Kiwa Digital Ltd.

Microsoft Excel | Cloud API | User Guide

This guide provides information to help you get started and understand VoiceQ integration with Microsoft Excel.

Updated May 2025

INTRODUCTION

Overview

VoiceQ Cloud excels at managing voiceover projects, while Microsoft Excel provides a robust document management and collaboration platform. Integrating these two platforms allows users to combine their strengths, facilitating efficient data transfer and automation of various workflows.

System Requirements

To seamlessly integrate Microsoft Excel with VoiceQ Cloud, users will need:

- A Microsoft 365 account with access to Microsoft Excel Online and VoiceQ Cloud.
- Basic understanding of Microsoft Excel document libraries and lists.
- Microsoft Power Automate experience or a willingness to learn.
- A stable internet connection.

Key benefits of this integration could include:

Automated Data Transfer: Send project details or voiceover files from VoiceQ Cloud to Microsoft Excel for centralized storage.

Enhanced Collaboration: Enable teams to review and provide feedback on voiceover projects directly within Microsoft Excel.

Streamlined Workflows: Trigger actions in Microsoft Excel (like approvals or notifications) based on events in VoiceQ Cloud.

SETTING UP MICROSOFT EXCEL (EXCEL)

Before integrating with VoiceQ Cloud, users should ensure that their Microsoft Excel (Excel) sheet is appropriately set up to accommodate the data exchange. Users must create a new document:

- Open Microsoft 365 and select Excel in the applications list.
- **Character data:** Once you have a blank sheet, you must download the script: Download here
- **Script data:** Once you have a blank sheet, you must download the script: Download here

INTEGRATING VOICEQ CLOUD WITH MICROSOFT EXCEL (CHARACTERS EXPORT)

This section will guide users through connecting VoiceQ Cloud with Microsoft Excel. It will cover steps such as creating the script, copying the authorization token, authorizing access, and establishing communication between the two platforms.

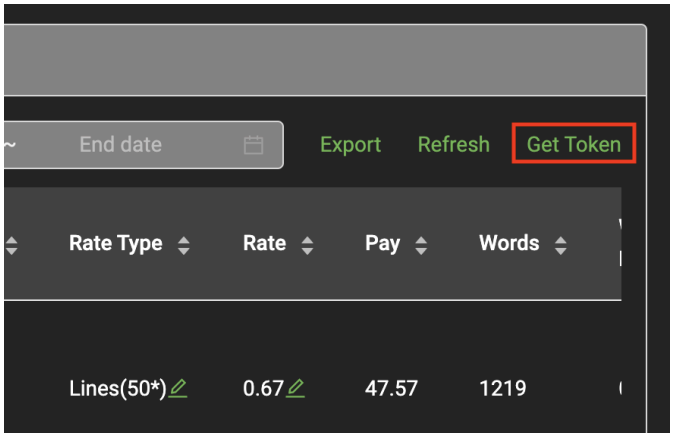
Template for API sheet (Characters)

We have supplied a public link to download the API template, which includes a working document and a Refresh script. Raw code can be found in ‘Appendix 1’

Download the template

Connecting to VoiceQ Cloud

1. Begin by opening the VoiceQ Cloud project to which you want to connect.
2. Locate and copy the project ID in the project URL. It typically appears as a string of characters, such as **'11e9f6a8-5fa8-fe32-a8bd-b71835bxxxxx'.**
3. Paste the ID into Cell A1 - **INSERT_PROJECT_ID**
4. Reopen the VoiceQ Cloud project.
5. Locate the 'Get Token' button positioned above the Characters section.
6. Click on the 'Get Token' button to generate a token.



7. Copy the token provided and paste it into Cell A2 - **INSERT_USER_TOKEN**

	A	B	C	D	E
1	11ee5caf-8871-1a91-82a1-7b3777c123de				
2	Bearer eyJhbGciOiJIUzUxMiJ9.eyJjaCI6ImViNDQ0MzVlYTJlMWFiOGM4YmJiOGYzOTVmMjQ2ZjhiMzY0Y2U				
3					
4	Name	Actor Name	Actor	List Id	Lines
5	SPEAKER 4, SPEAKER				2
6	SPEAKER 6(M30), SPEAKER 12(M28)				2
7	SPEAKER 14				16
8	SPEAKER 15				5
9	SPEAKER 13				2
10	SPEAKER, SPEAKER 11				1
11	SPEAKER 11	Johnny DEPP		1015	5
12	SPEAKER	Robert DOWNEY		1005	50
13	SPEAKER 13, SPEAKER				1
14	SPEAKER 12(M28), SPEAKER 4				1
15	SPEAKER 11, SPEAKER 12(M28)				1
16	SPEAKER 14, SPEAKER 15				1

ADVANCED CONFIGURATION

Configuring Automate Scripts

Microsoft Power Automate serves as the connector between Microsoft Excel and VoiceQ Cloud. To configure this integration, follow these steps:

1. Download and import the script labeled “Resfresh.osts” - You may do this by navigating to the ‘Automate’ tab and selecting “New Script.”
2. Add the script to the Code Editor and select ‘Run’ to test.

Note: If the option to add or import a script is absent, you must create a new one and then import it.

RUNNING SCRIPTS AND AUTOMATING TASKS

Once the integration is complete and scripts are configured, users can execute them to perform various actions automatically. This section will explain how to run scripts manually and set up triggers for automation based on specific events or schedules.

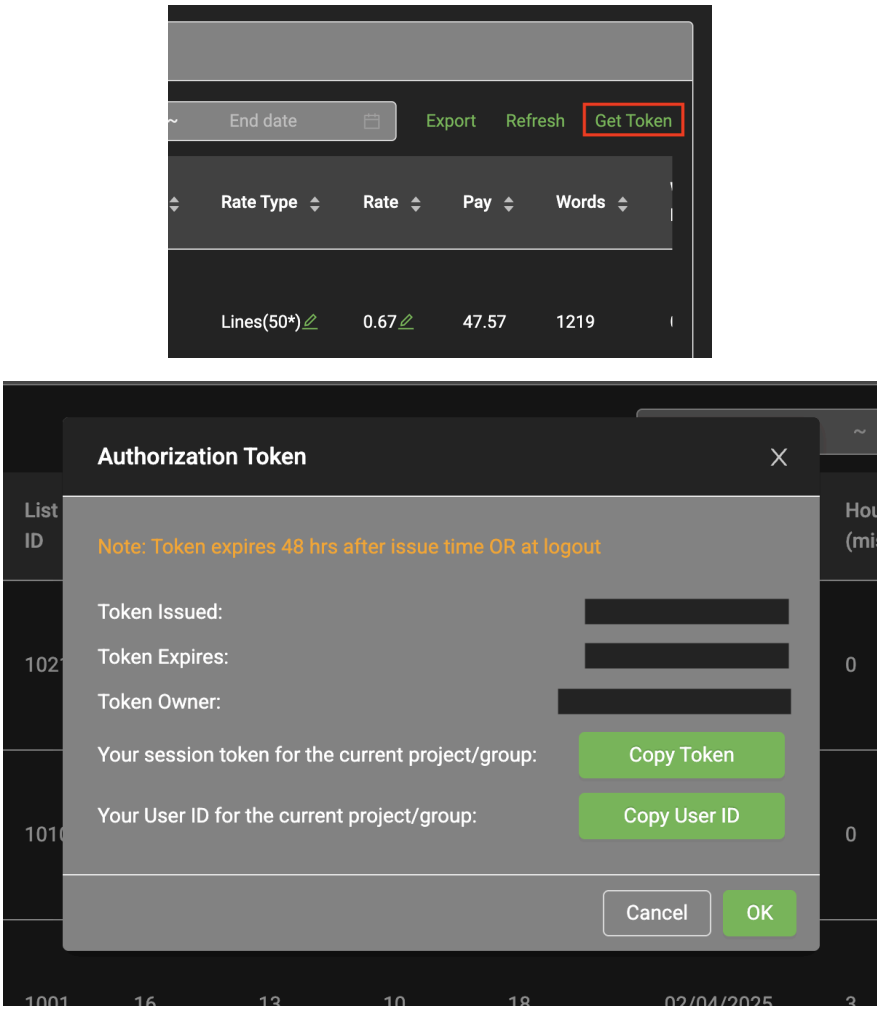
Connecting to VoiceQ Cloud

8. Begin by opening the VoiceQ Cloud project to which you want to connect.

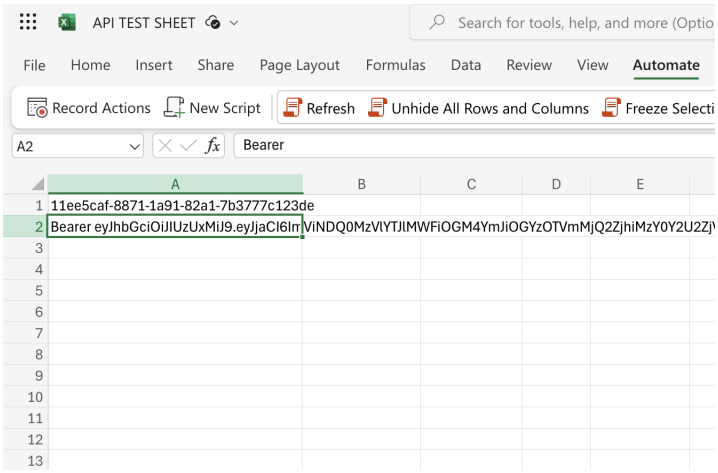
9. In the project URL, locate and copy the Project ID. It typically appears as a string of characters at the end of your URL, for example, “app.voiceqcloud.com/app/#/projects/**YOUR_PROJECT_ID**”
10. Paste this Project ID into Cell A1 of your Google Sheets document.

Obtaining and Using the Token

11. Reopen the VoiceQ Cloud project.
12. Locate the 'Get Token' button positioned above the Characters section.
13. Click on the 'Get Token' button to generate a token.



14. Press ‘Copy Token’
15. Open Microsoft Excel and paste the token into Cell A2 of your document.



Configuring Script Execution

16. Once you import the Project ID and Token, you may select ‘Run’ to test the API connection.
17. This function will execute the script for importing characters' data from VoiceQ Cloud into your document.

A4 Name

	A	B	C	D	E	F	G	H
1	11ee5caf-8871-1a91-82a1-7b3777c123de							
2	Bearer eyJhbGciOiJIUzUxMiJ9.eyJjaCI6ImViNDQ0MzVlYTJlMWFiOGM4YmJiOGYzOTVmMjQ2ZjhiMzY0Y2U2ZjVmOGFmMzI4NzJiNjBmNWU5MTViZDdmYmEiLCJ0							
3								
4	Name	Actor Name	Actor	List Id	Lines	Lines (50*)	Lines (65*)	Completion Date
5	SPEAKER 4, SPEAKER				2	2	2	N/A
6	SPEAKER 6(M30), SPEAKER 12(M28)				2	2	2	N/A
7	SPEAKER 14				16	13	10	N/A
8	SPEAKER 15				5	3	2	N/A
9	SPEAKER 13				2	3	2	N/A
10	SPEAKER, SPEAKER 11				1	1	1	N/A
11	SPEAKER 11	Johnny DEPP		1015	5	3	2	N/A
12	SPEAKER	Robert DOWNEY		1005	50	25	19	29/09/2023
13	SPEAKER 13, SPEAKER				1	2	1	N/A
14	SPEAKER 12(M28), SPEAKER 4				1	1	1	N/A
15	SPEAKER 11, SPEAKER 12(M28)				1	1	1	N/A
16	SPEAKER 14, SPEAKER 15				1	1	1	N/A
17	SPEAKER 3(M28), SPEAKER 4				1	1	1	N/A
18	SPEAKER 12(M28), SPEAKER 3(M28)				1	1	1	N/A
19	SPEAKER, SPEAKER 6(M30)				1	1	1	N/A
20	SPEAKER 14, SPEAKER 4				1	1	1	N/A
21	SPEAKER, SPEAKER 4				1	1	1	N/A

Token Expiry and Replacement

- Note that the token obtained in step 3 will expire after 48 hours for security reasons.

- When the token expires, you must **repeat steps 3 and 4** to obtain and update a new token in your document.

Repeat Steps for Future Integration

- Follow the steps above for future integrations or connections to VoiceQ Cloud projects.
- Copy the Project ID from the project URL and paste it into Cell A1 of your Google Sheets document.
- Obtain a new token by clicking the 'Get Token' button in the VoiceQ Cloud project and paste it into Cell A2.

By following these steps, users can seamlessly connect their documents to VoiceQ Cloud projects, obtain and use tokens for authentication, and automate importing character data into their sheets. This ensures that character information remains updated and easily accessible for users working within the VoiceQ Cloud environment.

CONCLUSION

Following the instructions outlined in this manual, users can harness the combined power of Microsoft Excel (Excel) and VoiceQ Cloud to streamline their workflow, increase productivity, and achieve better project management outcomes. Whether managing voiceover projects, tracking data, or automating tasks, this integration offers a versatile solution for various needs.

INTEGRATING VOICEQ CLOUD WITH Microsoft Excel (Script export)

This section will guide users through connecting VoiceQ Cloud with Microsoft Excel. It will cover steps such as creating the script, copying the authorization token, authorizing access, and establishing communication between the two platforms.

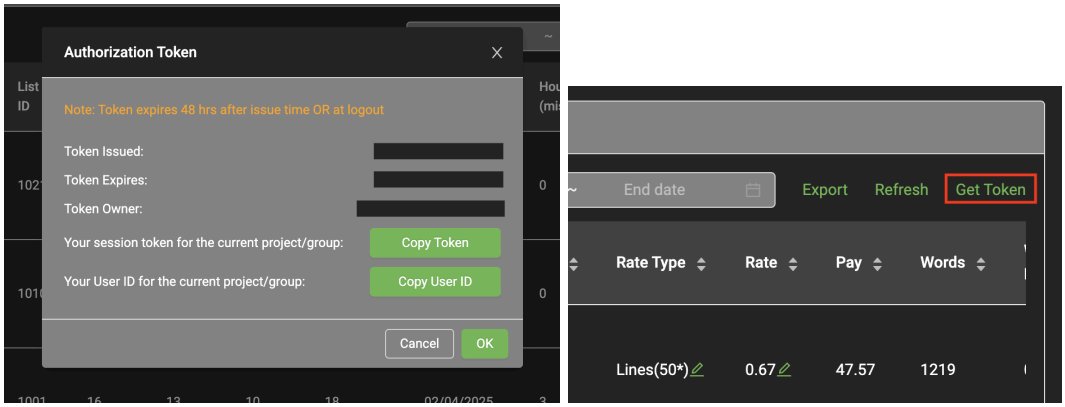
Template for API sheet (Scripts)

We have supplied a public link to download the API template, which includes a working document and a Refresh script. Raw code can be found in ‘Appendix 2’.

Download the template

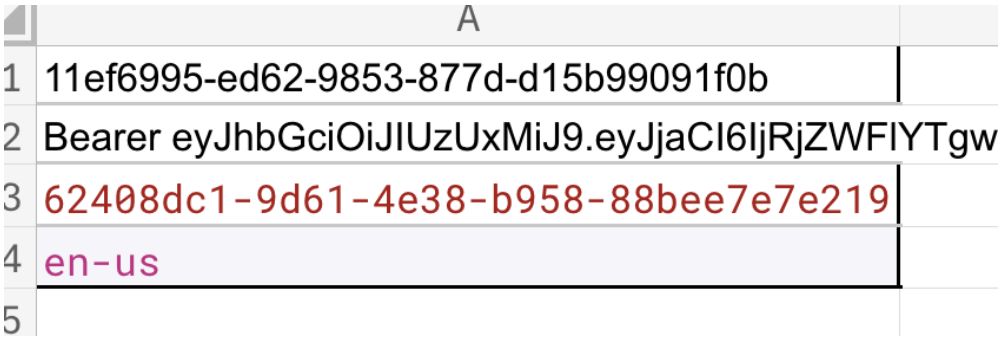
Connecting to VoiceQ Cloud

1. Begin by opening the VoiceQ Cloud project to which you want to connect.
2. Locate and copy the project ID in the project URL. It typically appears as a string of characters, such as **'11e9f6a8-5fa8-fe32-a8bd-b71835bxxxxx'.**
3. Paste the ID into Cell A1 - **INSERT_PROJECT_ID**
4. Reopen the VoiceQ Cloud project.
5. Locate the 'Get Token' button positioned above the Characters section.
6. Click on the 'Get Token' button.
7. You will see two options - One is the Token for authentication, the other is you personal user ID.



8. Copy the tokens provided and paste them into Cell A2 and A3.

a. Cell A2 - **INSERT_USER_TOKEN**
b. Cell A3 - **INSERT_USER_ID**



9. In Cell A4 - Enter the language code you wish to pull. Example ‘EN-US’ will pull English (United States) or ‘ES’ will pull in Spanish.

ADVANCED CONFIGURATION

Configuring Automate Scripts

Microsoft Power Automate serves as the connector between Microsoft Excel and VoiceQ Cloud. To configure this integration, follow these steps:

3. Download and import the script labeled “Excel_Script.osts” - You may do this by navigating to the ‘Automate’ tab and selecting “New Script.”
4. Add the script to the Code Editor and select ‘Run’ to test.

Note: If the option to add or import a script is absent, you must create a new one and then import it.

RUNNING SCRIPTS AND AUTOMATING TASKS

Once the integration is complete and scripts are configured, users can execute them to perform various actions automatically. This section will explain how to run scripts manually and set up triggers for automation based on specific events or schedules.

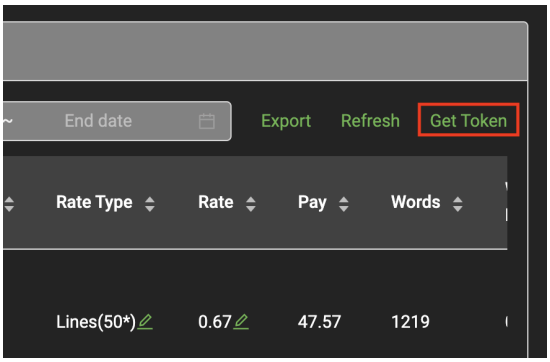
Connecting to VoiceQ Cloud

10. Begin by opening the VoiceQ Cloud project to which you want to connect.

11. In the project URL, locate and copy the Project ID. It typically appears as a string of characters, for example, '11e9f6a8-5fa8-fe32-a8bd-b71835b230d9'.
12. Paste this Project ID into Cell A1 of your Google Sheets document.

Obtaining and Using the Token

13. Reopen the VoiceQ Cloud project.
14. Locate the 'Get Token' button positioned above the Characters section.
15. Select it, and then you will see two options - One is the Token for authentication, and the other is your personal user ID.



Configuring Script Execution

16. Once you import the Project ID, Token and User ID, enter the language code of the script data you wish to pull.
17. Then select ‘Run’ to test the API connection.
18. This function will execute the script for importing script data from VoiceQ Cloud into your document.

	A	B	C	D	E	F	
1	11ef6995-ed62-9853-877d-d15b99091f0b						
2	Bearer eyJhbGciOiJIUzUxMiJ9.eyJjaCI6IjRjZWFlYTgwYWFhNzUwZmQ0OWMwOGUzMmI0MmZiNmEzNTA1YzhjNTg4MDNjODZmYmM2M2E3NTR						
3	62408dc1-9d61-4e38-b958-88bee7e7e219						
4	en-us						
5							
6	Scene	Character	Actor	Timecode	Timecode	Duration	Script
7	Scene 0001	JUDGE BATTLE COMMIS	Colin Murdock	00:00:15:(	00:00:18:1	00:00:03:(	Coordinates. W08 x2
8	Scene 0001	JUDGE BATTLE COMMIS	Colin Murdock	00:00:18:1	00:00:22:1	00:00:03:2	Zoid battle request in
9	Scene 0001	JUDGE BATTLE COMMIS	Colin Murdock	00:00:29:2	00:00:32:1	00:00:02:1	Battle mode 0982.
0	Scene 0001	JUDGE BATTLE COMMIS	Colin Murdock	00:00:55:1	00:01:00:(	00:00:04:1	Judge. Capsule releas
1	Scene 0001	BIT CLOUD	Marina LAARIF	00:01:08:2	00:01:10:(	00:00:01:(	Lucky me.
2	Scene 0001	BIT CLOUD	Marina LAARIF	00:01:15:(	00:01:19:1	00:00:04:(	The judge capsule.
3	Scene 0001	JUDGE BATTLE COMMIS	Colin Murdock	00:01:45:(	00:01:49:(	00:00:04:(	The area within a 30 mil
4	Scene 0001	JUDGE BATTLE COMMIS	Colin Murdock	00:01:49:1	00:01:50:1	00:00:01:(	The zone is trap restri
5	Scene 0001	JUDGE BATTLE COMMIS	Colin Murdock	00:01:50:2	00:01:54:1	00:00:03:2	Only competitors anc
6	Scene 0001	JUDGE BATTLE COMMIS	Colin Murdock	00:01:54:2	00:01:56:2	00:00:02:(	All others must leave
7	Scene 0001	BIT CLOUD	Marina LAARIF	00:01:57:(	00:01:59:(	00:00:02:(	You can't just boot us
8	Scene 0001	JUDGE BATTLE COMMIS	Colin Murdock	00:01:59:(	00:02:02:(	00:00:02:2	This zone is now restr

Token Expiry and Replacement

- Note that the token obtained in step 3 will expire after 48 hours or when you log out for security reasons.
- When the token expires, you must **repeat steps 3 and 4** to obtain and update a new token in your document.

Repeat Steps for Future Integration

- Follow the steps above for future integrations or connections to VoiceQ Cloud projects.
- Copy the Project ID from the project URL and paste it into Cell A1 of your Google Sheets document.
- Obtain a new token by clicking the 'Get Token' button in the VoiceQ Cloud project and paste it into Cell A2.

By following these steps, users can seamlessly connect their documents to VoiceQ Cloud projects, obtain and use tokens for authentication, and automate importing character data into their sheets. This ensures that character information remains updated and easily accessible for users working within the VoiceQ Cloud environment.

CONCLUSION

Following the instructions outlined in this manual, users can harness the combined power of Microsoft Excel (Excel) and VoiceQ Cloud to streamline their workflow, increase productivity, and achieve better project management outcomes. Whether managing voiceover projects, tracking data, or automating tasks, this integration offers a versatile solution for various needs.

APPENDIX A - CHARACTER EXPORT

This code can be customized for use with Microsoft Excel documents.

```
async function main(workbook: ExcelScript.Workbook) {

 const sheet = workbook.getActiveWorksheet();

 var id = sheet.getRange("A1").getValue().toString();

 var auth = sheet.getRange("A2").getValue().toString();


 // Get the active cell and worksheet.
 let selectedCell = workbook.getActiveCell();
 let selectedSheet = workbook.getActiveWorksheet();

 // Retrieve sample JSON data from a test server.
 let project = await fetch('https://app.voiceqcloud.com/api/v1/project/' + id +
'/stats/characters', {
   method: "GET",
   headers: {
     'Authorization': auth
   }
 });

 let group = await fetch('https://app.voiceqcloud.com/api/v1/group/' + id +
'/stats/characters', {
   method: "GET",
   headers: {
     'Authorization': auth
   }
 });

 let contacts = await fetch('https://app.voiceqcloud.com/api/v1/contacts/', {
   method: "GET",
   headers: {
     'Authorization': auth
   }
 });

 let contactsJson: ContactsJSON = await contacts.json()


 // Convert the returned data to the expected JSON structure.
```

```
 var finalJson: JSONData
 try{

  finalJson  = await project.json();
 }
 catch(error){
 }

 try{

  finalJson  = await group.json()
 }
 catch(error){
 }

 const headers = [
   "Name", "Actor Name", "Actor", "List Id", "Lines", "Lines (50*)", "Lines (65*)",
"Rounded (65)", "Completion Date", "Hours/takes (misc)", "Rate Type", "Rate", "Pay",
"Words", "Words Recorded", "Spoken Lines", "Translated", "Recorded Letters",
"Letters", "Recorded", "Character Description", "Gender", "Uuid"
 ];

 // Write headers to the first row
 const headerRange = sheet.getRange("A4:W4");
 headerRange.setValues([headers]);


 for (const char in finalJson){



   sheet.getRange(`A${5 + parseInt(char,
10)}`).setValue(finalJson[char].characterName);
   sheet.getRange(`B${5 + parseInt(char, 10)}`).setValue(finalJson[char].actorName);
   sheet.getRange(`C${5 + parseInt(char, 10)}`).setValue(finalJson[char].actor);
   sheet.getRange(`D${5 + parseInt(char, 10)}`).setValue(getListId(char));
   sheet.getRange(`E${5 + parseInt(char, 10)}`).setValue(finalJson[char].lines);
   sheet.getRange(`F${5 + parseInt(char, 10)}`).setValue(finalJson[char].lines50);
   sheet.getRange(`G${5 + parseInt(char, 10)}`).setValue(finalJson[char].lines65);
   sheet.getRange(`H${5 + parseInt(char,
10)}`).setValue(finalJson[char].lines65Rounded);
   const completionDate: Number = finalJson[char].completionDate
 if(completionDate > 0){
   sheet.getRange(`I${5 + parseInt(char, 10)}`).setValue(new
Date(finalJson[char].completionDate*1000).toLocaleDateString());
 }else{
   sheet.getRange(`J${5 + parseInt(char, 10)}`).setValue("N/A")
```

```
 }
   sheet.getRange(`K${5 + parseInt(char, 10)}`).setValue(finalJson[char].miscInfo);
   sheet.getRange(`L${5 + parseInt(char, 10)}`).setValue(transformRateType(char));
   sheet.getRange(`M${5 + parseInt(char, 10)}`).setValue(finalJson[char].rate);
   sheet.getRange(`N${5 + parseInt(char, 10)}`).setValue(getPay(char));
   sheet.getRange(`O${5 + parseInt(char, 10)}`).setValue(finalJson[char].words);
   sheet.getRange(`P${5 + parseInt(char,
10)}`).setValue(finalJson[char].recordedWords);
   sheet.getRange(`Q${5 + parseInt(char, 10)}`).setValue(finalJson[char].spokenLines);
   sheet.getRange(`R${5 + parseInt(char, 10)}`).setValue(finalJson[char].translated);
   sheet.getRange(`S${5 + parseInt(char,
10)}`).setValue(finalJson[char].recordedLetters);
   sheet.getRange(`T${5 + parseInt(char, 10)}`).setValue(finalJson[char].letters);
   sheet.getRange(`U${5 + parseInt(char, 10)}`).setValue(finalJson[char].recorded);
   sheet.getRange(`V${5 + parseInt(char,
10)}`).setValue(finalJson[char].characterDesc);
   sheet.getRange(`W${5 + parseInt(char, 10)}`).setValue(transformGender(char));
   sheet.getRange(`X${5 + parseInt(char, 10)}`).setValue(finalJson[char].uuid);


  }


 function getPay(char: string) {

    switch (finalJson[char].rateType) {
     case 0:
       return finalJson[char].rate;
     case 1:
       return Number((finalJson[char].miscInfo * finalJson[char].rate).toFixed(2));
     case 2:
       return Number((finalJson[char].rate * finalJson[char].lines50).toFixed(2));
     case 3:
       return Number((finalJson[char].rate * finalJson[char].words).toFixed(2));
     case 4:
       return Number((finalJson[char].rate * finalJson[char].lines65).toFixed(2));
     case 5:
       return Number((finalJson[char].rate * finalJson[char].lines).toFixed(2));
     default:
       return '';
   }
 }


 function transformRateType(char: string) {
   switch (finalJson[char].rateType) {
     case 5:
```

```
      return 'Lines';
    case 1:
      return 'Hourly';
    case 2:
      return 'Lines(50*)';
    case 4:
      return 'Lines(65*)';
    case 0:
      return 'Flat Rate';
    case 3:
      return 'Words';
    default:
      return '';
  }
}


function transformGender(char: string) {
  switch (finalJson[char].gender) {
    case 0:
      return 'N/A';
    case 1:
      return 'Female';
    case 2:
      return 'Male';
    case 3:
      return 'Other';
    default:
      return '';
  }
}


function getListId(char: string) {

  for (const contact in contactsJson){

    const contactString: String = contactsJson[contact].nameLast ?
contactsJson[contact].nameFirst + ' ' + contactsJson[contact].nameLast :
contactsJson[contact].nameFirst;



  if (contactString && finalJson[char].actorName) {
```

```
    if (contactString.toLowerCase().trim() ===
finalJson[char].actorName.toLowerCase().trim()) {

      return contactsJson[contact].listId
    }
  }

  }

  return ''


 }
```

```
/**
* An interface that matches the returned JSON structure.
* The property names match exactly.
*/
interface JSONData {
 uuid: String;
 characterName: String
 actorName: String;
 actor: String;
 words: Number;
 recordedWords: Number;
 letters: Number;
 recordedLetters: Number;
 lines: Number;
 lines50: Number;
 lines65: Number;
 linesRounded65: Number;
 spokenLines: Number;
 translated: Number;
 recorded: Number;
 color: String;
 miscInfo: String;
 completionDate: String;
 talent: String;
 characterDesc: String;
 gender: Number;
 characterType: Number;
 rateType: Number;
 rate: Number;
 pay: Number;
```

```
}

 interface ContactsJSON {
   nameLast: String;
   nameFirst: String
   listId: String;
 }
}



// // Write data to the worksheet
// sheet.getRange("A1").setValue("API Data");
// sheet.getRange("A2").setValue(JSON.stringify(data, null, 2));
// }
```

APPENDIX B - SCRIPT EXPORT

This code can be customized for use with Microsoft Excel documents.

```
async function main(workbook: ExcelScript.Workbook) {
    let sheet = workbook.getActiveWorksheet();

    // Explicitly declare types for the variables
    let auth: string = sheet.getRange("A2").getValue() as string;
    let id: string = sheet.getRange("A1").getValue() as string;
    let userId: string = sheet.getRange("A3").getValue() as string;
    let langCode: string = sheet.getRange("A4").getValue() as string;
    let finalObj: object[] = []; // Array of objects to store the final data


    let url: string = `https://app.voiceqcloud.com/api/v1/project/${id}/sync/`;

    let params = {
        id: id,
        remoteVersion: -1,
        remoteVersionUpdatedBy: userId
    };

    // Build query string
    let queryString: string = Object.keys(params).map(key => `${key}=${params[key]}`).join("&");
    let fullUrl: string = `${url}?${queryString}`;

    let headers: { Authorization: string; Accept: string } = {
        Authorization: auth,
        Accept: "application/json"
    };

    try {
        // Fetch response with specific type for the response
        let response: Response = await fetch(fullUrl, {
            method: "GET",
            headers: headers
        });

        let responseCode: number = response.status;
        let responseText: string = await response.text();

        // Define an interface for the expected response structure
        interface ApiResponse {
            scenes: Scene[];
            characters: { name: string, actorName: string }[];
            project: { fps: number };
        }

        interface Scene {
            uuid: string;
```

```
        name: string;
        children: FilmedLine[];
    }

    interface FilmedLine {
        uuid: string;
        character: string;
        startTimecode: string;
        endTimecode: string;
        cue: number;
        note: string;
        children: LanguageLine[];
    }

    interface LanguageLine {
        text: string;
        done: boolean;
        doneAt: string | null;
        adapted: boolean;
        adaptedAt: string | null;
        qc: boolean;
        qcAt: string | null;
        pickUp: boolean;
        pickUpAt: string | null;
        vo: boolean;
        voAt: string | null;
        useOV: boolean;
        useOVAt: string | null;
        onScreenNote: string | null;
        eventType: string;
        onScreenOption: string;
        screenPosition: string | null;
        dubAnnotations: string | null;
        language: string | null;
    }

    // Use the ApiResponse type for the object variable
    let object: ApiResponse;

    object = JSON.parse(responseText) as ApiResponse;


    // let finalObj: object[] = []; // Array of objects to store the final data

    for (let scene of object.scenes) {
        for (let filmedLine of scene.children) {
            for (let languageLine of filmedLine.children) {
                if(languageLine.language.toLocaleLowerCase() == langCode.toLowerCase()){
                let newObj: { [key: string]: string | number | boolean | null } = {}; // Explicitly define the type of
newObj

                newObj["Scene"] = scene.name;
                newObj["Character"] = filmedLine.character;
```

```
                newObj["Actor"] = getActorName(filmedLine.character, object.characters);
                newObj["Timecode In"] = filmedLine.startTimecode;
                newObj["Timecode Out"] = filmedLine.endTimecode;
                newObj["Duration"] = calcDuration(filmedLine.startTimecode, filmedLine.endTimecode,
object.project.fps);
                newObj["Script"] = languageLine.text;
                newObj["Cue Number"] = filmedLine.cue;
                newObj["Recorded"] = languageLine.done === true ? "Y" : "N";
                newObj["Recorded at"] = languageLine.doneAt;
                newObj["Adapted"] = languageLine.adapted === true ? "Y" : "N";
                newObj["Adapted at"] = languageLine.adaptedAt;
                newObj["QC"] = languageLine.qc === true ? "Y" : "N";
                newObj["QC at"] = languageLine.qcAt;
                newObj["Pick up"] = languageLine.pickUp === true ? "Y" : "N";
                newObj["Pick up at"] = languageLine.pickUpAt;
                newObj["V/O"] = languageLine.vo === true ? "Y" : "N";
                newObj["V/O at"] = languageLine.voAt;
                newObj["Use OV"] = languageLine.useOV === true ? "Y" : "N";
                newObj["Use OV at"] = languageLine.useOVAt;
                newObj["Comments"] = filmedLine.note;
                newObj["On Screen Note"] = languageLine.onScreenNote;
                newObj["Event Type"] = languageLine.eventType;
                newObj["On/Off screen"] = languageLine.onScreenOption;
                newObj["Screen Position"] = languageLine.screenPosition;
                newObj["Dub Annotations"] = languageLine.dubAnnotations;
                newObj["Scene ID"] = scene.uuid;
                newObj["Line ID"] = filmedLine.uuid;

                finalObj.push(newObj);
                }
            }
        }
    }

    console.log(finalObj);

    // Call a function to add the data into Excel cells

  } catch (error) {
    console.log(error);
  }
  finally {

    const headers = [
      "Scene",
      "Character",
      "Actor",
      "Timecode In",
      "Timecode Out",
      "Duration",
      "Script",
      "Cue Number",
      "Recorded",
```

```
    "Recorded at",
    "Adapted",
    "Adapted at",
    "QC",
    "QC at",
    "Pick up",
    "Pick up at",
    "V/O",
    "V/O at",
    "Use OV",
    "Use OV at",
    "Comments",
    "On Screen Note",
    "Event Type",
    "On / Off screen",
    "Screen Position",
    "Dub Annotations",
    "Scene ID",
    "Line ID"
];

// Write headers to the first row (row 1)
const headerRange = sheet.getRange("A6:AB6");
headerRange.setValues([headers]);

// Function to write data in batches
const BATCH_SIZE = 100; // Define your batch size here
for (let i = 0; i < finalObj.length; i += BATCH_SIZE) {
    // Create a batch of data
    const batch = finalObj.slice(i, i + BATCH_SIZE).map((item) => [
        item.Scene,
        item.Character,
        item.Actor,
        item["Timecode In"],
        item["Timecode Out"],
        item.Duration,
        item.Script,
        item["Cue Number"],
        item.Recorded,
        item["Recorded at"],
        item.Adapted,
        item["Adapted at"],
        item.QC,
        item["QC at"],
        item["Pick up"],
        item["Pick up at"],
        item["V/O"],
        item["V/O at"],
        item["Use OV"],
        item["Use OV at"],
        item["Comments"],
        item["On Screen Note"],
        item["Event Type"],
```

```
                item["On / Off screen"],
                item["Screen Position"],
                item["Dub Annotations"],
                item["Scene ID"],
                item["Line ID"]
            ]);

            // Determine the range for the batch
            const rowStart = 7 + i; // Starting row based on index
            const range = sheet.getRange(`A${rowStart}:AB${rowStart + batch.length - 1}`);
            range.setValues(batch);
        }
    }

    // Helper function to get actor name
    function getActorName(characterName: string, charactersList: { name: string, actorName: string }[]): string
{
        for (let i = 0; i < charactersList.length; i++) {
            if (charactersList[i].name === characterName) {
                return charactersList[i].actorName;
            }
        }
        return "";  // Return empty string if no match is found
    }

    // Helper function to calculate duration
    function calcDuration(startTimecode: string, endTimecode: string, fps: number): string {
        function timecodeToFrames(timecode: string, fps: number): number {
            let parts: string[] = timecode.split(":");
            let hours: number = parseInt(parts[0], 10);
            let minutes: number = parseInt(parts[1], 10);
            let seconds: number = parseInt(parts[2], 10);
            let frames: number = parseInt(parts[3], 10);

            let totalFrames = (hours * 3600 * fps) + (minutes * 60 * fps) + (seconds * fps) + frames;
            return totalFrames;
        }

        let startFrames: number = timecodeToFrames(startTimecode, fps);
        let endFrames: number = timecodeToFrames(endTimecode, fps);
        let durationFrames: number = endFrames - startFrames;

        let hours: number = Math.floor(durationFrames / (3600 * fps));
        let minutes: number = Math.floor((durationFrames % (3600 * fps)) / (60 * fps));
        let seconds: number = Math.floor((durationFrames % (60 * fps)) / fps);
        let frames: number = Math.round(durationFrames % fps);

        return (
            `${String(hours).padStart(2, '0')}:${String(minutes).padStart(2, '0')}:${String(seconds).padStart(2,
'0')}:${String(frames).padStart(2, '0')}`
        );
    }
}
```