestCo-DocTesting": {
    "Test01": {
      "apps": [],
      "asroptions": {
        "billing": "DocTestCo-DocTesting-Test01"
      },
      "audiotype": "Mono",
      "callback": {
        "aws_id": "123456789012345678901",
        "aws_secret": "123456789012345678901/12345678901234567890",
        "posturl": "S3:///joeuser/test",
        "sendaudio": "no",
        "sendtext": "no"
      },
      "created": "2017-05-18",
      "custom_meta": [],
      "mode": "active",
      "modelchan0": "devel:callcenter",
      "nspeakers": 1,
      "purifyaudio": true,
      "purifytext": true,
      "status": "OK",
      "servers": [
        "asrsrvr1"
      ]
    },...
  },...
},...
```

Figure 3.7. Sample Folder output from the /config/folders API

The fields in per-folder JSON output contain but are not limited to the information that you provide when creating a folder using the V-Spark graphical user interface. For example, when providing JSON to the API in order to create a folder, the `created` field need not be specified because it defaults to the current time. [Figure 3.8, “Creating a Folder in the V-Spark GUI”](#) shows the first dialog used when creating a folder in the V-Spark user interface.

The screenshot shows a 'Create New Folder' dialog box. At the top, it says 'Create New Folder' with a close button. Below that, the 'Organization' is set to 'Technologies / Demos'. The 'Folder Name' field contains 'TestFolder' and has a note below it: 'Can only contain A-Z 0-9 with no spaces'. The 'Servers' dropdown is set to 'All selected (2)'. There are two dropdown menus: '# of speakers' set to '2' and 'Audio Source' set to 'Stereo'. At the bottom, there are two checkboxes: 'Use Purify Text' and 'Use Purify Audio', both of which are unchecked. A blue 'Next' button is centered at the bottom of the dialog.

Figure 3.8. Creating a Folder in the V-Spark GUI

The `custom_meta` field in a JSON folder definition enables you to define custom metadata fields that are associated with that folder, just as you could have done in the GUI by checking the **Add/Remove Custom Metadata Fields** field during folder creation, and then populating the subsequent dialog with field names. When defining or modifying custom metadata fields in JSON, the contents of the `custom_meta` field are a comma-separated list of double-quoted field names.

For detailed information about the information that is part of the definition of a folder, see the section entitled "Creating a Folder" in the *V-Spark 3.4.3 Management Guide*.

The excerpt from the output of calling the `/config/folders` API, shown in [Figure 3.7, "Sample Folder output from the /config/folders API"](#), 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 that shown in [Figure 3.9, "Sample Folder output from the /config/CO_SHORT/ORG_SHORT/FOLDERNAME API"](#), which differs only from the output shown in [Figure 3.7](#) 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.

```

{
  "apps": [],
  "asroptions": {
    "billing": "DocTestCo-DocTesting-Test01"
  },
  "audiotype": "Mono",
  "callback": {
    "aws_id": "0SNASZ8CQFC6AF2NWHR2",
    "aws_secret": "Lc1dDilTsNAcnVePn8nS/VooRumYoGX6vn9SRdsd",
    "posturl": "S3://wvh/test",
    "sendaudio": "no",
    "sendtext": "no"
  },
  "created": "2017-05-18",
  "custom_meta": [],
  "mode": "active",
  "modelchan0": "devel:callcenter",
  "nspeakers": 1,
  "purifyaudio": true,
  "purifytext": true,
  "servers": [
    "asrsrvr1"
  ]
}

```

Figure 3.9. Sample Folder output from the /config/CO_SHORT/ORG_SHORT/FOLDERNAME API

The mode field in the JSON output for a Folder indicates whether processing of that folder is "active" or "paused". Use 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.

3.1.2.4. Sample /config/apps JSON Output for an Application

Figure 3.10, "Sample Application output from the /config/apps API" shows sample output for a single application from the /config/apps API for the applications that have been defined for a single organization.

```

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

```

Figure 3.10. Sample Application output from the /config/apps API

The fields in application-related JSON output contain but are not limited to the information that you provide when creating an application using the V-Spark graphical user interface. Figure 3.11, "Creating an Application in the V-Spark GUI" shows the first dialog used when creating an application in the V-Spark user interface.

Create New Application [X]

1 Organization
Technologies ▾ / Demos ▾

2 Application Name
testapp
Can only contain A-Z 0-9

3 Default Score Type
 Hit/Miss Coverage

4 Template Options
Use preset template ▾

5 Templates
Agent Scorecard ▾

Please be aware that applications with **custom metadata filters** may result in scores of 0 if linked folders do not share the custom metadata fields used.

6 Link to Folders
Demos ▾

Create

Figure 3.11. Creating an Application in the V-Spark GUI

For detailed information about the information that is part of the definition of an application, see the section entitled "Creating an Application" in the *V-Spark 3.4.3 Management Guide*. For information about retrieving and uploading applications using the API, see [Chapter 9, Configuring V-Spark Applications](#).



Note

The template option to **Copy from existing organization** is not supported in the API.

The excerpt from the output of calling the `/config/apps` API shown in [Figure 3.10, "Sample Application output from the /config/apps API"](#) is very similar to the output that you would have received had you requested 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 [Figure 3.12, "Sample Application output from the /config/CO_SHORT/ORG_SHORT/apps/APPNAME API"](#). 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" had been defined. (When supplied as part of a URL, the `%20` is required in order to URL-encode the space in the name.) 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.

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

Figure 3.12. Sample Application output from the /config/CO_SHORT/ORG_SHORT/apps/APPNAME API



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.

3.1.2.5. Sample /config/users JSON Output for a User



Important

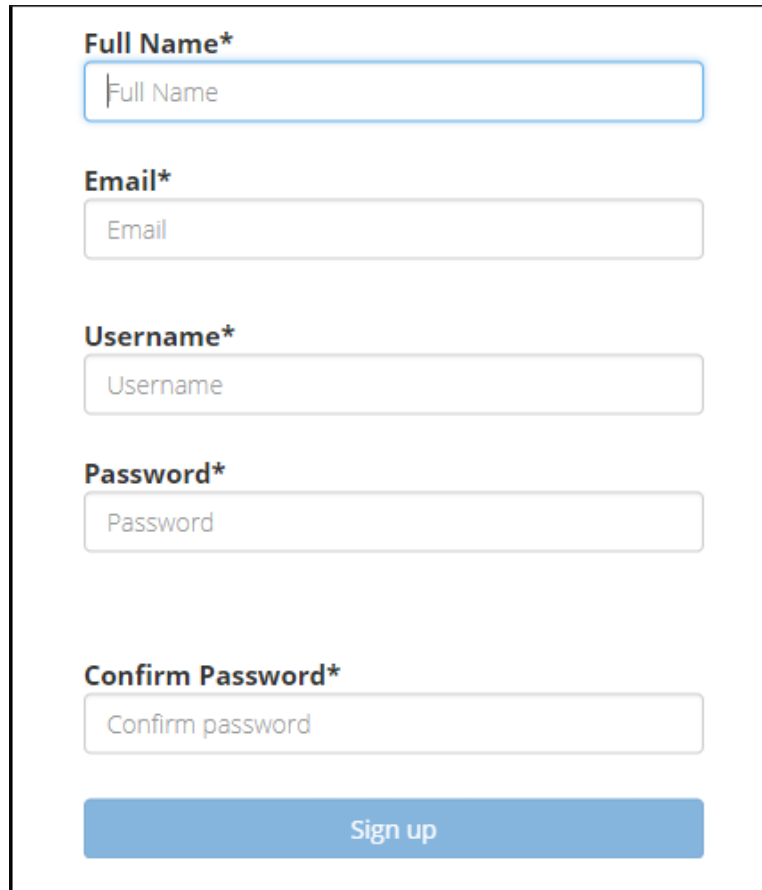
Users are defined within companies, and are therefore deleted when the company that they are associated with is deleted.

Figure 3.13, “Sample User Information from the /config/users API” shows an excerpt that contains sample output for a single user 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"
        ]
      }
    }
  }
}
}...
```

Figure 3.13. Sample User Information from the /config/users API

The fields in per-user JSON output contain but are not limited to the information that you provide when registering for a user account using the V-Spark graphical user interface (with the exception of the password information), and also when an administrator assigns you to a company, associates you with one or more organizations, and defines the permissions that you have within each organization. Figure 3.14, “Registering for a V-Spark Account” shows the first dialog used when registering for an account in the V-Spark user interface.



The image shows a registration form with the following fields and a button:

- Full Name***: A text input field with the placeholder text "Full Name".
- Email***: A text input field with the placeholder text "Email".
- Username***: A text input field with the placeholder text "Username".
- Password***: A text input field with the placeholder text "Password".
- Confirm Password***: A text input field with the placeholder text "Confirm password".
- Sign up**: A blue button with white text.

Figure 3.14. Registering for a V-Spark Account

For detailed information about registering for an account, see the section entitled "*Create a User Account*" in the *V-Spark 3.4.3 Management Guide*. For information about types of V-Spark accounts, see the section entitled "*User Account Types*" in the *V-Spark 3.4.3 Management Guide*.

The output shown in [Figure 3.13, "Sample User Information from the /config/users API"](#), an excerpt from the output of calling the `/config/users` API, is very similar to the first part of the output that you would have received had you called the `/config/Technologies/users` API on a V-Spark installation where the "Technologies" company (with the company short name, "Technologies") had been defined. The latter call would have returned JSON, as shown in [Figure 3.15, "Sample User output from the /config/CO_SHORT/users API"](#), 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.

```

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

```

Figure 3.15. Sample User output from the `/config/CO_SHORT/users` API

3.1.2.6. Sample `/config/system/readonly` JSON Output for System Status



Important

Because only the `readonly` API lives under `/config/system`, there is no more general `/config/system` API. This may be added in a future release if other system settings are added under `/config/system`. Attempting to GET, POST, or DELETE to the `/config/system` API directly will return HTTP error code 400.

The following is sample output retrieved using the `/config/system/readonly` API:

```

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

```

3.2. Permissions and Capabilities in the `/config/users` API

This section discusses permissions and capabilities that are specific to the `/config/users` API. For general information about the authorization tokens that are used by the V-Spark API, see [Section 1.2, “V-Spark API Permission Requirements”](#). For general information about the authorization tokens that are used by the V-Spark API, see [Section 1.2, “V-Spark API Permission Requirements”](#).

As an administrative API, the `/config/users` API provides capabilities that are specific to its use in the context of a V-Spark installation and which make the API easier to use in an enterprise environment of any size. The next few sections discuss aspects of the `/config/users` API that are particular to both that API and to its use as a programmatic mechanism for configuring V-Spark in a corporate environment.

3.2.1. V-Spark Permissions and the /config/users API

The `/config/users` API enables you to set the read/write permissions (referred to as View and Create/Edit permissions, respectively, within the V-Spark GUI) for each user within three different scopes:

- **System admin** - a system administrator role that gives a user read/write permissions to any aspect of a V-Spark installation that can be configured within the V-Spark GUI. This enables them to create, delete, and modify V-Spark users, companies, and organizations, as well as add system-wide announcements or put the system into read-only mode. [Figure 3.16, “Sample User JSON for a System Administrator”](#) shows sample JSON that describes a user with system administrator permissions. Note that system administration permissions are in a special section that is labeled `system`.

In [Figure 3.16](#), you'll note that there is no View (read) permission in the `System admin` group. That is because the read permission is inherently available at the system level when a user already has the privilege to Create/Edit (write) to any part of the V-Spark configuration data.

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

Figure 3.16. Sample User JSON for a System Administrator

- **company-level permissions** - gives a user View and Create/Edit permissions within the specified company. Write permission enables the user to create and add users to that company, and set the permissions of those users. Write permissions at the company level also grant the ability to create and edit organizations, folders, and applications within the company. Read permission enables the user to view dashboards and transcripts for any existing or newly created organization within the specified company. [Figure 3.17, “Sample User JSON with Company-Level Permissions”](#) shows the JSON for a user with company-level permissions for the company `DocTestCo`.

```

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

```

Figure 3.17. Sample User JSON with Company-Level Permissions

- **organization-level permissions** - gives a user View and Create/Edit permissions within the specified organization. Write permission enables the user to create and modify folders and applications that are associated with that organization. Read permission enables the user to view dashboards and transcripts for that organization. [Figure 3.18, “Sample User JSON with Organization-Level Permissions”](#) shows the JSON for a user with organization-level permissions for the organization DocTestCo-DocTesting.

```

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

```

Figure 3.18. Sample User JSON with Organization-Level Permissions

V-Spark provides a sophisticated and easy-to-use API for creating companies, organizations, and users. The GUI also makes it very easy to set and modify user permissions. See the "V-Spark 3.4.3 Management Guide" for detailed information about using the GUI.



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. You will only be able to see changes that have been made at a level that is equal to or lower than your current authorization level.

3.2.2. Differences between GET and POST JSON for the /config/users API

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 shown in [Figure 3.13, “Sample User Information from the /config/users API”](#):

- `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. [Figure 3.19, “Sample User JSON Including a Password Field”](#) shows sample JSON for a user who has system administration permissions for their home company (DocTestCo, in this case) and has a default password of `changeme`. See [Section 3.2.1, “V-Spark Permissions and the /config/users API”](#) for a discussion of user permissions and roles in the API.

```
{
  "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"
        ]
      }
    }
  }
},...
```

Figure 3.19. Sample User JSON Including a Password Field

- `method` - enables you to specify the authorization method that will be used to verify the user's identity when they log in. This name must have one of the following values:
 - `standard` - for internal V-Spark authorization
 - `ldap` - for external authorization through a Lightweight Directory Access Protocol server such as Microsoft Active Directory.

This option is only useful when you are creating a new account. Once the 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 system. This username may be a simple username, an email address, or a "user principle name" (UPN), depending on the service.

The additional name/value pair(s) discussed in the previous list can also be used in JSON input that is provided to calls to the `/config/CO-SHORT/users` API. The primary difference between the JSON that is provided in calls to that API and to the `/config/users` API is the JSON that is provided in calls to the `/config/users` API must specify the company under which each user is to be created.

3.3. Using the `/config` API with cURL

The next few topics discuss how to retrieve configuration information for V-Spark using the `/config` API's GET method, how to create or update V-Spark configuration information using the `/config` API's POST method, and how to delete V-Spark configuration information using the `/config` API's DELETE method.

If you are unfamiliar with the cURL command, see [Section 1.3, “Using cURL for REST API Testing”](#) for a short introduction and an explanation of how cURL examples are displayed. See [Section 1.3.2, “Tips for Debugging and Managing cURL Calls”](#) for suggestions about how to debug and manage cURL calls.

3.3.1. GET'ing Information Using cURL and the `/config` API

This section discusses how to use the `/config` API to retrieve configuration information from a specified V-Spark installation. A sample cURL command to retrieve information about all of the organizations in a V-Spark installation is the following:

```
curl -s $PROTOCOL://$HOST:$PORT/config/orgs?token=TOKEN
```

The variables in this command are the following:

- *PROTOCOL* - the protocol that your V-Spark installation uses to communicate over the network, one of `http` or `https`. V-Spark installation use the `http` protocol by default.
- *HOST* - the host on which your V-Spark installation is running, specified by host name or IPv4 IP address.
- *PORT* - the network port that the V-Spark installation is listening on. The default is port **3000**, which must still be specified in V-Spark REST API calls.
- *TOKEN* - an authorization token that enables calls to the `/config` API to access all data in a V-Spark installation. If you are using cURL to use the API to retrieve or modify information that is associated with a specific company, you can provide that company's authorization token to authorize your access. If you are requesting higher-level information, you can always use the V-Spark installation's root token to obtain the information that you are requesting. The root token for a default V-Spark installation is located in the file `/opt/voci/state/vspark/apitoken`. Finding a company's authorization token is shown in [Figure 1.1, “Location of a Company Authorization Token”](#).

Note that no REST verb (GET, POST, DELETE) has been specified in the preceding cURL command. That is because the cURL command defaults to performing a GET operation when no REST verb is specified.

As an example, the cURL command to use the HTTP protocol to retrieve configuration information for the organizations that have been defined in a V-Spark installation on the host 192.168.6.46, which is listening on port 3000 (the default port) is the following:

```
curl -s http://192.168.6.64:3000/config/orgs?token=123456789012345678901234567890123
```

The previous curl commands retrieved information about all of the organizations within an entire V-Spark installation, and therefore required providing the root token for that installation. To retrieve information about organizations within a specific company, you can use that company's token, so an example would be the following:


```
curl -s http://192.168.6.64:3000/config/CO_SHORT/orgs?token=company-token
```

See [Section 1.2, “V-Spark API Permission Requirements”](#) for information about retrieving the type of token that you want to use.

When calling the config API and providing entries that you want to create or modify in JSON format, you must make sure that you are calling the API with JSON that corresponds to the URL that you are specifying. For example, the `/config/users` API can also be called as `/config/CO_SHORT/users` to get information about the users within a specific company. When calling the `/config/users` API to create or modify user information, the user information must be top-level JSON information about the company with which you want to associate the user, as shown in [Figure 3.20, “Sample User JSON for use with the /config/users API”](#).

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

Figure 3.20. Sample User JSON for use with the /config/users API

When calling the `/config/CO_SHORT/users` API to create or modify user information, the user information must be top-level JSON information about the user, and does not need to nest the user information within the company information, because you are specifying the company that the user is associated with as part of the URL. [Figure 3.21, “Sample User JSON for use with the /config/CO_SHORT/users API”](#) shows the same information about a user that was shown in [Figure 3.20, “Sample User JSON for use with the /config/users API”](#), except that the JSON in [Figure 3.21](#) does not need to nest the user information within a company identifier.

```
{
  "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"
        ]
      }
    }
  }
}
```

Figure 3.21. Sample User JSON for use with the /config/CO_SHORT/users API

If you deliver JSON at the wrong level to any V-Spark API, you will receive an error code (400). Ensuring that you have sent the level of JSON that corresponds to the API URL call that you are making is the first thing that you should check when an API call is failing and you are sending JSON that you know to be valid.

3.3.2. POST'ing Information Using cURL and the /config API

Use the `/config` API to update the configuration of a V-Spark installation.



Note

The `/config/system` API does not accept POST requests. At this time, the `/config/system` API only returns system read only settings and read only settings must be updated directly using the `/config/system/readonly` API.

The cURL commands to POST data to the V-Spark API are slightly more complex than commands to GET or DELETE portions of a V-Spark installation. As an example, a sample cURL command to use the JSON file `config.json` to make sure that the companies that it describes are defined in the V-Spark installation on the host 192.168.6.64 is the following:

```
curl -s -X POST -H "Content-Type:application/json" \
"http://192.168.6.64:3000/config?token=TOKEN" --data @config.json
```

When using cURL and a command like this one to POST data to a host, the information about the protocol, host, and port is required, as is the `token` that ensures that you have rights to access the V-Spark installation.

You must also use the following cURL options:

- `-X` - identifies the request method to use (POST) when communicating with the target HTTP server
- `-H` - identifies the type of content that you are sending (`"Content-Type:application/json"`).
- `-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).

The cURL command's `-s` command-line argument is optional, causing the cURL command to run in silent mode, where it does not display progress information or error messages.

Data that is written to a V-Spark installation is additive - if some of the objects that are described by portions of the data that you are writing already exist, they will be updated (if necessary) to reflect their descriptions in your JSON data. Only objects that do not exist will be created. Objects that already exist but are not described in your JSON input will be preserved in their current configuration.

3.3.3. DELETE'ing Information Using cURL and the /config API

Use the `/config` API to delete information about companies, organizations, folders, applications, and users.

The cURL commands to DELETE V-Spark configuration data using the V-Spark API enable you to delete all, or specified, companies, organizations, folders, and applications. These commands do not require JSON data as an input, but simply require that you identify the object or objects that you want to delete. The `/config` API's DELETE methods provide two options that enable you to refine the scope of what is being deleted:

1. `multi=true/false` - The `multi` parameter defines whether multiple objects at the same level can be deleted by an operation. For example, if you want to delete a single, specified object, such as a single company, organization, folder, or application, you do not need to pass an option as a parameter to the URL because `multi=false` is the default value.
2. `tree=true/false` - The `tree` parameter defines whether non-empty objects can be deleted by an operation. For example, if you want to delete an object and everything that is logically stored in and "below" that object, you would pass the parameter `tree=true` as a parameter to the URL that your cURL command is calling.

For example, a cURL command to delete the single folder `Test01` and everything below it is the following:

```
curl -s -X DELETE 192.168.6.64:3000/config/DocTestCo/DocTestCo/Test01\
?token=123456789012345678901234567890123
```

In this example command, you are only deleting a single folder, so you do not need to specify the `multi=true` parameter. You do not need to specify the `tree=true` options because there is nothing that is hierarchically under a single folder.

As another example, a cURL command to delete all folders and everything below them is the following:

```
curl -s -X DELETE 192.168.6.64:3000/config/folders \
?token=123456789012345678901234567890123&multi=true
```

In this example, you do not need to specify the `tree=true` parameter because nothing is hierarchically located under a folder, but you do need to specify the `multi=true` parameter because you are deleting all folders that exist on the target host.

When using cURL to DELETE data from a host using the `/config` API, you must use the following cURL option:

- `-X` - identifies the request method to use (DELETE) when communicating with the target HTTP server

The cURL command's `-s` options is optional, causing the cURL command to run in silent mode, in which it does not display progress information or error messages.

3.3.3.1. Getting DELETE Status Information

The JSON returned by GET calls to the `/config` API for companies, organizations, and folders include a `status` field that provides high-level status information about the object that you are enquiring about. This is useful for long-running operations such as a DELETE operation. The value of the `status` field will be one of the following:

- `OK` - no operations are in progress regarding the queried object
- `deleting` - the queried object is in the process of being deleted
- `deleting (PERCENTAGE)` - 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

Figure 3.22, “Folder Status When No Delete Operation is In Progress” shows sample JSON that is returned by a call to the `/config/COSHORT/ORGSHORT/FOLDER` API when no delete operation is in progress.

```

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

```

Figure 3.22. Folder Status When No Delete Operation is In Progress

Figure 3.23, “Folder Status When a Delete Operation is In Progress” shows sample JSON that is returned by a call to the `/config/COSHORT/ORGSHORT/FOLDER` API when a delete operation is in progress.

```

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

```

Figure 3.23. Folder Status When a Delete Operation is In Progress

After using the `/config` API's DELETE method, it is a good idea to call the `/config` API's GET verb for the object that you requested deletion of. If the GET call returns JSON like that shown in Figure 3.23, the folder is in the process of being deleted. If the call returns "Folder not found: *FOLDERNAME*", you can be sure 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.

3.4. Using the `/config` API with Python

Section 3.3, “Using the `/config` API with `cURL`” and subsequent sections explained how to access the V-Spark API from the command-line by using the `cURL` command. This is a common way of integrating API calls into scripts and V-Spark maintenance processes, but calling the API directly from application code is equally common. The next few sections provide examples of using the GET, POST, and DELETE verbs with the `/config` API from within applications that are written using the Python programming language.

3.4.1. GET'ing Information Using Python and the /config API

Figure 3.24, “Sample Python code to query the /config API” shows example Python code that takes multiple parameters, and which enables you to retrieve information from a specified portion of the /config API. This code takes the following parameters, in order:

- *HOST* - the hostname or IP address of the host that is running the V-Spark installation which you want to query
- *ROOT_TOKEN* - the root token for the V-Spark installation that you are querying. The root token for a V-Spark installation is stored in the file `/opt/voci/state/vspark/apitoken`
- *API-TO-CALL* - You can use the example code to call any aspect of the /config API. For example, you could pass `/config`, `/config/orgs`, and so on, or explicitly request information about a specific company by passing `/config/DocTestCo` (if **DocTestCo** is a valid company on the V-Spark installation that you are querying).
- *HTTP_CODE_TARGET* - this is the HTTP return code that you expect to receive, and is only used in this example so that you can display its value and visually compare it against what you received from the actual API call to determine if your API call worked correctly. You could expand your code to react appropriately if you received an HTTP return code other than this one.
- *OUTPOST_FILE* - the file to which you want to write the JSON that you retrieved. The sample code also pretty-prints this JSON, to make it easier to read.

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

# Copyright 2017 Voci Technologies, Inc. All rights reserved.
# Contains confidential company information.
# 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 ): ❶
    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 ❷
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) ❸
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)) ❹

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

target.close()
```

Figure 3.24. Sample Python code to query the /config API

The sample code shown in [Figure 3.24, “Sample Python code to query the /config API”](#) is one of the applications that are used to help test the /config API that is discussed in this section. Therefore, its focus is on providing simple, linear code that shows how to get data from a specified host. The major steps in this sample Python application are the following:

- ❶ Check if the right number of command-line arguments have been provided, assign them to appropriate variables if so and identifying the expected arguments if not.
- ❷ Assemble the URL that you will use to call the specified API
- ❸ Make the get request to the V-Spark installation on the host that you specified on the command-line so that you can get the HTTP response and return code to display in an output message
- ❹ Issue the get call in another fashion so that you can retrieve the JSON that it returns and pretty-print that to a file.

3.4.2. Integrating Multiple GET Results Using Python

To read data from a V-Spark installation, you can use the root token, but that's analogous to running every Linux command as the superuser - it is cleaner to use the authorization token that is associated with each company.

[Figure 3.25, “Sample Python code to combine data read from the /config API”](#) provides example code that enables you to explore a V-Spark installation by using the top-level JSON configuration data for a V-

Spark installation along with the JSON that describes all of the folders in the V-Spark installation. You could use the sample code shown in [Figure 3.24, “Sample Python code to query the /config API”](#) to extract the JSON information that you need by executing the following two commands, assuming that the code shown in [Figure 3.24, “Sample Python code to query the /config API”](#) has been saved to the file `config-get-info.py`.

Once you have retrieved the JSON for the companies and folders that have been defined in the V-Spark installation on the host `192.168.6.64`, you can combine these two aspects of JSON configuration information about your V-Spark installation to explore that installation. Sample Python code that enables you to do this is shown in [Figure 3.25, “Sample Python code to combine data read from the /config API”](#). This example merits a walkthrough of the code.

```
#!/usr/bin/env python
#
# Copyright 2017 Voci Technologies, Inc. All rights reserved.
# Contains confidential company information.
# Unsupported example code - Not for production use.
#
# Application that reads company and folder information about a
# product installation on a specified host, then uses the company's
# short name to link the two. The application then 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): ❶
    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): ❷
    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): ❸
    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(): ❹
        print comp+" (Token: "+tokens[comp]+")"
        for org, org_data in comp_data.iteritems(): ❺
            print "\t", org
            for folder, folder_data in org_data.iteritems(): ❻
                print "\t\t", folder

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

Figure 3.25. Sample Python code to combine data read from the /config API

The sample code shown in [Figure 3.25, “Sample Python code to combine data read from the /config API”](#) does the following:

- ❶ The `main` function provides a traditional main routine that shows the order in which functions are called in the application

- ② Uses the `/config` API to retrieve the top-level configuration information from the host that was specified on the command-line, and returns a dictionary that contains only the company names and their associated authorization tokens (stored in the `uuid` field of the per-company information).
- ③ Uses the `/config/folders` API to retrieve the folder-level configuration information from the host that was specified on the command-line
- ④ Initiates the primary loop for the application, which is controlled by the companies that were found in the information that was retrieved from the host specified on the command-line. Each company has an associated authorization token (originally stored in the `uuid` name/value pair), which is the other field for each company entry in the dictionary that was constructed in the `gettokens()` function. The short name for each company is the data item in the company JSON that provides the linkage between the data from the company and folder sources. This loop prints out the name of each company that was found on the remote V-Spark installation and its associated authorization token.
- ⑤ Initiates a second loop for the application, which is controlled by the organizations that were retrieved in the company information which was retrieved from the host specified on the command-line. This loop prints the name of each organization that was found under the current company.
- ⑥ Initiates the final internal loop that iterates through all of the folders that are associated with each organization that was retrieved in the company information which was retrieved from the host specified on the command-line. This loop prints the name of each folder found under the current organization.

After the final `print` entry in the code shown in [Figure 3.25, “Sample Python code to combine data read from the `/config` API”](#), you could expand this sample application for testing purposes by adding other API calls at this point that would require company, authorization token, organization, and folder information.

The sample application shown in [Figure 3.25, “Sample Python code to combine data read from the `/config` API”](#) is provided as an example that shows how you can programmatically integrate the JSON output that you receive regarding different aspects of a V-Spark installation. As noted in the comments at the beginning of the source code, this code is only provided as an example.

3.4.3. POST'ing Information Using Python and the `/config` API

Use the `/config` API to update the configuration of a V-Spark installation.



Note

The `/config/system` API does not accept POST requests. At this time, the `/config/system` API only returns system read only settings and read only settings must be updated directly using the `/config/system/readonly` API.

This section provides sample Python code that shows how to put information from a sample JSON file to the V-Spark installation on the host that you provide as a command-line argument. You can prepare this JSON file manually, or you can use Python code like that shown in [Figure 3.24, “Sample Python code to query the `/config` API”](#) to retrieve V-Spark configuration from a V-Spark installation on another host.


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

# Copyright 2017 Voci Technologies, Inc. All rights reserved.
# Contains confidential company information.
# Unsupported example code - Not for production use.

import sys
import json
import requests

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

if ( len(sys.argv) != 6 ): ❶
    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 ❷
url = "%s%s:%s%s&token=%s" % (PROTOCOL, HOST, PORT, API_TO_CALL, ROOT_TOKEN)

print "Checking " + API_TO_CALL + ", POST'ing input from " + INPOST_JSON_FILE
print " Whole URL: "+url

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

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

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

Figure 3.26. Sample Python code to write (POST) data with the /config API

The sample code shown in [Figure 3.26, “Sample Python code to write \(POST\) data with the /config API”](#) is one of the applications that are used to help test the /config API that is discussed in this section. Therefore, its focus is on providing simple, linear code that shows how to put data to a specified host. The major steps in this sample Python application are the following:

- ❶ Check if the right number of command-line arguments have been provided, assign them to appropriate variables if so and identifying the expected arguments if not.
- ❷ Assemble the URL that you will use to call the specified API
- ❸ Open the JSON file that was specified on the command-line (and which contains that data that you want to POST to the specified host). Read that JSON into a JSON object.
- ❹ Call the specified URL, passing the JSON object as a parameter. Next, print a summary that identifies the API that the program called, the HTTP message and return code that the call returned, and prints the expected return code that you provided as a parameter.

Some examples of calling this Python script from the command-line are the following:

- `config-put-tests.py 192.168.6.64 123456789012345678901234567890123 /config/orgs 200 \
config-orgs.json`

Enables you to write the organization-level JSON configuration information stored in the file `config-org.json` to the V-Spark installation of the host `192.168.6.64` using the root token `123456789012345678901234567890123`, and also says that you expect that command to return success (HTTP error code 200).

- `config-put-tests.py 192.168.6.64 123456789012345678901234567890123 /config 200 \
config.json`

Enables you to write the company-level (top-level) JSON configuration information stored in the file `config.json` to the V-Spark installation of the host `192.168.6.64` using the root token

123456789012345678901234567890123, and also says that you expect that command to return success's (HTTP error code 200).

3.4.4. DELETE'ing Information Using Python and the /config API

Use the /config API to delete information about companies, organizations, folders, applications, and users.

This section provides sample Python code that shows how to delete all configuration information at a certain level of the product or a specific company, organization, folder, or application by specifying the full hierarchy of that object as part of the API that you specify on the command-line for this application. You do not need to provide a JSON file as one of the arguments to the delete operation - only the full path to the object that you want to delete is required.

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

# Copyright 2017 Voci Technologies, Inc. All rights reserved.
# Contains confidential company information.
# Unsupported example code - Not for production use.

import sys
import requests

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

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

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

print "Deleting based on " + API_TO_CALL
print " URL is ", url

response = requests.delete(url) ❸
print ' SUMMARY (DELETE): ' + API_TO_CALL, ': HTTP message: ', response.reason, ' HTTP return code: ',
    str(response.status_code), ' expected ' + TARGET_HTTP_CODE
```

Figure 3.27. Sample Python code to delete data using the /config API

The sample code shown in [Figure 3.27, “Sample Python code to delete data using the /config API”](#) is one of the applications that are used to help test the /config API that is discussed in this section. Therefore, its focus is on providing simple, linear code that shows how to delete data to a specified host. The major steps in this sample Python application are the following:

- ❶ Check if the right number of command-line arguments have been provided, assign them to appropriate variables if so and identifying the expected arguments if not.
- ❷ Assemble the URL that you will use to call the specified API. Remember that you have to pass the `&multi=true` parameter if you are trying to delete an object that contains multiple other objects, the `&tree=true` option if you are trying to delete an object that has one or more descendants, or both parameters if both of these are true.
- ❸ Call the specified URL, and print a summary that identifies the API that the program called, the HTTP message and return code that the call returned, and prints the expected return code that you provided as a parameter.

Some examples of calling this Python script from the command-line are the following:

- ```
config-delete-tests.py 192.168.6.64 123456789012345678901234567890123 /config/orgs 200 \
 "&multi=true&tree=true"
```

Enables you to delete the organizations from the V-Spark installation of the host 192.168.6.64 using the root token 123456789012345678901234567890123, and also says that you expect that command to return success (HTTP error code 200). (The extra parameters are passed correctly to the Python script, which just appends them to the `?token` parameter.)

- ```
config-put-tests.py 192.168.6.64 123456789012345678901234567890123 \
  /config/DocTestCo/DocTestCo-DocTesting/Test01 200 ""
```

Enables you to delete the folder `Test01` from the `docTestCo-DocTesting` organization under the `DocTestCo` from the V-Spark installation of the host 192.168.6.64 using the root token 123456789012345678901234567890123. The command also says that you expect that command to return success (HTTP error code 200).

3.5. Listing Configuration Information

The `/list` API provides high-level configuration information about a V-Spark installation. The `/list` API provides the names of items within the configuration of a V-Spark installation and does not include the detailed configuration information that the `/config` API's `GET` verb provides. The `/list` API does not support any REST verb other than `GET`.

3.5.1. Reference for the `/list` API

The `/list` API enables you to read (`GET`) the names used in the configuration of a V-Spark installation.

Synopsis

```
/list
/list/users
/list/orgs
/list/folders
/list/apps
/list/CO_SHORT/users
/list/CO_SHORT/orgs || /list/CO_SHORT
/list/CO_SHORT/folders
/list/CO_SHORT/apps
/list/CO_SHORT/ORG_SHORT/folders || /list/CO_SHORT/ORG_SHORT
/list/CO_SHORT/ORG_SHORT/apps
```

As shown in the synopsis, calls to the V-Spark `/list` API can include optional entries that enable you to specify the short name of a company (`CO_SHORT`) and the short name of an organization within a company (`ORG_SHORT`) to limit the amount of information that you are retrieving, updating, or adding. See [Section 3.1.1.1, “Refining by Companies, Organizations, Folders, and Apps”](#) for more information.

Calls to the V-Spark API without specific company or organization values return all of the name information for the part of the API that you are calling. In other words, a call to `/list/DocTestCo` only returns information about the organizations under the company whose short name is "DocTestCo". A call to `/list` returns information about all of the companies that have been defined in the V-Spark installation that you are querying.

Description

The preceding calls can `GET` the following types of name-level information about a V-Spark installation:

- `/list` - returns information about companies in a V-Spark installation
- `/list/orgs` - returns information about the organizations that have been defined under companies in a V-Spark installation
- `/list/folders` - returns information about the folders that have been defined under organizations in a V-Spark installation
- `/list/apps` - returns information about any apps that have been defined in a V-Spark installation
- `/list/users` - returns information about any users that have been defined in a V-Spark installation

See [Section 1.1, “Overview of V-Spark Organization”](#) for information about how a V-Spark installation is hierarchically organized.

Options

(None)

3.5.2. Sample JSON Output from the `/list` API

The `/list` API returns a high-level JSON representation of the V-Spark installation that is being queried. The following sections describe the JSON output from the `/list` API for each aspect of a V-Spark installation:

- [Section 3.5.2.1, “Sample `/list` JSON Output for Companies”](#)
- [Section 3.5.2.2, “Sample `/list/orgs` JSON Output for a Company”](#)
- [Section 3.5.2.3, “Sample `/list/folders` JSON Output for a Company”](#)
- [Section 3.5.2.4, “Sample `/list/apps` JSON Output”](#)
- [Section 3.5.2.5, “Sample `/list/users` JSON Output for an Installation”](#)

3.5.2.1. Sample `/list` JSON Output for Companies

The following is an example of output produced by the `/list` API for the companies in a V-Spark installation:

```
[
  "Testing",
  "DocTestCo",
  "WebAPITest",
  "Limitedhours",
  "CNCO",
  "JWebAPITest"
]
```

The `/list` API enables code to quickly extract a high-level view of the companies in a V-Spark installation, but does not itself provide enough information for you to drill down into that installation.

3.5.2.2. Sample `/list/orgs` JSON Output for a Company

The following is sample output for a single company from the `/list/orgs` API:

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

In the same way that you can use the `/config/CO_SHORT/orgs` API to retrieve information about the organizations within a company in a V-Spark installation, you can use the `/list/CO_SHORT/orgs` API to retrieve the names of the organizations within a company, as shown in the following example, which was produced by calling the `/list/DocTestCo/orgs` URL, and lists the organizations that have been defined within a sample company known as DocTestCo:

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



Tip

This example was produced by passing `/list/DocTestCo` as the *API-TO-CALL* parameter to the sample code shown in [Figure 3.24, “Sample Python code to query the /config API”](#). Though the discussion of that example in [Section 3.4.1, “GET’ing Information Using Python and the /config API”](#) was used to show calling the `/config` API, it can just as easily be used to call the `/list` API.



Important

The `/list/CO_SHORT` and `/list/CO_SHORT/orgs` API calls produce identical output. This is by design, because the only editable V-Spark items that are directly located under a specific company are the organizations that have been defined within that company.

3.5.2.3. Sample /list/folders JSON Output for a Company

The following is sample output for a single folder in a sample V-Spark installation, produced as part of a call to the `/list/folders` API:

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

In the same way that you can use the `/config/CO_SHORT/ORG_SHORT/folders` API to retrieve detailed information about the folders that have been defined within an organization, you can use the `/list/CO_SHORT/ORG_SHORT/folders` API to retrieve the names of such folders, as shown in the following example, which was produced by calling the `/list/DocTestCo/DocTestCo-DocTesting/folders` URL, and lists the folders that have been defined for the DocTestCo-DocTesting organization within a sample company known as DocTestCo:

```
[
  "Test01"
]
```



Note

In this case, only one folder has been defined within the DocTestCo-DocTesting organization. If multiple folders had been defined within that organization, all of their names would be displayed by the output of this command.



Tip

This example was produced by passing `/list/DocTestCo/DocTestCo-DocTesting` as the *API-TO-CALL* parameter to the sample code shown in [Figure 3.24](#),

“[Sample Python code to query the /config API](#)”. Though the discussion of that example in [Section 3.4.1](#), “[GETting Information Using Python and the /config API](#)” was used to show calling portions of the `/config` API, it can just as easily be used to call portions of the `/list` API.

3.5.2.4. Sample /list/apps JSON Output

The following is sample output for the applications that are associated with companies from the output of the `/list/apps` API for a V-Spark installation:

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

In the same way that you can use the `/config/CO_SHORT/apps` API to retrieve information about the applications that have been defined within a company in a V-Spark installation, you can use the `/list/CO_SHORT/apps` API to retrieve the names of the applications within a company, as shown in the following example, which was produced by calling the `/list/DocTestCo/apps` URL, and lists the applications that have been defined within a sample company known as DocTestCo:

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

3.5.2.5. Sample /list/users JSON Output for an Installation

The following is sample output for the users in a V-Spark installation from the `/list/users` API:

```
"DocTestCo": [
  "joe.user",
  "bill.vonhagen"
],
```

To extract this same information in a company-specific way, you could call the `/list/CO_SHORT/users` API to extract information about the users that have been defined within the `CO_SHORT` company. For example, calling the `/list/DocTestCo/users` API directly in the same sample V-Spark installation would produce the following output:

```
[
  "joe.user",
  "bill.vonhagen"
]
```

3.5.3. Using the /list API with cURL

The cURL utility makes it easy to test using the V-Spark API by providing a command-line mechanism for invoking APIs such as the `/list` API. The next few sections provide examples of using the GET verbs with the `/list` API from the command-line via the cURL command.

The cURL utility is freely available for operating systems including [Linux](#), [Windows](#), and [Mac OS X](#) [<https://curl.haxx.se/download.html>]. The command-line command for invoking the cURL utility is

`curl` on Linux or Apple Mac OS X systems. The executable command for invoking the `cURL` utility is `curl.exe` on Microsoft Windows systems.



Note

Escaped newlines (that is, lines in `cURL` commands or example output that end with a backslash) are added for readability. They must not be present in `cURL` commands, and are also not present in the output of those commands.

Example commands are shown in normal monospaced text. Example output from each command is shown in **bold** monospaced text.



Tip

See [Section 1.3.2, “Tips for Debugging and Managing `cURL` Calls”](#) for suggestions about how to debug and manage `cURL` calls.

This section discusses how to use the `/list` API to retrieve high-level configuration information from a specified V-Spark installation. A sample `cURL` command to retrieve information about all of the organizations in a V-Spark installation is the following:

```
curl -s $PROTOCOL://$HOST:$PORT/list/orgs?token=TOKEN
```

The variables in this command are the following:

- *PROTOCOL* - the protocol that your V-Spark installation uses to communicate over the network, one of `http` or `https`. V-Spark installation use the `http` protocol by default.
- *HOST* - the host on which your V-Spark installation is running, specified by host name or IPv4 IP address.
- *PORT* - the network port that the V-Spark installation is listening on. The default is port **3000**, which must still be specified in V-Spark REST API calls.
- *TOKEN* - an authorization token that enables calls to the `/list` API to access all data in a V-Spark installation. If you are using `cURL` to access the API for information within a specific company, you can provide that company's authorization token to get the information that you need. If you are requesting higher-level information, you can always use the V-Spark installation's root token to obtain the information that you are requesting. The root token for a default V-Spark installation is located in the file `/opt/voci/state/vspark/apitoken`.

Note that no REST verb has been specified in the preceding `cURL` command. That is because the `cURL` command defaults to performing a GET operation when no REST verb is specified, and the `/list` API only supports the GET verb.

As an example, the `cURL` command to use the HTTP protocol to retrieve configuration information for the organizations that have been defined in a V-Spark installation on the host 192.168.6.64, which is listening on port 3000 (the default port) is the following:

```
curl -s http://192.168.6.64:3000/list/orgs?token=123456789012345678901234567890123
```

If you are using the root token to enable access to the V-Spark configuration information, it is a 33-character string that is passed as an option using the `token` parameter.

3.5.4. Using the `/list` API with Python

The sample test script that was provided in [Figure 3.24, “Sample Python code to query the `/config` API”](#) was used to test the `/config` API. However, it can just as easily be used to query a V-Spark installation and

retrieve name-level information by specifying an API URL that begins with `/list` as the third parameter to the sample code, *API-TO-CALL*. [Figure 3.28](#), “Invoking the code in [Figure 3.24](#) to use the `/list` API” shows an example of invoking the code shown in [Figure 3.24](#) to use it to explore the `/list` API, assuming that you have saved the code in that figure to a file named `get-tests.py` in the current directory and made that file executable.

```
./get-tests.py HOST_NAME ROOT_TOKEN /list 200 config.json
./get-tests.py HOST_NAME ROOT_TOKEN /list/users 200 config-users.json
./get-tests.py HOST_NAME ROOT_TOKEN /list/orgs 200 config-orgs.json
./get-tests.py HOST_NAME ROOT_TOKEN /list/folders 200 config-folders.json
./get-tests.py HOST_NAME ROOT_TOKEN /list/apps 200 config-apps.json
```

Figure 3.28. Invoking the code in [Figure 3.24](#) to use the `/list` API

To execute this snippet as directed, you would have to replace *HOST_NAME* with the name of the host on which your V-Spark installation is running, and replace *ROOT_TOKEN* with the root token for your V-Spark installation.

The examples in this section (which call the code in [Figure 3.24](#)) are fairly simple because the `/list` API only supports the GET HTTP verb, and only returns name-level information.

Chapter 4. Submitting audio and metadata for processing

V-Spark uses the HTTP POST method to submit audio and optional metadata files for processing. Please refer to the "*V-Spark 3.4.3 Management Guide*" for a discussion of supported audio formats and metadata formatting details.

When using the `/transcribe` API to submit files for transcription, single audio files and JSON transcripts can be submitted individually. Files submitted individually will not be associated with each other. Multiple files must be encapsulated into a single "zip" file. These zip files can contain both audio data and metadata - any metadata that you provide must be formatted as described in the "Metadata Management" section of "*V-Spark 3.4.3 Management Guide*". Audio files and metadata files submitted as parts of a "zip" file will remain associated with each other as parts of a single submission.



Tip

If you want to directly submit audio files in various audio formats, V-Spark's GUI enables you to submit files to a specific folder by using the **Settings** menu's **Folders** command to display your folders and clicking the **Upload audio** button to the right of a folder's name. See the "*V-Spark 3.4.3 Management Guide*" for more information.



Note

The maximum size of a file that can be submitted for transcription using the `transcribe` API is 250 MB.

When using the `transcribe` API to submit zip files for transcription of the audio files that they contain, the POST must be encoded as a multipart/form-data request, with the zip file name provided in a file field and a V-Spark authorization token provided in the token field. You can use either the root token for your V-Spark installation or the token for the company that is associated with the organization and folder to which you are submitting your transcription request. See [Section 1.2, "V-Spark API Permission Requirements"](#) for information about locating these tokens and the rights that these tokens give you.

The following is an example of calling the **transcribe** API method using the cURL (crawl URL) command-line utility:

```
curl -F token=0123456789abcde0123456789abcde01 \
-F "file=@/path/to/audio_and_meta.zip:type=application/zip" \
-X POST https://hostname:3000/transcribe/org_shortcode/folder_name
```

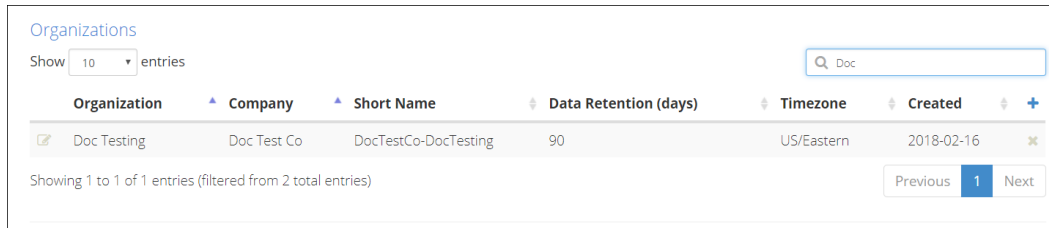
The cURL utility is freely available for operating systems including [Linux](#), [Windows](#), and [Mac OS X](#) [<https://curl.haxx.se/download.html>].



Note

Items shown as *replaceable* in the sample cURL command are example settings only and must be replaced with real values that are appropriate for your environment.

In the example command, note that `org_shortcode` refers to the Short Name assigned to the target Organization, which can be found on the V-Spark Settings page in the Organization section of V-Spark - see [Figure 4.1, "Location of the V-Spark Organization Short Name"](#). The `folder` refers to the folder for the organization into which you want to upload the audio that is contained in the zip file that you are uploading.



The screenshot shows a web interface for 'Organizations'. At the top, there is a search bar with 'Doc' entered and a dropdown menu set to '10 entries'. Below the search bar is a table with columns: Organization, Company, Short Name, Data Retention (days), Timezone, and Created. A single row is visible with the following data: Organization: Doc Testing, Company: Doc Test Co, Short Name: DocTestCo-DocTesting, Data Retention (days): 90, Timezone: US/Eastern, Created: 2018-02-16. At the bottom, it says 'Showing 1 to 1 of 1 entries (filtered from 2 total entries)' and has 'Previous', '1', and 'Next' navigation buttons.

Organization	Company	Short Name	Data Retention (days)	Timezone	Created
Doc Testing	Doc Test Co	DocTestCo-DocTesting	90	US/Eastern	2018-02-16

Figure 4.1. Location of the V-Spark Organization Short Name

The cURL command exits after transmission of the zip file to the V-Spark instance has completed. The POST returns a Universally Unique ID (UUID) that identifies the transcription request. All transcripts produced as a result of the request will include a "requestid" field with its value set to this UUID. The requestid enables you to correlate specific transcriptions with specific transcription requests.

Once the audio has been transcribed, the transcripts (along with optional metadata) are loaded into V-Spark. Please refer to the "*V-Spark 3.4.3 Review and Analysis Guide*" for details regarding browsing, searching, and analyzing your calls and metadata within V-Spark. All parameters that control transcription options are specified in the V-Spark **Folder** definition. These include language models used to decode each audio channel, number of speakers, number of audio channels (i.e. mono or stereo), etc. It is therefore unnecessary to provide these parameters when POSTing zip files to V-Spark. Please refer to the "*V-Spark 3.4.3 Management Guide*" for more details.

4.1. Reference for the transcribe API

The transcribe API enables you to submit files for transcription.

Synopsis



Note

The transcribe API can not be called as a single URL from the command-line due to its combination of resource and query parameters. See [Section 4.1.1, “Examples of calling the transcribe API”](#) for examples of using the **curl** command to test this API from the command-line by using special **curl** options to pass form and multi-part data.

Variables used in a call to the transcribe API are the following:

- **SERVER**: the name or IP address of the computer system on which V-Spark is installed
- **ORG_SHORT**: the short name of the organization that you are using. Finding that information is shown in [Figure 4.1, “Location of the V-Spark Organization Short Name”](#).
- **FOLDER**: the name of the V-Spark Folder to which you want to upload the zip file containing your audio file(s).
- **FILE**: the name of the individual file or zip file containing the multiple files that you want to transcribe.
- **AUTH_TOKEN**: the V-Spark authorization token that you are using to establish permission to retrieve information. The token that is used to authorize transcription requests can either be the root token for your V-Spark installation, or the company authorization token. Locating an authorization token for the company associated with the folder that you are uploading to is shown in [Figure 1.1, “Location of a Company Authorization Token”](#).

Description

The `transcribe` method takes the following parameters to provide authentication information and to identify the zip file that you are attempting to upload:

- `token`: the authorization token that you are using to establish permission to upload a zip file
- `file`: the name of the zip file containing the audio information that you want to be transcribed and analyzed

Upon success, the `transcribe` method returns a Universally Unique ID (UUID) that identifies the transcription request. All transcripts produced as a result of the request will include a "requestid" field with its value set to this UUID.

4.1.1. Examples of calling the transcribe API

The following example shows calling the `transcribe` method using the `cURL` (crawl URL) command-line command. Sample output is also provided, but depends on the host on which the API is running, the organization and folder that you are uploading to, and the authorization token that you are using. The `cURL` utility is freely available for operating systems including [Linux, Windows, and Mac OS X](https://curl.haxx.se/download.html) [https://curl.haxx.se/download.html].



Note

Escaped newlines (that is, lines in the `cURL` command or the example output that end with a backslash) are added for readability. They must not be present in `cURL` commands, and are also not present in the output of those commands.

Example commands are shown in normal monospaced text. Example output from each command is shown in **bold** monospaced text.

The following is a `cURL` example that shows calling the `transcribe` API:

```
curl -F token=0123456789abcde0123456789abcde01 \  
-F 'file=../SAMPLES/CallTEST.zip?type=application/zip' \  
-X POST http://example.company.com:3000/transcribe/Test-Testing/Test01  
  
9700fc31-f608-48b5-aaaf-bd264e811d9a
```

4.2. Using the transcribe API with AWS S3

The Amazon Web Service (AWS) Simple Storage Service (S3) is a common location for archiving audio files and metadata together in zip files. If you already have such files stored in S3, you can use the `transcribe` API's support for S3 to process them from that location, which can save upload time because V-Spark typically must upload your files to S3 for processing. In order to use the `transcribe` API with S3 from the command-line, you must pass your `AWS_ACCESS_KEY` (referred to in the `transcribe` API as your `aws_id`) and `AWS_SECRET_KEY` (referred to as your `aws_secret`) by using the `curl` command's support for filling in forms.

The following is the general format of a `cURL` command that calls the `transcribe` API to transcribe a file or directory that is stored in S3:

```
curl -F token=AUTH_TOKEN \  
-F aws_id=AWS_ACCESS_KEY \  
-F aws_secret=AWS_SECRET_KEY \  
-F s3key=s3://BUCKET/path/to/file/or/directory \  
-X POST http://SERVER:3000/transcribe/ORG_SHORT/FOLDER
```

The user-specific fields that you need to provide are the following:

- *AUTH_TOKEN*: the authorization token that you are using to retrieve information. Locating your authorization token is described in [Chapter 4, *Submitting audio and metadata for processing*](#). Locating an authorization token for the company associated with the folder that you are uploading to is shown in [Figure 1.1, “Location of a Company Authorization Token”](#).
- *AWS_ACCESS_KEY*: the Amazon key for the bucket in which the file that you want to transcribe is stored
- *AWS_SECRET_KEY*: the secret Amazon key for the bucket in which the file that you want to transcribe is stored
- *BUCKET*: the Amazon S3 bucket in which the file that you want to transcribe is stored
- *path/to/file/or/directory*: the path to the file that you want to process, a zip file that contains the audio file that you want to process (and an optional metadata file), or to a directory that contains a hierarchy of files that you want to process. If you specify a directory, all of the files that are located under that directory will be queued for transcription. Files that are submitted for processing but which are not in a format that is supported by V-Spark will not be processed and will be listed in the V-Spark folder's process log as being **UNSUPPORTED**.
- *SERVER*: the name or IP address of the computer system on which V-Spark is installed
- *ORG_SHORT*: the short name of the organization that you are using. Finding that information is shown in [Figure 4.1, “Location of the V-Spark Organization Short Name”](#).
- *FOLDER*: the V-Spark folder in which you want the transcript and audio output that is produced by V-Spark to be stored. organization that you are using.

The following is a specific example of calling the `transcribe` API to transcribe a zip file that is stored in S3:

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

9700fc31-f608-48b5-aaaf-bd264e811d9a
```

This example transcribes the audio in the zip file named `documentation-TEST.zip` in the bucket `example.company.com` and puts the results of that transcription in the `Test01` folder of the organization `Test-Testing`. As with other calls to the `transcribe` API, it returns the request ID for your transcription request, which you can subsequently use with the `request` API, as discussed in [Section 5.2, “Reference for the request API”](#).

By default, any zip file in S3 that you have identified for transcription using the `transcribe` API remains stored on S3 after its contents have been transcribed. Keeping such files in S3 after their content has been transcribed may not be necessary, so the `transcribe` API includes a `"delete=true"` option that you can pass to delete a file after its content has been transcribed. In an application, you would pass this as an additional parameter to the `transcribe` API call. In a curl command, you would add the `-F delete=true` option to your command-line.

Chapter 5. Receiving transcripts and status information

You can receive JSON and text transcripts and audio files directly within V-Spark or by using the `request` API call to retrieve them manually. V-Spark uses the HTTP POST method to provide transcripts to a callback server. A callback server is a client-side HTTP server that is used to receive information from a known source and process it appropriately. In most cases, using callbacks rather than polling the results of submitting and analyzing audio files for transcription is simpler and more efficient.

Callback URLs are defined as part of V-Spark folder configuration. In most cases, you will not need to customize the callback URL that is configured when V-Spark folders are created - you will only want to do so if you want to integrate V-Spark with external applications (Filesystem, HTTP/S, SFTP), external storage locations for audio data (S3), and so on. Using a callback server with V-Spark is described in [Section 5.1, “Using Callbacks in V-Spark”](#), and is the preferred way of interacting with the V-Spark API.

For situations where a callback server cannot be used and you want to call the V-Spark API directly, [Section 5.2, “Reference for the request API”](#) provides reference material and examples of using the `request` API. It is not necessary to specify a callback URL as part of using the `request` API because results are returned directly by the API call, but you must wait until a call to the `request` method shows that processing has completed until you can retrieve complete results.



Note

Throughout the remainder of this section, HTTP refers to both of the HTTP and HTTPS protocols.

5.1. Using Callbacks in V-Spark

V-Spark uses the HTTP POST method to provide transcripts that have been enriched with analytical annotations to a callback server. This callback server is a client-side HTTP server used to receive information from a known source and process it appropriately.

In the simplest case, the callback server is configured to receive enriched transcript files from V-Spark and save them to a local file system for later use. The URL that identifies this folder supports the HTTP, HTTPS, Secure FTP, and Amazon Simple Storage Service protocols for on-premise deployments, as well as being able to write to the filesystem of the server on which V-Spark is running. Cloud-based deployments only support HTTP. Otherwise, the API is the same for both on-premise and cloud-based deployment methods. See [Section 5.1.1, “Configuring Callbacks in V-Spark”](#) for detailed information about how and where to specify these.



Note

If V-Spark cannot deliver results to a callback URL, it retries up to 100 times (by default), and then moves the results that it was trying to deliver to the folder `/var/lib/vspark/error/callback`, storing them with the same file name as that with which they would have been delivered. The number of times that callbacks are retried is configurable in the V-Spark configuration file.

Using callbacks to deliver enriched transcript (and other) files to a callback server can occasionally result in name collisions. A name collision results when files with the same name as the file that is

currently being processed already exist on the callback server. This can occur when, for example, two folders have the same callback settings and files with the same name are uploaded to those folders for transcription. V-Spark's callback mechanism automatically resolves name collisions. File that are delivered via callbacks automatically insert version number in file names to avoid name collisions. For example, if a file named `sample.json` already exists in the location where a callback writes files, the JSON file that the callback writes will be named `sample.1.json`. If both a file named `sample.json` and a file named `sample.1.json` already exist in the location where callbacks write files, the JSON file that the callback writes will be named `sample.2.json`. The number will increment until no file with a matching name is found.



Important

Callbacks that use the **File system**, **SFTP**, and **AWS S3** callback delivery methods automatically avoid name collisions. Name collisions in callbacks that use the **HTTP** and **HTTPS** callback delivery methods will over-write existing files.

By default, callbacks send transcription output from V-Spark's speech recognition engine to V-Spark on the same host for analysis and presentation within the V-Spark graphical interface. The callback server URL can be customized to enable feeding data from V-Spark to other applications for purposes such as performing additional analytics, archival, and so on.

5.1.1. Configuring Callbacks in V-Spark

The callback URL is specified as part of V-Spark folder configuration, which is accessed within V-Spark as shown in [Figure 5.1, “V-Spark Folder Settings”](#). To modify the settings for a folder:

- Move your mouse over the **Settings** menu at the top left of the V-Spark screen. The **Settings** menu displays.
- Select **Folders** from the **Settings** menu.
- Display the folder whose setting you want to modify. You can do this by selecting your **Company** and **Organization** from the drop-down menus below the V-Spark logo, or by (more easily) using the **Search folders** search box to search for and display information about a specified folder.

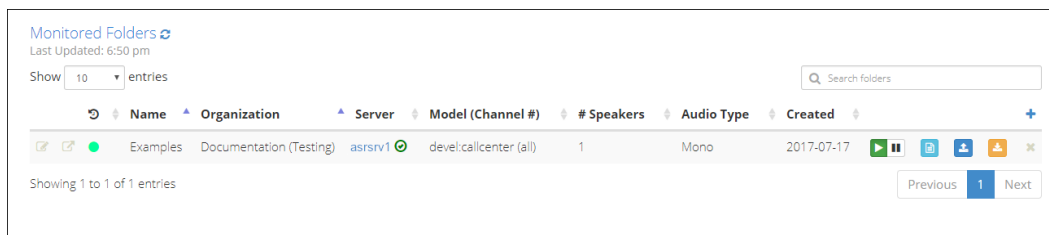


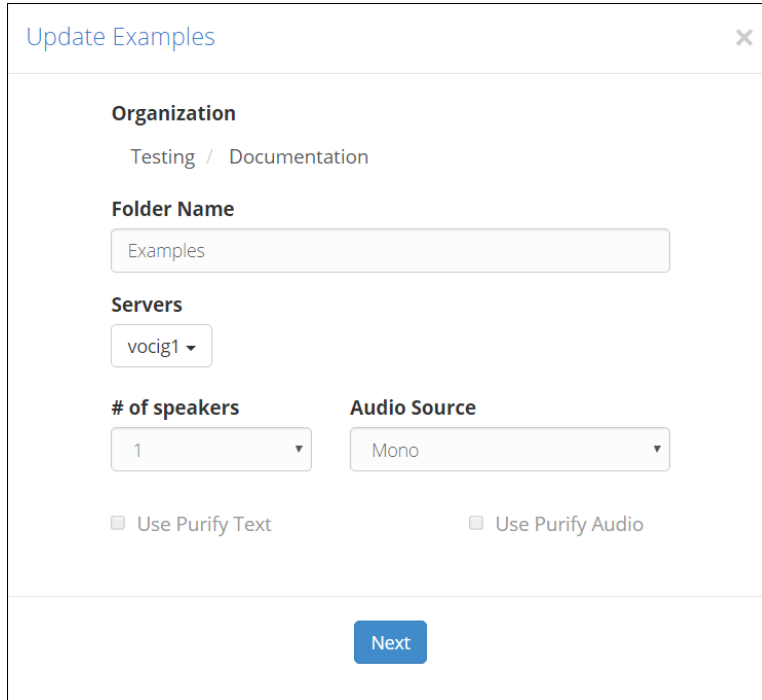
Figure 5.1. V-Spark Folder Settings



Note

Other aspects of working with V-Spark folders are discussed in the "[V-Spark 3.4.3 Management Guide](#)".

To edit the settings for an existing folder, click the **edit** icon. The dialog shown in [Figure 5.2, “Editing V-Spark Folder Settings”](#) displays.



Update Examples ✕

Organization
Testing / Documentation

Folder Name
Examples

Servers
vocig1 ▾

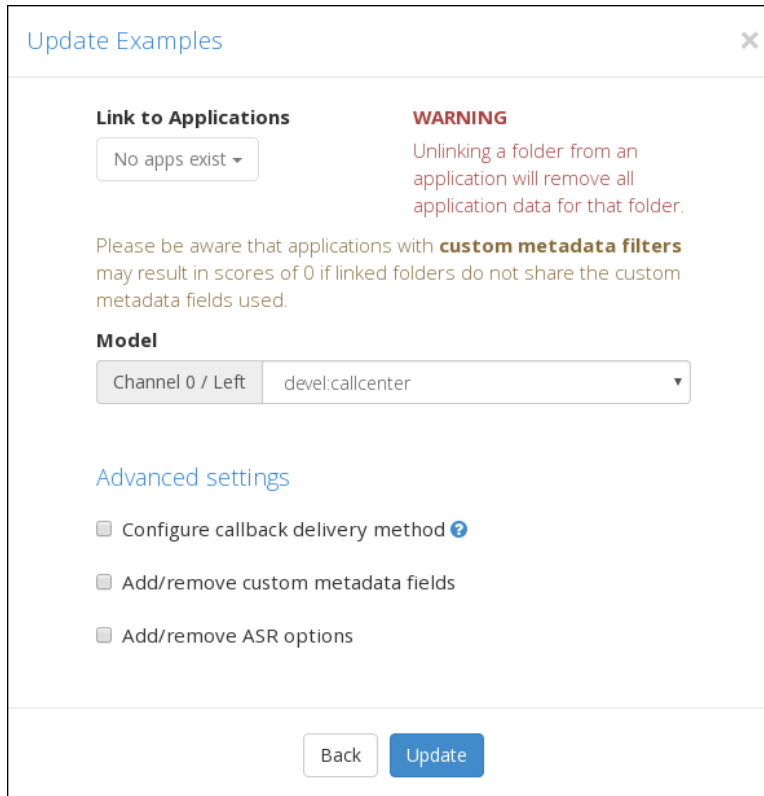
of speakers **Audio Source**
1 ▾ Mono ▾

Use Purify Text Use Purify Audio

[Next](#)

Figure 5.2. Editing V-Spark Folder Settings

Click **Next** to proceed to the part of this dialog in which you can specify the callback delivery method and select the types of data that V-Spark sends to it. The dialog shown in [Figure 5.3, “Further Customization of V-Spark Folder Settings”](#) displays.



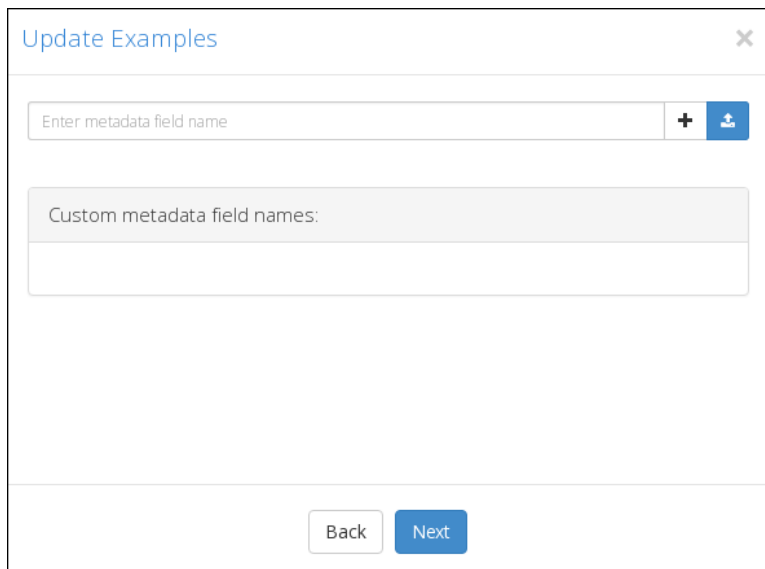
The screenshot shows a dialog box titled "Update Examples" with a close button (X) in the top right corner. The dialog is divided into several sections:

- Link to Applications:** A dropdown menu currently shows "No apps exist".
- WARNING:** A red warning message states: "Unlinking a folder from an application will remove all application data for that folder."
- Information:** A note says: "Please be aware that applications with **custom metadata filters** may result in scores of 0 if linked folders do not share the custom metadata fields used."
- Model:** A dropdown menu shows "Channel 0 / Left" and "dev:callcenter".
- Advanced settings:** A section with three unchecked checkboxes:
 - Configure callback delivery method [?](#)
 - Add/remove custom metadata fields
 - Add/remove ASR options
- Buttons:** "Back" and "Update" buttons are located at the bottom.

Figure 5.3. Further Customization of V-Spark Folder Settings

To display the portion of this dialog in which you can specify the parameters for and the location of the callback server used by V-Spark and set related options, select the checkbox beside **Configure callback delivery method**, and click **Next**. A dialog like the one shown in [Figure 5.6, “Configuring an HTTP Callback Server and related options”](#) displays.

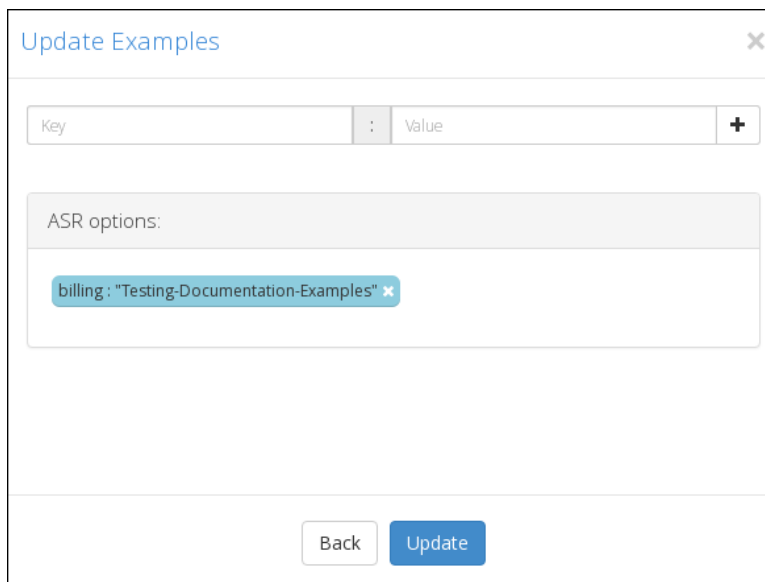
Other options at the bottom of this dialog are:



The screenshot shows a dialog box titled "Update Examples" with a close button (X) in the top right corner. At the top, there is a text input field labeled "Enter metadata field name" with a plus sign (+) and a blue button with a downward arrow. Below this is a section titled "Custom metadata field names:" with a large empty text area. At the bottom of the dialog, there are two buttons: "Back" and "Next".

Figure 5.4. Specifying custom metadata fields to include in output

- **Add/remove custom metadata fields** - selecting this checkbox and clicking **Next** displays the dialog shown in [Figure 5.4, “Specifying custom metadata fields to include in output”](#). This dialog enables you to identify custom metadata fields that you want to include in transcripts for this folder.
- **Add/remove ASR options** - selecting the checkbox beside this label and clicking **Next** displays the dialog shown in [Figure 5.5, “Specifying ASR options”](#). This dialog enables you to identify Automatic Speech Recognition (ASR) fields that you want to include in output for this folder.



The screenshot shows a dialog box titled "Update Examples" with a close button (X) in the top right corner. At the top, there is a text input field with "Key" on the left, a colon (:), "Value" on the right, and a plus sign (+). Below this is a section titled "ASR options:" with a list of options. One option is highlighted in blue: "billing: 'Testing-Documentation-Examples' ✕". At the bottom of the dialog, there are two buttons: "Back" and "Update".

Figure 5.5. Specifying ASR options



Note

If you have not selected one or more of the checkboxes beside **Configure callback delivery method**, **Add/remove custom metadata fields**, or **Add/remove ASR options** checkboxes and clicked **Next**, click **Update** to close this dialog and return to V-Spark.

By default, the callback configuration dialog shown in [Figure 5.6, “Configuring an HTTP Callback Server and related options”](#) displays the information that is required for a callback server which communicates using the HTTP protocol, including an example that identifies the format of a standard URL.

Figure 5.6. Configuring an HTTP Callback Server and related options

The **Callback Delivery Method** area consists of a dropdown at left which identifies the supported mechanisms for delivering callbacks, while the right portion of that area enables you to enter the path that V-Spark must use to deliver transcription and optional information. Supported callback delivery mechanisms are:

- **http://** - the standard Hypertext Transfer Protocol, which means that the callback server is a web server that is listening on a specified (or default) port
- **https://** - the Hypertext Transfer Protocol running over the Secure Sockets Layer, which means that the callback server is a web server that is listening on a specified (or default) port
- **File System** - transcription info is written to files that are located in a specified directory on the system on which the V-Spark software is installed. When using this protocol, make sure that the user group is set to **vocisrv** and that the directory is writable by that group so that V-Spark is able to write files into the specified directory.
- **SFTP** - transcripts and other requested information are sent to a server via the Secure File Transfer Protocol (SFTP), which requires additional authorization information, such as the username and password, or an SSH key, as shown in [Figure 5.7, “Configuring an SFTP Callback Server and related options”](#).
- **AWS S3** - transcription and other data is written to the Amazon Web Service (AWS) Simple Storage Service (S3). Specifying this callback delivery mechanism causes a dialog that is similar to the one shown in [Figure 5.7, “Configuring an SFTP Callback Server and related options”](#) to display fields in which you must specify the **AWS access key id** and **AWS secret access key** that are used to securely

communicate with AWS S3. See [Section 4.2, “Using the transcribe API with AWS S3”](#) for information about using the `transcribe` API to process archived audio files in zip format that are already stored in S3.

Figure 5.7. Configuring an SFTP Callback Server and related options

By default, the dialogs for each of these delivery mechanisms enable you to specify the type of transcription data that is being sent. By default, sending a JSON transcription is always enabled. You can also select:

- **MP3** - checking this box causes V-Spark to also send an MP3 version of the transcribed audio file to the callback server
- **Text** - checking this box causes V-Spark to also send a text version of the transcribed audio file to the callback server

Once you have made any changes that you want to make to the callback information, click **Update** to close this dialog and return to V-Spark.



Important

Make sure that the machine and port specified in the callback URL are accessible from the V-Spark instance that you are using. In some cases, this requires modifying client-side firewall rules.

5.1.2. Example Callback Server

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 usually used to enable another application

to receive and directly interact with the transcriptions produced by V-Spark. This section provides an example of setting up a simple callback server, submitting a sample audio file for transcription using the V-Spark `/transcribe` method, and then examining the results that are received by the callback server. This section concludes with suggestions for troubleshooting common problems when setting up and using a callback server.

5.1.2.1. 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](https://en.wikipedia.org/wiki/Netcat#Ports_and_reimplementations) [https://en.wikipedia.org/wiki/Netcat#Ports_and_reimplementations].

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 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.

5.1.2.2. Submitting a Sample File for Text Transcription

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/x-wav" \
http://vspark_host/transcribe
```

This sample command sends a text transcription 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:

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.

5.1.2.3. Receiving Transcription Results

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

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

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.

5.1.2.4. Troubleshooting a Callback Server

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 that was discussed in [Section 5.1.2.1, “Setting up a Sample Callback Server”](#).

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.zip;type=application/zip" \
-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.

5.2. Reference for the request API

If you have submitted audio files for processing using V-Spark or submitted zip files that contain audio files and optional metadata through the `transcribe` API method that is described in [Section 4.1, “Reference for the transcribe API”](#), the `request` API method enables you to retrieve overall status information, short and long status/summary information in JSON format, and various results of transcribing and analyzing those audio files. The `request` API is typically only directly used when you want to embed result retrieval in applications and you therefore cannot use the V-Spark GUI and its callback server mechanism.



Important

Before you can begin interacting with V-Spark via the REST API you must create a **Folder** using the V-Spark GUI. Each Folder has associated configuration information such as the language model to use for transcription and the names of metadata fields available for filtering. See the "*V-Spark 3.4.3 Quickstart Guide*" and "*V-Spark 3.4.3 Management Guide*" for more information. These documents are available under the Help pulldown immediately after logging into the V-Spark GUI.

Synopsis

```
http://SERVER:3000/request/ORG_SHORT/VERB?requestid=REQUESTID&token=AUTH_TOKEN
```

Variables used in this example command are the following:

- *SERVER*: the name or IP address of the computer system on which V-Spark is installed
- *ORG_SHORT*: the short name of the organization that you are using. Finding that information was discussed in [Chapter 4, *Submitting audio and metadata for processing*](#) and shown in [Figure 4.1, “Location of the V-Spark Organization Short Name”](#).
- *VERB*: the action that you want the `request` API to perform. These verbs and the data that they return are described in the next section (**DESCRIPTION**).
- *REQUESTID*: the unique identifier for the request about which you want to retrieve results or status information
- *AUTH_TOKEN*: the authorization token that you are using to retrieve information. Locating your authorization token is described in [Chapter 4, *Submitting audio and metadata for processing*](#). Locating an authorization token for the company associated with the folder that you are uploading to is shown in [Figure 1.1, “Location of a Company Authorization Token”](#).

Description

The `request` API enables you to retrieve status information and converted content for files that you have submitted for transcription.

The items in the rest of this section refer to retrieving status information or transcription results from both zip and audio files that have been submitted for transcription. Remember that while both the `transcribe` API and V-Spark enable you to submit audio and metadata files individually, only files that have been encapsulated into a single "zip" file and uploaded at the same time will be associated properly.

The `request` method takes the following verbs to identify the type of information that you are attempting to retrieve based on a `requestid`:

- `status`: Possible return values for this verb are the following:
 - `analyzing`: the zip or audio file associated with the specified `requestid` has been received and is in the process of being transcribed and analyzed by V-Spark
 - `done`: transcription of all of the audio files in the zip or audio file associated with the specified `requestid` has completed
 - `error`: no results will ever be available. Check the contents of the zip or audio file that is associated with the specified `requestid` to verify that it contains valid audio files.
 - `received`: the zip or audio file associated with the specified `requestid` has been received and the audio file(s) that it contains are in the process of being transcribed
- `summary`: returns a short response in JSON format that contains the following information about the request that was identified by the specified `requestid`:
 - a `time_submit` timestamp which identifies the time that the request was submitted
 - a `time_complete` timestamp which identifies the time that analyzing the contents of the zip or audio file was completed
 - counts of the following categories:
 - number of audio files that were **submitted** in the zip or audio file that is associated with the specified `requestid`
 - number of audio files that were successfully **processed** in the zip or audio file that is associated with the specified `requestid`
 - number of audio files that were successfully **analyzed** in the zip or audio file that is associated with the specified `requestid`
 - number of audio files in the zip or audio file that is associated with the specified `requestid` that generated an **error** when being processed or analyzed



Note

Count information is not provided for fields that are NULL.

- `status`: the over-all status of handling the zip or audio file associated with the request that is associated with the specified `requestid`. This is one of the possible values that can be returned by the `status` verb.
- `details`: returns a long response in JSON format that contains:
 - the information provided by the `summary` verb for the zip or audio file that was uploaded under the specified `requestid`

- a `filedetails` section that provides a `fullfilename` and the same information for each file that was part of the request that was identified by the specified `requestid`:
- `result`: a zip file containing different types of data that is associated with the transcription of the zip or audio file that was uploaded under the specified `requestid`. Optional Boolean parameters identify the type of data that is contained in this zip file:
 - `json`: include the complete transcription information (transcribed text, emotion, sentiment, and so on) in JSON format for each audio file. By default, this option is "on" ("1").
 - `mp3`: include the mp3 version of each audio file. By default, this option is "off" ("0").
 - `txt`: include a text file containing the transcription of each audio file. By default, this option is "off" ("0").

If the `request` method's `result` verb is called with all output formats disabled, the API call will return a 404 and the message "No file types were specified".

5.2.1. Examples of calling the request API

The following examples show calling the `request` method with some of its verbs, using the `cURL` (crawl URL) command-line command (executed as the `curl` command). Sample output is also provided, but depends on the host on which the API is running, the contents of the zip or audio file that you uploaded, the request ID, the organization to which you uploaded the file, and authorization token that you are using. The `cURL` utility is freely available for operating systems including [Linux, Windows, and Mac OS X](https://curl.haxx.se/download.html) [https://curl.haxx.se/download.html].



Note

Escaped newlines (that is, lines in the `cURL` command or the example output that end with a backslash) are added for readability. They must not be present in `cURL` commands, and are also not present in the output of those commands.

Example commands are shown in normal monospaced text. Example output from each command is shown in **bold** monospaced text.

A `cURL` example that shows calling the `request` API's `status` verb:

```
curl 'http://example.company.com:3000/request/Test-Testing/status \
?requestid=a0cf623d-9e5c-4890-886f-832acb29635e \
&token=0123456789abcde0123456789abcde01'
```

done

A `cURL` example that shows calling the `request` API's `summary` verb:

```
curl 'http://example.company.com:3000/request/Test-Testing/summary\
?requestid=a0cf623d-9e5c-4890-886f-832acb29635e\
&token=0123456789abcde0123456789abcde01'
```

**{ "time_submit": "2017-01-16T21:43:08.000Z", \
"time_complete": "2017-01-16T21:45:16.000Z", \
"submitted": 1, "processed": 1, "analyzed": 1, "error": 0, "status": "done" }**

A `cURL` example that shows calling the `request` API's `details` verb:


```
curl 'http://example.company.com:3000/request/Test-Testing/details?\
requestid=a0cf623d-9e5c-4890-886f-832acb29635e\
&token=0123456789abcde0123456789abcde01'

{"time_submit":"2017-01-16T21:43:08.000Z",\
"time_complete":"2017-01-16T21:45:16.000Z", \
"submitted":1,"processed":1,"analyzed":1,"error":0,"status":"done",\
"filedetails":[{"fullfilename":"Call14541511.mp3","status":"OK",\
"job_start":"2017-01-16T21:45:14.000Z",\
"job_finish":"2017-01-16T21:43:12.000Z",\
"analyze":"2017-01-16T21:45:14.000Z","size":4984848}]}
```

A cURL example that shows calling the request API's result verb to return the JSON transcript of your audio file and write it to the file `json_transcript.zip`:

```
curl 'http://example.company.com:3000/request/Test-Testing/result?\
requestid=a0cf623d-9e5c-4890-886f-832acb29635e\
&token=0123456789abcde0123456789abcde01' > json_transcript.zip

  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left    Speed
100 28304    100 28304    0     0    2780      0  0:00:10  0:00:10  ---:--:-- 6821

(Displays cURL's download status)
```


Chapter 6. Retrieving Log and Status Information

[Chapter 3, *Retrieving and Updating V-Spark Information*](#) provided an overview of the V-Spark REST APIs that enable you to read (and, in one case, update) configuration information about a V-Spark installation. [Chapter 7, *Searching V-Spark Data*](#) provides detailed information about the `/search` API. The next few sections provide detailed information about the `/log` and `/status` APIs that give you programmatic access to information about the state of a V-Spark installation's transcription and analysis of audio files. All of these APIs return and use information in JSON format.

As REST APIs, these APIs can be used in any programming language or with any application that supports both REST calls and which provides or can invoke a JSON parser that enables you to work with the output of these calls.

6.1. Retrieving Log Information

The `/log` API enables you to retrieve log information about any folder in a specified V-Spark installation.

6.1.1. Reference for the `/log` API

The `/log` API enables you to retrieve log information that was produced on a specified day from a specified folder, organization, and company in a specific V-Spark installation. Information is returned as a JSON list.

Synopsis

```
/log/CO_SHORT/ORG_SHORT/FOLDERNAME?token=TOKEN&date=YYYYMMDD
```

Description

As shown in the synopsis, the V-Spark `/log` API only supports a single GET call, which retrieves log information that was produced on a certain day from a specific folder, associated with a certain organization, and located under a certain company. Variables used in a call to the `/log` API are the following:

- `CO_SHORT`: the short name of the company for which you want to retrieve log data
- `ORG_SHORT`: the short name of the organization that you are using. Finding that information is shown in [Figure 4.1, “Location of the V-Spark Organization Short Name”](#).
- `FOLDERNAME`: the name of the V-Spark Folder from which you want to retrieve log information
- `TOKEN`: the V-Spark authorization token that you are using to establish permission to retrieve information. You can either use the root token for the target V-Spark installation (located in the file `/opt/voci/state/vspark/apitoken`) or the authorization token for the company under which the specified `FOLDERNAME` is located. Locating your company authorization token is shown in [Figure 1.1, “Location of a Company Authorization Token”](#).
- `YYYYMMDD`: the date for which you want to retrieve log data, expressed as *four-digit-year**two-digit-month**two-digit-day*. This date is passed to the API using the `date` parameter.

6.1.2. Sample JSON Output for a folder from the /log API

Figure 6.1, “Sample Folder Log Output from the /log API” shows an excerpt of the output that is produced by a call to the log API using a folder which contains audio files that were processed on a specified date (or the current date if the date parameter was not specified).

```
[
  [
    "FileName",
    "FileProperties",
    "Duration",
    "Size",
    "FileDate",
    "Time",
    "Status"
  ],
  [
    "CallTEST.mp3",
    "MPEG Audio (Layer 3), CBR, 24.0 Kbps, 8000 Hz, , 1 ch",
    "836.14",
    "2509272",
    "2017-06-15_09:05:11",
    "2017-06-15_09:05:52",
    "OK"
  ],...
]
```

Figure 6.1. Sample Folder Log Output from the /log API

As shown in Figure 6.1, the /log API returns audio processing log data as a list rather than as a JSON dictionary. The first entry in the list identifies the names of the entries which exists in every subsequent row. You will need to convert the list returned by the /log API to the data format in which you want to use the report. A useful (and free) tool for previewing this JSON is the [free JSON Table viewer](http://json2table.com) [http://json2table.com]. Figure 6.2, “Sample Folder Log Output from the /log API” shows a sample list from the /log API in this tool.

	CallTEST.mp3						
FileName	MPEG Audio (Layer 3), CBR, 24.0	audio-and-metadata/file1json.wav	audio-and-metadata/file1json.wav	audio-and-metadata/file1json.wav	audio-and-metadata/file1json.wav	audio-and-metadata/file1json.wav	audio-and-metadata/file1json.wav
FileProperties	Kbps, 8000 Hz, , 1 ch	PCM, CBR, 256 Kbps, 8000 Hz, 16 bits, 2 ch	PCM, CBR, 256 Kbps, 8000 Hz, 16 bits, 2 ch	PCM, CBR, 256 Kbps, 8000 Hz, 16 bits, 2 ch	PCM, CBR, 256 Kbps, 8000 Hz, 16 bits, 2 ch	PCM, CBR, 256 Kbps, 8000 Hz, 16 bits, 2 ch	PCM, CBR, 256 Kbps, 8000 Hz, 16 bits, 2 ch
Duration	836.14	325.38	325.38	325.38	325.38	325.38	325.38
Size	2509272	10412206	10412206	10412206	10412206	10412206	10412206
FileDate	2017-06-15_09:05:11	2017-06-15_12:23:09	2017-06-15_12:40:18	2017-06-15_12:42:27	2017-06-15_12:47:13	2017-06-15_12:49:22	2017-06-15_12:50:43
Time	2017-06-15_09:05:52	2017-06-15_12:23:41	2017-06-15_12:40:51	2017-06-15_12:43:00	2017-06-15_12:47:45	2017-06-15_12:49:54	2017-06-15_12:51:16
Status	OK	OK	OK	OK	OK	OK	OK

Figure 6.2. Sample Folder Log Output from the /log API

See Section 6.1.3, “Using the /log API with cURL” for examples of retrieving /log data for a folder using cURL, where you supply the company, organization, folder, and date. See Section 6.1.4, “Using the /log API with python” for an example of a Python application that prints log data for all folders in a V-Spark installation that contain audio which was processed on a specified date.

6.1.3. Using the /log API with cURL

If you are unfamiliar with the cURL command, see Section 1.3, “Using cURL for REST API Testing” for a short introduction and an explanation of how cURL examples are displayed. See Section 1.3.2, “Tips for Debugging and Managing cURL Calls” for suggestions about how to debug and manage cURL calls.

The `/log` API makes it easy to programmatically retrieve log data for a specific folder on a specific date. See [Section 6.1.1, “Reference for the /log API”](#) for detailed information about the values and parameters that you need to supply when making a call to the `/log` API.

An example of a cURL command to retrieve log information from June 15, 2017 (20170615) from the company "DocTestCo", organization "DocTestCo-DocTesting", folder "Test01" on the host 192.168.6.64 where V-Spark is listening on port 3000, is the following:

```
curl -s 'http://192.168.6.64:3000/log/DocTestCo/DocTestCo-DocTesting/Test01 \
?token=012345678901234567890123456789012&date=20170615'
```

To produce this output in a pretty-printed form that is more usable, and write that output to the file `log.out`, you could execute a command like the following:

```
curl -s 'http://192.168.6.64:3000/log/DocTestCo/DocTestCo-DocTesting/Test01 \
?token=012345678901234567890123456789012&date=20170615' | \
python -m json.tool > log.out
```

6.1.4. Using the /log API with python

V-Spark's `/log` API is a read-only API that provides programmatic access to log data about audio files that have been transcribed on a certain date. As explained in [Section 6.1.1, “Reference for the /log API”](#) and shown in the examples in [Section 6.1.3, “Using the /log API with cURL”](#), the API enables you to specify the company, organization, and folder that you are interested in as part of the call, and specify the data that you are interested in as the `date` parameter to the `/log` API call



Tip

To read data from a V-Spark installation, you can use the root token, but that's analogous to running every Linux command as the superuser - it is cleaner to use the authorization token that is associated with each company. You only need the root token when reading multi-company information, such as when retrieving the root token for each company in the V-Spark installation.

As shown previously in [Figure 3.25, “Sample Python code to combine data read from the /config API”](#), you can dynamically retrieve company information from a V-Spark installation (in the `gettokens()` function) and folder information for the organizations in that company (retrieved in the `getfolderinfo()` function). You can then combine the company and token information with the folder information in order to explore all of the companies, organizations, and folders in a V-Spark installation (in the `printfolders()` function).

[Figure 6.3, “Sample Python Code to Search for Audio Processed on a Given Day”](#) builds on the code from [Figure 3.25](#) in several ways:

- Sharing the `gettokens()` and `getfolderinfo()` functions shows that extracting company and token information and establishing the relationship between company token information and the hierarchical aspects of companies, organizations, and folders are common tasks when writing many V-Spark applications.
- Replacing the `printfolderinfo()` function with a `findfolders()` function makes it easy to call another function to interact with each folder.
- Adds a `findandprintloginfo()` function to perform the key functionality of the example, writing log information for audio files that were processed on the specified date.

As with the Python code example shown in [Figure 3.25](#), a quick code walkthrough highlights how this example works.

```
#!/usr/bin/env python
#
# Copyright 2017 Voci Technologies, Inc. All rights reserved.
# Contains confidential company information.
# Unsupported example code - Not for production use.
#
# Application that reads company and folder information about a
# product installation on a specified host, then uses the company's
# short name to link the two. The application then uses the company's
# auth token to walk through all of the folders in the installation,
# and checks each for audio files that were processed on the specified
# date. When any are found that match, the application prints the
# associated log information to a folder named
# CO_SHORT-ORG_SHORT-FOLDER-log.json.

import requests
import json
import urllib2

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

def main(argv): ❶
    if len(argv) != 4: usage(argv) ❷
    host, token, date = argv[1:]
    tokens = gettokens(host,token)
    folderinfo = getfolderinfo(host,token)
    printfolders(host, folderinfo, tokens, date)

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

def findfolders(host, folder_info, tokens, date): ❺
    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():
                findandprintloginfo(host, tokens[comp], comp, org, folder, date)

def findandprintloginfo(host, token, comp, org, folder, date): ❻
    url = "http://%s/log/%s/%s/%s?token=%s&date=%s" % (host, comp, org, folder, token, date)
    response = requests.get(url)
    if response.status_code == 200:
        print " URL is "+url
        OUTPUT_FILE = comp+"-"+org+"-"+folder+"-log.json"
        print " Writing JSON 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)
```

Figure 6.3. Sample Python Code to Search for Audio Processed on a Given Day

The major steps in the Python application shown in [Figure 6.3](#), “Sample Python Code to Search for Audio Processed on a Given Day” are the following:

- ❶ The main function provides a traditional main routine that shows the order in which functions are called in the application

- ② Check if the right number of command-line arguments have been provided, assign them to appropriate variables if so and identifying the expected arguments if not.
- ③ Uses the `/config` API to retrieve the company information from the V-Spark installation and build a dictionary that only contains the company name and associated token information from the host that was specified on the command-line
- ④ Uses the `/config/folders` API to retrieve the folder-level configuration information from the host that was specified on the command-line
- ⑤ Initiates the primary loop for the application, which is controlled by the companies that were found in the information that was retrieved from the host specified on the command-line. Each company has an associated authorization token (originally stored in the `uuid` name/value pair), which is the other field for each company entry in the dictionary that was constructed in the `gettokens()` function. The short name for each company is the data item in the company JSON that provides the linkage between the data from the company and folder sources. This loop then uses this information to walk the company/organization/folder hierarchy in the V-Spark installation that was specified on the command-line, calling the function that represents the core functionality of this application.
- ⑥ Tests each folder for audio that was processed on the date that was specified on the command-line, and tests the result of the HTTP call to the REST API to determine if a match was found. If so, pretty-prints the log information for that folder.

6.2. Retrieving Status Information

The `/status` API enables you to retrieve status information about the transactions that are associated with a company, organization, or folder within a specified V-Spark installation.

6.2.1. Reference for the `/status` API

The `/status` API enables you to retrieve status information about the activities that are associated with a company, organization, or folder within a specified V-Spark installation.

Synopsis

```
/status
/status/CO_SHORT
/status/CO_SHORT/ORG_SHORT
/status/CO_SHORT/ORG_SHORT/FOLDERNAME
```

Description

As shown in the synopsis, the V-Spark `/status` API enables you to retrieve status information about an entire V-Spark installation, a specific company, a specific organization, or a specific folder. Variables used in a call to the `/status` API are the following:

- `CO_SHORT`: the short name of the company for which you want to retrieve status information
- `ORG_SHORT`: the short name of the organization that you are using. Finding that information is shown in [Figure 4.1, “Location of the V-Spark Organization Short Name”](#).
- `FOLDERNAME`: the name of the V-Spark Folder from which you want to retrieve status information
- `TOKEN`: the V-Spark authorization token that you are using to establish permission to retrieve information. You can either use the root token for the target V-Spark installation (located in the file /

opt/voci/state/vspark/apitoken) or the authorization token for the company under which the specified FOLDER is located. Locating your company authorization token is shown in [Figure 1.1, “Location of a Company Authorization Token”](#).

- `json` or `csv`: specifies the format in which you want status information to be delivered. Format information is passed to the API using the `format` parameter.

When status information is returned in CSV form, the first row in the output provides heading for the data in the other rows in the table.



Tip

Being able to retrieve information from the `/status` API in CSV format (by passing the `format=csv` parameter) makes it easy to quickly preview any level of status information. Tools such as spreadsheets typically support importing CSV data. As an example, [Figure 6.4, “Previewing /status information in a spreadsheet”](#) shows the CSV output for a folder being previewed in the Libre Office spreadsheet application, **Calc**.

The next few sections illustrate the `/status` information that is retrieved from the various levels of the hierarchy of a V-Spark installation.

6.2.2. Sample JSON and CSV Output from the `/status` API

The `/status` API returns a JSON or CSV (comma-separated value) representation of the V-Spark installation status data that is being requested. The following sections show the JSON and CSV output for each aspect of a V-Spark installation:

- [Section 6.2.2.1, “Sample /status JSON and CSV Output for a Company”](#)
- [Section 6.2.2.2, “Sample /status JSON and CSV Output for an Organization”](#)
- [Section 6.2.2.3, “Sample /status JSON and CSV Output for a Folder”](#)



Note

In the CSV examples, linebreaks have been inserted in both the heading and data entries at the same output rows.

6.2.2.1. Sample `/status` JSON and CSV Output for a Company

The following is sample output for a company in a V-Spark installation from the `/status` API. This shows the status of the organizations that have been defined within that company:

```
{
  "DocTestCo-DocTesting": {
    "Test01": {
      "mode": "active",
      "servers": {
        "asrsrvr1": "OK"
      },
      "queued": {
        "count": 0,
        "lastactive": "2017-06-16 09:06:45.584507060 -0400"
      },
      "decoding": {
        "count": 0,

```



```

        "lastactive": "2017-06-16 09:05:45.451651713 -0400"
      },
      "analyzing": {
        "count": 0,
        "lastactive": "2017-06-16 09:15:24.235266461 -0400"
      },
      "error": {
        "count": 1,
        "lastactive": "2017-05-18 12:46:55.780155897 -0400"
      }
    },...
  },...
}

```

6.2.2.2. Sample /status JSON and CSV Output for an Organization

The following is sample output for an organization within a company in a V-Spark installation from the /status API. This shows the folders that are defined within that organization:

```

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

```

The CSV equivalent of this data is the following:

```

"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"

```

6.2.2.3. Sample /status JSON and CSV Output for a Folder

The following is sample JSON output for a folder in a V-Spark installation from the /status API:

```

{
  "mode": "active",
  "servers": {
    "asrsrvr1": "OK"
  },
  "queued": {
    "count": 0,
    "lastactive": "2017-06-16 09:06:45.584507060 -0400"
  },
  "decoding": {
    "count": 0,
    "lastactive": "2017-06-16 09:05:45.451651713 -0400"
  }
}

```

```

    },
    "analyzing": {
      "count": 0,
      "lastactive": "2017-06-16 09:15:24.235266461 -0400"
    },
    "error": {
      "count": 1,
      "lastactive": "2017-05-18 12:46:55.780155897 -0400"
    }
  }
}

```

```

"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"

```

Figure 6.4, “Previewing /status information in a spreadsheet” shows the CSV output for a folder being previewed in the Libre Office spreadsheet application, **Calc**.

	A	B	C	D	E	F	G	H	I
1	company	organization	folder	mode	servers	queued.count	queued.lastactive	decoding.count	decoding.lastactive
2	DocTestCo	DocTestCo-DocTesting	Test01	active	{\"vocig1\": \"OK\"}	0	2017-06-16 09:06:45.584507060 -0400	0	2017-06-16 09:05:45.451651713 -0400
3									
4									
5									
6									
7									

Figure 6.4. Previewing /status information in a spreadsheet

6.2.3. Using the /status API with cURL

The cURL utility makes it easy to test using the V-Spark API by providing a command-line mechanism for invoking APIs such as the /status API. The next few sections provide examples of using the GET verb with the /status API from the command-line via the cURL command.

If you are unfamiliar with the cURL command, see [Section 1.3, “Using cURL for REST API Testing”](#) for a short introduction and an explanation of how cURL examples are displayed. See [Section 1.3.2, “Tips for Debugging and Managing cURL Calls”](#) for suggestions about how to debug and manage cURL calls.

An example of a cURL command to retrieve status information from June 15, 2017 (20170615) from the company "DocTestCo", organization "DocTestCo-DocTesting", folder "Test01" on the host 192.168.6.64 where V-Spark is listening on port 3000, is the following:

```

curl -s 'http://192.168.6.64:3000/status/DocTestCo/DocTestCo-DocTesting/Test01 \
?token=01234567890123456789012345678901'

```

To produce this output in a pretty-printed form that is more usable, and write that output to the file `log.out`, you could execute a command like the following:

```

curl -s 'http://192.168.6.64:3000/status/DocTestCo/DocTestCo-DocTesting/Test01 \
?token=012345678901234567890123456789012' | \
python -m json.tool > log.out

```

6.2.4. Using the /status API with Python

V-Spark's `/status` API is a read-only API that provides programmatic access to status information about various levels of the hierarchy of a V-Spark installation. As explained in [Section 6.2.1, “Reference for the /status API”](#) and shown in the examples in [Section 6.2.3, “Using the /status API with cURL”](#), the API enables you to pass the level of the hierarchy that you are interested in as part of the URL of the `/status` call, and specify the format in which you want the output data to be returned as the `format` parameter to the call.

To read data from a V-Spark installation, you can use the root token, but that's analogous to running every Linux command as the superuser - it is cleaner to use the authorization token that is associated with each company. You only need the root token when reading multi-company information, such as when retrieving the root token for each company in the V-Spark installation.

As shown previously in [Figure 3.25, “Sample Python code to combine data read from the /config API”](#), you can dynamically retrieve company information from a V-Spark installation (in the `gettokens()` function) and folder information for the organizations in that company (retrieved in the `getfolderinfo()` function). You can then combine the company and token information with the folder information in order to explore all of the companies, organizations, and folders in a V-Spark installation (in the `printfolders()` function).

[Figure 6.5, “Sample Python Code to Retrieve /status Information”](#) builds on the code from [Figure 6.3](#) and [Figure 3.25](#) in several ways:

- Sharing the `gettokens()` and `getfolderinfo()` functions shows that extracting company and token information and establishing the relationship between company token information and the hierarchical aspects of companies, organizations, and folders are common tasks when writing many V-Spark applications.
- Replacing the `printfolderinfo()` function with a `findfolders()` function makes it easy to call another function to interact with each folder.
- Adds a `writefolderstatus()` function to perform the key functionality of the example, writing status information for that level of their hierarchy in the format that you specified on the command-line.

As with the Python code example shown in [Figure 3.25](#) and [Figure 6.3](#), a quick code walkthrough highlights how this example works.

```

#!/usr/bin/env python
#
# Copyright 2017 Voci Technologies, Inc. All rights reserved.
# Contains confidential company information.
# Unsupported example code - Not for production use.
#
# Application that reads company and folder information about a
# product installation on a specified host, then uses the company's
# short name to link the two. The application then uses the company's
# short name, the organization's short name, and each folder name to
# call the /status API for each folder. The output format (json or
# csv) is specified as the third argument, and determines both the
# extension of the files that are written to and the format of their
# content.
#

import requests
import json

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

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

def gettokens(host, token): ❸
    url = "http://%s:3000/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): ❹
    url = "http://%s:3000/config/folders?token=%s" % (host,token)
    return requests.get(url).json()

def findfolders(host, folder_info, tokens, output_format): ❺
    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): ❻
    url = "http://%s:3000/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__': ❼
    from sys import argv
    main(argv)

```

Figure 6.5. Sample Python Code to Retrieve /status Information

The major steps in the Python application shown in [Figure 6.5, “Sample Python Code to Retrieve /status Information”](#) are the following:

- ❶ The `main` function provides a traditional main routine that shows the order in which functions are called in the application

- ② Check if the right number of command-line arguments have been provided, assign them to appropriate variables if so and identifying the expected arguments if not.
- ③ Uses the `/config` API to retrieve the company information from the V-Spark installation and build a dictionary that only contains the company name and associated token information from the host that was specified on the command-line
- ④ Uses the `/config/folders` API to retrieve the folder-level configuration information from the host that was specified on the command-line
- ⑤ Initiates the primary loop for the application, which is controlled by the companies that were found in the information that was retrieved from the host specified on the command-line. Each company has an associated authorization token (originally stored in the `uuid` name/value pair), which is the other field for each company entry in the dictionary that was constructed in the `gettokens()` function. The short name for each company is the data item in the company JSON that provides the linkage between the data from the company and folder sources. This loop then uses this information to walk the company/organization/folder hierarchy in the V-Spark installation that was specified on the command-line, calling the function that represents the core functionality of this application.
- ⑥ Writes status information in the format that was requested on the command-line to a file whose name is `CO_SHORT-ORG_SHORT-FOLDER-status.output-format`.
- ⑦ The name of the scope in which the top-level code executes. This enables the sample application to execute standalone or as a part of another application. This is a standard Python notation, and is not anything that is specific to V-Spark.

Chapter 7. Searching V-Spark Data

V-Spark provides several REST APIs that enable you to read (and, in one case, update) configuration, status, and log information about a V-Spark installation, and also to search that installation. [Chapter 3, Retrieving and Updating V-Spark Information](#) provided an overview of these APIs and then provided details about the `/config` and `/list` APIs. [Chapter 6, Retrieving Log and Status Information](#) provides detailed information about the `/log` and `/status` APIs. The next few sections provide detailed information about the `/search` API which enables you to programmatically search the files that contain transcripts in specified folders in a V-Spark installation. All of these APIs return and use information in JSON format, many also support Comma-Separated Value (CSV) output, and some support direct exporting of matching transcript files in the ZIP archive format.

As REST APIs, these APIs can be used in any programming language or with any application that supports both REST calls and which provides or can invoke a JSON parser that enables you to work with the output of these calls.

7.1. Reference for the `/search` API

The `/search` API enables you to search all transcripts for an organization or those that are located in a folder within an organization. You can search using two different REST methods:

- GET: search parameters are specified as options that are passed as parameters which are part of a query
- POST: search parameters are specified name/value pairs in a JSON file that is passed to the API

Synopsis

```
GET /search/CO_SHORT/ORG_SHORT?token=TOKEN&OPTIONS...
GET /search/CO_SHORT/ORG_SHORT/FOLDERNAME?token=TOKEN&OPTIONS...
POST /search/CO_SHORT/ORG_SHORT?token=TOKEN&OPTIONS...
POST /search/CO_SHORT/ORG_SHORT/FOLDERNAME?token=TOKEN&OPTIONS...
```

Description

As shown in the synopsis, the V-Spark `/search` API supports both GET and POST calls, both of which search a specified V-Spark installation, but which pass parameters to the API in different ways. See [Section 7.3, “Using the `/search` API with cURL”](#) for examples of passing parameters in both ways using the cURL command. See [Section 7.4, “Using the `/search` API with Python”](#) for examples of using both the GET and POST models in Python.

Variables used in a call to the `/search` API are the following:

- `CO_SHORT`: the short name of the company whose transcripts you want to search
- `ORG_SHORT`: the short name of the organization that you are interested in. Finding that information is shown in [Figure 4.1, “Location of the V-Spark Organization Short Name”](#).
- `FOLDERNAME`: the name of the V-Spark Folder whose transcripts you want to search
- `TOKEN`: the V-Spark authorization token that you are using to establish permission to retrieve information. You can either use the root token for the target V-Spark installation (located in the file `/opt/voci/state/vspark/apitoken`) or the authorization token for the company under which the specified `ORG_SHORT` or `FOLDERNAME` is located. Locating a company's authorization token is shown in [Figure 1.1, “Location of a Company Authorization Token”](#).

Subsequent sections provide detailed information about the many GET options (which map to POST name/value pairs) that you can specify in calls to the `/search` API. Options are discussed in the form that they would be specified as part of a GET call. The value that you specify for the `output` option determines what other sets of options you can specify.

7.1.1. Output Type Options

After identifying the `token` that you want to use to access the V-Spark installation or company data that you want to query (discussed in [Section 7.1, “Reference for the /search API”](#), the most basic option that you will want to specify is the `output` option. The value that you provide for this option determines the other options that you can provide when calling the `/search` API:

- `output=count | details | summary | zip` - Specifies the type of results that you want to receive from a search query:
 - `count` - returns the number of search results that matched your query. After specifying the `output=count` option on the command-line or in a JSON file, you can also specify any of the options discussed in [Section 7.1.2, “Search Term Options”](#).
 - `details` - returns a collection of the Voci JSON for the search results for each result that matched your query. After specifying the `output=details` option on the command-line or in a JSON file, you can also specify any of the options discussed in [Section 7.1.2, “Search Term Options”](#) or [Section 7.1.5, “Output Sorting Options”](#).
 - `summary` - returns a summary of the matching search results in JSON or Comma-Separated Value (CSV) format. This 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 any of the options discussed in [Section 7.1.2, “Search Term Options”](#), [Section 7.1.5, “Output Sorting Options”](#), [Section 7.1.4, “Output Field Options”](#), or [Section 7.1.3, “Output Format Options”](#).
 - `zip` - returns the files that matched your query, returning them in the ZIP archive format. The ZIP archive 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 any of the options discussed in [Section 7.1.2, “Search Term Options”](#) or [Section 7.1.5, “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"
```

7.1.2. Search Term Options

The types of searches that you can perform using the V-Spark `/search` API demonstrate its power and flexibility. Search parameters and potential values are the following:

- `daterange=START-END` - Enables you to specify a date range that your search should be restricted to. The `START` and `END` values, though optional, are both expressed as `YYYYMMDD[HHmmss]` values where the year (YYYY) month (MM) and day (DD) values are required, the hours (HH), minutes (mm), and seconds (ss) are optional. Date ranges are always assumed to be positive (where `START` is less than

END). No verification is done to ensure that this is 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.

- terms *.FIELD=VALUE* - fields that can identified for searching are the following:
 - agent
 - agentid
 - client
 - file
 - requestid
 - speakers
 - tag
 - tid
- *CLIENT-DATA* - search any of the custom metadata fields that have been entered within the folder with which your audio files are associated
- *terms.op* - multiple terms (and therefore, their associated fields) can be identified to be searched by combining them with the logical and and or operators, where and is the default operator.
- *TYPE emotion | gender=enumerated-value* - possible values for *TYPE* are agent, client, and overall (where appropriate - overall does not make sense in the context of gender); possible value for emotion are Improving, Negative, Positive, and Worsening; possible values for gender are Female and Male. Only one emotion and one gender filter can be used within a single search.
- *duration | overtalk | silence | agent_clarity | client_clarity = N-M* - numerical searches based on these fields, where:
 - *duration* - expressed as *N* and *M*, which are a common colon delimited duration format *hh:mm:ss* that define a numeric range in seconds, where seconds (ss) is required, hours (hh), and minutes (mm) are optional, and colons are only included as appropriate when hours or minutes are included.
 - *overtalk | silence | agent_clarity | client_clarity* - expressed as *N* and *M*, which are floating point values that define a numeric range as a percentage.
- *diarization* - a floating point value that defines a percentage. Only results with the defined diarization score or greater will be included. A *diarization* value of "2" will only return calls that were not diarized.
- *app.name=APPNAME* and *app.APPNAME.TOP CATEGORY[.LOWER CATEGORY...]* = All | High | Medium | Low | None - *APPNAME* specifies the name of the application that you want to search. *TOP CATEGORY* and (optionally) *LOWER CATEGORY* identify various scoring categories:
 - All - greater than 0
 - High - greater than 0.66
 - Medium - greater than 0.33

- Low - between 0 and 0.33
- None - 0

7.1.3. Output Format Options

The format in which search results are delivered is specified by the following option:

- `format=csv | json` - identifies the output format in which your search results are delivered. The default value is `json`. Producing Comma-Separated Value (`csv`) output simplifies importing search results into other software, such as spreadsheets, that support CSV input.



Important

If custom metadata fields are associated with the folders in which search results are found, JSON output will only include fields that have values. CSV output will include all fields, with an empty value for fields without an explicit value.

7.1.4. Output Field Options

Depending on the output format that you specified in your query, the `/search` API enables you to identify the fields that you want to display when showing the results of your search. You can specify these parameters when using `summary` as the `/search` API's output type.

Field specifications are a comma-separated list of specified fields to include or exclude in output. Fields to exclude are preceded by a `-` sign.) Field specifications are the following:

- `voci` - fields that are specific to audio data that has been transcribed by V-Spark software. This is the default behavior if no value for the `fields` option is specified. These are:
 - `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, and having one of the following values: `Positive`, `Worsening`, `Improving`, or `Negative`
 - `agent_gender` - gender of the agent, one of `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, and having one of the following values: `Positive`, `Worsening`, `Improving`, or `Negative`
 - `client_gender` - identification of the gender of the client, one of `Male` or `Female`
 - `datetime` - transcript date and time, expressed in Coordinated Universal Time (UTC)
 - `diarization` - only provided in 2 speaker, 1 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` - duration of the call
 - `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 in which the search request was matched
 - `folder` - name of the folder where the file was uploaded. *Not included when `voci` is specified as return field.*

- `overall_emotion` - overall emotional intelligence assessment for a call, derived from both acoustic and linguistic information, and having 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 transcription 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 would be 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.*
- `CLIENT-DATA` - custom metadata fields, specified by name, that have been entered within the folder with which your audio files are associated
- `all` - get all `voci` and `CLIENT-DATA` fields in the audio files in the specified folder or in all folders associated with an organization
- `apps|app.APPNAME[.TOP CATEGORY[.LOWER CATEGORY...]] - APPNAME` specifies the name of the application that you want to get scores for. Optionally, get scores for a specific category by specifying the category's fullname, which includes its parents' names. For example, `TOP CATEGORY.LOWER CATEGORY`. To get scores for all applications, specify `apps`.

7.1.5. Output Sorting Options

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 `sort FIELD`, `score` is *always* used as a secondary sort option. The default `sort` value is `datetime`. The following Voci `FIELD` are available sort options:
 - `agent_clarity`
 - `agent_emotion`
 - `agent_gender`

- `client_clarity`
 - `client_emotion`
 - `client_gender`
 - `datetime`
 - `diarization`
 - `duration`
 - `filename`
 - `overall_emotion`
 - `overtalk`
 - `score`
 - `silence`
- `sortdir=asc | desc` - specifies whether entries should be sorted in `asc` (ascending) or `desc` (descending) order, based on their data type. The default `sortdir` value is `desc`.

7.2. Sample JSON Output for a query from the /search API

The following is an excerpt from the output of a call to the `/search` API, showing a file that matches the query that was submitted, and which shows the default fields that are returned in a response:

```
[
  {
    "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": {}
  },...
]
```

7.3. Using the /search API with cURL

The `cURL` utility makes it easy to test using the V-Spark API by providing a command-line mechanism for invoking APIs such as the `/config` API. The next few sections provide examples of using the `GET`, `POST`, and `DELETE` verbs with a V-Spark API from the command-line via the `cURL` command.

If you are unfamiliar with the cURL command, see [Section 1.3, “Using cURL for REST API Testing”](#) for a short introduction and an explanation of how cURL examples are displayed. See [Section 1.3.2, “Tips for Debugging and Managing cURL Calls”](#) for suggestions about how to debug and manage cURL calls.

The following is a cURL example that shows calling the `/search` API using the GET method, which supplies the `/search` parameters as options on the URL:

```
curl http://example.company.com:3000/search/Test/Test-Testing/Test01? \
  token=0123456789abcdef0123456789abcdef&duration=4.5-5.50
```

The following is a cURL example that shows calling the `/search` API using the POST method, which supplies the `/search` parameters in a JSON file:

```
curl -F token=0123456789abcdef0123456789abcdef \
  -F 'file=@search-params.json' \
  -H 'Content-type: application/json' \
  -X POST http://example.company.com:3000/search/Test/Test-Testing/Test01
```

The JSON file containing the parameters for this call looks like the following:

```
{
  "output": "summary",
  "client_emotion": "positive"
}
```

To return just the number of matching items that were found, you could simply modify the `search-params.json` file to contain the following:

```
{
  "output": "count",
  "client_emotion": "positive"
}
```

Specifying `count` as the value of the `output` parameter was explained in [Section 7.1.1, “Output Type Options”](#). Being able to programmatically retrieve both matching data and the number of matching data items requires two API calls, but makes it easy for you to use the `count` as an iterator for processing the matching `summary` data.

7.4. Using the `/search` API with Python

You can also use the `/search` API using either GET or POST from programming languages such as Python. The next sections provide sample Python code for a simple application that uses each of these HTTP verbs.

7.4.1. Using the `/search` API via GET with Python

Figures [Figure 7.1](#) and [Figure 7.2](#) show a sample application that uses the `/search` API's GET method to search the folders associated with a specified company (passed as a parameter) in a V-Spark installation and saves matching results to a file. Whenever matching items are found, the application calls the API with the output type set to `count` to identify how many matches were found.

```

#!/usr/bin/env python
#
# Copyright 2017 Voci Technologies, Inc. All rights reserved.
# Contains confidential company information.
# 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): ❶
    if len(argv) != 5: usage(argv) ❷
    host, token, company, searchparams = argv[1:]
    tokens = gettokens(host,token)
    folderinfo = getfolderinfo(host,token)
    findfolders(host, folderinfo, tokens, company, searchparams)

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

def findfolders(host, folder_info, tokens, company, searchparams): ❺
    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)

```

Figure 7.1. Sample Python Code to search for Audio using GET, Part 1

The major steps in the portion of the Python application shown in [Figure 7.1](#), “[Sample Python Code to search for Audio using GET, Part 1](#)” are the following:

- ❶ The main function provides a traditional main routine that shows the order in which functions are called in the application
- ❷ Check if the right number of command-line arguments have been provided, assign them to appropriate variables if so and identifying the expected arguments if not.
- ❸ Uses the `/config` API to retrieve the company information from the V-Spark installation and build a dictionary that only contains the company name and associated token information from the host that was specified on the command-line
- ❹ Uses the `/config/folders` API to retrieve the folder-level configuration information from the host that was specified on the command-line
- ❺ Initiates the primary loop for the application, which is controlled by the companies that were found in the information that was retrieved from the host specified on the command-line. Each company has an associated authorization token (originally stored in the `uuid` name/value pair), which is the other field for each company entry in the dictionary that was constructed in the `gettokens()` function. The short name for each company is the data item in the company JSON that provides the linkage between the data from the company and folder sources. This loop then uses this information to search for organization/folder data that is associated with the company whose name was specified on the command-line, calling the function that represents the core functionality of this application (`searchandprintresults`) for each organization and folder.

```

def searchandprintresults(host, token, comp, org, folder, searchparams):
    url = "http://%s/search/%s/%s/%s?token=%s" % (host, comp, org, folder, token) ❶
    response = requests.get(url)
    if response.status_code == 200: ❷
        print " URL is " +url
        counturl = url+"&output=count"
        countresponse = requests.get(counturl) ❸
        OUTPUT_FILE = comp+"-"+org+"-"+folder+"-search.json"
        print " Writing Matching JSON for "+countresponse.text+" matches to "+OUTPUT_FILE
        target = open(OUTPUT_FILE, 'w') ❹
        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)

```

Figure 7.2. Sample Python Code to search for Audio using GET, Part 2

The major steps in the remainder of the Python application, shown in [Figure 7.2](#), “[Sample Python Code to search for Audio using GET, Part 2](#)”, are the following:

- ❶ Assembles the URL that will be called with the GET method, and then calls that URL.
- ❷ Tests each folder for audio that matches the search parameters that were specified on the command-line, and tests the result of the HTTP call to the REST API to determine if the search was successful.
- ❸ If the search was successful, the application calls the same URL, appending the `output=count` parameter in order to retrieve the number of matches found. This number is used in an informative message.
- ❹ If the search was successfully, the application also saves the matching search results to a file whose name is made up of the company, organization, and folder in which matching results were found.

The following is an example of executing this application, assuming that the code shown in [Figures Figure 7.1](#) and [Figure 7.2](#) was concatenated and saved to an executable file named `search-get-searches.py`:

```

./search-get-searches.py vspark.example.com:3000 1656744ac845cbe185d1a50a0225d7ac \
DocTestCo '&client_emotion=positive'

```

Output from executing this application depends on a V-Spark installation: the company that you are running it against and the folder data that is associated with that company. That output might look something like the following:

```

Searching folders under DocTestCo (Token: d457aa9c65a602254e9810c8d08025ad)
URL is http://vspark.example.com:3000/search/DocTestCo/DocTestCo-DocTesting/Test01?
token=d457aa9c65a602254e9810c8d08025ad&client_emotion=positive
Writing Matching JSON for 4 matches to DocTestCo-DocTestCo-DocTesting-Test01-log.json

```

7.4.2. Using the /search API via POST with Python

[Figures Figure 7.3](#) and [Figure 7.4](#) show an application with exactly the same functionality as the application shown in [Figures Figure 7.1](#) and [Figure 7.2](#), but using the `/search` API's POST method rather than the GET method that was used in [Figures Figure 7.1](#) and [Figure 7.2](#). In this application a JSON file containing the `/search` parameters is passed as a command-line argument rather than the parameters themselves. As this example shows, reading parameters from a JSON file make it easy to programmatically modify those parameters if you need to issue the same call in a slightly different fashion.

```

#!/usr/bin/env python
#
# Copyright 2017 Voci Technologies, Inc. All rights reserved.
# Contains confidential company information.
# 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): ❶
    if len(argv) != 5: usage(argv) ❷
    host, token, company, searchparamfile = argv[1:]
    tokens = gettokens(host,token)
    folderinfo = getfolderinfo(host,token)
    findfolders(host, folderinfo, tokens, company, searchparamfile)

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

def findfolders(host, folder_info, tokens, company, searchparamfile): ❺
    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)

```

Figure 7.3. Sample Python Code to Search for Audio using POST, Part 1

The major steps in the portion of the Python application shown in [Figure 7.3, “ Sample Python Code to Search for Audio using POST, Part 1 ”](#) are the following:

- ❶ The main function provides a traditional main routine that shows the order in which functions are called in the application
- ❷ Check if the right number of command-line arguments have been provided, assign them to appropriate variables if so and identifying the expected arguments if not.
- ❸ Uses the `/config` API to retrieve the company information from the V-Spark installation and build a dictionary that only contains the company name and associated token information from the host that was specified on the command-line
- ❹ Uses the `/config/folders` API to retrieve the folder-level configuration information from the host that was specified on the command-line
- ❺ Initiates the primary loop for the application, which is controlled by the companies that were found in the information that was retrieved from the host specified on the command-line. Each company has an associated authorization token (originally stored in the `uuid` name/value pair), which is the other field for each company entry in the dictionary that was constructed in the `gettokens()` function. The short name for each company is the data item in the company JSON that provides the linkage between the data from the company and folder sources. This loop then uses this information to search for organization/folder data that is associated with the company whose name was specified on the command-line, calling the function that represents the core functionality of this application (`searchandprintresults`) for each organization and folder.


```

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: ❶
        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: ❷
        print " URL is "+url

        param_data["output"] = "count".decode('utf-8') ❸
        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: ❹
            localfile.write(response.content)
        localfile.close()

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

```

Figure 7.4. Sample Python Code to Search for Audio using POST, Part 2

The major steps in the remainder of this sample Python application, shown in [Figure 7.4, “ Sample Python Code to Search for Audio using POST, Part 2 ”](#), are the following:

- ❶ Assembles the URL that will be called with the POST method, reads in the JSON file that contains the parameters with which you want to call the API, sets the correct header value that is required to identify the type of data that you are passing to the POST call, and then calls that URL.
- ❷ Tests each folder for audio that matches the search parameters that were specified on the command-line, and tests the result of the HTTP call to the REST API to determine if the search was successful.
- ❸ If the search was successful, the application modifies the in-memory representation of the JSON file to specify that the next call that uses that data will request the number of matching results rather than the matching data. This number is used in an informative message.
- ❹ If the search was successful, the application also saves the matching search results to a file whose name is made up of the company, organization, and folder in which matching results were found.

The following is an example of executing this application, assuming that the code shown in [Figures Figure 7.3 and Figure 7.4](#) was concatenated and saved to an executable file named `search-post-searches.py`:

```

./search-post-searches.py vspark.example.com-ng:3000 1656744ac845cbe185d1a50a0225d7ac \
DocTestCo summary-and-client-emotion.json

```

Output from executing an application depends on a V-Spark installation: the company that you are running it against and the folder data that is associated with that company. That output might look something like the following:

```

Searching folders under DocTestCo (Token: d457aa9c65a602254e9810c8d08025ad)
URL is http://vspark.example.com:3000/search/DocTestCo/DocTestCo-DocTesting/Test01?
token=d457aa9c65a602254e9810c8d08025ad
Writing Matching JSON for 4 matches to DocTestCo-DocTestCo-DocTesting-Test01-search.json

```


Chapter 8. Retrieving Folder and Application Statistics Information

This section describes V-Spark's APIs that enable you to programmatically check and retrieve folder statistics (`/stats`) and statistics for agent application and category scores (`/appstats`).

8.1. Retrieving Folder Statistics

The `/stats` API enables you to retrieve daily statistics for folders. The next few sections provide reference information for this API, and examples of calling this API from the [Section 8.1.3, “Using the /stats API with cURL”](#) and [Section 8.1.4, “Using the /stats API with Python”](#).

8.1.1. Reference for the /stats API

The `/stats` API enables you to retrieve daily statistics for folders by specifying a date or date range for which you want to retrieve information. Statistics are returned in JSON format, include call volume and average call duration, and also includes agent information if calls include agent id metadata.

Synopsis

```
GET /stats/CO_SHORT/ORG_SHORT?token=TOKEN&OPTIONS...
GET /stats/CO_SHORT/ORG_SHORT/FOLDERNAME?token=TOKEN&OPTIONS...
```

Description

Variables used in the URL of a call to the `/stats` API are the following:

- `CO_SHORT`: the short name of the company whose statistics you would like to retrieve
- `ORG_SHORT`: the short name of the organization that you are interested in. Finding that information is shown in [Figure 4.1, “Location of the V-Spark Organization Short Name”](#).
- `FOLDERNAME`: the name of the V-Spark Folder whose statistics you would like to retrieve. If you do not specify the name of a folder, matching statistics for all folders under the specified `ORG_SHORT` will be returned.
- `TOKEN`: the V-Spark authorization token that you are using to establish permission to retrieve information. You can either use the root token for the target V-Spark installation (located in the file `/opt/voci/state/vspark/apitoken`) or the authorization token for the company under which the specified `ORG_SHORT` is located. Locating a company's authorization token is shown in [Figure 1.1, “Location of a Company Authorization Token”](#).

Options

The following options can be passed as parameters to calls to the `/stats` API:

- `daterange=START-END` - Enables you to specify a date range for daily stats. The `START` and `END` values are both expressed as `YYYYMMDD` values, where the year (`YYYY`) month (`MM`) and day (`DD`) values are required. Date ranges are always assumed to be positive (where `START` is less than `END`).

No verification is done to ensure that this is correct. Invalid date ranges will simply return no values. If *START* is not specified, the default value is today's date. If *END* is not specified, the start date is used, and only 1 day of stats will be returned. If this option is not specified, today's date is used.



Important

No information is returned for dates in the specified range that do NOT contain any calls.

8.1.2. Sample JSON from the /stats API

Figure 8.1, “Sample Folder Statistics Output from the /stats API” shows an excerpt of the output that is produced by a call to the /stats API, specifying a folder to which audio files have been uploaded and processed on a specified date (or the current date if the date parameter was not specified).

```
[
  {
    "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
      }
    }
  }
]
```

Figure 8.1. Sample Folder Statistics Output from the /stats API

8.1.3. Using the /stats API with cURL

This section discusses how to use the /stats API to retrieve high-level folder statistics information from a specified V-Spark installation.

If you are unfamiliar with the cURL command, see [Section 1.3, “Using cURL for REST API Testing”](#) for a short introduction and an explanation of how cURL examples are displayed. See [Section 1.3.2, “Tips for Debugging and Managing cURL Calls”](#) for suggestions about how to debug and manage cURL calls.

An example of a cURL command to retrieve daily stats information from September 02, 2017 (20170902) from the company "DocTestCo", organization "DocTestCo-DocTesting", folder "Test01" on the host 192.168.6.64 where V-Spark is listening on port 3000, is the following:

```
curl -s 'http://192.168.6.64:3000/stats/DocTestCo/DocTestCo-DocTesting/Test01\
?token=01234567890123456789012345678901&daterange=20170925'
```

To produce this output in a pretty-printed form that is more usable, and write that output to the file stats.json, you could execute a command like the following:

```
curl -s 'http://192.168.6.64:3000/stats/DocTestCo/DocTestCo-DocTesting/Test01 \
?token=01234567890123456789012345678901&daterange=20170605' | \
python -m json.tool > stats.json
```



Tip

The `json.tool` module that is provided by Python sorts keys in JSON output alphabetically. If you want to pretty-print JSON output without reorganizing key values, you may want to use a Python command such as **jsonlint**, which includes pretty-printing along with other capabilities and is provided as part of the `python-demjson-1.6-1.el6.noarch` package on CentOS systems.

Example output from the previous command would look something like the following:

```
[
  {
    "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
      }
    }
  }
]
```

8.1.4. Using the /stats API with Python

Figure 8.2, “Sample Python Code for Retrieving Folder Statistics from the /stats API” shows a generic Python application that can be used to call any V-Spark API. While you will probably want to develop more specialized Python applications to use specific V-Spark APIs directly, this sample code shows the basic mechanisms that make it easy to interact with the /stats API from Python. This sample application enables you to pass both the API that you want to call and the parameters that you want to provide. In this case, the parameter is the date or the date range that you want to retrieve statistics for, based on the folder that is specified in the previous argument.

The arguments to the sample application shown in Figure 8.2 are the following:

- *HOST* - the name of the host that is running V-Spark
- *TOKEN*: the V-Spark authorization token that you are using to establish permission to retrieve information. You can either use the root token for the target V-Spark installation (located in the file /opt/voci/state/vspark/apitoken) or the authorization token for the company that is associated with the application that you are working with. Locating a company's authorization token is shown in Figure 1.1, “Location of a Company Authorization Token”.
- *API_TO_CALL* - the API that you want to call, along with the *CO_SHORT*, *ORG_SHORT*, and *FOLDER* that you want to retrieve statistics about
- *PARAMS* - the parameters that you want to pass to your call to the stats API. For this sample application, the parameters that you pass should be enclosed within single quotation marks so that the Linux shell does not attempt to interpret them.

```

#!/usr/bin/env python

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:] ❶

# Define the URL in a single variable for JSON load ❷
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: ❸
    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( ❹
    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)) ❺
target.close()
print " Output written to: "+tmp_str[1:]

```

Figure 8.2. Sample Python Code for Retrieving Folder Statistics from the /stats API

The core functionality of this application is the following:

- ❶ If the previous test showed that the right number of command-line arguments were provided, assign the command-line arguments to the relevant variables
- ❷ Assemble the URL that you want to call from the parameters that were supplied on the command-line and a few internal default settings
- ❸ Make the API call using the URL that you assembled, convert the object that it returned into JSON, and catch any exception that is returned. If an exception was raised, display the associated HTTP error message and status code that was returned before exiting,
- ❹ Build the name of the output file to which you want to write the JSON that was returned by the API call. This code is present to avoid constructing filenames which contain special characters that have other implications on a Linux system. Note that when this output file name is used in the following `open()` statement, the first character in its name is skipped because that character is a safe conversion of the leading `'/'` in the name of the API.
- ❺ Write the JSON object to the output file to which you want to write the JSON that was returned by the API call, using standard formatting parameters like indenting each entry by four spaces and sorting the entries by key. These enable a Python application to produce output that is already pretty-printed.

An example of executing this application is the following:

```

statistics-get-info.py HOST TOKEN /stats/DocTestCo/DocTestCo-DocTesting/Test01 \
'&daterange=20170901-20171011'

```

This call to the example function would retrieve statistics about activity in the folder `DocTestCo/DocTestCo-DocTesting/Test01`, for the date range `20170901-20171011` and write it to an output file named `stats_DocTestCo_DocTestCo-DocTesting_Test01_daterange=20170901-20171011.json`.

Sample output from this call would look like the following:

```
[
  {
    "agent": {
      "avgcalls": "4.0",
      "emotion": {
        "improving": 4,
        "negative": 0,
        "positive": 0,
        "worsening": 0
      },
      "talk": {
        "avg": "0:02:11",
        "max": {
          "duration": "0:08:44",
          "id": "105"
        },
        "min": {
          "duration": "0:08:44",
          "id": "105"
        }
      }
    },
    "avgduration": "0:05:25",
    "avgsilence": "0:02:16",
    "avgwords": "643.0",
    "calls": 4,
    "client": {
      "emotion": {
        "improving": 0,
        "negative": 4,
        "positive": 0,
        "worsening": 0
      },
      "talk": {
        "avg": "0:00:58"
      }
    },
    "date": "20170925"
  }, ...
]
```

This sample output has been abbreviated to save space.

8.2. Retrieving Agent Application Statistics and Category Scores

The `/appstats` API enables you to retrieve daily statistics for agent application and category scores. The next few sections provide reference information for this API, and examples of calling this API from the [Section 8.2.3](#), “Using the `/appstats` API with `cURL`” and [Section 8.2.4](#), “Using the `/appstats` API with Python”.

8.2.1. Reference for the `/appstats` API

The `/appstats` API enables you to retrieve daily statistics for agent application and category scores. These statistics are returned in JSON format.

A number of optional parameters are available to allow you to retrieve specific category scores or a select group of agent scores.

Synopsis

```
GET /appstats/CO_SHORT/ORG_SHORT/APPNAME?token=TOKEN&OPTIONS...
GET /appstats/CO_SHORT/ORG_SHORT/APPNAME/FOLDERNAME?token=TOKEN&OPTIONS...
```

Description

Variables used in a call to the `/appstats` API are the following:

- `CO_SHORT`: the short name of the company whose statistics you would like to retrieve
- `ORG_SHORT`: the short name of the organization that you are interested in. Finding that information is shown in [Figure 4.1, “Location of the V-Spark Organization Short Name”](#).
- `APPNAME`: the name of the V-Spark application whose agent scores you would like to retrieve.
- `FOLDERNAME`: the name of the V-Spark Folder whose statistics you would like to retrieve
- `TOKEN`: the V-Spark authorization token that you are using to establish permission to retrieve information. You can either use the root token for the target V-Spark installation (located in the file `/opt/voci/state/vspark/apitoken`) or the authorization token for the company under which the specified `ORG_SHORT` is located. Locating a company's authorization token is shown in [Figure 1.1, “Location of a Company Authorization Token”](#).

Options

- `daterange=START-END` - Enables you to specify a date range for daily stats. The `START` and `END` values, though optional, are both expressed as `YYYYMMDD` values where the year (`YYYY`) month (`MM`) and day (`DD`) values are required. Date ranges are always assumed to be positive (where `START` is less than `END`). No verification is done to ensure that this is correct. Invalid date ranges will simply return no values. If `START` is not specified, the default value is today's date. If `END` is not specified, the start date is used, and only 1 day of stats will be returned.



Important

No information is returned for dates in the specified range that do NOT contain any calls.

- `category=CATEGORY[.LOWER-LEVEL CATEGORY...]` - Enables you to return Agent Stats for a particular category. The category name should be a string that also contains the category's upper level categories. 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 - (default value if category option is specified) Return only the level of the category specified.
 - *positive non-zero integer* - Return the specified number of lower levels of the category
 - -1 - (default value if no category option is specified) Return all levels of category stats
- `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, each `AGENTID` must be separated from the next by a comma.
- `zeros=true|false` - (default is false) - Include zero scores in the JSON that is returned for categories in which the agent did not 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.

8.2.2. Sample JSON from the /appstats API

Figure 8.3, “Sample Application Statistics and Category Score Output from the /appstats API” shows an excerpt of the output that is produced by a call to the /appstats API, specifying an application that has been run against audio files have been processed or reprocessed on a specified date (or the current date if the date parameter was not specified).

```
[
  {
    "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"
      },...
    },...
  },...
]
```

Figure 8.3. Sample Application Statistics and Category Score Output from the /appstats API

8.2.3. Using the /appstats API with cURL

This topic discusses how to retrieve application statistics information from a V-Spark installation. For each agent, the output will return the number of calls the agent had and the overall app scores, along with any category scores that were specified using category and depth parameters.

See the *Using the Agents View* section of the "V-Spark 3.4.3 Management Guide" for more information about the category statistics that are returned.

If you are unfamiliar with the cURL command, see [Section 1.3, “Using cURL for REST API Testing”](#) for a short introduction and an explanation of how cURL examples are displayed. See [Section 1.3.2, “Tips for Debugging and Managing cURL Calls”](#) for suggestions about how to debug and manage cURL calls.

An example of a cURL command to retrieve application scores from the `Agent ScoreCard` application on September 25, 2017, which is associated with the company "DocTestCo" and organization "DocTestCo-DocTesting" on the host 192.168.6.64 where V-Spark is listening on port 3000, is the following:

```
curl -s 'http://192.168.6.64:3000/appstats/DocTestCo/DocTestCo-DocTesting/Agent%20Scorecard \
?token=01234567890123456789012345678901&daterange=20170925'
```

The format of the JSON output that is returned by this call is the following:

```
[
  {
    "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"
      },...
    },...
  }
]
```

The contents of this example have been abbreviated to save space. Each matching agent is identified numerically (001 in the previous example), with each category for which scores are available listed under the call identifier.

An example of a cURL command to retrieve category scores for the `Communication Skills.Client Informed` category from the `Agent ScoreCard` application on September 25, 2017, which is associated with the company "DocTestCo" and organization "DocTestCo-DocTesting" on the host 192.168.6.64 where V-Spark is listening on port 3000, is the following:

```
curl -s 'http://192.168.6.64:3000/appstats/DocTestCo/DocTestCo-DocTesting/Agent%20Scorecard\
?token=01234567890123456789012345678901&daterange=20170925\
&category=Communication%20Skills.Client%20Informed&depth=1'
```

As mentioned earlier, requesting category information for a subcategory also returns information about its parent categories. In this example, requesting scores for `Communication Skills.Client Informed` automatically also returns scores for the `Communication Skills` category. Since a depth of "1" is specified, the call also returns the specified number of lower levels of the `Client Informed` category. In this case, there is only one such lower-level category: `Communication Skills.Client Informed.Agent Actions`.

The format of the JSON output that is returned by this call is the following:

```
[
  {
    "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"
    },
    },
    "Communication Skills.Client Informed": {
      "ncalls": 1,
      "hitmiss": "1.0000",
      "coverage": "0.3333",
      "duration": "0:11:55",
      "silence": "0:03:13"
    },
    },
    "Communication Skills.Client Informed.Agent Actions": {
      "ncalls": 1,
      "hitmiss": "1.0000",
      "coverage": "1.0000",
      "duration": "0:11:55",
      "silence": "0:03:13"
    },
    }
  }, ...
]

```

You can further refine the output of a call to the `/appstats` API by restricting the category to a lower-level one that does not contain subcategories, as in the following example:

```

statistics-get-info.py HOST TOKEN \
  /appstats/DocTestCo/DocTestCo-DocTesting/Clone%20Test%2002 \
  'daterange=20171005&category=Politeness'

```

This call uses the `daterange` parameter to limit results to those from files processed on a single date (05 Oct 2017), and only lists specific and summary (overall) information about the category, `Politeness`, which has no subcategories:

```

[
  {
    "105": {
      "Politeness": {
        "coverage": "0.5714",
        "duration": "0:05:25",
        "hitmiss": "1.0000",
        "ncalls": 19,
        "silence": "0:02:19"
      },
      "calls": 19,
      "overall": {
        "coverage": "0.3293",
        "duration": "0:05:25",
        "hitmiss": "1.0000",
        "ncalls": 19,
        "silence": "0:02:19"
      }
    },
    "agents": 1,
    "date": "20171005"
  }
]

```

8.2.4. Using the `/appstats` API with Python

The same Python example that was provided for calling the `/stats` API, [Figure 8.2, “Sample Python Code for Retrieving Folder Statistics from the `/stats` API”](#), can also be used to call the `/appstats` API. This generic Python application can be used to call any V-Spark API. While you will probably want to develop more specialized Python applications to use specific V-Spark APIs directly, the sample code shows the basic mechanisms that make it easy to interact with the `/appstats` API from Python.

The sample application provided in [Figure 8.2](#) enables you to specify both the API that you want to call and the parameters that you want to provide as arguments to the Python code. In this case, the parameters

are the daterange that you want to retrieve statistics for and other refinements on matching output, based on the application that is specified as the previous argument.

An example of executing this application is the following:

```
statistics-get-info.py HOST TOKEN \  
  /appstats/DocTestCo/DocTestCo-DocTesting/Clone%20Test%2001 \  
  daterange=20170901-20171015
```

This call to the example function would retrieve statistics about activity for the application *Clone Test 01* that is associated with folders within the company and organization *DocTestCo/DocTestCo-DocTesting*, for the date range 20170901–20171011.

Sample output from this call would look like the following:

```
[  
  {  
    "105": {  
      "Communication Skills": {  
        "coverage": "0.5250",  
        "duration": "0:05:25",  
        "hitmiss": "1.0000",  
        "ncalls": 2,  
        "silence": "0:02:19"  
      },  
      "Communication Skills.Ask for Call Reason": {  
        "coverage": "1.0000",  
        "duration": "0:05:25",  
        "hitmiss": "1.0000",  
        "ncalls": 2,  
        "silence": "0:02:19"  
      },  
      "Communication Skills.Client Informed": {  
        "coverage": "1.0000",  
        "duration": "0:05:25",  
        "hitmiss": "1.0000",  
        "ncalls": 2,  
        "silence": "0:02:19"  
      },  
      "Communication Skills.Client Informed.Agent Actions": {  
        "coverage": "1.0000",  
        "duration": "0:05:25",  
        "hitmiss": "1.0000",  
        "ncalls": 2,  
        "silence": "0:02:19"  
      },  
      ...  
    },  
    "agents": 1,  
    "date": "20171002"  
  },...  
]
```

To see how to call the `/appstats` API, retrieve the output from a call to the API, and write the JSON output to a file, see the sample Python code that is provided in [Figure 8.2](#). The core functionality of the primary steps in this sample application are explained after the sample code is presented.

Chapter 9. Configuring V-Spark Applications

V-Spark applications are customizable analytics tools that are associated with one or more folders. When using the V-Spark GUI, you can download the JSON configuration data that is associated with a custom application by visiting the **Settings** menu's **Applications** page and selecting the **Application Editor** button to the right of the application that you want to edit. This displays the Application Editor dialog, which provides **Download** and **Upload** buttons in the upper right-hand corner. You can download the JSON for your custom application for archival purposes, or modify it using your favorite text editor and re-upload the updated file to integrate your improvements.

The `/appedit` API provides a programmatic mechanism for retrieving the category configuration of a custom application in JSON format. You can then make any modifications that you want to the JSON text which describes the configuration of that application and re-upload it to incorporate those changes into the V-Spark.

9.1. Reference for the `/appedit` API

The `/appedit` API enables you to retrieve the category/phrase configuration of a custom application for archival purposes, or so that you can modify it using an external editor. You can then upload the modified application for use within V-Spark.

Synopsis

```
GET /appedit/CO_SHORT/ORG_SHORT/APPNAME?token=TOKEN
POST /appedit/CO_SHORT/ORG_SHORT/APPNAME?token=TOKEN
```

Description

The variables used in a call to the `/appedit` API are the following:

- `CO_SHORT`: the short name of the company whose category configuration you would like to retrieve or upload
- `ORG_SHORT`: the short name of the organization that you are interested in. Finding that information is shown in [Figure 4.1, “Location of the V-Spark Organization Short Name”](#).
- `APPNAME`: the name of the V-Spark application whose category configuration you would like to retrieve or upload
- `TOKEN`: the V-Spark authorization token that you are using to establish permission to retrieve information. You can either use the root token for the target V-Spark installation (located in the file `/opt/voci/state/vspark/apitoken`) or the authorization token for the company under which the specified `ORG_SHORT` is located. Locating a company's authorization token is shown in [Figure 1.1, “Location of a Company Authorization Token”](#).

9.2. Using the `/appedit` API with cURL

If you are unfamiliar with the cURL command, see [Section 1.3, “Using cURL for REST API Testing”](#) for a short introduction and an explanation of how cURL examples are displayed. See [Section 1.3.2, “Tips for Debugging and Managing cURL Calls”](#) for suggestions about how to debug and manage cURL calls.

Application category configurations can be retrieved and updated using the `/appedit` API. The following examples demonstrates saving an application's category configuration to a JSON file so that it can be edited, either manually or programmatically, and then updated.

For more information about application category configuration, see the "*Additional Features of Custom Applications*" section of the "*V-Spark 3.4.3 Application Development Guide*".

Application category configurations can be retrieved using a GET call to the `/appedit` API. The following command demonstrates retrieving the configuration of an application named "AppEdit Test", located under the *Technologies* company, in the *Technologies-RD* organization. The JSON that is retrieved is written to a JSON file named `AppEdit-Test.json`:

```
curl -s "http://192.168.6.64:3000/appedit/Technologies/Technologies-RD/AppEdit%20Test?token=TOKEN" > AppEdit-Test.json
```



Note

Because the names of applications can contain spaces, you must URL-encode each space in the name of an application by replacing it with `%20`.

Application category configurations can be updated using a POST call to the `/appedit` API. The configuration contain the entire application and therefore require every category to be present in order to preserve that structure. You can not POST a configuration for only one category at a time.

The following command demonstrates uploading the 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://192.168.6.64:3000/appedit/Technologies/Technologies-RD/AppEdit%20Test?token=TOKEN"
```

When using cURL and a command like this one to POST data to a host, the information about the protocol, host, and port should also include the mandatory `token` parameter that ensures that you have rights to access the V-Spark installation to upload information.

You must also use the following cURL options:

- `-X` - identifies the request method to use (POST) when communicating with the target HTTP server
- `-H` - identifies the type of content that you are sending (`"Content-Type:application/json"`).
- `-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).

The cURL command's `-s` command-line argument is optional, causing the cURL command to run in silent mode, where it does not display progress information or error messages.

9.2.1. Creating and Populating an Application Using cURL

While the `/appedit` API only enables you to download or upload the configuration of an existing application, you can combine JSON configuration data and calls to the `/config/apps`, `/appedit`,

and `/config/folders` APIs to create an application, upload its configuration, and associate it with a folder. [Figure 9.1, “Sample JSON that defines an Application”](#) shows sample JSON for an application named `New AppEdit Test`.

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

Figure 9.1. Sample JSON that defines an Application

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. [Figure 9.2, “Sample JSON that Associates an Application with a Folder”](#) shows the JSON that associates a folder with the application that was defined in [Figure 9.1, “Sample JSON that defines an Application”](#).

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

Figure 9.2. Sample JSON that Associates an Application with a Folder

[Figure 9.3, “Sample JSON for an Application”](#) shows an abbreviated version of the JSON configuration of 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": {}
  }
}

```

Figure 9.3. Sample JSON for an Application

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 definite its configuration with three cURL calls like the following:

```

curl -s -X POST -H "Content-Type:application/json" \
  "http://vtorch2:3000/config/apps?token=566af08d0d6ca7436c9117f09571cad" \
  --data @app-definition.json

curl -s -X POST -H "Content-Type:application/json" \
  "http://vtorch2:3000/config/folders?token=566af08d0d6ca7436c9117f09571cad" \
  --data @app-folder-binding.json

curl -s -X POST -H "Content-Type:application/json" \
  "http://vtorch2:3000/appedit/Technologies/Technologies-RD/AppEdit%20Test?
  token=566af08d0d6ca7436c9117f09571cad" \
  --data @app-configuration-sample.json

```

The cURL commands in this example, respectively:

1. Call the `/config/apps` API to create the application, using the contents of the file `app-definition.json`, which is shown in [Figure 9.1](#).
2. Call the `/config/folders` API to associate the new application with a specified folder, using the contents of the file `app-folder-binding.json`, which is shown in [Figure 9.2](#).
3. Call the `/appedit` API to upload the configuration of the application, using the contents of the file `app-configuration-sample.json`, which is shown in [Figure 9.3](#).

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 popular Python programming language.

9.3. Using the /appedit API with Python

Figure 9.4, “Sample Python Application for POST'ing JSON to APIs” shows a generic Python application that can be used to call any V-Spark API. While you will probably want to develop more specialized Python applications to use specific V-Spark APIs directly, this sample code shows the basic mechanisms that make it easy to interact with the V-Spark APIs from Python.

The arguments to the sample application shown in Figure 9.4 are the following:

- *HOST* - the name of the host that is running V-Spark
- *TOKEN*: the V-Spark authorization token that you are using to establish permission to retrieve information. You can either use the root token for the target V-Spark installation (located in the file `/opt/voci/state/vspark/apitoken`) or the authorization token for the company that is associated with the application that you are working with. Locating a company's authorization token is shown in Figure 1.1, “Location of a Company Authorization Token”.
- *API_TO_CALL* - the API to which you want to POST the JSON input
- *INPUT_JSON_FILE* - the JSON file that contains the data that you want to post. If the JSON file is not in the current directory, you must specify a full or relative path to the file.

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

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 "POST'ing 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)
```

Figure 9.4. Sample Python Application for POST'ing JSON to APIs

To perform the same activity as the cURL commands in the previous section and re-use the sample JSON files that were shown in Section 9.2, “Using the /appedit API with cURL”, you could call this application (saved as the file `post-json.py`) three times:

1. `post-json.py host token /config/apps app-create.json`

Calls the `/config/apps` API to create the application that is specified in the JSON shown in Figure 9.1, “Sample JSON that defines an Application”.

2. `post-json.py host token \
/appedit/Technologies/Technologies-RD/New%20AppEdit%20Test appedit-sample.json`

Calls the `/appedit` API to upload the configuration for the application `/Technologies/Technologies-RD/New%20AppEdit%20Test` from the JSON file shown in Figure 9.3, “Sample JSON for an Application”.

3. `post-json.py host token /config/folders app-folder-associate.json`

Calls the `/config/folders` API to associate the application with the folder identified in the JSON shown in [Figure 9.2, “Sample JSON that Associates an Application with a Folder”](#).

Chapter 10. Retrieving System Information

The `/sysinfo` API provides a programmatic mechanism for retrieving (in JSON format) the status and configuration of the V-Spark host and version levels of installed Voci software.

10.1. Reference for the `/sysinfo` API

The `/sysinfo` API enables you to retrieve system status, system configuration, and software version information from the running V-Spark server. The information is returned in JSON format, and can be saved for archival purposes or transmitted to Voci support for diagnostic use.

Synopsis

```
GET /sysinfo
GET /sysinfo?full
```

Description

The call returns basic system information. Using the "full" option returns extended system information.

When calling this API, you must provide the *root authorization token*, which proves that you are authorized to perform system administration operations. For information about the authorization tokens that you can provide for use with the V-Spark API, see [Section 1.2, “V-Spark API Permission Requirements”](#).

10.2. Sample JSON from the `/sysinfo` API

[Figure 10.1, “Sample Basic System Output from the `/sysinfo` API”](#) shows an excerpt of the JSON output that is produced by a basic call to the `/sysinfo` API.

```
{
  "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"
}
```

Figure 10.1. Sample Basic System Output from the `/sysinfo` API

The fields in the JSON output are described as follows:

`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/sdal            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 output:

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

Sample full 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"
],
"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.

hostname (basic, full)

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

The hostname of the server, in this case *analysis*.

```
"hostname": "analysis",
```

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 output:

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

Sample full 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"
```

10.3. Using the /sysinfo API with cURL

If you are unfamiliar with the cURL command, see [Section 1.3, “Using cURL for REST API Testing”](#) for a short introduction and an explanation of how cURL examples are displayed. See [Section 1.3.2, “Tips for Debugging and Managing cURL Calls”](#) for suggestions about how to debug and manage cURL calls.

Retrieve basic system information by using a GET call to the `/sysinfo` API. The following command demonstrates retrieving basic system information from a V-Spark server. The JSON that is retrieved is written to a JSON file named `Sysinfo-Test.json`:

```
curl -s "http://HOSTNAME:3000/sysinfo?token=TOKEN" > Sysinfo-Test.json
```

Retrieve full system information including service status and the version levels of all installed packages by using the `full` switch on a GET call to the `/sysinfo` API. The following command demonstrates retrieving full system information from a V-Spark server. The JSON that is retrieved is written to a JSON file named `Sysinfo-Full-Test.json`:

```
curl -s "http://HOSTNAME:3000/sysinfo?full&token=TOKEN" > Sysinfo-Full-Test.json
```


Appendix A. Sample transcribe/request API Shell Script

This appendix 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 &quot;TEST: Running API tests for transcribe/result mode...&quot;
echo &quot;&quot;
#
if [ $# != 2 ] ; then
    echo &quot;Usage: $0 host token&quot;
    exit
fi

SERVER=$1
TOKEN=$2

# Uncomment the value of FILE that you want to test with: the
# &quot;standard&quot; version with a file extension, the &quot;guess what kind of
# file&quot; 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
# &quot;cp&quot; 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 $SAMPLEFILE ] ; then
    echo &quot; Specified upload ($SAMPLEFILE) does not exist!&quot;
    exit
fi

SHORTORG=DocTestCo-DocTesting
FOLDER=Test01

DEBUG=&quot;--trace-ascii debug.out&quot;

echo -n &quot;Type of file to upload is: &quot;
file $SAMPLEFILE

echo &quot;&quot;
echo &quot;Submitting $SAMPLEFILE for transcription:&quot;

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

```

echo &quot;CMD is $CMD&quot;
echo &quot;;&quot;

echo &quot; SUMMARY: Uploading $SAMPLEFILE for transcription at approximately $(date)&quot;

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

# REQUESTID=`$CMD`

echo &quot;;&quot;

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

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

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

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

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

echo $REQUESTID &gt; .REQUESTID

echo &quot; SUMMARY: Transcription completed successfully at $(date)&quot;

```

Appendix B. Possible Error Codes from the V-Spark API

B.1. Possible Error Codes from the /transcribe API

The `transcribe` API returns HTTP error codes when called with incorrect or invalid parameters:

- `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. Locating your authorization token is shown in [Figure 1.1, "Location of a Company Authorization Token"](#).
- `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 250 MB` -
- `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.

B.2. Possible Error Codes from the /request API

The `request` API returns HTTP error codes when called with incorrect or invalid parameters:

- 400 - the `request` API 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. Locating your authorization token is shown in [Figure 1.1, “Location of a Company Authorization Token”](#).
- 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 `txt`)
 - `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.

B.3. Possible Error Codes from the /config/folders API

The `config/folders` API returns HTTP error codes when called with incorrect or invalid parameters:

- 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.

B.4. General Error Codes from the V-Spark APIs

V-Spark APIs other than the `/request` and `/transcribe` APIs return HTTP error messages when called with incorrect, invalid, or missing parameters:

- 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 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.

