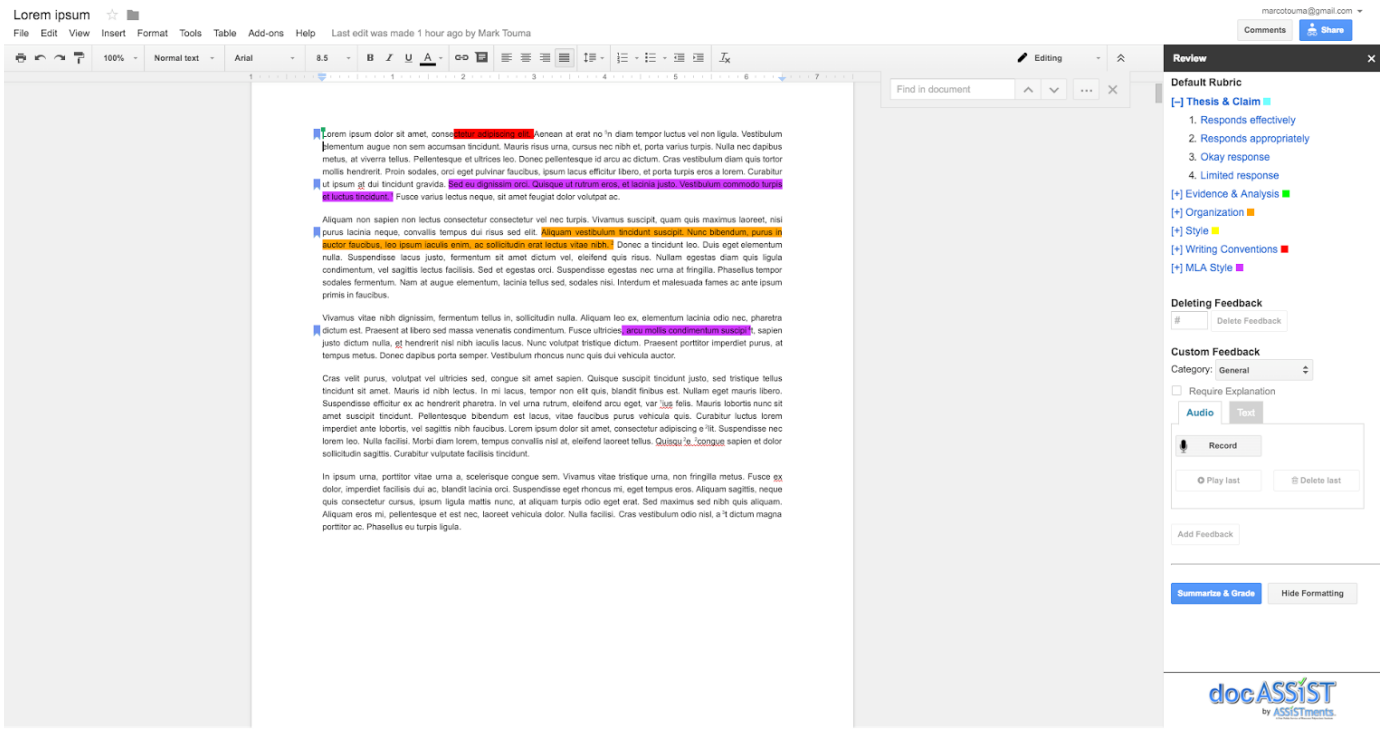
Developing docASSIST: A Google Doc Add-on



By
Christian Roberts
Jean Marc Touma

WPI

docASSIST by ASSISTments
A Free Public Service of Worcester Polytechnic Institute

Developing docASSIST: A Google Doc Add-on

An Interactive Qualifying Project Report Submitted to the Faculty of the
WORCESTER POLYTECHNIC INSTITUTE
in partial fulfillment of the requirements for the Degree of Bachelor of Science

by

_____
Christian Roberts

_____
Jean Marc Touma

Submitted on

Approved:

_____
Professor Neil T. Heffernan, Advisor

_____
Cristina Heffernan, Advisor

Table of Contents

- Storage.gs
- addComment.gs
- rubric.html

Created Code

- Grading.gs
- Languages.gs
- RubricPreview.gs
- saveOptions.gs
- grading.html
- options.html

List of Figures

Abstract

docAssist is a Google Doc Add-on created by Nick McMahon and Sam La with the intention of helping teachers organize and expedite the reviewing and grading process. The goal of this project was to further develop the Add-on, fixing any bugs and introducing new features. Those features included Grading, Deleting Feedback, Rubric Preview, and the Rubric Manager. After having conversations with teachers who were users of docAssist, we were able to tailor the features to their workflow in our effort to make the tool the most efficient it could be.

Acknowledgements

We would like to acknowledge our advisors, Neil and Cristina Heffernan, for their guidance over the course of the project; they were both very hands-on throughout the project, giving us fair criticism or praise when it was deserved. We would also like to acknowledge Nick McMahon and Sam La for creating the tool, trusting us to carry on the project, and for being present and helpful over the course of our development. We want to acknowledge Zi Wang for her work on developing the Rubric Preview feature, We would like the acknowledge the teachers who gave us feedback: Janine Campbell, Stephanie Marando, Coralie Stopford, Tanya Dolata, Amy Lyons, Ian Schwartz, and those who met with us, either personally, or through video calls: Susan Andrews, Tosca Necoechea, Lisa Hogan, and Althea Danielski. Finally, we would like to acknowledge Google for having such excellent documentation for their APIs that made our development process significantly smoother.

Authorship

Christian Roberts and Jean Marc Touma wrote the entirety of this paper in conjunction.

Introduction

Google Docs, though now a common resource in educational settings, does not provide the best support for the logistical work demanded by teaching. Some features, such as commenting, provide a very nice aesthetic, however they do not cut down time that teachers must often take in order to provide their students feedback. When a teacher grades a written assignment, they often use a rubric with a predetermined set of notes they will give the students. This was the original philosophy behind docAssist, providing teachers with the power to create rubrics tailored to the feedback that they frequently note (such as spelling or grammatical errors).

At the time that we began working on docAssist, it was a few months old. We were able to collect feedback from its initial users and add features in line with their needs and expectations. Our goal in this project was to continue to develop docAssist and strive to achieve its initial design philosophy of simplifying work for teachers when grading assignments.

The Development Process

Getting Started

When we began work on docAssist, we had to become familiar with the add-on, Google Apps Script, and Google APIs. The initial code for the add-on, while well commented, was very difficult to comprehend most of the time. This was for a few reasons; firstly, we did not have the familiarity with Google Apps Script that would allow us to read and understand the code. The other reason was that, as Nick and Sam (the previous developers) put work into docAssist, they were learning just as we would, so the readability of their code was only ideal on parts of the code that were written later on in their development process.

When we first began work on docAssist, we tackled relatively simple bugs instead of major features to help us learn the purposes of the different parts of the code. These were our initial lists of features and bugs that we worked towards in our first term of development:

Title	Description	Requestor	Requestor Email	Status
Batch Rubric Assign	User should be able to add the same rubric to a group of essays. (Most likely a folder in Google Docs)	Jeri A. Weiss	jaweiss@eduhsd.k12.ca.us	**Not Started**
Delete Feedback	Users should be able to delete their own feedback notes in case they make a mistake	Jeri A. Weiss	jaweiss@eduhsd.k12.ca.us	Deployed
Rubric Point Values	so when clicked on, the points for each category would appear in the summary and then automatically be totaled.	Jeri A. Weiss	jaweiss@eduhsd.k12.ca.us	Deployed
Editable Titles/Sections	Users should be able to edit the titles and sections of the rubric after creating it for easy editing.	Cristina Heffernan, Michelle Munro	cristina.heffernan@gmail.com, michelle.munro@btps.ca	Deployed
Drag and Drop	Rubric setup should allow for dragging and dropping of sections	Cristina Heffernan	cristina.heffernan@gmail.com	**Not Started**
User rubric database access	Allow users to share rubrics globally, and find other popular rubrics created by other users	Christian Roberts, Mark Touma	caroberts@wpi.edu, jatouma@wpi.edu	**Not Started**
ASSISTments integration	Allow submission, and returning graded assignments via ASSISTments	Christian Roberts, Mark Touma	caroberts@wpi.edu, jatouma@wpi.edu	**Not Started**
Google classroom integration	Allow submission, and returning graded assignments via google classroom	Christian Roberts, Mark Touma	caroberts@wpi.edu, jatouma@wpi.edu	**Not Started**
Create a google doc to visually represent the rubric	Students should be able to understand their rubric as they are writing th paper			Deployed
Find a way to attach a rubric to a doc when it is being sent out	Rubrics and other document properties don't copy when a template is made. Perhaps use a marker and auto assign a rubric on start			**Not Started**
Delete General Feedback	Deleting feedback that was made without highlighting.	Christian Roberts, Mark Touma, Zi Wang	caroberts@wpi.edu, jatouma@wpi.edu, zwang7@wpi.edu	Deployed
Grading Tool	Teacher can grade with a seperate grading tool, automatically saving to a spreedsheet selected from a list.			Deployed

Figure 1: Initial list of features

Date added	Title	Description	Reporter	Reporter Email	Link to Doc	Status
A-B term	Resolving Error	This looks like it has something to do with invalid text ranges. The following error sometimes occurs when trying to resolve feedback. "Index (-2) value must be greater than or equal to zero"	Neil Heffernan	nth@WPI.EDU	https://docs.google.com/d	Ready to Deploy
A-B term	Window Resize	On some computers, the rubric setup window is too large, preventing users from clicking "Save"	Kim Estes	kestes@bisdmail.net	N/A	Ready to Deploy
A-B term	Refresh rubrics	Selecting a new rubric does not automatically refresh the sidebar if it is open	Nick & Sam	N/A	N/A	Ready to Deploy
A-B term	Undefined in Grade table	For some reason the grade table said undefined.				Ready to Deploy
A-B term	Custom comment button bug	When making a custom comment, the feedback button would remain enabled, when it should be disabled	Christian & Mar	caroberts@wpi.edu jatouma@wpi.edu		Ready to Deploy
A-B term	Delete or Resolve Highlight Bug	When multiple comments are made over the same range and one is resolved or deleted, the color of the highlight is changed to white, when it should be the colors of the comments beneath it.	Christian Mark Zi	caroberts@wpi.edu jatouma@wpi.edu zwang7@wpi.edu		Requires Review
A-B term	getRange on null bug	If a section of text that had a comment associated with it was deleted and someone tries to unformat it (or do anything that requires its range) the function fails.	Christian Mark Zi	caroberts@wpi.edu jatouma@wpi.edu zwang7@wpi.edu	https://docs.google.com/d	Not Started
A-B term	Grade total alert	When grades are totalled, we get a weird alert.	Christian & Mar	caroberts@wpi.edu jatouma@wpi.edu		In Progress
1/22/2016	Enter in rubric setup	When changing a category name, pressing enter should exit the text box	Christian & Mar	caroberts@wpi.edu jatouma@wpi.edu		Ready to Deploy
1/22/2016	Name change in rubric setup	When a category name is changed, it does not save although it visually displays the new name.	Christian & Mar	caroberts@wpi.edu jatouma@wpi.edu		Ready to Deploy
1/22/2016	Alphabetical need to include lower and upper case	This is in the GAE backend	Christian & Mar	caroberts@wpi.edu jatouma@wpi.edu		Ready to Deploy
1/22/2016	Selected rubric name should appear in rubric setup	When opening rubric setup, the selected rubric does not fill in the name automatically as it does when selecting a rubric from the drop-down.	Christian & Mar	caroberts@wpi.edu jatouma@wpi.edu		In Progress

Figure 2: Initial list of bugs

We proceeded to fix most bugs, and then we began our work on adding our first feature: Deleting Feedback.

Delete Feedback

Prior to our work on docAssist, feedbacks made by teachers could not be deleted individually. Instead, a teacher had to delete all feedback that had been attached so far. We received an email from a Spanish teacher, Tosca Necoechea, at Berkeley High in California in which she asked “Can I undo feedback once it's given by any means other than resetting the add-on and erasing all my other feedback?”. We then proceeded to develop this feature.

When writing the back-end code for this feature, we had to go through every feedback that had been made so far and find a unique identifier that the user also knew. Every time a feedback was made, it was given an ID as well as a reference number. Though searching by ID would have been an easier approach from a technical perspective, users could not see the ID of a feedback, so we had to develop the code using reference numbers. Writing this part of the code helped us understand better how feedbacks were being saved.

After writing the function in charge of deleting feedback, we created an interface as seen in figure 3. This provided teachers with an easy and intuitive way to delete feedback. We integrated this feature alongside the space for creating feedbacks so a teacher would immediately grasp its functionality.

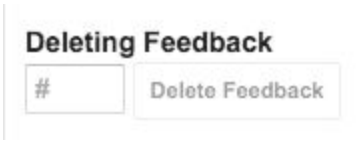


Figure 3: Deleting Feedback Interface

The following figures demonstrate the way in which the feature is used:

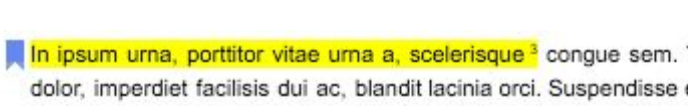


Figure 4: A feedback with reference number 3

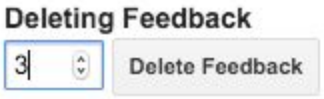


Figure 5: Deleting feedback 3

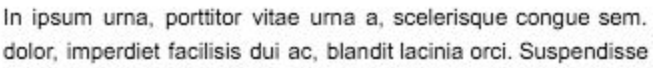


Figure 6: Unformatted (Deleted) text

In the example above, feedback number 3 would be removed from the document and the text would return to its original format (white background with no superscript).

Deleting feedback uncovered a bug in unformatting which was later fixed with the help of Nick McMahon.

Rubric Preview

On our second term we wanted to create a feature that would help integrate Learning Management Systems (LMS), like Google Classroom, with docAssist. The way that Classroom works with Google Docs is that when an assignment is created, each student receives a copy of the assignment document. We wanted to come up with a way for the students to then have easy access to the information docAssist provides without needing to get the Add-on (only requiring work from the teacher). Doing this proved to be difficult, as when a document is copied, the add-on does not stay attached.

This inspired our Rubric Preview feature, where the teacher would create their assignment document, attach the rubric they planned on using, and then send out the copies to the students. This process did not require any more work of the teacher than they were already putting into sending out these assignments, as we attach the preview automatically when they attach the rubric.

During the development of the feature, Zi Wang, a graduate student at WPI, had been working with us in making the add-on. She took charge of making this feature while we developed others, we gave her guidance and designed what the Rubric Preview should look like. In its early stages, the Preview looked similar to the Summary:

Default Rubric:

Thesis & Claim
1. Responds effectively 2. Responds appropriately 3. Okay response 4. Limited response
Evidence & Analysis
1. Strong evidence 2. Minimal evidence
Organization
1. Strong organization 2. Weak Organization
Style
1. Effective style 2. Weak Style
Writing Conventions
1. Strong conventions 2. Weak conventions
MLA Style
1. Consistent MLA 2. Weak MLA

Figure 7: Original Rubric Preview look

Summary of "Example doc":

Feedback:

Total Feedback Notes: 5

Thesis & Claim
Reviewer 1: • Responds effectively (Note 3)
Evidence & Analysis
Organization
Reviewer 1: • Strong organization (Note 1)
Style
Reviewer 1: • Effective style (Note 4) • Effective style (Note 5)
Writing Conventions
MLA Style
Reviewer 1: • Consistent MLA (Note 2)
General Comments

Figure 8: Feedback Summary look

We did not like how, when both of these were present at the bottom of the document, it was hard to tell them apart. For that reason, we designed a new and unique aesthetic for the Rubric Preview that would allow it to be immediately identifiable and distinguishable:

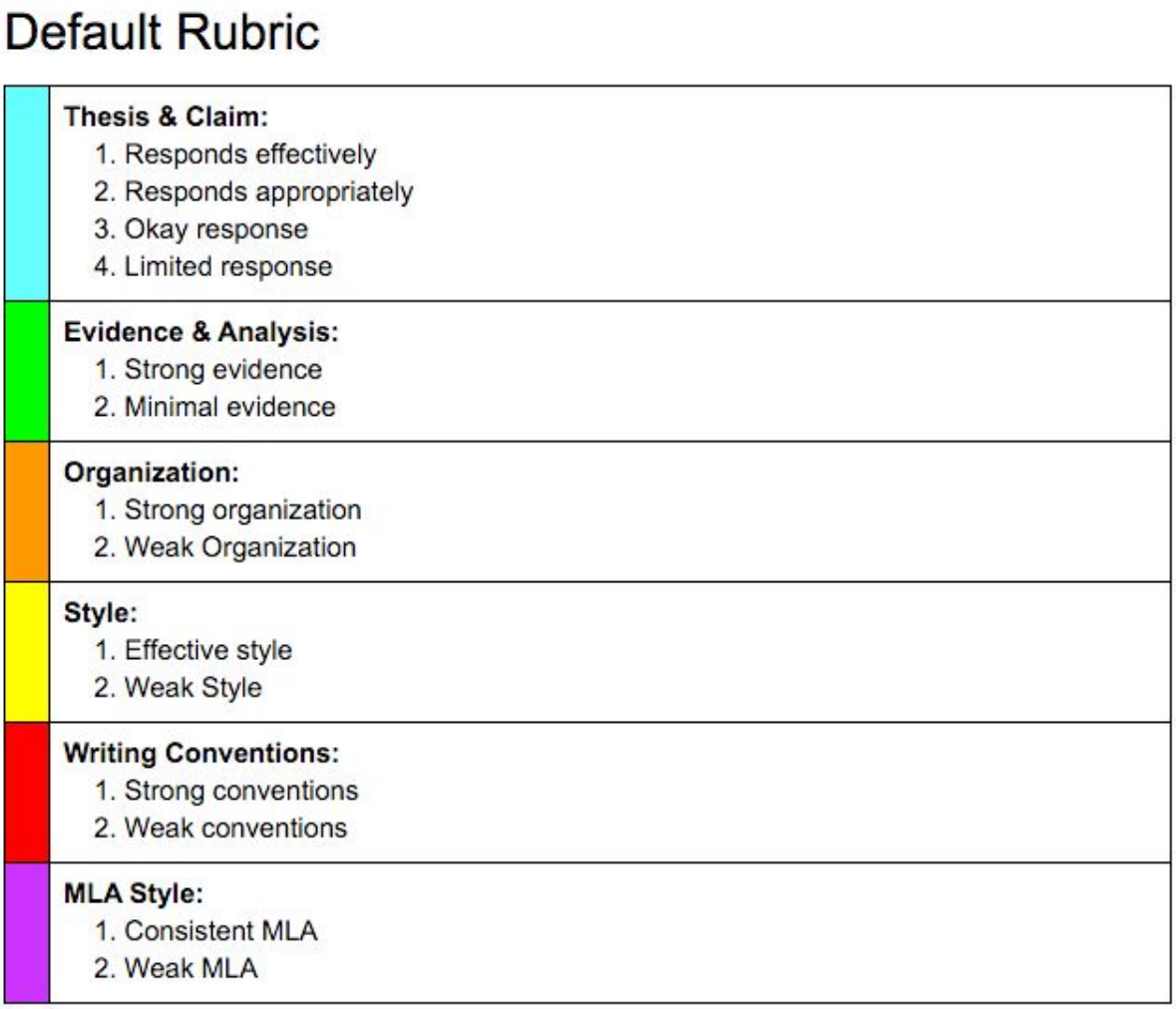


Figure 9: Final Rubric Preview look

Grading

One of the difficulties of creating an add-on for Google Docs was that we had no good way to distinguish what type of user was interacting with the document. For that reason, it was very hard to implement any sort of security or permissions. This means that we had no way of knowing if a user was a student or a teacher, and thus could not keep students from changing the values in the grading table. The other issue that that the old grading feature had was that the grades were saved and attached to only the document that had been graded. If the teacher wanted to check the grades they had given their students, they would need to go through every document they had graded. Instead, we wanted teachers to have the option of saving as many grades as they like, for any amount of assignments and/or students, in one place.

Grade:

Thesis & Claim	5
Evidence & Analysis	7
Organization	9
Style	8
Writing Conventions	8
MLA Style	10
Total	47

Figure 10: Grading Table. This table gets attached to the bottom of the document when the assignment is graded.

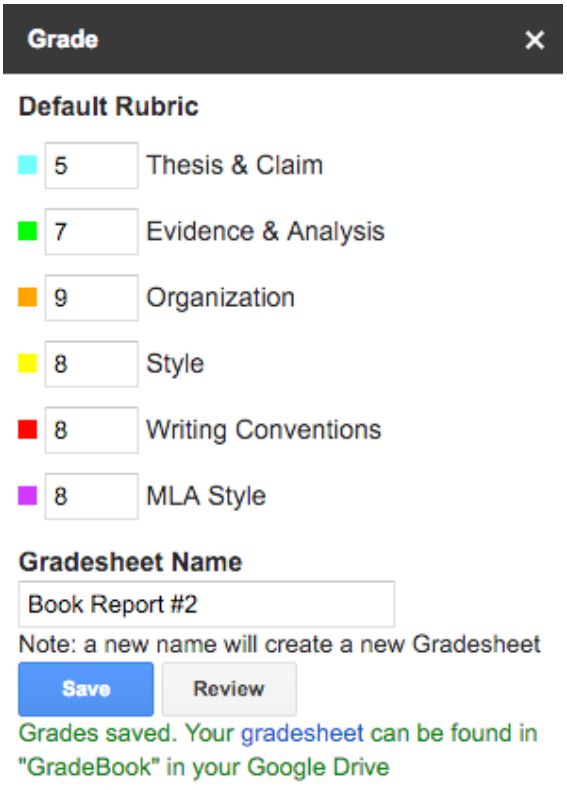
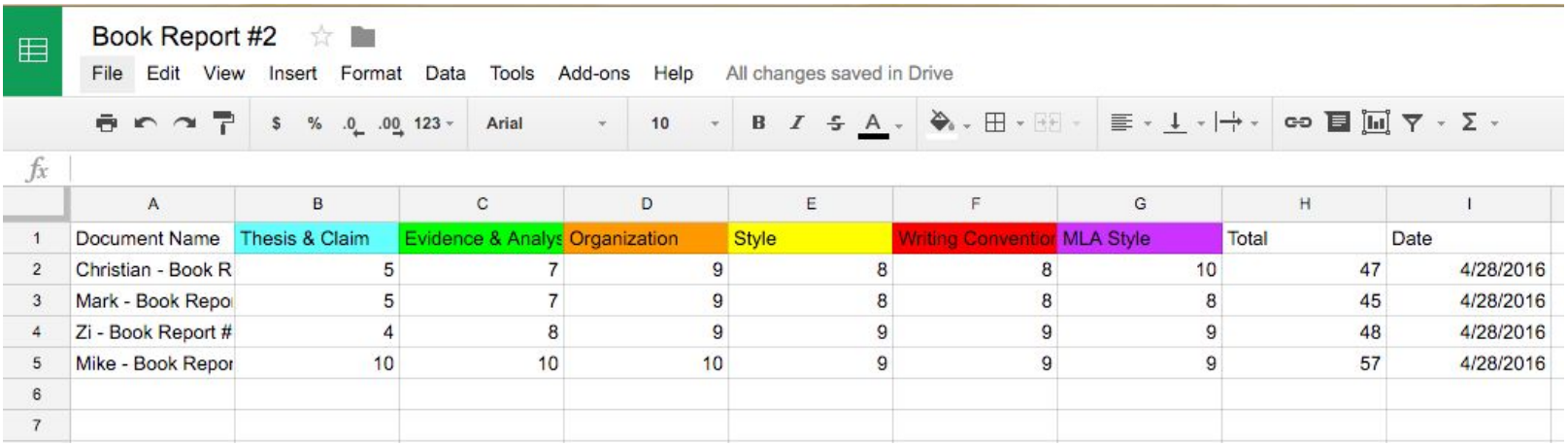


Figure 11: Grading tab

Because we didn't want students changing their grades and we wanted a way for teachers to have their grades together, we came up with the Grading feature as it is now. Teachers can fill out their grades in a form after creating the summary of their feedback for the student. Teachers specify the name of the Grade Sheet they would like that information saved to, and then the add-on automatically creates a Google Sheet with that name in their Google Drive, with the filepath "docAssist > docAssist Gradebook" as seen in figure 13. (we create the folders the first time they do this).

The table shown in figure 12 shows one way in which the teacher can choose to organize their assignments. The title of the Gradesheet can be the assignment name and every row in the table would be the set of grades each student received on that assignment. We thought, however, that teachers might have a different approach to how they organize their grades. For



	A	B	C	D	E	F	G	H	I
1	Document Name	Thesis & Claim	Evidence & Analys	Organization	Style	Writing Convention	MLA Style	Total	Date
2	Christian - Book R	5	7	9	8	8	10	47	4/28/2016
3	Mark - Book Repo	5	7	9	8	8	8	45	4/28/2016
4	Zi - Book Report #	4	8	9	9	9	9	48	4/28/2016
5	Mike - Book Repor	10	10	10	9	9	9	57	4/28/2016
6									
7									

Figure 12: Gradesheet. Students can put their names in the document name for ease of sorting.

example, the Gradesheet name could be the name of the class, and there may be more than one entry per student, as each student can do more than one assignment. Another way we thought a teacher might organize their grades is creating a spreadsheet for every student, where each entry is a different assignment. We wanted to allow for that freedom, so we let teachers grade any assignment into any Gradesheet, they just need to specify the name.

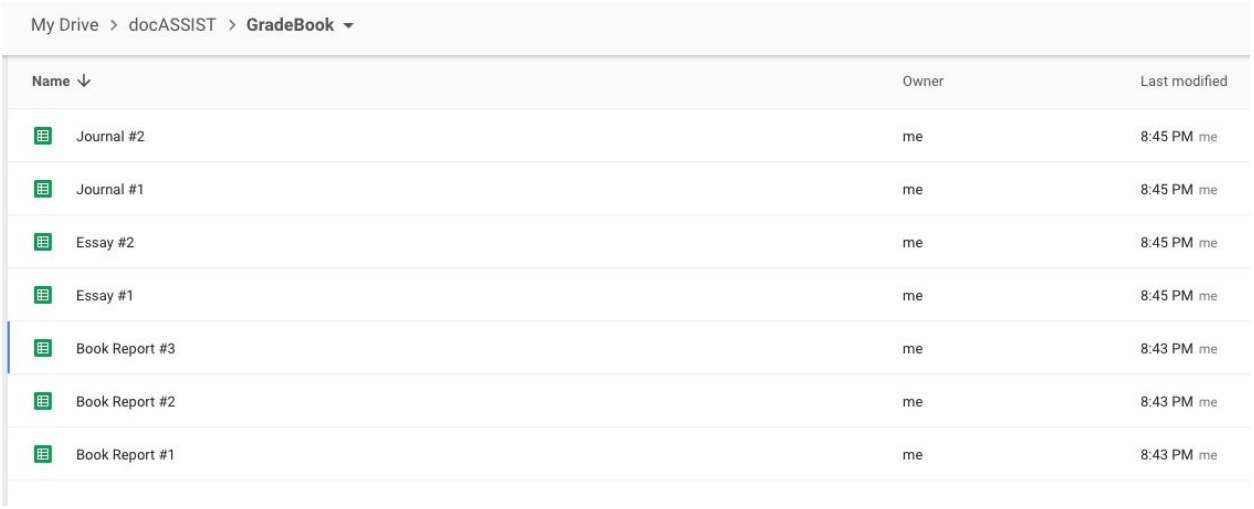


Figure 13: Gradebook Folder in Google Drive

The grading feature, Rubric Preview, and many bug fixes, were scheduled for release before our December break. We picked this time so that we would have the maximum amount of time to work on the features prior to release as well as releasing at a time where most teachers would not be working. This gave us a nice window in which we could test our new features along with any teachers that wanted to spend any of their time trying the new content. It was very fortunate that we did this because there were many bugs and issues that had

completely escaped us during development. We discuss the bugs that we found in the “Bugs and Feedback” section of this paper.

During our interview with Susan Andrews she mentioned to us that her school was making a change to a system called Standards-Based Learning. In that system, teachers strive away from grading students numerically, and instead focus on their effectiveness to accomplish a task and their improvement. We wanted to allow teachers to grade using text instead, if that was what they preferred, but we did not like that option to be constantly available, cluttering the interface. This inspired us to create an options feature, where users can define options that would persist whenever they use docAssist. One of these options is whether you will do your grading with numbers or with text.

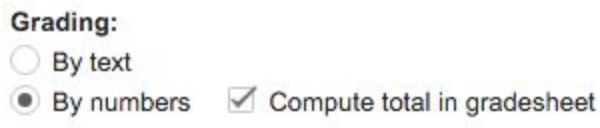


Figure 14: Grading Options

As can be seen in figure 14, we also allow teachers to choose whether or not they would like their grades to be totalled. This stemmed from conversations we had with Tosca Necoechea and Lisa Hogan in which she told us that her school used a similar system to Standards-Based learning, where different skills were given numeric values, but they didn’t have any sort of cardinal significance. If they choose not to total the grades, we simply don’t add a total column to their gradesheet.

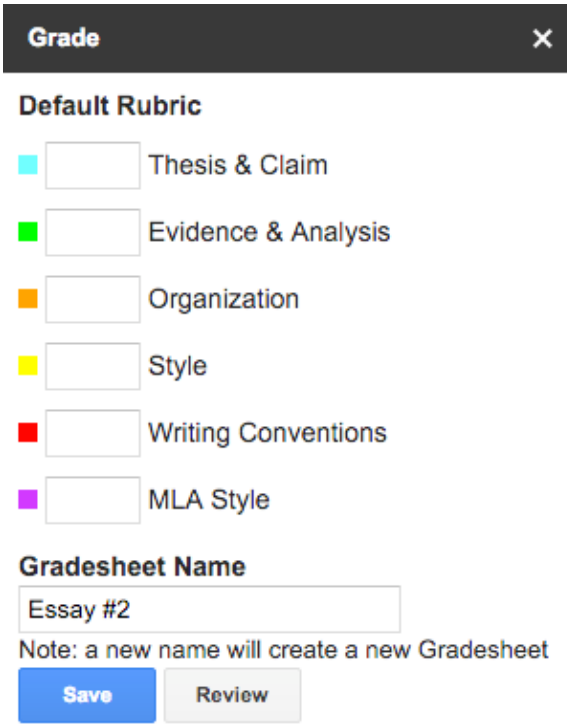


Figure 15: Grading tab with text fields

Options

As mentioned above, we wanted teachers to be able to set defaults. That way, whenever they start up docAssist and use any of our features, they won’t have to change anything if it’s something they always want. Besides grading, we came up with a few other cases in which teachers would want some sort of smart default. We had been wanting to make docAssist available internationally, but we recognized that it may frustrate non-English speaking users. For that reason, we decided to take on the ambitious task of providing language options. We had also been considering allowing teachers to leave audio notes for their students. Because we knew teachers would have a preference for audio or text notes, we wanted to give teachers the option of choosing the default.

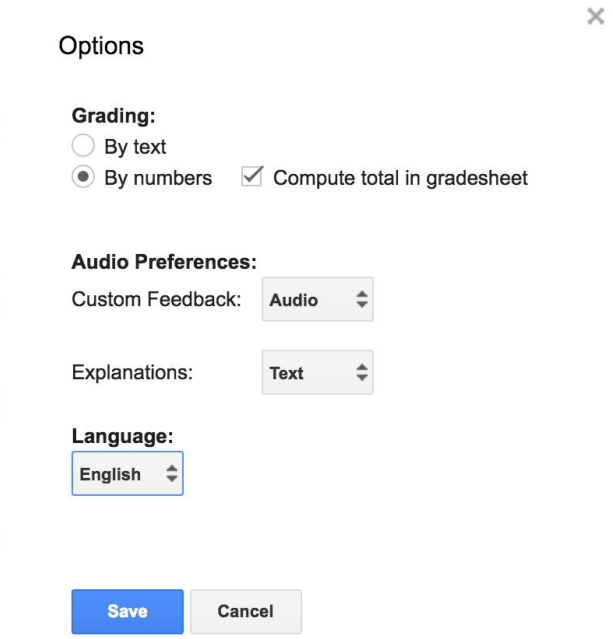


Figure 16: Options Panel

In figure 16, we show the available options. The first, grading, gives teachers the choice to grade by text or numbers, giving them the additional choice to total the scores if they pick the latter. The audio preferences allow the user to pick whether or not they want to give Audio or Text feedback by default.

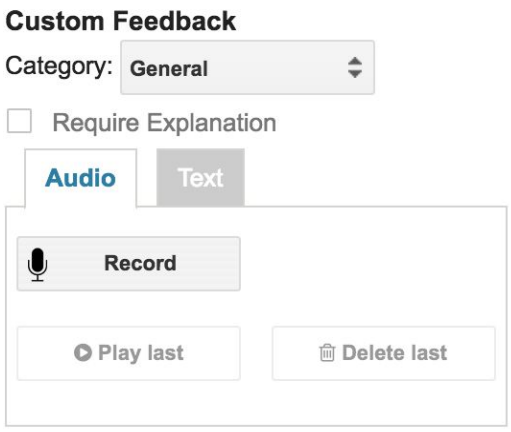


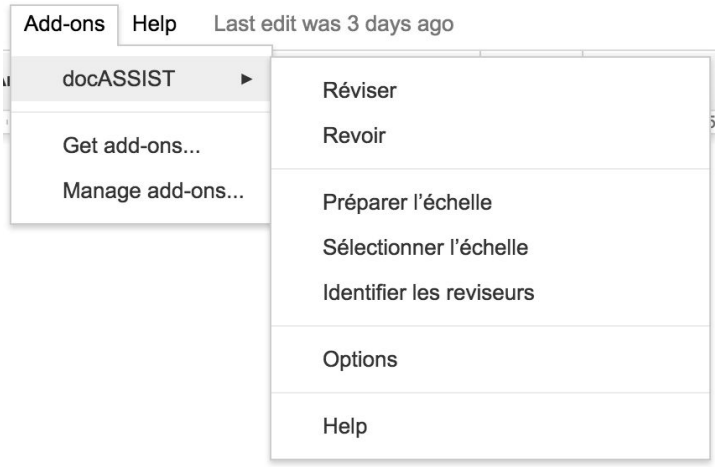
Figure 17: Audio selected as default Custom Feedback[1]

The last option in this list is the language option where, currently, users can pick English, Spanish, French, and Arabic. In our implementation of making rubrics publicly available, we considered adding an option to let the user decide if their rubrics are visible or hidden by default. However, because we wanted to push users to share their rubrics, we always default to visible rubrics, and allow them to change them on a rubric-by-rubric basis.

Languages

When we decided to add language options, we knew and understood that it would be a challenge. This is especially true because we recognized that translations provided by any external service would not be guaranteed to be clean and in-context. To provide appropriate translations, we would need people who speak the languages in question as well as they do English and have a good understanding of the system. In this case, the people most suited to do this task were us, the developers[2]. We knew it would cut into development time, but the translations needed to be to our standard if we wanted to release them.

[1] Audio Feedback was developed by Nick McMahon.
[2] Christian Roberts is a native Spanish speaker and Jean Marc Touma a native Arabic speaker with French as a second language.



We wanted the localization to be automatic so, if the user had never set a language option, we detect the language of their google docs and set their default language for docAssist to that. In the case that we don't support the language, we'll default to English. If the user had gone into the options tab and set the language to any other language, docAssist will always default to that choice in every document they use it on. The full list of translations is viewable in the function "translate()" in the Languages.gs section of Appendix C.

Figure 18: The docAssist tab translated into French.

Rubric Manager

Some of the features that existed prior to us working on docAssist, such as the Select Rubric and Rubric Setup features, felt disjointed and unintuitive. They feel like they should do similar things, however they have fundamentally different functionality. As a result, when we were thinking of how to implement Rubric Sharing, we weren't sure where it would fit in in the workflow of docAssist. We realized it would be ideal to centralize all things rubric related; thus, the Rubric Manager was born. We overhauled the previous system, replacing the separate Rubric Setup and Select Rubric menu items with a single Rubric Manager menu item. In the Rubric Manager, there are four tabs: Select, Create & Edit, Get, and Share. These tabs cover the functionality of the features we replaced as well as integrating the Sharing features.

Select Tab

To replace Select Rubric, we needed the Rubric Manager to be just as quick and simple. The Rubric Manager also needed to handle the user's list of shared rubrics and/or attaching rubrics with a code. To make sure that the workflow remained smooth, the Select Tab is the first tab to be displayed when opening the Rubric Manager. If a user already has a rubric attached to the document, however, the Create & Edit tab will be displayed first.

The idea behind the Select Tab was that it would be as similar as going through Select Rubric as possible. We had originally intended to have them be the same size, but when we created the tabs, it felt more natural if the tab contents were all larger than the original Select Rubric size. Part of the reason that the new size was necessary was the fact that it felt better to have the Shared Rubrics list in its own space. On the left, we still load up My Rubrics as we used to, and on the right, we load the Shared Rubrics with an option to include a rubric using its Share Code (more on that in the Get Tab and Share Tab sections of the paper).

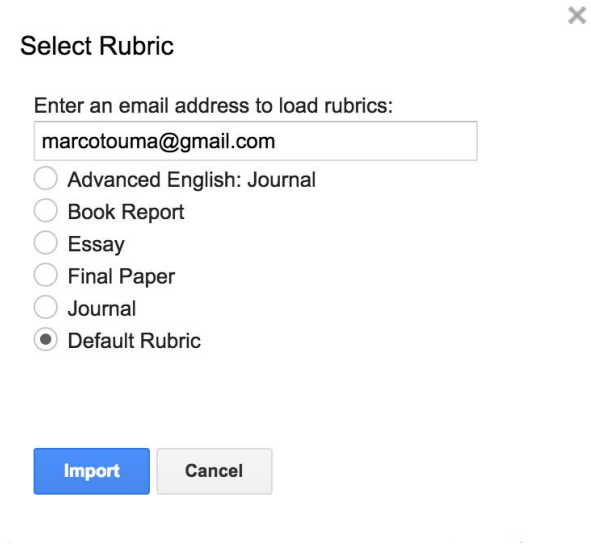


Figure 19: Old Select Rubric Window

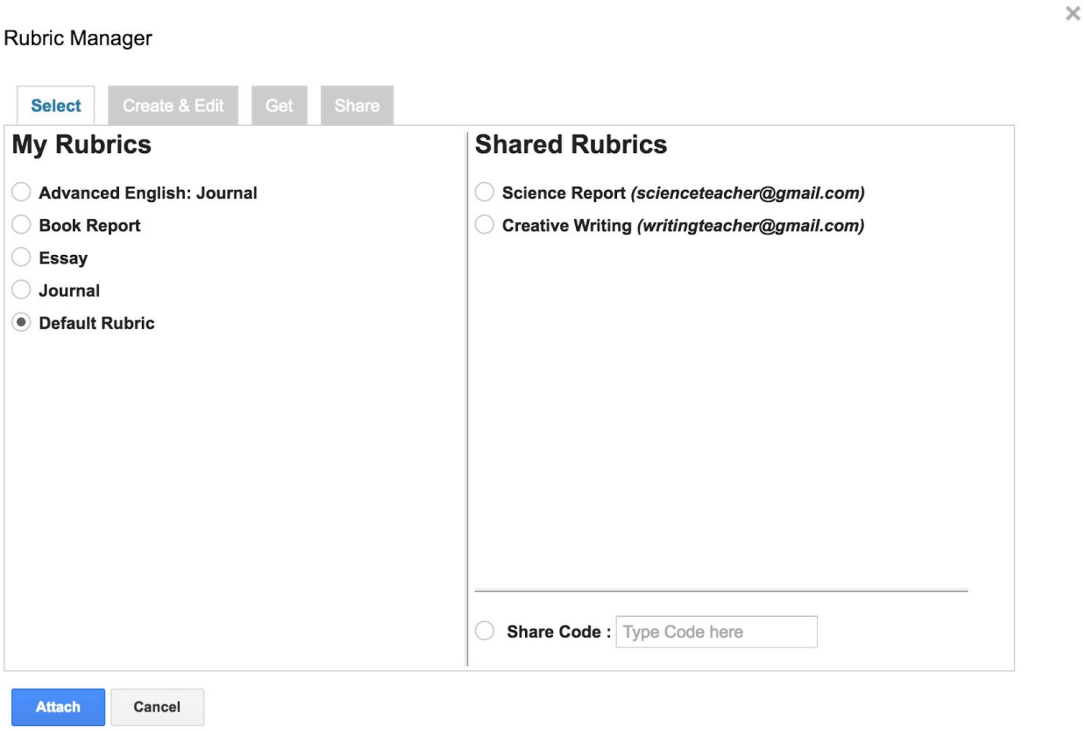


Figure 20: Rubric Manager - Select Tab

Create & Edit Tab

The Create & Edit tab was designed to replace the entire functionality of Rubric Setup with a few added features to streamline the workflow. First, we added a Create New button that starts a new rubric from a very basic template. The other thing we did was remove the need to double-click on a category or feedback to edit it, and instead made that option always available.

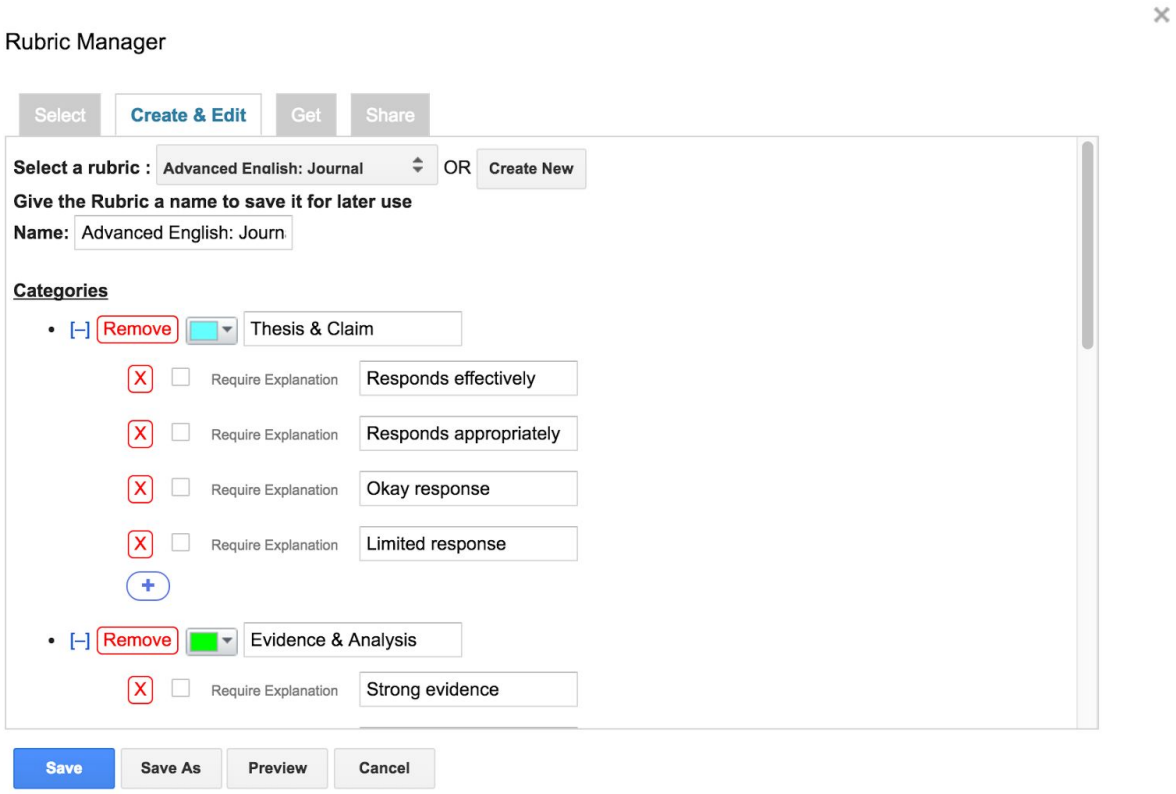


Figure 21: Rubric Manager - Create & Edit Tab

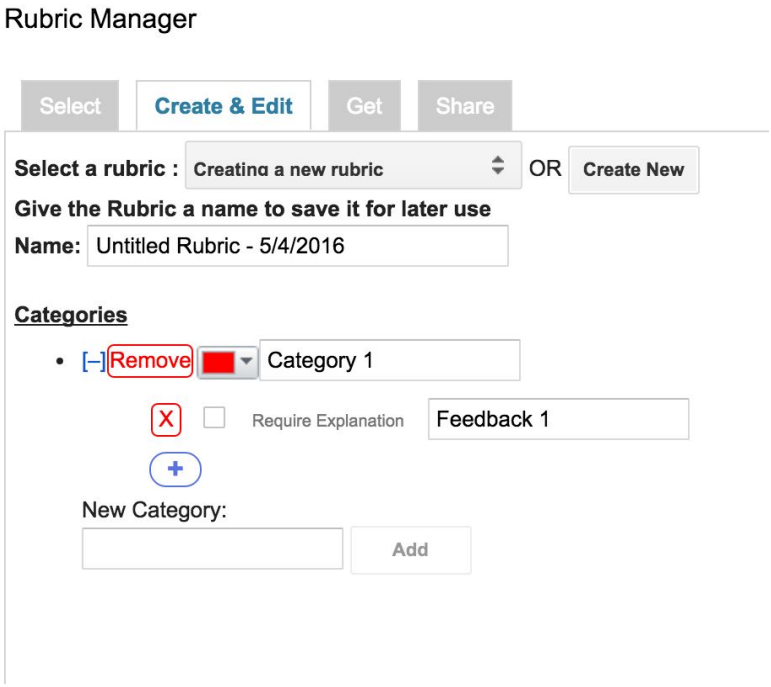


Figure 22: Create New Default Format

Get Tab

To implement Rubric Sharing, we needed to give the user the ability to get rubrics from other users and to decide what rubrics they would like others to see. To get rubrics from others, we created the Get Tab of the Rubric Manager. From here, the user can type in the email of the user that they want to get emails from, and a list of that other user's visible rubrics will appear. The user can then preview these rubrics, add them to their Shared Rubrics list, or make copies and add them to their My Rubrics list. They also have the option of doing any of these things with a Share Code, a unique code corresponding to a rubric.

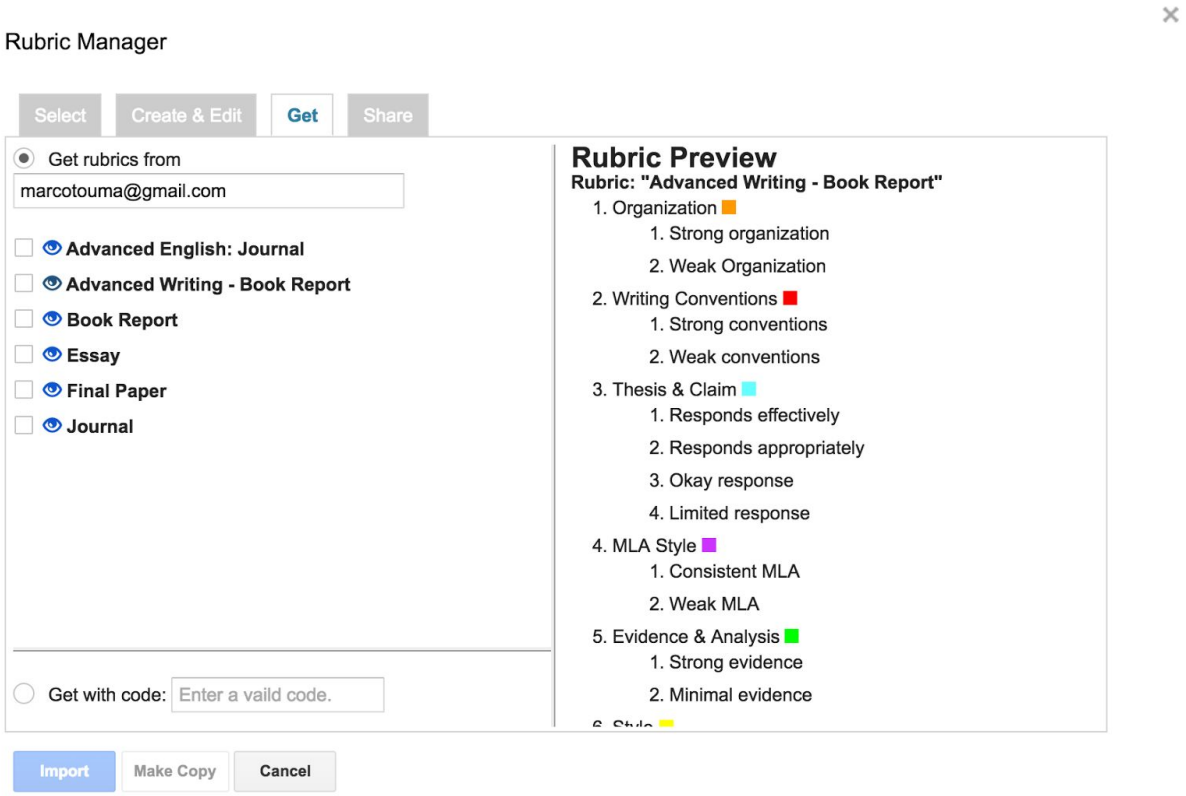


Figure 23: Rubric Manager - Get Tab. A rubric is being previewed on the right.

Share Tab

In the Share Tab, the user can manage their own rubrics and decide whether or not they want other people to be able to find them. Here, each rubric can be toggled between publicly visible or hidden. Also, the Share Codes for each rubric can be found here. A user can send another user a Share Code if they want that other user to use their rubric. There are two reasons for why a user would share their rubric this way. Firstly, it is more convenient than having to give the other user your email and having them sort through all of your visible rubrics. The second thing that we find great about these Share Codes is that a user can give another user their rubric without needing to make it publicly visible.

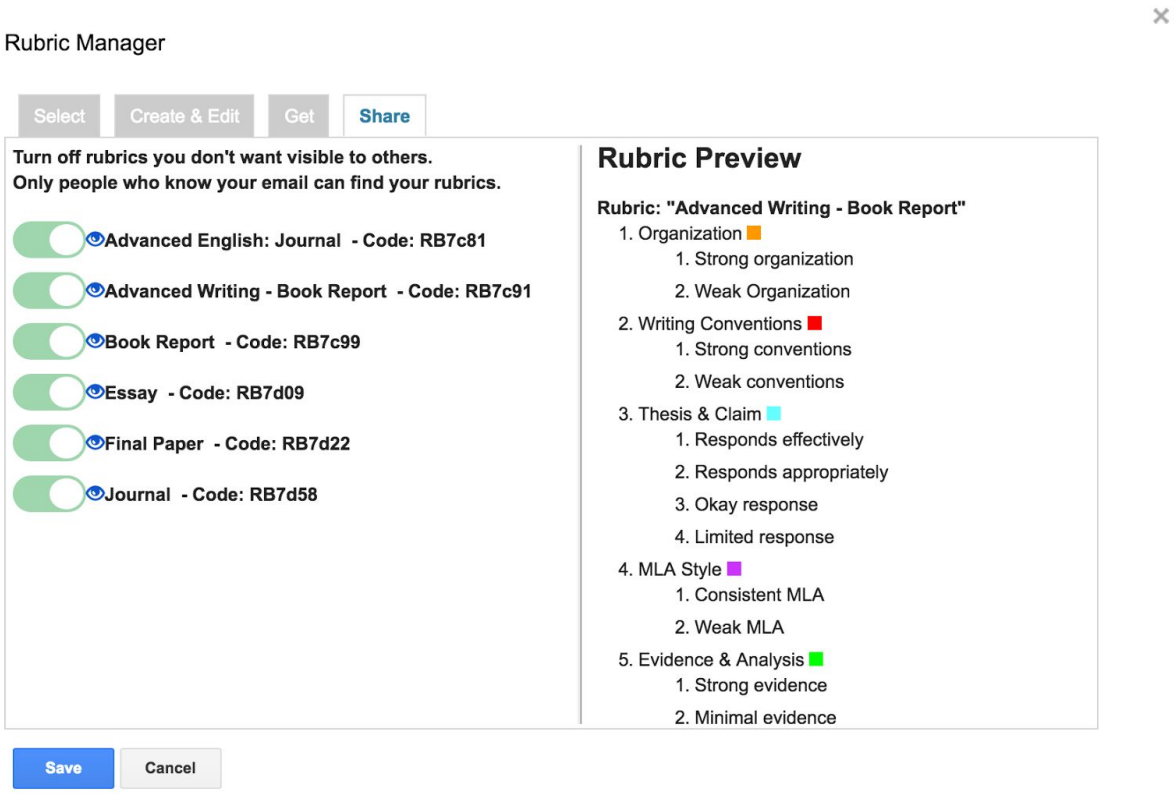


Figure 24: Rubric Manager - Share Tab.

User Relations

User Feedback

Throughout our year of development, we've been receiving emails and user reviews. For the most part, these pieces of feedback tend to be bug reports or simple questions on how to use docAssist. Some of the key bug reports and requests that we received were:

- Rubric Preview was being attached twice to the same document.
- docAssist was not loading properly in some cases.
- Changes made to Category names and feedbacks in Rubric Setup would not save.
- Certain characters (such as "\") could not be in any part of a rubric.
- A link to the Gradesheet should appear after grading.
- Users should be able to hide feedback notes on the document (for printing purposes).
- If text with feedback associated with it was deleted, docAssist would throw an error when looking for the feedback.
- When creating a rubric, docAssist would sometimes throw a message intended for debugging.

- There should be a separate place for saving grades that students can't edit.
- If the Summary was deleted, the Update Summary button wouldn't bring it back.

We responded to these users as soon as we could and, in most cases, were able to resolve any issues they were having.

The docAssist Website

To help users get familiar with docAssist, we maintain a website that can be found at http://docassist.assistments.org/. In Appendix B, we show the tutorial slides that we made for the main page of the website.

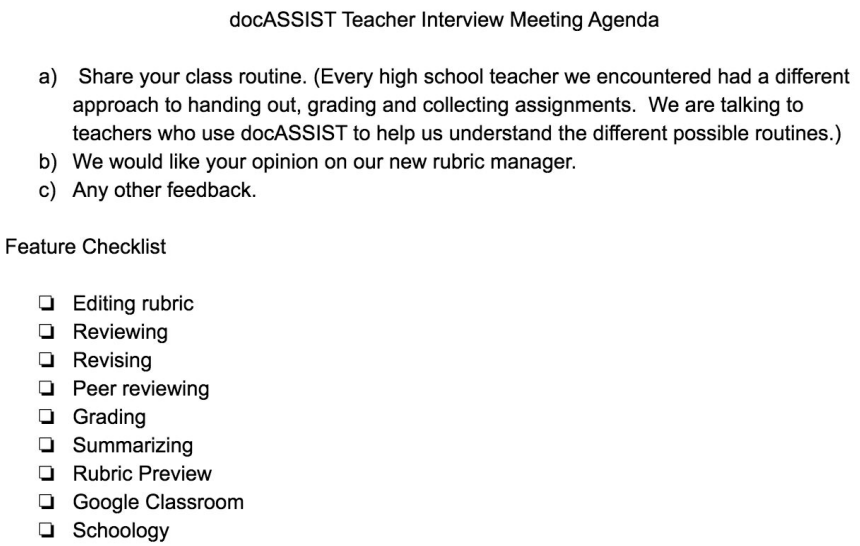
Usage Data

We wanted to reach out to some of our expert users to get feedback from people familiar with the add-on. We didn't want to send out surveys as the data they provide can be unreliable. Instead, we wanted to determine who our expert users are by some criteria and interview them personally. Because we do not keep a record of every time a rubric is used, we could not use frequency as a way to determine user expertise. We wrote a datastore query (an inquiry into the database) that would go through all of our user's rubrics and determine which users have the most rubrics. Once we got the list of most active users, we sent them an email, asking if they could meet in person or via Google Hangout.

To run the query, simply navigate to the following link to use your browser
https://assistmentsdocfeedback.appspot.com/docassist/api/query/mostRubrics/n
Where n is the number of users who have the most rubrics that you want returned. You must be authorized to run this query.

Teacher Interviews

We met with three teachers with the express purpose of getting feedback for any issues or new features they wanted to see addressed. We had an agenda we wanted to go through for each of those meetings.



docASSIST Teacher Interview Meeting Agenda

a) Share your class routine. (Every high school teacher we encountered had a different approach to handing out, grading and collecting assignments. We are talking to teachers who use docASSIST to help us understand the different possible routines.)
b) We would like your opinion on our new rubric manager.
c) Any other feedback.

Feature Checklist

- ❏ Editing rubric
- ❏ Reviewing
- ❏ Revising
- ❏ Peer reviewing
- ❏ Grading
- ❏ Summarizing
- ❏ Rubric Preview
- ❏ Google Classroom
- ❏ Schoology

Figure 25: Teacher Interview Meeting Agenda

We had three main points we wanted addressed during those meetings. First, we wanted to know the teacher's specific workflow, disregarding docAssist. That way, we could know if what we envisioned as we developed our tool met the necessities of our users. The second topic to be discussed was our Rubric Manager which was still in its early stages of its development. We would show the teachers a mockup of the new feature so we could get their opinions and ask whether or not they believed it would be useful to them. Finally, we wanted to see if they had any other feedback they personally wanted to express to us. If they didn't bring any of them up during the meeting, we would ask the teachers about the features in the "Feature Checklist" of Figure 19.

The following are the main points from each of those meetings:

Interview with Susan Andrews

Ms. Andrews uses schoology to assign work to her students. Her students have access to copies of her rubrics, handed out in class and available online as a pdf. She uses a traditional rubric, with large, wordy categories, and we were very surprised at the way in which she translated it into docAssist. She had typed the entirety of the rubric as her rubric in docAssist, resulting in very long pieces of feedback that made remarks on the entirety of the paper. In our work, we had expected users to write very short pieces of feedback intended for specific parts of the paper. For turning in assignments, Ms. Andrews has students write their papers in a Google Doc and save that document in a shared Google Drive with their name in the document's name; this is an ideal environment for integrating docAssist.

We realized that, with the way Ms. Andrews used docAssist, there was a lot of work that she put into it that, at times, would go to waste. When creating a rubric, if she had written the feedbacks and/or categories out of order, she would have to delete one of them and rewrite it in the correct place. This gave us the idea to add a drag-and-drop feature in the Rubric Manager,

more details on the feature are available in the “Conclusion and Future Development” section of this paper.

At the meeting, Ms. Andrews let us know that our (new at the time) grading feature did not support decimals. A teacher could not give a non-integer grade, a problem that was fairly easy to resolve that we didn’t catch prior to release. She also told us that, while she was interested in our peer review feature, implementing peer reviews in a classroom setting was very difficult. We talk about potential improvements to make the peer review feature more accessible in the “Conclusion and Future Development” section.

Interview with Lisa Hogan

In Ms. Hogan’s classroom, her students use iPads to write and submit their assignments. The assignments are written in the iPad Google Doc app. Google Docs for iPads does not support the use of add-ons, so we realized that this could make teachers believe that they can’t use docAssist. This, however, is not the case; teachers can use docAssist for grading on a computer regardless of what platform the assignment was written on. The only limitation that presents itself when students use iPads is that they can’t use any of the docAssist functionality intended for student use (Revision and Peer Review). The students can see the Rubric Preview as well as the Summary, even when they don’t have docAssist.

Ms. Hogan mentioned to us that the Revise feature was also unintuitive, saying that teachers have a hard time knowing what students can or need to do. This made us realize that our add-on did not provide enough information about its use and that showed us the importance of having tutorials available.

The way Ms. Hogan used our rubrics made excellent use of the dynamic nature of the add-on. She had students turn in the assignments in parts with the understanding that she would only pay attention to specific details of their papers instead of the product as a whole. For example, she would tell students to write their paper integrating figurative language and grade their submission using only that criteria. That way, she would have students resubmit their papers several times while only focusing on a specific aspect on any given submission. For each of these submissions, she would use a rubric tailored to her expectations, using a master rubric only for final submissions. She was very pleased not only with the way docAssist made that easier for her, but also how having those readily available tailored rubrics helped her concentrate on grading what she cared about.

Ms. Hogan, being a teacher at a Standards-Based learning oriented school, didn’t like the fact that the grading feature only allowed for numerical grades. She suggested that we allow grading by text. It was from this conversation that we really got the ball rolling on making text-based grading available. Not all LMS systems are very friendly towards Standards-Based grading, so she suggested that we look into connecting to systems such as PowerSchool and JumpRope.

Interview with Tosca Necoechea

Tosca Necoechea is a Spanish teacher for Berkeley High in California. For this reason, we were very excited to speak with her given she represents a very interesting use case for

docAssist. Not only does she teach in a language that we were interested in integrating, but she was also teaching non-native speakers of the language. We found this exciting because it meant her approach to docAssist would involve more helping students develop a familiarity with the language instead of improving their general writing skills. This was apparent in the rubrics she made in docAssist. She gave feedback with short, simple notes under every category. She had categories intended to give notes on the entirety of the paper and others meant to point out simple issues in specific parts of the writing.

Rubric: "Anécdota"

Conjugación: 1. Error 2. Éxito
Normas de escritura: 1. Número/Género 2. Acento 3. Ortografía 4. Mayúscula/Minúscula 5. Puntuación 6. Infinitivo 7. Palabra que falta 8. Orden de palabras 9. Imperfecto/Pretérito 10. Idiomas 11. Modelos de oraciones
Anecdota: 1. Excelente - Todos los elementos de la anécdota están. Si hay cambios, no hay errores. El pretérito y el imperfecto se usan bien. 2. Bueno - Tres de cuatro de los elementos de la anécdota están bien. Hay pocos errores. Si hay cambios, tienen sentido. 3. Más o menos - El autor ignora los elementos de la anécdota. Hay errores y la anécdota empieza no tener sentido. 4. Pobre

Figure 26: A rubric made by Tosca Necoechea. Note that a category can be as simple as Error/Success or you can have an overarching category with feedbacks as reminders.

In her classes, Ms. Necoechea uses Google Classroom to assign work to her students. She would create a Google Doc and attach one of her rubrics using our Rubric Preview feature; Google Classroom would then make a copy of that document for each of her students. She didn't, however, use Google Classroom for assignment submissions, she would receive those and organize them separately. Once she graded an assignment, she sent it back to the students and had them revise the paper using docAssist.

Tosca shared with us some bugs with our newer features. Firstly, she let us know that our Rubric Preview was problematic and not always available in the case that another Preview had been attached earlier. This was an issue with the fact that the function that attached the Preview did not check if one had been previously attached. Another bug she mentioned to us was that a deleted comment would not always unformat the page correctly, often leaving a

footnote or some colored text. Tosca also mentioned to us that the grading feature was not allowing for decimals.

In terms of grading, however, Tosca let us know that she really enjoyed the feature. One piece of feedback she gave us was that the option to opt out of having the grades automatically totalled should be available. We found it very interesting that, although she did like to use numbers for grading, she did not like totalling them. Though she did use a numeric grading system, she also suggested that we allow the use of words in grading to track skills.

Conclusion and Future Development

After our work on docAssist, we are very happy with the product we have built. However, there is still much that we feel can be done with the add-on, and we would like to address some of those ideas here. The first feature that we would really like future developers to implement is the ability to drag-and-drop categories and subcategories when creating or editing rubrics. Currently, if a user does not like the order that they have made the elements of a rubric in, they have to jump through hoops to fix it.

Another feature we'd love to see implemented is a tab system, such as the one in the Rubric Manager, to switch between the Review and Grade sidebars. We received a question from a user who mentioned that they did not like to Summarize their comments prior to putting in some of the grades. The example she gave us was if part of the rubric was dedicated solely to the introduction, she would like to put in the grade right after she gave feedback. For that reason, the tab system would be great to allow users to be filling in the grades as they go along and to not force them through an order that they may feel is suboptimal.

Finally, the last feature that would make docAssist great is a Rubric Store. This was an idea that we had had very early in development but, because there was so much other work to be done and because there was not the infrastructure for it, we never had the time to put it together. The concept behind this feature would be to allow users to search a database of popular rubrics. The Store would look very much like a typical application store, with filters, suggestions, some "Developer's Top Picks", and a search function. This will encourage users to build very clean rubrics and keep them from doing work that others have already done. It will also foster a sense of pride within the users. Our implementation of Rubric Sharing created the infrastructure necessary for this feature, so it is definitely our first step towards its development.

Appendix A: General Information

Important Links

- Add-ons Market Listing
    - https://chrome.google.com/webstore/detail/docassist/niencjmhincjchnncioldoihpgkgfejb
- docASSIST Help Website
    - https://sites.google.com/site/assistmentsfeedbacktool/

Contact Information

For support, questions, or feedback, please email docassist@wpi.edu.

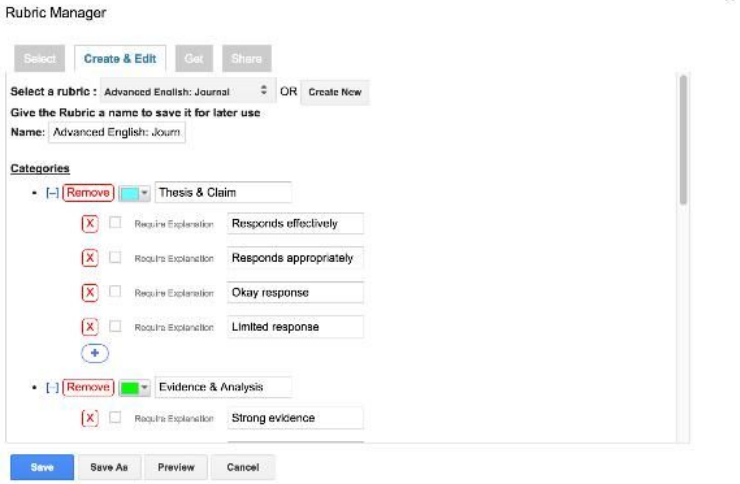
Appendix B: docAssist: A Tutorial Slides

docASSiST
by ASSiSTments
A Free Public Service of Worcester Polytechnic Institute

A Tutorial

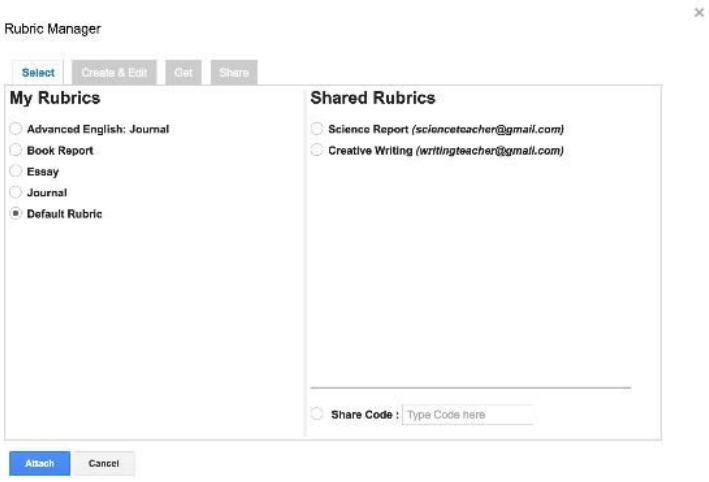
Rubric Manager: Create & Edit

- Create rubrics here.
- Create New button starts a new rubric from scratch.
- Press the blue "+" to add subcategories and the Add Category button to add new Categories.
- Save button will save your Rubric to your My Rubrics list.



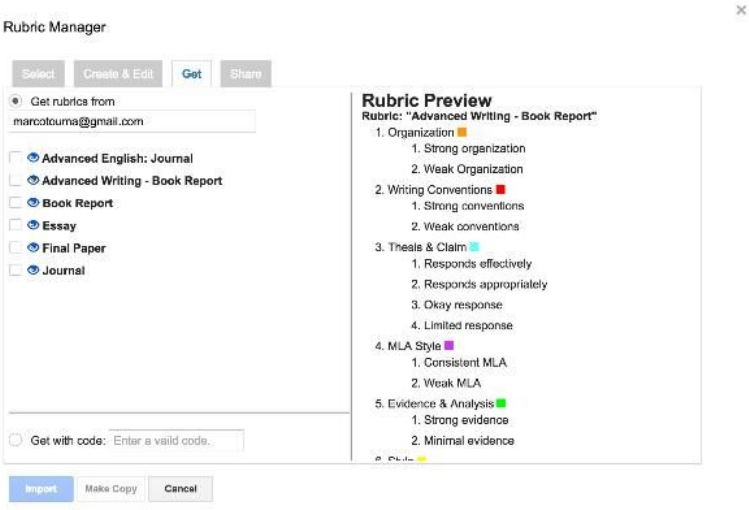
Rubric Manager: Select

- Select the rubric you want to attach to the document.
- If someone has given you their Share Code, you can use it here.



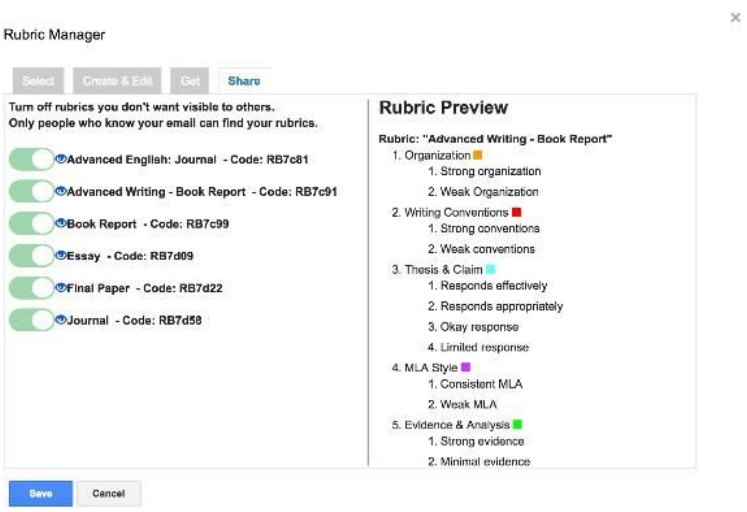
Rubric Manager: Get

- Type in the email of the person whose rubrics you want to use.
- You can import the rubrics to your Shared Rubrics list or copy them to your My Rubrics list.
- You can also get a rubric with a Share Code.
- Click the eye to see a preview of the rubric.



Rubric Manager: Share

- Toggle your rubrics off if you want to hide them from others.
- You can find the Share Codes for your rubrics here. A rubric does not need to be visible to use its Share Code.



Rubric Preview

- A rubric preview is generated automatically at the bottom of the document whenever you attach a rubric.

Default Rubric

Thesis & Claim:
1. Responds effectively
2. Responds appropriately
3. Okay response
4. Limited response

Evidence & Analysis:
1. Strong evidence
2. Minimal evidence

Organization:
1. Strong organization
2. Weak Organization

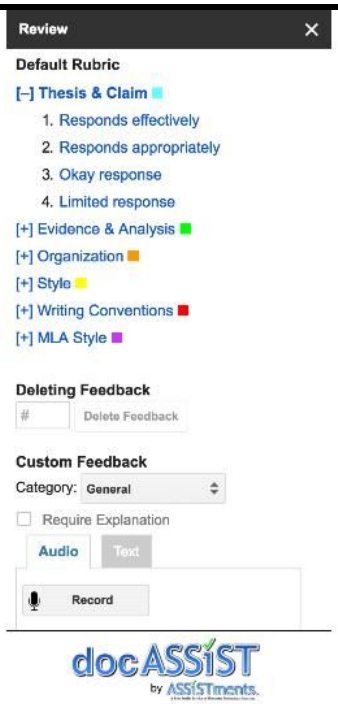
Style:
1. Effective style
2. Weak Style

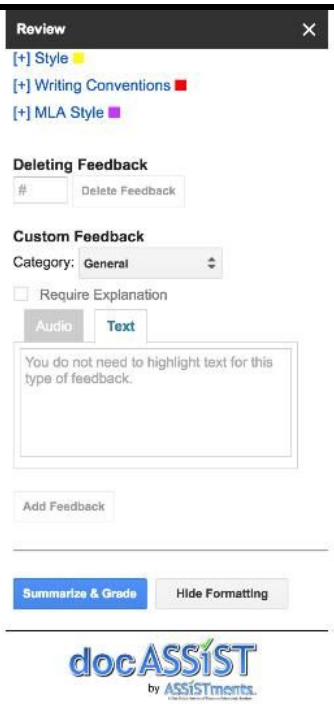
Writing Conventions:
1. Strong conventions
2. Weak conventions

MLA Style:
1. Consistent MLA
2. Weak MLA

Review

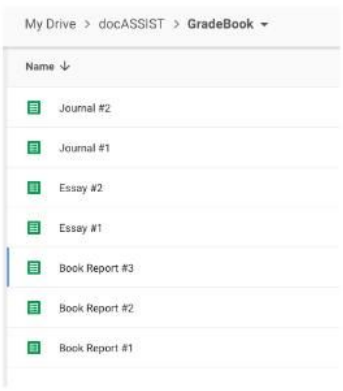
- Highlight text and choose the feedback you would like to make.
- You can make custom audio or text feedback.
- Once you're done giving feedback, click Summarize & Grade to insert a feedback summary.
- Hide Formatting hides your feedback.

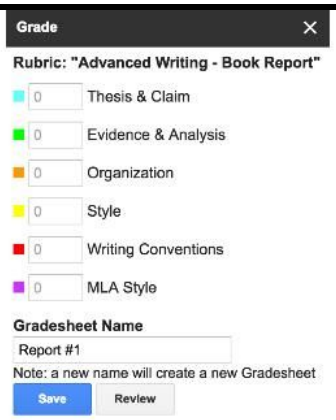


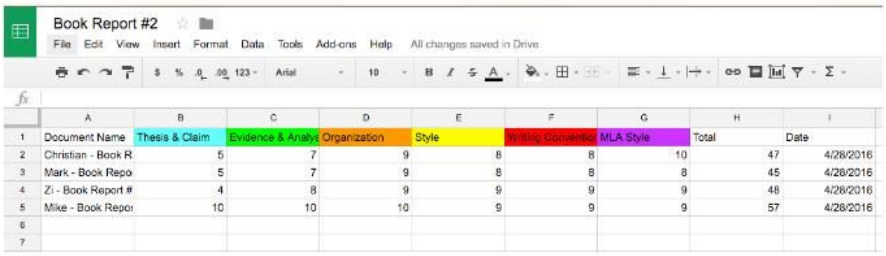


Grade

- Type in the points students earned in each section.
- A Gradesheet will be generated in your Google Drive.
- You can also grade with text by changing the setting in the Options menu.

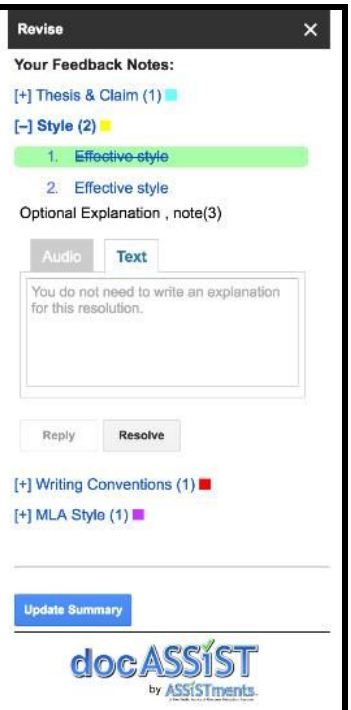






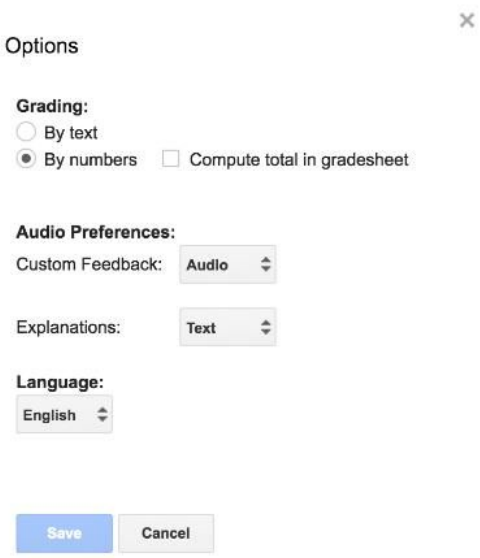
Revise

- All feedbacks will be shown on the right.
- The writer can go through each feedback and reply to/resolve feedbacks.
- Replies can be made by text or audio.
- If the reviewer requires explanation, a feedback cannot be resolved without a reply.



Options

- Preferences will be saved here for whenever you open docAssist.
- You can choose to grade by numbers or text and whether or not to total the grades.
- You can choose the default feedback type (audio or text).
- You can select your language here.



Appendix C: Source Code

Modified Code

Code.gs

```
/**
 * Runs when a document is opened.
 */
function onOpen(e) {
  var menu = DocumentApp.getUi()
      .createAddonMenu()
      .addItem(translate('Review', e), 'showSidebar')
//      .addItem('Grade', 'showGrading')
      .addItem(translate('Revise', e), 'showPeerReview')
      .addSeparator()
      .addItem(translate('Rubric Manager', e), 'showFullRubricManager')
      .addItem(translate('Rubric Setup', e), 'showRubric')
      .addItem(translate('Select Rubric', e), 'promptForImport')
      .addItem(translate('Show Reviewers\' Identity', e), 'showReviewerEmails')
      .addSeparator()
      .addItem(translate('Options', e), 'promptForOptions')
//      .addSeparator()
//      .addItem('DEVELOPER-Delete Comments', 'deleteAllComments')
//      .addItem('DEVELOPER-Delete Summary Key', 'deleteSummaryKey')
//      .addItem('DEVELOPER-Delete Rubric', 'deleteRubric')
//      .addItem('DEVELOPER-Delete Saved User Rubrics', 'deleteUserRubrics')
//      .addItem('DEVELOPER-Delete Email Key', 'deleteEmailKey');
//      .addItem('DEVELOPER-Delete Grading Syntax', 'deleteGradingSyntax')
//    .addItem('DEVELOPER-Delete Rubric Selection', 'deleteSelectedRubric')
//    .addItem('DEVELOPER-Log Summary', 'highlightSum');
      ;
  menu.addToUi();
}

/**
 * Action for on install
 */
function onInstall(e) {
  onOpen(e);
}

/**
 * Clears formatting in a document.
 */
function clearFormatting() {
  var comments = getAllComments();
  comments.forEach( function (comment) {
    comment.unformat();
  });
  PropertiesService.getDocumentProperties().setProperty("FormattingState", "Cleared");
}

/**
 * Reapplys formatting in a document
 */
```

```
function restoreFormatting() {
  var comments = getAllComments();
  var colors = getRubricColors();
  var doc = DocumentApp.getActiveDocument();
  comments.forEach( function (comment) {
    if (!comment.resolved){
      comment.format(colors.getColor(comment.ccss), doc);
    }
  });
  PropertiesService.getDocumentProperties().setProperty("FormattingState", "Formatted");
}

/**
 * Gets the docASSIST folder in the user's drive. If one does not exist, it is created.
 *
 * @return {Folder} The docASSIST folder
 */
function getDriveFolder() {
  var folders = DriveApp.getFoldersByName("docASSIST");
  if (folders.hasNext() == true){
    return folders.next();
  } else {
    return DriveApp.createFolder("docASSIST");
  }
}

/**
 * DEVELOPER USE
 * Deletes the last email list key
 */
 function deleteEmailKey(){
   PropertiesService.getDocumentProperties().deleteProperty("ASSISTmentsLastEmailList");
 }

/**
 * DEVELOPER USE
 * Deletes the grading Syntax preference
 */
 function deleteGradingSyntax(){
   PropertiesService.getUserProperties().deleteProperty("GradingSyntax");
 }

/**
 * DEVELOPER USE
 * Deletes the rubric that is saved within the document.
 */
function deleteRubric(){
  PropertiesService.getDocumentProperties().deleteProperty("ASSISTmentsRubric");
  PropertiesService.getDocumentProperties().deleteProperty("ASSISTmentsRubricOwner");
  PropertiesService.getDocumentProperties().deleteProperty("ASSISTmentsRubricColorMap");
}
/**
 * DEVELOPER USE
 * Deletes the saved rubrics for a user.
 */
function deleteUserRubrics(){
  PropertiesService.getUserProperties().deleteProperty("ASSISTmentsUserRubrics");
}

/**
```

```
 * DEVELOPER USE
 * Delets the key that indicates a summary has been generated
 */
function deleteSummaryKey(){
  PropertiesService.getDocumentProperties().deleteProperty("ASSISTmentsSummary");
}

/**
 * DEVELOPER USE
 * Delets the key that indicates a summary has been generated
 */
function deleteSelectedRubric(){
  PropertiesService.getDocumentProperties().deleteProperty("ASSISTmentsRubric");
}

/**
 * Shows the sidebar for Peer Review
 */
function showPeerReview() {
  var html = HtmlService.createTemplateFromFile('peerReview');

  var rubric = getRubric();
  if (rubric == null){
    promptForImport("showPeerReview");
    return;
  }

  var secs = rubric.getKeys();
  if (getFilteredComments("ccss", "ASSISTMENTS_GENERAL_FEEDBACK").length > 0){
    secs.push("ASSISTMENTS_GENERAL_FEEDBACK");
  }

  html.sections = secs;
  var evalHtml = html.evaluate()
      .setSandboxMode(HtmlService.SandboxMode.IFRAME)
      .setTitle(translate('Revise'))
      .setWidth(400);
  DocumentApp.getUi()
      .showSidebar(evalHtml);
  PropertiesService.getDocumentProperties().setProperty("currentSidebar", "Revise");
}

/**
 * Shows the sidebar to add a comment
 */
function showSidebar() {

  var html = HtmlService.createTemplateFromFile('addComment');
  var rubric = getRubric();
  if (rubric == null){
    promptForImport("showSidebar");
    return;
  }
  html.categories = rubric;
  var evalHtml = html.evaluate()
      .setSandboxMode(HtmlService.SandboxMode.IFRAME)
      .setTitle(translate('Review'))
      .setWidth(400);
  DocumentApp.getUi()
      .showSidebar(evalHtml);
```

```
  PropertiesService.getDocumentProperties().setProperty("currentSidebar", "Review");


}

/**
 * Shows the sidebar to grade
 */
function showGrading() {
  var gradingSyntax = PropertiesService.getUserProperties().getProperty("GradingSyntax");
  var html = HtmlService.createTemplateFromFile('grading');
  var rubric = getRubric();
  if (rubric == null){
    promptForImport("showGrading");
    return;
  }
  html.isNumeric = (gradingSyntax == "Numbers") || (gradingSyntax == null);
  html.categories = rubric;
  var evalHtml = html.evaluate()
      .setSandboxMode(HtmlService.SandboxMode.IFRAME)
      .setTitle(translate('Grade'))
      .setWidth(400);
  DocumentApp.getUi()
      .showSidebar(evalHtml);
  PropertiesService.getDocumentProperties().setProperty("currentSidebar", "Grade");
}

/**
 * Reveals the identity of each feedback person.
 * This is only allowed by people who can edit the rubric.
 */
function showReviewerEmails(){
  var authors = getAllRubricAuthors();
  if (authors.indexOf(Session.getEffectiveUser().getEmail()) == -1 && authors.length != 0){
    var html = HtmlService.createHtmlOutputFromFile('rubric-error');
    var evalHtml = html.setSandboxMode(HtmlService.SandboxMode.IFRAME)
                       .setTitle(translate("Reviewer Emails"))
                       .setHeight(200)
                       .setWidth(800);
    DocumentApp.getUi().showModalDialog(evalHtml, translate("Reviewer Emails"));
  } else {
    var html = HtmlService.createTemplateFromFile('author-list');
    html.authors = getCommentAuthors(getAllComments());
    var evalHtml = html.evaluate()
        .setSandboxMode(HtmlService.SandboxMode.IFRAME)
        .setTitle(translate("Reviewer Emails"))
        .setWidth(500)
        .setHeight(250);
  }
  DocumentApp.getUi().showModalDialog(evalHtml, translate("Reviewer Emails"));
}

/**
 * Saves the recently used emails as a document property
 * @param {string[]} emails A list of email addresses.
 */
 function saveEmailList(emails){
   if (emails.length > 0)
      PropertiesService.getDocumentProperties().setProperty("ASSISTmentsLastEmailList",
emails.join("\n"));
```

```
 }


/**
 * Shows the popup Rubric generator
 */
function showRubric(){
  var authors = getAllRubricAuthors();
  if (authors.indexOf(Session.getEffectiveUser().getEmail()) == -1 && authors.length != 0){
      promptForImport("showRubric");
  } else {

    var html = HtmlService.createTemplateFromFile('rubric');
    var rubric = getRubric();
    var colors = getRubricColors();
    if (rubric == null || colors == null){
      rubric = getDefaultRubric();
      colors = getDefaultRubricColors();
    }
    html.rubric = rubric;
    html.map = colors;
    html.authors = getOtherRubricAuthors();
    var evalHtml = html.evaluate()
        .setSandboxMode(HtmlService.SandboxMode.IFRAME)
        .setTitle('Rubric Setup')
        .setHeight(500)
        .setWidth(800);
    DocumentApp.getUi().showModalDialog(evalHtml, "Rubric Setup");
  }
}

/**
 * Here is the default rubric used if a custom one is not implemented
 */
function getDefaultRubric(){
  var defaultRubric = new Rubric();
  defaultRubric.addCategory("Thesis & Claim")
               .addCategory("Evidence & Analysis")
               .addCategory("Organization")
               .addCategory("Style")
               .addCategory("Writing Conventions")
               .addCategory("MLA Style");

  defaultRubric.addComment("Thesis & Claim", "Responds effectively", false)
               .addComment("Thesis & Claim", "Responds appropriately", false)
               .addComment("Thesis & Claim", "Okay response", false)
               .addComment("Thesis & Claim", "Limited response", false);

  defaultRubric.addComment("Evidence & Analysis", "Strong evidence", false)
               .addComment("Evidence & Analysis", "Minimal evidence", false);

  defaultRubric.addComment("Organization", "Strong organization", false)
               .addComment("Organization", "Weak Organization", false);

  defaultRubric.addComment("Style", "Effective style", false)
               .addComment("Style", "Weak Style", false);

  defaultRubric.addComment("Writing Conventions", "Strong conventions", false)
               .addComment("Writing Conventions", "Weak conventions", false);
```

```
  defaultRubric.addComment("MLA Style", "Consistent MLA", false)
               .addComment("MLA Style", "Weak MLA", false);

  return defaultRubric;
}

/**
 * Here is the default rubric color map used if a custom one is not implemented
 */

function getDefaultRubricColors(){
  var defaultMap = new RubricColorMap();
  defaultMap["Thesis & Claim"] = "#66ffff";
  defaultMap["Evidence & Analysis"] = "#00ff00";
  defaultMap["Organization"] = "#ff9900";
  defaultMap["Style"] = "#ffff00";
  defaultMap["Writing Conventions"] = "#ff0000";
  defaultMap["MLA Style"] = "#cc33ff";
  return defaultMap;
}

/**
 * Reports an error to the user.
 * @param {string} error to report
 */

 function reportError(string){
   DocumentApp.getUi().alert(string);
 }

 /**
  * Checks if the email is in the correct format
  * @param {email} the email to be validated
  */

  function validateEmail(email){
      try{
         DocumentApp.getActiveDocument().addViewer(email);
         if(email == "test")
           return false;
         return true;
      }
      catch(e){
         return false;
      }
  }

  /**
   * Opens the options menu
   */
  /**
   * Prompts the user to select their options.
   * Save options to user properties
   * @param {string} func Name of function to call at the end
   */
  function promptForOptions(){
    //if (PropertiesService.getDocumentProperties().getProperty("ASSISTmentsRubric") ==
null){
      var html = HtmlService.createTemplateFromFile("options");
      var evalHtml = html.evaluate()
```

```
          .setSandboxMode(HtmlService.SandboxMode.IFRAME)
          .setTitle(translate("Options"))
          .setHeight(350)
          .setWidth(325);
      DocumentApp.getUi().showModalDialog(evalHtml, translate("Options"));
  }

/**
   * Prompts the user to import a rubric or use the default.
   * Saved the rubric to the document properties
   * @param {string} func Name of function to call at the end
   */
  function showRubricManager(func){
    if (PropertiesService.getDocumentProperties().getProperty("ASSISTmentsRubric") == null){
      var html = HtmlService.createTemplateFromFile("managerSelectRubric");
      html.func = func;
      var evalHtml = html.evaluate()
          .setSandboxMode(HtmlService.SandboxMode.IFRAME)
          .setTitle(translate("Select Rubric"))
          .setHeight(250)
          .setWidth(325);
      DocumentApp.getUi().showModalDialog(evalHtml, translate("Select Rubric"));
    } else {
      showFullRubricManager();
    }
  }

/**
 * Shows the popup Rubric generator
 */
function showFullRubricManager(func, value){
  var authors = getAllRubricAuthors();
  if (authors.indexOf(Session.getEffectiveUser().getEmail()) == -1 && authors.length != 0){
      promptForImport("showRubric");
  } else {
    var html = HtmlService.createTemplateFromFile('rubricManager');
    html.func = func;
    var rubric = getRubric();
    var colors = getRubricColors();
    if (rubric == null || colors == null){
      rubric = getDefaultRubric();
      colors = getDefaultRubricColors();
    }
    html.rubric = rubric;
    if(value){html.rubric = getUserRubric(value);}
    html.map = colors;
    html.authors = getOtherRubricAuthors();
    var evalHtml = html.evaluate()
        .setSandboxMode(HtmlService.SandboxMode.IFRAME)
        .setTitle(translate('Rubric Manager'))
        .setHeight(500)
        .setWidth(800);

    DocumentApp.getUi().showModalDialog(evalHtml, translate("Rubric Manager"));
  }
}
```

Comment.gs

```
/**
 * Describes the Comment object.
 * Comments can be constracted using the given parameters.
 * If an author is not supplied, it will be determined from the Session.
 * @constructor
 * @param {string} text Text of the comment
 * @param {string} ccss Category of the comment (according to the Rubric)
 * @param {string} author Email address of the person who wrote the comment.
 * @param {number} id Optional. For use with testing.
 */
var Comment = function (text, ccss, req, author, id){
  //ID logic to ensure unique
  if (id == null){
    var tempId = Math.random() * 1000000;
    while (getFilteredComments("id", tempId).length != 0){
      tempId = Math.random() * 1000000;
    }
  } else {
    tempId = id;
  }
  this.id = tempId;
  //Range of text associated with the comment
  this.range == null;
  //Bookmark ID for link reference. Set along with range.
  this.bookmark;
  //Number for reference
  this.refNum = null;
  //text of comment
  this.text = text;
  //A link to an audio file, if the comment contains audio.
  this.audioId;
  //Common Core State Standard
  this.ccss = ccss;
  //Determines if the comment requires a response
  this.req = req;
  //Is this comment resolved?
  this.resolved = false;
  //author's email
  this.author;
  //author's explanations
  //array of explanation objects
  this.explanations = [];
  if(author == null){
    this.author = Session.getEffectiveUser().getEmail();
  } else {
    this.author = author;
  }
}

Comment.prototype = {
  /**
   * Highligets the text associated with the comment
   */
  highlight: function() {
```

```
    DocumentApp.getActiveDocument().setSelection(this.getRange());
  },

  /**
   * Gets the range of text that a comment refers to.
   * @return {Range} Text associated with a comment
   */
  getRange: function() {
    return DocumentApp.getActiveDocument().getNamedRangeById(this.range).getRange();
  },

  /**
   * Sets a comment's range based on the currently selected text
   * Saves the range ID in the comment.
   * @param {boolean} isPositive Is the comment positive? (Used for formatting)
   * @return {Comment} This comment, for chaining
   */
   setRange: function(doc) {
     if (doc == null){
       doc = DocumentApp.getActiveDocument();
     }
     var color = getRubricColors().getColor(this.ccss);
     var range = doc.getSelection();
     if(range == null){
       this.range = null; //Range is null if an overall comment
//       this.bookmark = null;
     } else {
       this.range = doc.addNamedRange("ASSISTments Comment", range).getId();
       var elts = doc.getNamedRangeById(this.range).getRange().getRangeElements();
       if (elts[0].isPartial()){
         var position = doc.newPosition(elts[0].getElement(), elts[0].getStartOffset());
       } else {
         var position = doc.newPosition(elts[0].getElement(), 0);
       }
       this.bookmark = doc.addBookmark(position).getId();
       this.format(color, doc);
     }
     return this;
   },

  /**
   * @override
   * @return {string} The text of a comment
   */
  toString: function(){
    return this.text;
  },

  /**
   * Summary string for reports. Includes a note number
   * @return {string} String of the comment for reports
   */
  toStringSummary: function() {
    var string = this.toString();
    if(this.range && this.refNum){
      string += " (Note " + this.refNum + ")";
    }
    return string;
  },
```

```
  /**
   * Gets the note reference string;
   * @return {string} The note number formatted like '(Note ##)'. Returns 1 space string if
not available (Prevents problems adding empty elements)
   */
  getNotation: function () {
    if(this.range){
      return "(Note " + this.refNum + ")";
    }
    return "(General Note " + this.refNum + ")";
  },

  /**
   * Formats the text in the given range with the given color.
   * This is used for showing where in a document a comment is.
   * @param {string} color Color to highlight text. (Hex color code)
   */
  format: function(color, doc){
    if (doc == null){
      doc = DocumentApp.getActiveDocument();
    }
    if (!this.range){
      return;
    }
    var namedRange = doc.getNamedRangeById(this.range);
    if (!namedRange){
      return;
    }
     var elements = namedRange.getRange().getRangeElements();
     var lastElement;
     for (var i = 0; i < elements.length; i++) {
       var element = elements[i];
       // Only modify elements that can be edited as text; skip images and other non-text
elements.
       if (element.getElement().editAsText) {
         var text = element.getElement().editAsText();
         lastElement = element;

         // Bold the selected part of the element, or the full element if it's completely
selected.
         if (element.isPartial()) {
           text.setBackgroundColor(element.getStartOffset(),
element.getEndOffsetInclusive(), color);
         } else {
           text.setBackgroundColor(color);
         }
       }
     }
     //Everything below adds a superscript number to the selection
     var text = lastElement.getElement().editAsText();
     var pos = lastElement.getEndOffsetInclusive() + 1;
     var position = DocumentApp.getActiveDocument().newPosition(lastElement.getElement(),
pos);
     if (this.refNum == null){
       this.refNum = getNextCommentNumber();
     }
     var numLength = this.refNum.toString().length;

     //Look for existing reference number. If there is none, then add it.
     var textString = text.getText().substr(pos - numLength, numLength);
```

```
    if (textString.indexOf(this.refNum) == -1) {
      text.insertText(pos, " " + this.refNum); //added + " "
      text.setTextAlignment(pos, pos + numLength, DocumentApp.TextAlignment.SUPERSCRIPT);
    }


    doc.setCursor(position);
  },

 /**
  * Unformats the text in the given range.
  * This is used when resolving a comment
  */
 unformat: function(shouldRemoveBookmark){
   var doc = DocumentApp.getActiveDocument();

   if (!this.range){
     Logger.log("No range available for comment " + this.id);
     return;
   }

   var namedRange = doc.getNamedRangeById(this.range);

   if (!namedRange){
     Logger.log("Could not find range associated with comment " + this.id);
     return;
   }
    var elements = namedRange.getRange().getRangeElements();

    var lastElement;
    for (var i = 0; i < elements.length; i++) {
      var element = elements[i];
      // Only modify elements that can be edited as text; skip images and other non-text
elements.
      if (element.getElement().editAsText) {
        var text = element.getElement().asText().editAsText();
        lastElement = element;

        // Unhighlight the selected part of the element, or the full element if it's
completely selected.
        if (element.isPartial()) {
          text.setBackgroundColor(element.getStartOffset(),
element.getEndOffsetInclusive(), "#ffffff");
        } else {
          text.setBackgroundColor("#ffffff");
        }
      }
    }

    var text = lastElement.getElement().asText().editAsText();
    var numLength = this.refNum.toString().length;

    //Determine the starting position and length of the text in question
    var startingPos = 0;
    var textLength = text.getText().length;
    if (lastElement.isPartial()){
      startingPos = lastElement.getStartOffset();
      textLength = lastElement.getEndOffsetInclusive() - startingPos + 1;
    }
```

```
    //Trim the text to only search our desired location and then search for the refNum in
superscript
    //This search starts from the end of the string and works its way forward
    var searchString = text.getText().substr(startingPos, textLength);
    var result = searchString.lastIndexOf(this.refNum);
    var searchIndex = startingPos + result;
    while (result != -1) {
      if (text.getTextAlignment(searchIndex) == DocumentApp.TextAlignment.SUPERSCRIPT){
        //Delete the extra space if we find it
        if (text.getText().charAt(searchIndex - 1) == ' '){
          text.deleteText(searchIndex - 1, searchIndex + numLength - 1);
        } else {
          text.deleteText(searchIndex, searchIndex + numLength - 1);
        }
        break;
      }
      result = searchString.lastIndexOf(this.refNum, result);
      searchIndex = startingPos + result;
    }

    if (shouldRemoveBookmark){
      var bookmark = DocumentApp.getActiveDocument().getBookmark(this.bookmark);
      if (bookmark) {
        try {
          bookmark.remove();
        } catch (e) {
          Logger.log(e.message);
        }
      }
    }
  },
}

/**
 * Makes a comment with the given text and category.
 * @param {string} value The text of the comment
 * @param {string} cat The category (ccss) of a comment.
 * @param {bool} req Does this require a response?
 */
function makeComment(value, cat, req){
  var range = DocumentApp.getActiveDocument().getSelection();
  if (range != null){
    saveComment(new Comment(value, cat, req).setRange());
  } else {
    DocumentApp.getUi().alert("Please select the text to associate with this comment");
  }
}

/**
 * Makes a custom comment that is not bound to a category.
 * @param {string} text Text of the comment
 * @param {string} cat Category that the comment is associated with
 * @param {boolean} req True if the comment requires a response
 */
 function makeCustomComment(text, cat, req){
   saveComment(generateCustomComment(text, cat, req));
 }

 /**
```

```
  * Generates a new custom comment
  * @param {string} text Text of the comment
  * @param {string} cat Category that the comment is associated with
  * @param {boolean} req True if the comment requires a response
  */
 function generateCustomComment(text, cat, req){
   var range = DocumentApp.getActiveDocument().getSelection();
   if(text.trim() == ""){
     DocumentApp.getUi().alert("Please enter text to comment.");
     return null;
   }
   var comment;
   if (cat == "General"){
     comment = new Comment(text, "ASSISTMENTS_GENERAL_FEEDBACK", req);
   } else {
     comment = new Comment(text, cat, req);
   }

   comment.refNum = getNextCommentNumber();

   if (range != null){
     comment.setRange(null);
   }

   return comment;
 }

 /**
  * Makes an audio comment that may or may not be bound to a category.
  * @param {string} encodedAudio Base64 encoded audio file (mp3)
  * @param {string} cat Category that the comment is associated with
  * @param {boolean} req True if the comment requires a response.
  * @return The reference number of the created comment.
  */
function makeAudioComment(encodedAudio, cat, req){
  var comment = generateCustomComment("Audio feedback", cat, req);
  comment.audioId = storeAudio(encodedAudio, comment.refNum);
  saveComment(comment);
  return comment.id
}

/**
 * Deletes a previously made comment by reference number
 * @param {string} refNum Reference number of comment to delete
 */
function deleteComment(refNum){
  var comment = getCommentByRefNum(refNum);
  //If doesn't exist
  if (!comment) {
    DocumentApp.getUi().alert("There is no comment " + refNum + ".");
  }
  //If no permission
  if (comment.author != Session.getEffectiveUser().getEmail()){
    DocumentApp.getUi().alert("You can't delete comments you're not the author of.");
  }

  if(comment.range){
    var bookmark = DocumentApp.getActiveDocument().getBookmark(comment.bookmark);
    if(bookmark){
      bookmark.remove();
```

```
    }

    comment.unformat(true);
  }

  if (comment.audioId){
    deleteAudioFileWithId(comment.audioId);
  }

  deleteCommentByRefNum(refNum);
}
/**
 * Sets the selection to the given range
 * @param {string} id Comment id to highlight
 */
function highlightRange(id){
  var comment = getFilteredComments("id", id)[0];
  if (comment.range == null){
    Logger.log("No range assocrated with Comment ID: " + id);
    return;
  }
  var namedRange = DocumentApp.getActiveDocument().getNamedRangeById(comment.range);
  if (namedRange == null){
    Logger.log("No range assocrated with Comment ID: " + id);
    return;
  }
  DocumentApp.getActiveDocument().setSelection(namedRange.getRange());
}

/**
 * Get the authors from the comment array
 * @param {string} array Array of comments
 * @return {string[]} Array of authors that made the comments
 */
function getCommentAuthors(array){
  var authors = [];
  for (var i = 0; i < array.length; i++){
    if(authors.indexOf(array[i].author) == -1){
      authors.push(array[i].author);
    }
  }
  return authors;
}
```

Rubric.gs

```
/**
 * This file describes what a rubric is. There is no default constructor, rather,
 * we supply methods to build up a rubric. Rubrics will be stored using XML,
 * so this seemed to make sense.
 * @constructor
 */

 function Rubric() {
```

```
this.responseRequired = {};
/**
 * Adds an empty category to a rubric.
 * @param {string} name Name of the category
 * @throws {Error} if the category already exists
 */
this.addCategory = function(name){
  if (Object.keys(this).indexOf(name) != -1){
    throw new Error("Category \"" + name + "\" already exists!");
  }
  if (Object.keys(new Rubric()).indexOf(name) != -1
      || name == "ASSISTMENTS_GENERAL_FEEDBACK"
      || Object.keys(new RubricColorMap()).indexOf(name) != -1){
    throw new Error(cat + " is reserved for system use!");
  }
  this[name] = [];
  this.responseRequired[name] = [];
  return this;
}

/**
 * Adds a comment to a give category.
 * @param {string} cat Category to add the rule to.
 * @param {string} name Name of the rule (e.g. Period, Comma, Run-on)
 * @param {boolean} required Does this comment require a response?
 */
this.addComment = function(cat, name, required){
  if(Object.keys(new Rubric()).indexOf(cat) != -1) {
    throw new Error(cat + " is reserved for system use!");
  }
  if (this[cat] == null){
    throw new Error("Category " + cat + " does not exist.");
  }
  this[cat].push(name);
  this.responseRequired[cat].push(required);
  return this;
}

/**
 * Creates an XML document and returns it so that the rubric can be stored.
 * @return {Document} XML Document representing the Rubric.
 */
this.toXML = function(){
  var root = XmlService.createElement("Rubric");
  var cats = this.getKeys();
  Logger.log("We HAVE to get here");
  for (var i = 0; i < cats.length; i++){
    Logger.log(cats[i]);
    var elt = XmlService.createElement("Category").setText(cats[i].trim());
    for(var j = 0; j < this[cats[i]].length; j++){
      var item = XmlService.createElement("Item").setText(this[cats[i]][j].trim());
      item.setAttribute("required", this.responseRequired[cats[i]][j]);
      elt.addContent(item);
    }
    root.addContent(elt);
  }
  return XmlService.createDocument(root);
}
```

```
   /**
   * Gets the name of the rubric.
   *@return {String} Rubric name.
   */
   this.getName = function(){
     return this.name;
   }

   /**
    * Returns all of the keys of the rubric (category names) while
    * omitting the keys for functions.
    * @return {string[]} Set of rubric categories.
    */
    this.getKeys = function(){
      var originalKeys = Object.keys(new Rubric());
      var result = [];
      var allKeys = Object.keys(this);
      Logger.log(allKeys);
      for (var i = 0; i < allKeys.length; i++){
        if (originalKeys.indexOf(allKeys[i]) == -1){
          result.push(allKeys[i]);
        }
      }
      return result;
    }

 }

 /**
  * Generates a rubric given the categories and their rules. Saves the rubric.
  * @param {string[]} cats Categories for the given rubric.
  * @param {string[][]} rules An array of arrays. Each array is a ruleset for a given
category.
  *                      They index of the ruleset must match the index of the category.
  * @param {string[]} authors Email addresses of those that can edit the rubric.
  * @param {string[]} colors Colors for each rubric category
  * @param {string} name Name of the rubric
  * @param {boolean[][]} required An array of arrays. Each array is a rulset for a category.
  *                       The index of the requirement matches the index of the subcategory.
  *                       Determines if a comment requires a response.
  */
 function makeRubric(cats, rules, authors, colors, name, required){
   Logger.log(cats[0] + rules[0] + name);
   var rubric = new Rubric();
   var map = new RubricColorMap();
   for (var i = 0; i < cats.length; i++){
     rubric.addCategory(cats[i]);
     map.addColorMapping(cats[i], colors[i]);
     for(var j = 0; j < rules[i].length; j++){
       rubric.addComment(cats[i], rules[i][j], required[i][j]);
     }
   }

   if(name != null && name.trim() != ""){
     Logger.log("Making a user rubric");
     makeUserRubric(rubric, map, authors, name.trim());
   }

   Logger.log("Saving Rubric Data");
   saveRubricData(rubric, map, authors, name.trim());
```

```
   showSidebar();
   generateRubricPreview();
 }

 /**
  * Format the list of authors to be put into a Property
  * @param {string[]} authors An array of author email addresses.
  * @return String to store in Property
  */
  function authorToString(authors){
    var string = "";
    for (var i = 0; i < authors.length - 1; i++){
      string += authors[i] + "\n";
    }
    string += authors[authors.length - 1];
    return string;
  }

 /**
  * Returns an array of author email addresses from the Property string
  * @param {string} string String to be converted to author array
  * @return Author array
  */
  function stringToAuthor(string){
    return string.split("\n");
  }

 /**
  * Gets all of the rubric authors.
  * @return An array of author email addresses that can edit the rubric.
  */
  function getAllRubricAuthors(){
    if(PropertiesService.getDocumentProperties().getProperty("ASSISTmentsRubricOwner") ==
null){
      return []
    }
    return
stringToAuthor(PropertiesService.getDocumentProperties().getProperty("ASSISTmentsRubricOwner
"));
  }


 /**
  * Gets the authors of the Rubric. Excludes current user.
  * @return An array of author email addresses that can edit the rubric
  */
  function getOtherRubricAuthors(){
    if(PropertiesService.getDocumentProperties().getProperty("ASSISTmentsRubricOwner") ==
null){
      return []
    }
    var authors =
stringToAuthor(PropertiesService.getDocumentProperties().getProperty("ASSISTmentsRubricOwner
"));
    var index = authors.indexOf(Session.getEffectiveUser().getEmail());
    if (index != -1){
      authors.splice(index, 1);
    }
    return authors;
  }
```

```
/**
 * Loads a User rubric
 * @param {string} name Name of the rubric to load
 */
function loadUserRubric(name, manager){
  var rubric = getUserRubric(name);
  var map = getUserRubricColorMap(name);
  var authors = getUserRubricAuthors(name);
  var idx = authors.indexOf(Session.getEffectiveUser().getEmail());

  if (idx != -1){
    authors.splice(idx, 1);
  }
  var html = HtmlService.createTemplateFromFile('rubric');
  if (manager != null){
    html = HtmlService.createTemplateFromFile('rubricManager');
  }
  html.rubric = rubric;
  html.map = map;
  html.authors = authors;
  html.loadedRubric = name;
  var evalHtml = html.evaluate()
      .setSandboxMode(HtmlService.SandboxMode.IFRAME)
      .setTitle('Rubric Setup')
      .setHeight(500)
      .setWidth(800);
  DocumentApp.getUi().showModalDialog(evalHtml, "Rubric Setup");
}
```

RubricImport.gs

```
var RECENT_OWNERS_PROP = "ASSISTmentsRecentRubricOwners";

/**
 * Prompts the user to import a rubric or use the default.
 * Saved the rubric to the document properties
 * @param {string} func Name of function to call at the end
 */
function promptForImport(func){
  if (PropertiesService.getDocumentProperties().getProperty("ASSISTmentsRubric") == null){
    var html = HtmlService.createTemplateFromFile("rubricTool");
    html.func = func;
    var evalHtml = html.evaluate()
        .setSandboxMode(HtmlService.SandboxMode.IFRAME)
        .setTitle("Select Rubric")
        .setHeight(250)
        .setWidth(325);
    DocumentApp.getUi().showModalDialog(evalHtml, "Select Rubric");
  } else {
    var html = HtmlService.createTemplateFromFile("rubricReset");
    if (func == null){
      func = "promptForImport";
```

```
      }
      html.func = func;
      var evalHtml = html.evaluate()
          .setSandboxMode(HtmlService.SandboxMode.IFRAME)
          .setTitle("Reset Rubric?")
          .setHeight(175)
          .setWidth(325);
      DocumentApp.getUi().showModalDialog(evalHtml, "Reset Rubric?");
    }
  }

/** ayy */

function requestRubricAttachment(name, func){
  var email = Session.getEffectiveUser().getEmail();
  if (PropertiesService.getDocumentProperties().getProperty("ASSISTmentsRubric") != null){
  var html = HtmlService.createTemplateFromFile("rubricReset");
      if (func == null){
        func = "";
      }
      html.func = func;
      var evalHtml = html.evaluate()
          .setSandboxMode(HtmlService.SandboxMode.IFRAME)
          .setTitle("Reset Rubric?")
          .setHeight(175)
          .setWidth(325);
      DocumentApp.getUi().showModalDialog(evalHtml, "Reset Rubric?");
  }
  else{


      var rubricDataXML = fetchUserRubricData(name, email);
      var rubricXML = rubricDataXML.getChild("Rubric");
      if (rubricXML == null){
        throw new Error("Rubric data was not found");
      }

      var colorXML = getUserRubricColorMap(name, email).toXML();

      var authorXML = rubricDataXML.getChildren("Author");
      if (authorXML == null){
        throw new Error("Malformed XML: Authors missing");
      }
      var authors = [];
      for (var i = 0; i < authorXML.length; i++){
        authors.push(authorXML[i].getText())
      }

      var stringRubric = XmlService.getPrettyFormat().format(rubricXML);
      var stringColor = XmlService.getPrettyFormat().format(colorXML);
      var stringAuthor = authorToString(authors)

      importedRubric = XMLtoRubric(stringRubric);
      importedColorMap = XMLtoRubricColorMap(stringColor);

      PropertiesService.getDocumentProperties().setProperties({
        "ASSISTmentsRubricName": name,
        "ASSISTmentsRubricOwner": stringAuthor,
        "ASSISTmentsRubricColorMap": stringColor,
        "ASSISTmentsRubric": stringRubric,
```

```
      });
      var owners = PropertiesService.getUserProperties().getProperty(RECENT_OWNERS_PROP);
      if (owners == null || typeof owners == 'undefined'){
        var emailList = authorToString([email]);
      } else {
        var array = owners.split("\n");
        if (array.indexOf(email) == -1){
          array.push(email);
        }
        var emailList = authorToString(array);
      }
      PropertiesService.getUserProperties().setProperty(RECENT_OWNERS_PROP, emailList);
      if (func != null){
        eval(func+"()");
      }
     //showSidebar();
     generateRubricPreview();
   }

}
  /**
   * Imports and stores the rubric into the document properties.
   * @param {string} email Email address associated with the rubric.
   * @param {string} name Name associated with the rubric.
   * @param {string} func Name of function to call at the end
   */
   function requestRubric(email, name, func){
      var rubricDataXML = fetchUserRubricData(name, email);
      var rubricXML = rubricDataXML.getChild("Rubric");
      if (rubricXML == null){
        throw new Error("Rubric data was not found");
      }

      var colorXML = getUserRubricColorMap(name, email).toXML();

      var authorXML = rubricDataXML.getChildren("Author");
      if (authorXML == null){
        throw new Error("Malformed XML: Authors missing");
      }
      var authors = [];
      for (var i = 0; i < authorXML.length; i++){
        authors.push(authorXML[i].getText())
      }

      var stringRubric = XmlService.getPrettyFormat().format(rubricXML);
      var stringColor = XmlService.getPrettyFormat().format(colorXML);
      var stringAuthor = authorToString(authors)

      importedRubric = XMLtoRubric(stringRubric);
      importedColorMap = XMLtoRubricColorMap(stringColor);

      PropertiesService.getDocumentProperties().setProperties({
        "ASSISTmentsRubricName": name,
        "ASSISTmentsRubricOwner": stringAuthor,
        "ASSISTmentsRubricColorMap": stringColor,
        "ASSISTmentsRubric": stringRubric,
      });
      var owners = PropertiesService.getUserProperties().getProperty(RECENT_OWNERS_PROP);
      if (owners == null || typeof owners == 'undefined'){
        var emailList = authorToString([email]);
```

```
      } else {
        var array = owners.split("\n");
        if (array.indexOf(email) == -1){
          array.push(email);
        }
        var emailList = authorToString(array);
      }
      PropertiesService.getUserProperties().setProperty(RECENT_OWNERS_PROP, emailList);
      if (func != null){
        eval(func+"()");
      }
     showSidebar();
   }

   /**
    * Sets the rubric to our default templates.
    * @param {string} func Name of function to call at the end
    */
    function setRubricToDefault(func){
      var rubric = XmlService.getPrettyFormat().format(getDefaultRubric().toXML());
      var colors = XmlService.getPrettyFormat().format(getDefaultRubricColors().toXML());
      var authors = authorToString([Session.getEffectiveUser().getEmail()]);
      PropertiesService.getDocumentProperties().setProperty("ASSISTmentsRubricName",
"Default Rubric");
      PropertiesService.getDocumentProperties().setProperties({
        "ASSISTmentsRubricOwner": authors,
        "ASSISTmentsRubricColorMap": colors,
        "ASSISTmentsRubric": rubric,
        "ASSISTmentsRubricName": "Default Rubric",
      });

      if (func != null && func.trim() != ""){
        eval(func+"()");
      }
    }

   function requestSettingRubricToDefault(func){
     if(PropertiesService.getDocumentProperties().getProperty("ASSISTmentsRubric") != null){
       var html = HtmlService.createTemplateFromFile("rubricReset");
      if (func == null){
        func = "";
      }
      html.func = func;
      var evalHtml = html.evaluate()
          .setSandboxMode(HtmlService.SandboxMode.IFRAME)
          .setTitle("Reset Rubric?")
          .setHeight(175)
          .setWidth(325);
      DocumentApp.getUi().showModalDialog(evalHtml, "Reset Rubric?");
     }
     else{
      var rubric = XmlService.getPrettyFormat().format(getDefaultRubric().toXML());
      var colors = XmlService.getPrettyFormat().format(getDefaultRubricColors().toXML());
      var authors = authorToString([Session.getEffectiveUser().getEmail()]);
      PropertiesService.getDocumentProperties().setProperty("ASSISTmentsRubricName",
"Default Rubric");
      PropertiesService.getDocumentProperties().setProperties({
        "ASSISTmentsRubricOwner": authors,
        "ASSISTmentsRubricColorMap": colors,
        "ASSISTmentsRubric": rubric,
```

```
        "ASSISTmentsRubricName": "Default Rubric",
      });
      generateRubricPreview();
      if (func != null && func.trim() != ""){
        eval(func+"()");
      }
     }
   }
    /**
     * Removes an email from the ASSISTments recent rubric owners.
     * @param {string} email
     */
     function removeEmailFromRecent(email){
       if (email.trim() != ""){
         var owners = PropertiesService.getUserProperties().getProperty(RECENT_OWNERS_PROP);
         if (owners == null) {
           throw new Error("Could not get recent email list");
         } else {
           var array = owners.split("\n");
           var idx = array.indexOf(email);
           if (idx == -1){
             throw new Error("Could not find email " + email + " in recent rubric owner
list");
           } else {
             array.splice(idx, 1);
             if (array.length == 0){
               PropertiesService.getUserProperties().deleteProperty(RECENT_OWNERS_PROP);
             } else {
               PropertiesService.getUserProperties().setProperty(RECENT_OWNERS_PROP,
authorToString(array));
             }
           }
         }
       }
     }

     /**
      * Resets the rubric on the current document.
      * @param {string} [func] Function to resume
      */
     function resetRubric(removeSummary, func){

       var comments = getAllComments();
       comments.forEach(function(comment){
         comment.unformat(true);
       });

       if (removeSummary){
         removeRange(PropertiesService.getDocumentProperties().getProperty(SUMMARYKEY));
         removeRange(PropertiesService.getDocumentProperties().getProperty(PREVIEWKEY));
       }

       PropertiesService.getDocumentProperties().deleteAllProperties();
       if (func){
         eval(func+"()");
       }
     }
```

Storage.gs

```
/** Base URL for rubric saving */
var API_BASE_URL = "https://assistmentsdocfeedback.appspot.com/docassist/api/rubric/"
var COMMENT_BASE = "FeedbackNote";

/**
 * Get saved comments
 * @return {Comment[]} An array of comments saved in the properties
 */
function getSavedComments() {
  //Backwards compatibility
  var properties = PropertiesService.getDocumentProperties().getProperties();
  var keys = Object.keys(properties);

  var property = properties["ASSISTmentsComments"];
  if (property != null) {
    var doc = XmlService.parse(property);
    return convertXmlToJson(doc);
  }

  var feedbackKeys = getFeedbackKeys(keys);

  var comments = [];
  feedbackKeys.forEach(function(key) {
    comments.push(parseJson(properties[key], Comment.prototype));
  });

  return comments;
}

/**
 * Gets an object from JSON and inherits the given prototype
 * @param {String} json JSON representation of the object
 * @param {Object} prototype Prototype of the new object.
 * @return The object combining the JSON and the prototype
 */
function parseJson(json, prototype) {
  if (!json){
    return null
  }

  if (!prototype){
    return JSON.parse(json);
  }

  var obj = Object.create(prototype);
  var props = JSON.parse(json);
  var keys = Object.keys(props);
  keys.forEach(function(key) {
    obj[key] = props[key];
  });

  //Convert explanation objects to explanations, if they exist
  if (obj.explanations) {
    obj.explanations.forEach(function (exp, index, array) {
      var newExp = Object.create(Explanation.prototype);
```

```
        var keys = Object.keys(exp);
        keys.forEach(function(key) {
          newExp[key] = exp[key];
        });
        obj.explanations[index] = newExp;
      });
    }

    return obj;
  }

/**
 * Gets a comment by reference number
 * @param {Number} refNum Reference number of the comment
 * @return {Comment} the comment, or null if it does not exist
 */
 function getCommentByRefNum(refNum) {
    var commentJson = PropertiesService.getDocumentProperties().getProperty(COMMENT_BASE +
refNum);

    if (commentJson) {
      return parseJson(commentJson, Comment.prototype);
    }

    return null;
  }

/**
 * Deletes a comment by reference number
 * @param {Number} refNum Reference number of the comment
 */
 function deleteCommentByRefNum(refNum) {
    PropertiesService.getDocumentProperties().deleteProperty(COMMENT_BASE + refNum);
  }

/**
 * Get properties keys for feedback
 * @param {String[]} [keys] Optional. Keys to search through
 * @return {String[]} A string of feedback note keys in the Document Properties
 */
function getFeedbackKeys(keys) {
  if (!keys){
    keys = PropertiesService.getDocumentProperties().getKeys();
  }

  var feedbackKeys = keys.filter(function(element, index, array) {
    var regex = new RegExp("^" + COMMENT_BASE);
    return element.match(regex);
  });

  feedbackKeys.sort(function(first, second) {
    var regex = new RegExp("^" + COMMENT_BASE);
    var firstNum = Number(first.replace(regex, ""));
    var secondNum = Number(second.replace(regex, ""));
    return firstNum - secondNum;
  });

  return feedbackKeys;
}
```

```
/**
 * Converts XML comments to JSON and spreads them out in properties.
 * For backwards compatibility
 * @param {XmlDocument} doc Document to parse for comments.
 * @return {Comment[]} The comments in an array
 */
function convertXmlToJson(doc) {
  Logger.log("Converting legaxy XML comments to JSON style comments");
  var commentElements = doc.getRootElement().getChildren("Comment");
  var commentArray = [];
  var handleLater = [];
  var maxRefNum = 0;
  var propertiesToSet = {};
  for (var i = 0; i < commentElements.length; i++){
    var comment = XmlToComment(commentElements[i]);
    commentArray.push(comment);
    //If no reference number, add on to the end
    if (!comment.refNum) {
      handleLater.push(comment);
    }

    if (maxRefNum < comment.refNum){
      maxRefNum = comment.refNum;
    }
    propertiesToSet[COMMENT_BASE + comment.refNum] = JSON.stringify(comment);
  }

  //Handle any unreferenced comments
  handleLater.forEach(function (comment, index, array) {
    Logger.log("Handling comments without a refNum");
    comment.refNum = maxRefNum + index + 1;
    propertiesToSet[COMMENT_BASE + comment.refNum] = JSON.stringify(comment);
  });


  var docProperties = PropertiesService.getDocumentProperties();
  docProperties.setProperties(propertiesToSet);
  docProperties.deleteProperty("ASSISTmentsComments")
  return commentArray;
}

 /**
  * Gets the number of the superscript that will label a comment. This is stored
  * so that the count does not start from scratch each type a doc opens.
  * @return {number} Next comment reference number
  */
  function getNextCommentNumber(){
    var properties = PropertiesService.getDocumentProperties().getProperties();
    var keys = Object.keys(properties);

    var feedbackKeys = getFeedbackKeys(keys);
    for (var i = feedbackKeys.length - 1; i >= 0; i--){
      var key = feedbackKeys[i];
      var comment = parseJson(properties[key], Comment.prototype);
      if (comment.refNum){
        return comment.refNum + 1;
      }
    }
    return 1;
  }
```

```
function getSavedAssignmentNames() {
    var property = PropertiesService.getUserProperties().getProperty("ASSISTmentsNames");
    if (property == null) {
      return null;
    }
    return XmlService.parse(property);
  }
 /**
  * Saves a comment into document properties.
  * If no comment exists yet, new properties are created for the add on.
  * @param {Comment} comment object to be saved
  * @return {string} XML representation of currently saved comments
  */
 function saveComment(com){
   if (!com.refNum){
     com.refNum = getNextCommentNumber();
   }

   var json = JSON.stringify(com);
   PropertiesService.getDocumentProperties().setProperty(COMMENT_BASE + com.refNum, json);
   return json;
 }

 /**
  * Marks a comment as resolved or opened.
  * @param {string} comId Comment ID to add explanation to;
  * @param {boolean} resolvedTrue to resolve, false to reopen
  * @return {string} True if resolved, false otherwise (permission issue)
  */
 function markAsResolved(comId, resolved){
    var comment = getFilteredComments("id", comId)[0];

    if (comment.req){
      if (comment.explanations.length == 0) {
        DocumentApp.getUi().alert("Your teacher requires you to write an explanation of your
explanation.");
        return false;
      }
    }


    if (resolved && comment.range != null){
      comment.unformat();
    } else if (!resolved && comment.range != null) {
      comment.format(getRubricColors().getColor(comment.ccss), null);
    }

    comment.resolved = resolved;
    saveComment(comment);
    return true;
  }

 /**
  * Adds an explanation to an existing comment that is already saved.
  * @param {string} comId Comment ID to add explanation to;
  * @param {string} text Text of explanation to add.
  * @param {boolean} audio True if audio feedback
  * @return {string} Explanation added.
  */
```

```
  function addExplanation(comId, text, audio){
    var date = new Date();

    var fullExplanation = new Explanation(text);
    fullExplanation.audio = audio;

    var comment = getFilteredComments("id", comId)[0];
    comment.explanations.push(fullExplanation);
    saveComment(comment);

    return fullExplanation.toPrettyFormat();
  }

  /**
  * Given a rubric object, save it to the document properties
  * @param {Rubric} rubric Rubric to save
  */
 function saveRubric(rubric){
   var stringXML = XmlService.getPrettyFormat().format(rubric.toXML());
   PropertiesService.getDocumentProperties().setProperty("ASSISTmentsRubric", stringXML)
 }

  /**
  * Makes a rubric to the User's properties so that it can be used in future documents.
Stores it
  * @param {Rubric} rubric Rubric to save to the user.
  * @param {RubricColorMap} map Map of the colors to the rubric categories.
  * @param {string[]} authors A list of the email addresses that can access a rubric.
  * @param {string} name Name of the Rubric.
  */
 function makeUserRubric(rubric, map, authors, name){
   var newRubric = XmlService.createElement("UserRubric").setText(name);
   var rubricXML = rubric.toXML().detachRootElement();
   var children = rubricXML.getChildren();
   for (var i = 0; i < children.length; i++) {
     children[i].setAttribute("color", map.getColor(children[i].getText()));
   }

   newRubric.addContent(rubricXML);
   for (var i = 0; i < authors.length; i++){
     newRubric.addContent(XmlService.createElement("Author").setText(authors[i]));
   }
   saveUserRubric(newRubric, name);
 }

 /**
  * Saves a rubric to the user's User Properties so that it can be used
  * in future documents.
  * @param {Document} doc XML Doc to be stored to the user's properties.
  */
 function saveUserRubric(doc, name){
   var stringXML = XmlService.getPrettyFormat().format(doc);
   var response =
UrlFetchApp.fetch(API_BASE_URL+Session.getEffectiveUser().getEmail()+"/"+encodeURIComponent(
name),{
     "method":"put",
     "payload":stringXML,
   });
   if (response.getResponseCode() > 300){
     throw new Error("Saving rubric failed with response code " +
```

```
response.getResponseCode());
    }
 }

 /**
  * Gets a list of available saved user rubrics.
  * @param {string} email Email address to look for (optional)
  * @return {string[]} A list of rubric names that the user has saved
  */
 function getUserRubricNames(email){
   if (email == null){
     email = Session.getEffectiveUser().getEmail();
   }
   var response = UrlFetchApp.fetch(API_BASE_URL+email);
   if (response.getResponseCode() >= 300){
     return [];
   } else {
     var array = response.getContentText().split("\n");
     array.splice(array.length - 1);
     for (var i = 0; i < array.length; i++){
       array[i] = array[i].split("%20").join(" ").trim();
     }
     return array;
   }
 }


 /**
  * Gets the rubric, color map, and authors from the external source
  * @param {string} name Name of the user rubric
  * @param {XML} XML document that belongs to the name;
  */
 var lastKnownRubric = null;
 var lastKnownRubricName = "";
 var lastKnownRubricEmail = "";
 function fetchUserRubricData(name, email){
   if (email == null){
     email = Session.getEffectiveUser().getEmail();
   }
   if (!(name == lastKnownRubricName && email == lastKnownRubricEmail)){
     var response = UrlFetchApp.fetch(API_BASE_URL+email+"/"+encodeURIComponent(name));
     if (response.getResponseCode() >= 300){
       throw new Error("Rubric with that name and email could not be found");
     }
     lastKnownRubric = XmlService.parse(response.getContentText()).getRootElement();
     lastKnownRubricName = name;
     lastKnownRubricEmail = email;
   }
   return lastKnownRubric;
 }

 /**
  * Gets the saved rubric for the given name
  * @param {string} name Name of the user rubric
  * @return {Rubric} User rubric with the given name.
  */
 function getUserRubric(name){
   var XML = fetchUserRubricData(name);
   var rubricXML = XML.getChild("Rubric");
   if (rubricXML == null){
```

```
    throw new Error("Rubric data was not found");
  }
  return XMLtoRubric(XmlService.getPrettyFormat().format(rubricXML));
}


/**
* Returns the categories of an input rubric
* @param {string} name Rubric name to return keys of.
* @return {string[]} Set of rubric keys.
*/
function getRubricKeys(nam){
  var originalKeys = Object.keys(new Rubric());
  var result = [];
  var allKeys = Object.keys(nam);
  Logger.log(allKeys);
  for (var i = 0; i < allKeys.length; i++){
       if (originalKeys.indexOf(allKeys[i]) == -1){
         Logger.log(allKeys[i]);
         result.push(allKeys[i]);
       }
     }
  return result;
}

/**
 * Gets the saved rubric color map for the given name
 * @param {string} name Name of the rubric to get the color map for.
 * @param {string} email Optional. Email of the user.
 * @return {RubricColorMap} The appropriate rubric color map for the name
 */
function getUserRubricColorMap(name, email){
  var XML = fetchUserRubricData(name, email);
  var rubricXML = XML.getChild("Rubric");
  if (rubricXML == null){
    throw new Error("Rubric data was not found");
  }

  var map = new RubricColorMap();

  var children = rubricXML.getChildren();
  for (var i = 0; i < children.length; i++) {
    map.addColorMapping(children[i].getText().trim(),
children[i].getAttribute("color").getValue());
  }
  return map;
}

/**
 * Gets the authors that can edit the rubric for a user's saved rubric.
 * Note that this will not allow the user to change another user's saved rubric. Rather,
 * it lets them open the rubric on any document that the rubric is tied to.
 * @param {string} name Name of the rubric
 * @param {string[]} List of authors for the rubric with the given name.
 */
function getUserRubricAuthors(name){
  var XML = fetchUserRubricData(name);
  var authorXML = XML.getChildren("Author");
  if (authorXML == null){
    throw new Error("Malformed XML: Authors missing");
```

```
   }
   var result = [];
   for (var i = 0; i < authorXML.length; i++){
     result.push(authorXML[i].getText());
   }
   return result;
 }


  /**
   * Deletes a User rubric
   * @param {string} name Name of the rubric to erase.
   */
  function deleteUserRubric(name){
   var response =
UrlFetchApp.fetch(API_BASE_URL+Session.getEffectiveUser().getEmail()+"/"+encodeURIComponent(
name), {
   "method":"delete"
   })
   if (response.getResponseCode() >= 300){
     throw new Error("Could not delete rubric with name " + name);
   }
  }

 /**
  * Given a RubricColorMap, save it to the document properties
  * @param {RubricColorMap} map RubricColorMap to save.
  */
 function saveRubricColorMap(map){
   var stringXML = XmlService.getPrettyFormat().format(map.toXML());
   PropertiesService.getDocumentProperties().setProperty("ASSISTmentsRubricColorMap",
stringXML);
 }


 /**
  * Saves all the Rubric data at once.
  * @param {Rubric} rubric The rubric to save
  * @param {RubricColorMap} map The color map associated with the rubric
  * @param {String[]} authors Authors of the rubric
  */
  function saveRubricData(rubric, map, authors, name){
    var rubricXML = XmlService.getPrettyFormat().format(rubric.toXML());
    var mapXML = XmlService.getPrettyFormat().format(map.toXML());
    var authors = authorToString(authors);
    PropertiesService.getDocumentProperties().setProperties({
      "ASSISTmentsRubricOwner": authors,
      "ASSISTmentsRubricColorMap": mapXML,
      "ASSISTmentsRubric": rubricXML,
      "ASSISTmentsRubricName": name,
    });
  }

 /**
  * Fetch the rubric from the document properties. Returns a default rubric
  * if one does not exist
  * @return {Rubric} The saved rubric, or the default rubric if no saved rubric exists.
  */
  function getRubric(){
    var rubricXML =
```

```
PropertiesService.getDocumentProperties().getProperty("ASSISTmentsRubric");
    if (rubricXML == null){
      return null;
    } else {
      return XMLtoRubric(rubricXML);
    }
  }

  /**
   * Gets the rubric color mapping
   * @return {RubricColorMap} The color mapping stored in the document properties.
   */
  function getRubricColors(){
    var mapXML =
PropertiesService.getDocumentProperties().getProperty("ASSISTmentsRubricColorMap");
    if (mapXML == null){
      return null;
    } else {
      return XMLtoRubricColorMap(mapXML);
    }
  }

 /**
  * Return a rubric object based on a given XML string.
  * @param {string} XML Document representing the Rubric.
  * @return {Rubric} Rubric parsed from the XML Document
  */
  function XMLtoRubric(XML){
     var root = XmlService.parse(XML).getRootElement();
     var rubric = new Rubric();
     var cats = root.getChildren();
     for (var i = 0; i < cats.length; i++){
       rubric.addCategory(cats[i].getText().trim());
       var items = cats[i].getChildren();
       for (var j = 0; j < items.length; j++){
         rubric.addComment(cats[i].getText().trim(), items[j].getText().trim(),
items[j].getAttribute("required").getValue() == "true");
       }
     }

     return rubric;
  }

 /**
  * Returns a RubricColorMapping based on a given XML string
  * @param {string} XML XML of the Color Map
  * @return {RubricColorMap} Color mapping for the XML
  */
  function XMLtoRubricColorMap(XML) {
    var root = XmlService.parse(XML).getRootElement();
    var map = new RubricColorMap();
    var cats = root.getChildren();
    for (var i = 0; i < cats.length; i++){
      map.addColorMapping(cats[i].getText().trim(), cats[i].getChildren()[0].getText())
    }
    return map;
  }

 /**
  * Returns an array of comments that match the given ccss
```

```
  * (Actually, a wrapper for filterComments, where field is "ccss", and XML is the document
property
  * @param {string} ccss Category of the comment
  * @return {Comment[]} An array of comments that belong to that category (ccss)
  */
  function getComments(ccss) {
    return filterComments("ccss", ccss, getSavedComments());
  }

 /**
  * Returns an array of comments that match the filter criteria.
  * @param {string} field to filter by.
  * @param {string} value Value of the field.
  * @return {Comment[]} An array of comments satisfying the filter
  */
  function getFilteredComments(field, value){
    return filterComments(field, value, getSavedComments());
  }

  /**
   * Filter an XML element <Comments> based on a given field.
   * @param {string} field to filter by.
   * @param {string} value Value of the field.
   * @param {Comment[]} comments XML element containing all comments to filter.
   */

   function filterComments(field, value, comments) {
    var commentElements;
    if (comments == null){
      comments = [];
    }
    var commentArray = [];

    comments.forEach(function(element) {
      if (element[field] == value){
        commentArray.push(element);
      }
    });

    return commentArray;
   }

  /**
   * Gets the number of comments stored within the document
   * @return {number} Number of comments for this document
   */
  function getNumComments(){
    return getFeedbackKeys().length;
  }


  /**
   * Here for backwards compatibility.
   * Get all comments. Returns an array
   * @return {Comment[]} An array of comments saved in the Google Document
   */
   function getAllComments(){
      return getSavedComments();
   }
```

```
   function getAllAssignmentNames(){
     var XMLElement = getSavedAssignmentNames();
     var assignmentNameElements;
      if (XMLElement != null){
        assignmentNameElements = XMLElement.getRootElement().getChildren("AssignmentNames");
      } else {
        assignmentNameElements = []
      }
      var assignmentNameArray = [];
      for (var i = 0; i < assignmentNameElements.length; i++){
        assignmentNameArray.push(assignmentNameElements[i]);
      }
      return commentArray;
   }



  /**
   * Given an XML element, returns the same value
   * as a Comment (See Comment.gs)
   * @param {Element} element XML Element representing a comment.
   * @return {Comment} Comment that the XML element stored
   */
  function XmlToComment(element){
     var id = +element.getChild("id").getText();
     var range = element.getChild("range").getText();
     var bookmark = element.getChild("bookmark").getText();
     var author = element.getChild("author").getText();
     var ccss = element.getChild("ccss").getText();
     var text = element.getChild("text").getText();
     var refNum = +element.getChild("refNum").getText();
     var req = element.getChild("req").getText() == "true";
     var resolved = element.getChild("resolved").getText() == "true";
     var com = new Comment(text, ccss, req, author, id);
     if (range == "undefined"){
       com.range = null;
       com.bookmark = null;
     } else {
       com.range = range;
       com.bookmark = bookmark;
     }

     var audioLink = element.getChild("audioId");
     if (audioLink){
       com.audioId = audioLink.getText();
     }

     com.refNum = refNum;
     com.resolved = resolved;

     var children = element.getChildren("explanation");

     for (var i = 0; i < children.length; i++){
       com.explanations.push(XMLtoExplanation(children[i]));
     }
     return com;
   }

/**
 * Converts an XML element into an explanation object
```

```
 * @param {XMLService.Element} element Element to convert to an object.
 */
function XMLtoExplanation(element){
  if (element == null){
    throw new Error("Cannot parse a null XML element");
  }
  var date = element.getChild("date").getText();
  var user = element.getChild("user").getText();
  var text = element.getChild("text").getText();
  var exp = new Explanation(text, date, user);

  var audioAttribute = element.getAttribute("audio");

  if (audioAttribute && audioAttribute.getValue() == "true"){
    exp.audio = true;
  }

  return exp;
}

 /**
  * Deletes all comments.
  */
  function deleteAllComments() {
      var comments = getSavedComments();
      comments.forEach(function(comment){
        comment.unformat(true);
        PropertiesService.getDocumentProperties().deleteProperty(COMMENT_BASE +
comment.refNum);
      });
  }
```

addComment.gs

```
<?
var customAudio = PropertiesService.getUserProperties().getProperty("CustomAudio") ||
"Audio";
var formattingState =
PropertiesService.getDocumentProperties().getProperty("FormattingState") || "Formatted";
?>
<!DOCTYPE html>
<html>
<link rel="stylesheet" href="https://ssl.gstatic.com/docs/script/css/add-ons.css">
<link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/font-awesome/4.5.0/css/font-awesome.min.css">
<script src="https://ajax.googleapis.com/ajax/libs/jquery/2.1.3/jquery.min.js"></script>
<img id="loading-icon" src="https://www.assistments.org/images/indicator.gif"
class="loading" hidden>
<div id="div-wrapper">

<div class="titleName">
       <? if (PropertiesService.getDocumentProperties().getProperty("ASSISTmentsRubricName")
== "Default Rubric"){ ?>
       <b>Default Rubric </b>
```

```
    <?} else{?>
    <b> Rubric: "<?=
PropertiesService.getDocumentProperties().getProperty("ASSISTmentsRubricName") ?>" <?}?></b>
</div>

<style type="text/css">
.titleName {
  font-size: 14px;
}

.no-click {
  pointer-events: none;
  cursor: default;
}

.checkbox-sub {
  margin-top:4px;
  margin-bottom:4px;
}

.loading {
  position:fixed;
  top: 5px;
  right: 5px;
  height:25px;
  width:25px;
}

.standardButton {
    width: 200px;
    font-weight: bold;
}

.color-box {
    position: fixed;
    float: right;
    width: 10px;
    height: 10px;
    display: inline-block;
}

.boxed {
        width: 255px;
        margin-top: 5px;
        border: 1px solid #b9a894;
}


ol#menu {
  width: 255px;
  margin-top: 10px;
  padding: 0;
}


ol#menu ol {
  display: none;
  margin: 0;
  border-radius:7px;
}
```

```css
ol#menu li,
  ol#menu a {
  font-family: arial, arial;
  font-size: 14px;

  margin-bottom: 10px;
  margin-top: 10px;
}

ol#menu li {
  line-height: 15px;
}

ol#menu ol li {
  border-bottom: none;
}


ul#menu ol li:before {
  content: "";
}

ol#menu a:hover {
  font-weight: bold;
  text-decoration: none;
}

ol#menu a.active {
  font-weight: bold;
}

.main-box {
  overflow:auto;
}

.bottom-logo {
  position:fixed;
  bottom:0px;
  left: 60px;
}

.bottom-buttons {
  position:fixed;
  top:5px;
  width:255px;
}

#div-wrapper {
  position:absolute;
  left: 0px;
  right: 0px;
  top: 0px;
  padding-top: 5px;
  padding-left: 10px;
  overflow:auto;
  border-bottom: 1px solid black;
  height: calc(100% - 80px);
}
```

```css
#custom-feedback-select {
  max-width: 200px;
  text-align: left;
}

.bottom-btns {
  padding-bottom:20px;
}

#toggle-formatting-button {
  margin-left: 5px;
  width: 122px;
}

.error {
  color: red;
}

.success {
  color: green;
}

.micStatus {
  display: inline;
  font-weight: bold;
  transition: height 1s;
  height: 18px;
}

.pulse {
  animation-name: pulse;
  animation-duration: 2s;
  animation-iteration-count: infinite;
}

@keyframes pulse {
  0% {opacity: 1;}
  50% {opacity: 0.5;}
  100% {opacity: 1;}
}

.audioMsg {
  margin: 0px;
}

#audioButton {
    background:
url('https://sites.google.com/site/assistmentsfeedbacktool/resources/mic.png') center left
no-repeat, linear-gradient(top, #f5f5f5, #f1f1f1);
    background:
url('https://sites.google.com/site/assistmentsfeedbacktool/resources/mic.png') center left
no-repeat, -moz-linear-gradient(top, #f5f5f5, #f1f1f1);
    background:
url('https://sites.google.com/site/assistmentsfeedbacktool/resources/mic.png') center left
no-repeat, -webkit-linear-gradient(top, #f5f5f5, #f1f1f1);
    background-size: 20px auto;
    padding-left: 20px;
    width: 117px;
    margin-top: 10px;
}
```

```
.audioButton {
  width: 117px;
}

/* Begin Tab Styling */
.tabs-menu {
    height: 30px;
    float: left;
    clear: both;
    list-style-type: none;
    margin-bottom: 0px;
    margin-top: 0px;
    padding-left: 10px;
}

.tabs-menu li {
    height: 30px;
    line-height: 30px;
    float: left;
    margin-right: 10px;
    background-color: #ccc;
    border-top: 1px solid #d4d4d1;
    border-right: 1px solid #d4d4d1;
    border-left: 1px solid #d4d4d1;
}

.tabs-menu li.current {
    position: relative;
    background-color: #fff;
    border-bottom: 1px solid #fff;
    z-index: 5;
}

.tabs-menu li a {
    padding: 10px;
    color: #fff;
    text-decoration: none;
}

.tabs-menu .current a {
    color: #2e7da3;
}

.tab {
    border: 1px solid #d4d4d1;
    background-color: #fff;
    float: left;
    margin-bottom: 20px;
    width: auto;
}

.tab-content {
    width: 250px;
    padding: 5px;
    display: none;
}

#tab-<? if (customAudio=="Audio") {?>audio<?} else {?>text<?}?> {
 display: block;
```

```
}

/* End Tab Styling */

</style>

<script language="javascript">

/*
Created by: Travis Beckham :: http://www.squidfingers.com | http://www.podlob.com
version date: 06/02/03 :: If want to use this code, feel free to do so,
but please leave this message intact. (Travis Beckham) */

// Node Functions

if(!window.Node){
  var Node = {ELEMENT_NODE : 1, TEXT_NODE : 3};
}

function checkNode(node, filter){
  return (filter == null || node.nodeType == Node[filter] || node.nodeName.toUpperCase() ==
filter.toUpperCase());
}

function getChildren(node, filter){
  var result = new Array();
  var children = node.childNodes;
  for(var i = 0; i < children.length; i++){
    if(checkNode(children[i], filter)) result[result.length] = children[i];
  }
  return result;
}

function getChildrenByElement(node){
  return getChildren(node, "ELEMENT_NODE");
}

function getFirstChild(node, filter){
  var child;
  var children = node.childNodes;
  for(var i = 0; i < children.length; i++){
    child = children[i];
    if(checkNode(child, filter)) return child;
  }
  return null;
}

function getFirstChildByText(node){
  return getFirstChild(node, "TEXT_NODE");
}

function getNextSibling(node, filter){
  for(var sibling = node.nextSibling; sibling != null; sibling = sibling.nextSibling){
    if(checkNode(sibling, filter)) return sibling;
  }
  return null;
}
function getNextSiblingByElement(node){
        return getNextSibling(node, "ELEMENT_NODE");
}
```

```
// Menu Functions & Properties

var activeMenu = null;

function showMenu() {
  if(activeMenu){
    activeMenu.className = "";
    activeMenu.innerHTML = "[+]" + activeMenu.innerHTML.substring(3);
    getNextSiblingByElement(activeMenu.nextSibling).style.display = "none";
  }
  if(this == activeMenu){
    activeMenu = null;
  } else {
    this.className = "active";
    this.innerHTML = "[&#8211;]" + this.innerHTML.substring(3);
    getNextSiblingByElement(this.nextSibling).style.display = "block";
    activeMenu = this;
  }
  return false;
}

function initMenu(){
  var menus, menu, text, a, i;
  menus = getChildrenByElement(document.getElementById("menu"));
  for(i = 0; i < menus.length; i++){
    menu = menus[i];
    text = getFirstChildByText(menu);
    a = document.createElement("a");
    menu.replaceChild(a, text);
    a.appendChild(text);
    a.href = "#";
    a.onclick = showMenu;
    a.title = "Click to expand."
    a.onfocus = function(){this.blur()};
    if(i == 0){
      a.className = "active";
      a.innerHTML = "[&#8211;]" + a.innerHTML;
      getNextSiblingByElement(a.nextSibling).style.display = "block";
      activeMenu = a;
    }
    else{
      a.innerHTML = "[+]" + a.innerHTML;
    }
  }
}

$(document).ready(initMenu);

</script>
<body>

<div class="block">

<ol id="menu" style="list-style-type: none;">
  <? var sections = categories.getKeys(); ?>
  <? for (var i = 0; i < sections.length; i++) { ?>
        <li> <?= sections[i]?>
            <svg width="10" height="10"> <rect width="10" height="10" style="fill:<?=
getRubricColors().getColor(sections[i]); ?>"/></svg>
```

```
              <ol>
      <? for (var j = 0; j < categories[sections[i]].length; j++) { ?>

               <li><a onclick="makeComment('<?= categories[sections[i]][j] ?>', '<?=
sections[i]?>', '<?=categories.responseRequired[sections[i]][j]?>')"
id="<?=sections[i]?>-<?=categories[sections[i]][j]?>-button"> <?= categories[sections[i]][j]
?> </a></li>

      <? } ?>
              </ol>
         </li>

  <? } ?>
</ol>
<br>

<b style="font-size:14px"><?=translate("Deleting Feedback");?></b><br>
<input type="number" id="DeleteCommentBox" name="Delete comment input" style="width:50px;
position: relative;" onkeyup="updateButton(this)" onchange="updateButton(this)"
placeholder="#" min="1">
<input id="delete-comment-button" type="button" value= "<?=translate('Delete Feedback');?>"
onclick="deleteComment();" disabled>
<br>
<br>
<b style="font-size:14px"><?=translate("Custom Feedback");?></b>
<br>
<label for="custom-feedback-select"><?=translate("Category");?>: </label>
<select id="custom-feedback-select">
  <option><?=translate("General");?></option>
  <? for (var i = 0; i < sections.length; i++) { ?>
  <option><?= sections[i] ?></option>
  <? } ?>
</select>
<div class="checkbox-sub">
  <input id="general-checkbox" type="checkbox">
  <label for="general-checkbox" class="gray"><?=translate("Require Explanation");?></label>
</div>
<div id="tabs-container">
    <ul class="tabs-menu">
        <li <? if (customAudio == "Audio") {?>class="current"<?}?>><a
href="#tab-audio">Audio</a></li>
        <li <? if (customAudio == "Text") {?>class="current"<?}?>><a
href="#tab-text">Text</a></li>
    </ul>
    <div id="tab-content-container" class="tab">
        <div id="tab-audio" class="tab-content">
            <div id="audioDidYouKnow">
              <p>Audio feedback is not supported in your browser.</p>
              <img height="18" width="18"
src="https://sites.google.com/site/assistmentsfeedbacktool/resources/info-icon.png">
              <p style="margin-top:0px; display:inline-block;"><b>Tip:</b> Try Google Chrome
or Mozilla Firefox</p>
            </div>
            <div id="audioSection" style="display:none;">
              <!--<div style="margin-top: 10px">
                <label for="audio-feedback-select"><?=translate("Category");?>: </label>
                <select id="audio-feedback-select">
                  <option><?=translate("General");?></option>
                  <? for (var i = 0; i < sections.length; i++) { ?>
                  <option><?= sections[i] ?></option>
```

```
                <? } ?>
              </select>
            </div>
            <div class="checkbox-sub">
              <input id="audio-checkbox" type="checkbox">
              <label for="audio-checkbox" class="gray">Require Explanation</label>
            </div>
             -->
            <div>
              <button id="audioButton" class="audio" onclick="startRecording()">
                Record
              </button>
              <p id="recordingStatus" class="micStatus pulse error"
style="display:none;">Recording...</p>
              <p id="processingStatus" class="micStatus pulse"
style="display:none;">Processing...</p>
            </div>
            <br>
            <div>
              <audio id="audioPlayer"></audio>
              <button id="playLastButton" onclick="playLast()" class="audioButton"
disabled><i class="fa fa-play-circle"></i> Play last</button>
              <button id="deleteLastButton" onclick="deleteLast()" class="audioButton"
disabled><i class="fa fa-trash-o"></i> Delete last</button>
            </div>
            <br>
            <p id="micError" class="error" style="display:none;">Unable to access
microphone</p>
            <p id="audioSaveError" class="audioMsg error" style="display:none;">Unable to
save recording</p>
            <p id="audioSuccessMsg" class="audioMsg success"
style="display:none;">Recording saved successfully</p>
            <p id="audioDeleteError" class="audioMsg error" style="display:none;">Could
not delete last feedback</p>
            <p id="audioDeleteSuccess" class="audioMsg success"
style="display:none;">Successfully deleted last feedback</p>
          </div>
        </div>
        <div id="tab-text" class="tab-content">
          <textarea rows="6" id="CustomCommentBox" name="Comment" style="width:250px;
resize:none;" onkeyup="updateButton(this)" placeholder="<?=translate('You do not need to
highlight text for this type of feedback.');?>"></textarea>
        </div>
    </div>
</div>
<br>

<br>


<br>
<input id = "custom-feedback-button" type="button" value="<?=translate('Add Feedback');?>"
onclick="addCustomFeedback()" disabled>
<img id="loading-icon-for-custom" src="https://www.assistments.org/images/indicator.gif"
class="loading" hidden>
<span id="custom-comment-success" class="secondary" hidden>Your comment has been
submitted</span>
<br>
</div>
<br>
```

```
<hr>
<br>
<div class="bottom-btns">
  <input id="generate-summary-button" type="button" class="action"
value="<?=translate('Summarize & Grade');?>" onclick="generateSummary();">
  <input id="toggle-formatting-button" type="button" value="<?= formattingState ==
"Formatted" ? "Hide Formatting" : "Show Formatting"?>"
    onclick="<?!= formattingState == "Formatted" ? "clearFormat()" : "restoreFormat()"?>">
  <img id="loading-icon-for-bottom" src="https://www.assistments.org/images/indicator.gif"
class="loading" hidden>
</div>
</body>
</div>

<br>
<a href="http://docassist.assistments.org/" target="_blank" class="bottom-logo">
<img
src="https://sites.google.com/site/assistmentsfeedbacktool/resources/docASSIST-sidebar.png"
align="bottom" width="175">
</a>
<script language="javascript"
src="https://sites.google.com/site/assistmentsfeedbacktool/resources/recorder.js"></script>
<script language="javascript"
src="https://sites.google.com/site/assistmentsfeedbacktool/resources/lame.min.js"></script>
<script language="javascript">

  $(document).ready(init);

  var recorder;
  var audioTrack;
  var currentTabId = "#tab-audio";

  /**
   * Performs page initialization
   */
  function init(){
    detectAudio();

    $(".tabs-menu a").click(function(event) {
        event.preventDefault();
        $(this).parent().addClass("current");
        $(this).parent().siblings().removeClass("current");
        var tab = $(this).attr("href");
        currentTabId = this.getAttribute("href");
        updateCustomButton();

        //Fix for first animation
        var tabContainer = document.getElementById("tab-content-container");
        if (tabContainer.style.height == ""){
          tabContainer.style.height = $(tabContainer).outerHeight() + "px";
        }

        $(".tab-content").not(tab).css("display", "none");
        $(tab).fadeIn();

        if (!$(tab).height()){
          $(tab).height($(tab).outerHeight());
        }

        $(".tab").animate({
```

```
          height: $(tab).outerHeight(),
        });
    });

    <? if (PropertiesService.getUserProperties().getProperty("CustomAudio") == "Text") { ?>
    $("#tab-text-link").trigger("click");
    <? } ?>
  }

  /**
   * Initializes the audio feedback section
   */

  function detectAudio(){
    try {
      // webkit shim
      window.AudioContext = window.AudioContext || window.webkitAudioContext;
      navigator.getUserMedia = navigator.getUserMedia || navigator.webkitGetUserMedia ||
navigator.mozGetUserMedia;
      window.URL = window.URL || window.webkitURL;
      audio_context = new AudioContext;
    } catch (e) {
      console.log("No audio support. Audio panel is disabled.")
    }

    if (navigator.getUserMedia) {
      console.log("Enabling audio panel");
      document.getElementById("audioDidYouKnow").style.display = "none";
      document.getElementById("audioSection").style.display = "initial";
    }
  }

  /**
   * Initialize the recorder
   */
  function startUserMedia(stream){
    audioTrack = stream.getAudioTracks()[0];
    var input = audio_context.createMediaStreamSource(stream);

    recorder = new Recorder(input, {numChannels: 1} );

    $(document).trigger("microphoneReady");
  }

  /**
   * Start recording an audio comment
   */
  function startRecording(){
    hideAudioErrors();

    navigator.getUserMedia({audio: true}, startUserMedia, function(e) {
      console.log('No live audio input: ' + e);
      document.getElementById("audioButton").disabled = true;
      $("#micError").slideDown();
    });

    $(document).one("microphoneReady", function() {
      if (recorder){
        recorder.record();
        document.getElementById("processingStatus").style.display = "none";
```

```
      document.getElementById("recordingStatus").style.display = "initial";
      console.log("Beginning audio recording");
      var button = document.getElementById("audioButton");
      button.innerHTML = "Stop";
      button.onclick = stopRecording;
    } else {
      console.log("Recorder not present!");
    }
  });
}

var liblame = new lamejs();
var audioPlayer;
var mp3DataURL;

/**
 * Stop recording an audio comment
 */
function stopRecording() {
  if (recorder){
    document.getElementById("audioButton").disabled = true;
    document.getElementById("recordingStatus").style.display = "none";
    document.getElementById("processingStatus").style.display = "initial";

    recorder.stop();
    audioTrack.stop();

    console.log("Recording stopped")

    var arrayBuffer;
    var fileReader = new FileReader();

    var cat = document.getElementById("custom-feedback-select").value;
    var req = document.getElementById("general-checkbox").checked;

    //Encode to MP3
    fileReader.onload = function() {
        arrayBuffer = this.result;

        var wav = liblame.WavHeader.readHeader(new DataView(arrayBuffer));
        samples = new Int16Array(arrayBuffer, wav.dataOffset, wav.dataLen / 2);

        //Get mp3 blob
        var mp3 = encodeMono(wav.channels, wav.sampleRate, samples);

        console.log("MP3 Encoding complete");
        console.log("Uploading MP3 to Google Drive");

        var fr = new FileReader();
        fr.onloadend = function(){
          mp3DataURL = fr.result;
          updateCustomButton();

          var button = document.getElementById("audioButton");
          button.innerHTML = "Record";
          button.onclick = startRecording;
          document.getElementById("processingStatus").style.display = "none";
          document.getElementById("recordingStatus").style.display = "none";

          var PLButton = document.getElementById("playLastButton");
```

```
            PLButton.disabled = false;
            var DLButton = document.getElementById("deleteLastButton");
            DLButton.disabled = false;
            /*
            google.script.run
              .withSuccessHandler(audioLoaded)
              .withFailureHandler(audioFailed)
              .makeAudioComment(fr.result, cat, req);
            */
          };
          fr.readAsDataURL(mp3);

          var objectURL = URL.createObjectURL(mp3);

          if (!audioPlayer){
            audioPlayer = document.getElementById("audioPlayer");
          }

          audioPlayer.src = objectURL;
          audioPlayer.load();
      };

      console.log("Generating WAV file");
      //Make a wav file and send to encoder
      recorder.exportWAV(function(blob){
        console.log("Encoding MP3");
        fileReader.readAsArrayBuffer(blob);
      });
    }
  }

  /**
   * Hides all audio errors
   */
  function hideAudioErrors() {
    $("#audioSaveError").slideUp();
    $("#micError").slideUp();
    $("#audioSuccessMsg").slideUp();
    $("#audioDeleteError").slideUp();
    $("#audioDeleteSuccess").slideUp();
  }

  function encodeMono(channels, sampleRate, samples) {
    var buffer = [];
    //Encode at 64 Kbps since it is only a voice recording
    mp3enc = new lamejs.Mp3Encoder(channels, sampleRate, 64);
    var remaining = samples.length;
    var maxSamples = 1152;
    for (var i = 0; remaining >= maxSamples; i += maxSamples) {
        var mono = samples.subarray(i, i + maxSamples);
        var mp3buf = mp3enc.encodeBuffer(mono);
        if (mp3buf.length > 0) {
            buffer.push(new Int8Array(mp3buf));
        }
        remaining -= maxSamples;
    }
    var d = mp3enc.flush();
    if(d.length > 0){
        buffer.push(new Int8Array(d));
    }
```

```
    var blob = new Blob(buffer, {type: 'audio/mp3'});
    return blob;
   }

  /**
   * Plays the last recorded feedback
   */
   function playLast() {
     var audioPlayer = document.getElementById("audioPlayer");
     if (audioPlayer.readyState == 4){
       var playButton = document.getElementById("playLastButton");
       playButton.innerHTML = "<i class=\"fa fa-pause\"></i> Pause";
       playButton.onclick = pauseLast;

       audioPlayer.play();
       $(audioPlayer).one("ended", function() {
         playButton.innerHTML = "<i class=\"fa fa-play-circle\"></i> Play Last"
         playButton.onclick = playLast;
       });
     }
   }

  /**
   * Pauses the last recorded feedback
   */
   function pauseLast() {
     audioPlayer.pause();
     var playButton = document.getElementById("playLastButton");
     playButton.innerHTML = "<i class=\"fa fa-play-circle\"></i> Play Last";
     playButton.onclick = playLast;
   }

  /**
   * Deletes the last recorded feedback
   */
   function deleteLast() {
     mp3DataURL = null;
     updateCustomButton();

     recorder.clear();

     hideAudioErrors();
     var PLButton = document.getElementById("playLastButton");
     PLButton.disabled = true;
     var DLButton = document.getElementById("deleteLastButton");
     DLButton.disabled = true;

     var RButton = document.getElementById("audioButton");
     RButton.disabled = false;
   }

/**
 * Function wrapper for generate summary so that loading will properly display
 */
 function generateSummary(){
   showLoading("loading-icon-for-bottom");
   var btn = document.getElementById("generate-summary-button");
   google.script.run
         .withSuccessHandler(function(){
```

```
            btn.value = "Update Summary";
            hideLoading("loading-icon-for-bottom");

            google.script.run
                          .withSuccessHandler(function(){
hideLoading("loading-icon-for-bottom"); })
                          .withFailureHandler(function(error){
hideLoading("loading-icon-for-bottom"); alert(error.message);})
                          .showGrading();

          })
          .withFailureHandler(function(){ hideLoading("loading-icon-for-bottom"); })
          .generateSummary();
 }

/**
 * Function wrapper for generateRubricPreview so that loading will properly display
 */
 function generateRubricPreview(){
   showLoading("loading-icon-for-bottom");
   var btn = document.getElementById("rubric-preview-button");
   google.script.run
          .withSuccessHandler(function(){ btn.value = "Preview";
hideLoading("loading-icon-for-bottom"); })
          .withFailureHandler(function(){ hideLoading("loading-icon-for-bottom"); })
          .generateRubricPreview();
 }

/**
 * Function wrapper for making a comment so that loading will properly display
 */
 function makeComment(value, section, req) {
   showLoading("loading-icon");
   var isRequired = req == "true";
   var button = document.getElementById(section + "-" + value + "-button");
   $(button).addClass("no-click");
   google.script.run
          .withSuccessHandler(function(){ hideLoading("loading-icon");
$(button).removeClass("no-click");})
          .withFailureHandler(function(){ hideLoading("loading-icon"); })
          .makeComment(value, section, isRequired);
 }




/**
 * Makes an audio comment. Does not require a range
 */

 function makeAudioComment() {
   if (!mp3DataURL){
     return;
   }

   showLoading("loading-icon-for-custom");
   var button = document.getElementById("custom-feedback-button")
   button.disabled = true;

   var cat = document.getElementById("custom-feedback-select").value;
```

```
   var req = document.getElementById("general-checkbox").checked;
   google.script.run
                .withSuccessHandler(audioLoaded)
                .withFailureHandler(audioFailed)
                .makeAudioComment(mp3DataURL, cat, req);
 }

  /**
   * Callback for successful audio comment
   */
  function audioLoaded(id) {
    console.log("MP3 upload complete");

    mp3DataURL = null;

    document.getElementById("audioButton").disabled = false;
    updateCustomButton();
    document.getElementById("general-checkbox").checked = false;

    var PLButton = document.getElementById("playLastButton");
    PLButton.disabled = true;
    var DLButton = document.getElementById("deleteLastButton");
    DLButton.disabled = true;

    hideLoading("loading-icon-for-custom");
  }

  /**
   * Callback for failed audio comment
   */
  function audioFailed() {
    $("#audioSaveError").slideDown();
    var button = document.getElementById("audioButton");
    button.innerHTML = "Record";
    button.onclick = startRecording;
    document.getElementById("processingStatus").style.display = "none";
    document.getElementById("recordingStatus").style.display = "none";
    document.getElementById("audioButton").disabled = false;

    updateCustomButton();
    hideLoading("loading-icon-for-custom");
  }

/**
 * Adds custom feedback
 */
 function addCustomFeedback() {
   if (currentTabId == "#tab-audio"){
     makeAudioComment();
   } else if (currentTabId == "#tab-text") {
     makeCustomComment();
   }
 }


/**
 * Makes a custom comment. This comment does not require a range.
 */
 function makeCustomComment(){
   showLoading("loading-icon-for-custom");
```

```
   var button = document.getElementById("custom-feedback-button")
   button.disabled = true;
   var message = document.getElementById("custom-comment-success");
   message.hidden = true;
   var text = document.getElementById("CustomCommentBox").value;
   var cat = document.getElementById("custom-feedback-select").value;
   var req = document.getElementById("general-checkbox").checked;
   google.script.run
     .withSuccessHandler(function(returnVal){
         document.getElementById("CustomCommentBox").value = "";
         if (returnVal) {
           message.hidden = false;
         }
         document.getElementById("general-checkbox").checked = false;
         button.disabled = true;
         hideLoading("loading-icon-for-custom");
      })
     .withFailureHandler(function(error){ hideLoading("loading-icon-for-custom");
button.disabled = true; alert(error.message);})
     .makeCustomComment(text, cat, req);
 }

/**
 * Deletes a comment by reference number.
 */
 function deleteComment(){
 showLoading("loading-icon-for-custom");
 var button = document.getElementById("delete-comment-button");
 var refNumString = document.getElementById("DeleteCommentBox").value;
 var refNum = parseInt(refNumString, 10);
 google.script.run
    .withSuccessHandler(function(){
      document.getElementById("DeleteCommentBox").value = "";
      button.disabled = true;
      hideLoading("loading-icon-for-custom");
     })
    .withFailureHandler(function(error){ hideLoading("loading-icon-for-custom");
document.getElementById("DeleteCommentBox").value = ""; button.disabled = true;
alert(error.message);})
    .deleteComment(refNum); //TODO add check for string
 }

/**
 * Clears formatting on the document
 */
 function clearFormat() {
   debugger;
   showLoading("loading-icon-for-bottom");
   var button = document.getElementById("toggle-formatting-button");
   google.script.run
                .withSuccessHandler(function() {
                  button.value = "Show Formatting";
                  button.onclick = restoreFormat;
                  hideLoading("loading-icon-for-bottom");
                })
                .withFailureHandler(function(error) {
                  hideLoading("loading-icon-for-bottom");
                  alert(error.message);
                })
                .clearFormatting();
```

```
 }

/**
 * Restores the formatting on a document
 */
 function restoreFormat() {
   debugger;
   showLoading("loading-icon-for-bottom");
   var button = document.getElementById("toggle-formatting-button");
   google.script.run
                .withSuccessHandler(function () {
                  button.value = "Hide Formatting";
                  button.onclick = clearFormat;
                  hideLoading("loading-icon-for-bottom");
                })
                .withFailureHandler(function (error) {
                  hideLoading("loading-icon-for-bottom");
                  alert(error.message);
                })
                .restoreFormatting();
 }

/**
 * Updates the comment button based on the text in the text box
 * @param textBox The text box
 */

 function updateButton(textBox){
   var button;
   //switch (textBox){
     if (textBox.id == "CustomCommentBox"){
       button = document.getElementById("custom-feedback-button");
       }
     if (textBox.id == "DeleteCommentBox"){
       button = document.getElementById("delete-comment-button");
     }
   if (textBox.value.trim() == ""){
     button.disabled = true;
   } else {
     button.disabled = false;
   }
 }

 /**
  * Updates the comment button for custom feedback based on a variety of factors
  */
 function updateCustomButton() {
   var button = document.getElementById("custom-feedback-button");
   if (currentTabId == "#tab-audio"){
     if (!mp3DataURL) {
       button.disabled = true;
     } else {
       button.disabled = false;
     }
   } else if (currentTabId == "#tab-text"){
     var textArea = document.getElementById("CustomCommentBox");
     if (textArea.value.trim() == ""){
       button.disabled = true;
     } else {
       button.disabled = false;
```

```
      }
    }
  }


  /**
   * Shows the loading icon.
   */
  function showLoading(id){
    var icon = document.getElementById(id);
    icon.hidden = false;
  }

  /**
   * Hides the loading icon
   */
  function hideLoading(id){
    var icon = document.getElementById(id);
    icon.hidden = true;
  }

</script>


</html>
```

rubric.html

```
<html>
<link rel="stylesheet" href="https://ssl.gstatic.com/docs/script/css/add-ons.css">
<script
src="https://sites.google.com/site/assistmentsfeedbacktool/resources/mcColorPicker.js"
type="text/javascript"></script>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/2.1.3/jquery.min.js"></script>
<link rel="stylesheet"
href="https://sites.google.com/site/assistmentsfeedbacktool/resources/mcColorPicker.css">

<style type="text/css">

<? var rubricNames = getUserRubricNames();
   var loadingRubric = typeof loadedRubric !== 'undefined';
   var loadingRubricName;
   if (loadingRubric) {
     loadingRubricName = loadedRubric;
   } else {


       loadingRubricName = "";

   }
?>

.remove-cat {
  border-style:solid;
  border-width:1px;
```

```
  color:red;
  border-radius:4px;
  margin:1px;
}

.remove-cat:hover {
  text-decoration:none;
  margin:0px;
  border-width:2px;
}

.add-cat {
  text-align:center;
  font-size: 17px;
  font-weight:bold;
  color:#5973e3;
  border-style:solid;
  border-width:1px;
  border-radius:10px;
}

.add-cat:hover {
  text-decoration:none;
  border-width:2px;
}

.rubricBox {
  <? if (rubricNames.length > 0) { ?>
  height: 390px;
  <? } else { ?>
  height: 435px;
  <? } ?>
  width: 750px;
  border: 1px solid;
  padding: 10px;
  overflow:auto

}

.subcategory-div {
  height:40px;
}

.checkbox-sub {
  padding:2px;
  margin:5px;
}

.new-checkbox-sub {
  padding:2px;
  margin:5px;
  margin-left:9px;
  padding-right:5px;
  border-style:solid;
  border-width:1px;
  border-radius:6px;
  color:#616161;
}

.bottomButtons {
```

```
  position: fixed;
  bottom: 0px;
}

.input {
  border: 0;
}

p {
  display: inline;
}

.category {
  float:right;
  margin-right:100px;
}

select {
  width:200px;
}

li {
  margin-bottom:5px;
}

ul {
  margin-bottom:5px;
  margin-top:5px;
}

.add-to-cat-items {
  margin-bottom:15px;
}

.left-indent {
  margin-left:40px;
}

</style>

<? if (rubricNames.length > 0) { ?>
<label for="saved-rubric-select">Load a Saved Rubric:</label>
<select id="saved-rubric-select" onchange="loadUserRubric();">
  <option <? if (!loadingRubric) { ?>selected<? } ?>>Choose a saved rubric:</option>
  <? for (var i = 0; i < rubricNames.length; i++) { ?>
  <option <? if (loadingRubricName == rubricNames[i]) {?>selected<? } ?>><?= rubricNames[i]
?></option>
  <? } ?>
</select>
<input id="rubric-delete" type="button" value="Delete" onclick="deleteSelectedRubric()" <?
if (!loadingRubric) { ?>disabled<? } ?>>
<img id="loading-icon-user-rubric" src="https://www.assistments.org/images/indicator.gif"
class="loading" hidden>
<span id="rubric-delete-confirm" class="gray" hidden></span>
<br><br>
<? } ?>
<div class="rubricBox">
<div class="boxed">
    <p>Pick a highlight color for each category and add quick feedback messages.</p>
    <br>
```

```
    <p>Double click a category or feedback message to edit.</p>
    <br><br>
</div>
<b>Give the Rubric a name to save it for later use</b>
<br>
<label for="rubric-name">Name:</label>
<input id="rubric-name" type="text" onkeyup="validateName();" <? if (loadingRubric)
{?>value="<?=loadingRubricName?>"<? } ?>>
<span id="rubric-name-validate-message" class="error"  ><? if (!loadingRubric) {?>You must
give your rubric a name<? } else{ ?>You already have a rubric with this name. Saving will
overwrite that rubric.<?}?></span>
<br><br>
<b><u>Categories</u></b><br>

<form action="">
<ul id="CATEGORIES" class="expand-collapse">
  <? var categories = rubric.getKeys(); ?>
  <? for (var i = 0; i < categories.length; i++) { ?>
        <? var categoryID = categories[i].concat("-top");?>
        <li id="<?= categoryID?>" name="<?= categories[i] ?>">
          <a onclick="expandCollapse('<?= categories[i] ?>', this);">[&#8211;]</a>
          <a class="remove-cat" onclick="removeCategory('<?= categories[i]
?>')"> Remove </a>
          <input id="<?= categories[i]?>-color" type="hidden" class="color" value="<?=
map.getColor(categories[i]); ?>">
          <b ondblclick="editCategory(this)"><?= categories[i] ?></b>
          <? var subCategories = rubric[categories[i]]; ?>
          <div id="<?= categories[i] ?>-collapse1">
          <? for(var j = 0; j < subCategories.length; j++){ ?>
                <ul id="<?= categories[i] ?>-<?= subCategories[j] ?>" name="<?=
subCategories[j] ?>">
                <a onclick="removeItem('<?= categories[i] ?>-<?= subCategories[j] ?>');"
class="remove-cat"> X </a>
                <span class="checkbox-sub">
                <input id="<?= categories[i] ?>-<?= subCategories[j] ?>-checkbox"
type="checkbox" <? if (rubric.responseRequired[categories[i]][j]) { ?> checked<? } ?>>
                <label for="<?= categories[i] ?>-<?= subCategories[j] ?>-checkbox"
class="secondary">Require Explanation</label>
                </span>
                <p ondblclick="editCategory(this)"><?= subCategories[j] ?></p>
                </ul>
          <? } ?>
          </div>
        </li>
        <div id="<?= categories[i] ?>-collapse2">
          <ul id="<?= categories[i] ?>-input" class="add-to-cat-items">
            <input type="text" name="<?= categories[i] ?>text" style="height:23px;">
            <!-- <input type="button" value="Add to Category" class="action" id="<?=
categories[i] ?>" name="<?= categories[i] ?>text" onclick="addSubItem(id,name)"
style="text-align: center;"> -->
            <a id="<?= categories[i] ?>" name="<?= categories[i] ?>text"
onclick="addSubItem(id,name)" class="add-cat">  +  </a>
            <span id="<?= categories[i] ?>-subcategory-error-msg" class="error" hidden>That
item already exists for this category.</span>
          </ul>
        </div>
  <? } ?>
</ul>
</form>
<div class="left-indent">
```

```
New Category:<br>
<input type="text" id="category" onkeyup="addCatValidation(this);">
<input type="button" value="Add" id="category" name="add-category-button"
onclick="addItem(id, 0)" disabled/>
<span id="existing-category-err" class="error" hidden>A category with that name already
exists.</span>
</div>
<br>
<br>
<br>
<b><u>Who can edit this rubric?</b></u> (Enter an email address)
<ul id="author-list">
  <li
id="<?=Session.getEffectiveUser().getEmail()?>-DO_NOT_REMOVE"><?=Session.getEffectiveUser().
getEmail()?></li>
  <? for (var i = 0; i < authors.length; i++){ ?>
  <li id="<?=authors[i]?>">
  <a onclick="removeAuthor(<?=authors[i]?>)" style="border-style:solid; border-width:1px;
color:red">X</a>
  <?=authors[i]?></li>
  <? } ?>
</ul>
<div class="left-indent">
<input type="text" id="author-input" onkeyup="validateAuthor(this)">
<input id="author-add-button" type="button" value="Add" onclick="addAuthor('author-input')"
disabled>
<span id="Author-Error" class="error" hidden>Please enter a valid email address</span>
<span id="Author-Error2" class="error" hidden>That email address is already in the
list</span>
</div>
<br>
</div>

<div class="bottomButtons">
  <input type="button" value="Save & Set" class="action" id="SaveButton"
onclick="generateRubric();" <?if (!loadingRubric){?>disabled<?}?>/>
  <input type="button" value="Undo Changes" onclick="showLoading('loading-icon-save');
google.script.run.showRubric();">
  <input type="button" value="Cancel" onclick="showLoading('loading-icon-save');
google.script.host.close();">
  <input type="button" value="Reset to Default" onclick="showLoading('loading-icon-save');
google.script.run.resetRubric('showRubric');" style="position:fixed; right:20px;">
  <img id="loading-icon-save" src="https://www.assistments.org/images/indicator.gif"
class="loading" hidden>
</div>

<script language="javascript">

  /**
   * Enables the user to edit a category in the rubric
   */
  function editCategory(category){
    var input = document.createElement("input");
    input.type = "text";
    input.value = category.textContent;
    var tag = category.tagName;
    $(input).bind("blur", [input], function(event){
      var self = event.data[0];
      var msg;
      if(self.value.trim().length > 0){
```

```
        self.parentElement.setAttribute("name", self.value.trim());
        msg = " ondblclick=\"editCategory(this);\">" + self.value;
      }
      else
      {
        msg = " ondblclick=\"editCategory(this);\">" + category.textContent;
      }

      $(self).replaceWith("<" + tag + msg + "</" + tag + ">");
    });

    //category.getParentElement().setAttribute("name", input.value);
    $(category).replaceWith(input);

    //document.getElementById(id).setAttribute("name", input.value);
    //alert(document.getElementById(id).getAttribute("name"));
    input.focus();
  }

  /**
   * Enables and disabled the author add button accordingly.
   */
  function validateAuthor(elt){
    var button = document.getElementById("author-add-button");
    button.disabled = true;
    if (IsEmail(elt.value.trim())){
      button.disabled = false;
    }
  }

  function IsEmail(email) {
    var regex = /^([a-zA-Z0-9_.+-])+\@(([a-zA-Z0-9-])+\.)+([a-zA-Z0-9]{2,4})+$/;
    return regex.test(email);
  }


  /**
   * Removes assocaited author author from the list
   */
  function removeAuthor(email) {
    var elem = document.getElementById(email);
    elem.parentNode.removeChild(elem);
  };

/**
 * Adds an author to the author list
 */
function addAuthor(id){
  var authorError = document.getElementById("Author-Error");
  var authorError2 = document.getElementById("Author-Error2");
  authorError.hidden = true;
  authorError2.hidden = true;
  var elem = document.getElementById(id);
  var email = elem.value.toLowerCase();

  var at = email.indexOf("@");
  var dot = email.lastIndexOf(".");

  if(at > dot || at == -1 || dot == -1){
    authorError.hidden = false;
```

```
    return;
  }
  if (document.getElementById(email) != null ||
document.getElementById(email+"-DO_NOT_REMOVE") != null){
    authorError2.hidden = false;
    return;
  }

  elem.value = "";
  var list = document.getElementById("author-list");

  var removeButton = document.createElement('a');
  removeButton.appendChild(document.createTextNode("X"));
  removeButton.setAttribute("class","remove-cat");
  removeButton.onclick = function() {
    var elem = document.getElementById(email);
    elem.parentNode.removeChild(elem);
  };

  var li = document.createElement("li");
  li.id = email;
  li.appendChild(removeButton);
  var emailText = document.createTextNode(" " + email);
  li.appendChild(emailText);
  list.appendChild(li);
}

/**
 * Removes a subcategory from the rubric
 * @param id the subcategory to be removed
 */
function removeItem(id){
  var elem = document.getElementById(id);
  elem.parentNode.removeChild(elem);
}

/**
 * Removes a category from the rubric, including text box and buttons
 * @param id the category to be removed
 */
function removeCategory(id){
  var elem = document.getElementById(id+"-top");
  elem.parentNode.removeChild(elem);
  var div = document.getElementById(id+"-collapse2");
  div.parentNode.removeChild(div);
}

/**
 * Adds a custom category onto the rubric
 * @param id The id of the item to append
 * @param flag A flag indicating which list to use
 */
function addItem(id, flag){
    var list;
    var err = document.getElementById("existing-category-err");
    err.hidden = true;
    if(flag == 0)
      list = document.getElementById("CATEGORIES");
    else
      list = document.getElementById(id);
```

```
    var elem = document.getElementById(id).value;

    if(elem == "")
      return;

    if (document.getElementById(elem+"-top") != null){
      err.hidden = false;
      return;
    }

    document.getElementById(id).value = "";

    var entry = document.createElement('li');
    entry.id = elem+"-top";
    entry.setAttribute("name", elem);

    var buttonEntry = document.createElement('ul');
    buttonEntry.id = elem+"-input";

    var text = document.createElement('input');
    text.type = "text";
    text.id = elem;
    text.name = elem + "text";
    var name = text.name;
    var addButton = document.createElement('a');
    addButton.innerHTML = "  +  ";
    addButton.setAttribute("class","add-cat");
    var removeButton = document.createElement('a');
    removeButton.appendChild(document.createTextNode(" Remove "));
    removeButton.setAttribute("class","remove-cat");

    var subError = document.createElement('span');
    subError.id = elem + "-subcategory-error-msg";
    subError.setAttribute("class", "error");
    subError.innerHTML = " That item already exists for this category.";
    subError.hidden = true;

    var colorPicker = document.createElement('input');
    colorPicker.setAttribute("id", elem+"-color");
    colorPicker.setAttribute("type", "hidden");
    colorPicker.setAttribute("class", "color");
    colorPicker.setAttribute("value", "#ff0000");

    var collapse = document.createElement('a');
    collapse.setAttribute("onclick", "expandCollapse('" + elem + "', this);");
    collapse.innerHTML = "[&#8211;]";


    var collapseDiv1 = document.createElement('div');
    collapseDiv1.id = elem+"-collapse1";

    var collapseDiv2 = document.createElement('div');
    collapseDiv2.id = elem+"-collapse2";

    addButton.onclick = function(){addSubItem(elem, name)};
    removeButton.onclick = function(){removeCategory(elem, name)};
    buttonEntry.appendChild(text);
    buttonEntry.appendChild(addButton);
    buttonEntry.appendChild(subError);
    collapseDiv2.appendChild(buttonEntry);
```

```
    entry.appendChild(collapse);
    entry.appendChild(removeButton);
    entry.appendChild(colorPicker);
    var testBold = document.createElement("B");
    var testNode = document.createTextNode(" "+elem);
    testBold.setAttribute("ondblclick", "editCategory(this)");
    testBold.appendChild(testNode);
    entry.appendChild(testBold);
    entry.appendChild(collapseDiv1);
    list.appendChild(entry);
    list.appendChild(collapseDiv2);

    MC.ColorPicker.reload();
}

/**
 * Adds a sub section for categories onto the rubric
 * @param id The id of the item to append
 */
function addSubItem(id, name){
    var list = document.getElementById(id+"-collapse1");
    var elem = document.getElementsByName(name)[0].value;
    var subError = document.getElementById(id + "-subcategory-error-msg");
    subError.hidden = true;

    if(document.getElementById(id + "-" + elem) != null){
      subError.hidden = false;
      return;
    }


    document.getElementsByName(name)[0].value = "";
    if(elem == "")
      return;
    var entry = document.createElement('ul');

    entry.id = id+ "-" + elem;
    entry.setAttribute("name", elem);

    var removeButton = document.createElement('a');
    removeButton.innerHTML = " X ";
    removeButton.setAttribute("class", "remove-cat");
    removeButton.onclick = function(){removeItem(entry.id)};

    var span = document.createElement('span');
    span.setAttribute("class", "checkbox-sub");

    var checkbox = document.createElement('input');
    checkbox.setAttribute("type", "checkbox");
    checkbox.setAttribute("id", id + "-" + elem + "-checkbox");

    var label = document.createElement('label');
    label.setAttribute("for", "id" + "-" + elem + "-checkbox");
    label.setAttribute("class", "secondary");
    label.innerHTML = "Require Explanation";

    span.appendChild(checkbox);
    span.appendChild(label);
```

```
    entry.appendChild(removeButton);
    entry.appendChild(span);
    var pNode = document.createElement("P");
    pNode.innerHTML = elem;
    pNode.setAttribute("ondblclick", "editCategory(this)");
    entry.appendChild(pNode);
    list.appendChild(entry);
}

/**
 * Converts HTML list to an array
 * @param id The id of the list to convert
 */
function generateRubric(id){
  showLoading("loading-icon-save");
  var category = [];
  var subcategory = [];
  var required = [];
  var colors = [];
  var list = document.getElementById("CATEGORIES");
  for (var i = 0; i < list.childElementCount/2; i++){
    var catName = list.children[i*2].getAttribute("name");
    if(catName != null)
      category[i] = catName;
    colors.push(list.children[i*2].children[2].value);
    var subArray = [];
    var subArray2 = [];
      for(var j = 0; j < list.children[i*2].getElementsByTagName("UL").length; j++){
        var listItem = list.children[i*2].getElementsByTagName("UL")[j];
        var name = listItem.getAttribute("name");
        if(name != null){
          subArray.push(name);
          //var checkbox = document.getElementById(catName + "-" + name +
"-checkbox").checked;
          var checkbox = listItem.children[1].children[0].checked;
          subArray2.push(checkbox);
        }
      }
    subcategory.push(subArray);
    required.push(subArray2);
  }

  var authors = [];
  var authList = document.getElementById("author-list");
  authors.push(authList.children[0].textContent);
  for (var j = 1; j < authList.childElementCount; j++){
    authors.push(authList.children[j].id);
  }

  var name = "";
  var rName = document.getElementById("rubric-name");
  if (rName != null){
    name = rName.value;
  }
  if (validateName()){
    google.script.run
          .withSuccessHandler(google.script.host.close)
          .withFailureHandler(function(error){alert(error.message);})
          .makeRubric(category, subcategory, authors, colors, name, required);
```

```
  } else {
    google.script.run
          .withSuccessHandler(function() {
             google.script.run
                  .withSuccessHandler(google.script.host.close)
                  .withFailureHandler(function(error){alert(error.message);})
                  .makeRubric(category, subcategory, authors, colors, name, required);
          })
          .withFailureHandler(function(error){alert(error.message);})
          .deleteUserRubric(name);
  }
}

/**function saveAndPreview(){
  generateRubric();
  google.script.run.generateRubricPreview();

}*/

/**
 * Loads the user's selected rubric. This refreshes the window.
 */
function loadUserRubric() {
  showLoading("loading-icon-user-rubric");
  var select = document.getElementById("saved-rubric-select");
  if (select.selectedIndex > 0){
    google.script.run.loadUserRubric(select.value);
  } else {
    hideLoading("loading-icon-user-rubric");
  }
}

/**
 * Validates the name that the user wants to save the rubric as.
 * Displays an error next to the box if there is a conflict.
 * @return true if no conflicts. False otherwise.
 */
function validateName(){
   var label = document.getElementById("rubric-name-validate-message");
   var select = document.getElementById("saved-rubric-select");
   var button = document.getElementById("SaveButton");
   var name = document.getElementById("rubric-name").value;
   if (label == null || select == null){
     if (name.trim() != ""){
     button.disabled = false;
     label.hidden = true;
     } else{
     button.disabled = true;
     label.hidden = false;
     }
     return true;
   }
   label.hidden = true;
   for (var i = 1; i < select.length; i++){
     if (name.trim() == select.options[i].text.trim()){
       label.innerHTML = "You already have a rubric with this name. Saving will overwrite
that rubric.";
       label.hidden = false;
       button.disabled = false;
       return false;
```

```
      }
    }
    if(name.trim() == ""){
        label.innerHTML = "You must give your rubric a name";
        label.hidden = false;
        button.disabled = true;
        return false;
    }
    button.disabled = false;
    return true;
}

/**
 * Deletes the selected rubric from the user.
 */
function deleteSelectedRubric(){
  showLoading("loading-icon-user-rubric");
  var select = document.getElementById("saved-rubric-select");
  var confirm = document.getElementById("rubric-delete-confirm");
  var idx = select.selectedIndex;
  google.script.run
    .withSuccessHandler(function(){
        confirm.innerHTML = select.options[idx].text + " has been deleted successfully";
        confirm.hidden = false;
        select.remove(idx);
        hideLoading("loading-icon-user-rubric");
       })
    .deleteUserRubric(select.options[idx].text);
}

/**
 * Expands and collapses a section
 */
 function expandCollapse(id, elt){
   var div1 = document.getElementById(id+"-collapse1");
   var div2 = document.getElementById(id+"-collapse2");
   if (div1.hidden && div2.hidden){
     div1.hidden = false;
     div2.hidden = false;
     elt.innerHTML = "[&#8211;]";
   } else {
     div1.hidden = true;
     div2.hidden = true;
     elt.innerHTML = "[+]";
   }
 }

/**
 * Disables and enables the button depending on if there is text in the window
 */
function addCatValidation(elt){
//  var button = document.getElementById("add-category-button");
  var button = document.getElementsByName("add-category-button")[0];
  var err = document.getElementById("existing-category-err");
  err.hidden = true;
  button.disabled = false;

  if (elt.value.trim() == ""){
    button.disabled = true;
  }
```

```
  if (document.getElementById(elt.value+"-top") != null){
    err.hidden = false;
    button.disabled = true;
  }

}

 /**
  * Shows the loading icon.
  */
 function showLoading(id){
   var icon = document.getElementById(id);
   icon.hidden = false;
 }

 /**
  * Hides the loading icon
  */
 function hideLoading(id){
   var icon = document.getElementById(id);
   icon.hidden = true;
 }

</script>

</html>
```

Created Code

Grading.gs

```
function listFiles(assignmentName, grades){    //CHANGE THAT NAME
  var cats = getRubric().getKeys();
  var doc = DocumentApp.getActiveDocument();
  var totalFlag=PropertiesService.getUserProperties().getProperty("GradingTotal");
  var gradingSyntax=PropertiesService.getUserProperties().getProperty("GradingSyntax");
  var sheet;
  var oldLastRow;
  var file;
  if(gradingSyntax==null){
    gradingSyntax="Numbers";
  }
  if (assignmentName == ""){
    DocumentApp.getUi().alert("You can't grade an assignment without an assignment name.");
    return;
  }

  var folderIterator = DriveApp.getFoldersByName("docASSIST GradeBook");
  if (folderIterator.hasNext()){
```

```
    var folder = folderIterator.next();
    folder.setName("GradeBook");
    getDriveFolder().addFolder(folder);
    DriveApp.removeFolder(folder);

  }

  if(getDriveFolder().getFoldersByName("GradeBook").hasNext() == false){
    var docAssistFolder = getDriveFolder().createFolder("GradeBook");
    var spreadsheetName = assignmentName;
    var spreadsheet = SpreadsheetApp.create(spreadsheetName);
    sheet = spreadsheet.getActiveSheet();
    oldLastRow=sheet.getLastRow();
    var row = ["Document Name"];
    var row2 = [doc.getName()];
    var total = 0;
    for (var i = 0; i < cats.length; i++){
      row.push(cats[i]);
      if (grades[i]){
        row2.push(grades[i]);
      } else{
        if(gradingSyntax=="Numbers"){
          row2.push("0");
        }
        else{
          row2.push("");
        }
      }
      total += grades[i];
    }
    if(totalFlag=="true"){
    row.push("Total");
    }
    row.push("Date");
    var formattedDate = Utilities.formatDate(new Date(), "GMT", "yyyy-MM-dd");
      if(totalFlag=="true"){
      if (total == 0){total = "0";}
      row.push(total);
      }
      else if(sheet.getRange(1,cats.length + 2).getValues()[0][0] == "Total"){
      row.push(" ");
      }
      row.push(formattedDate);
    sheet.appendRow(row);
    for (var i = 0; i < cats.length; i++){
        var cell = sheet.getRange(1, 2 + i);
        var color = getRubricColors().getColor(cats[i]);
        cell.setBackground(color);
    }
    sheet.appendRow(row2);
    resizeColumns(sheet);
    file = DriveApp.getFilesByName(spreadsheet.getName()).next();
    docAssistFolder.addFile(file);

    //file.makeCopy(DocumentApp.getActiveDocument().getName() + " docASSIST spreadsheet",
DriveApp.getFoldersByName("docASSIST").next());
    DriveApp.removeFile(file);
  }
  else{ //Folder exists
    var docAssistFolder = getDriveFolder().getFoldersByName("GradeBook").next();
```

```
    var spreadsheetName = assignmentName;
    var spreadsheet;
    if (docAssistFolder.getFilesByName(spreadsheetName).hasNext()){ //Spreadsheet also
exists
      spreadsheet = docAssistFolder.getFilesByName(spreadsheetName).next();
      file = DriveApp.getFilesByName(spreadsheetName).next();
      spreadsheet = SpreadsheetApp.open(file);
      sheet = spreadsheet.getActiveSheet();
      oldLastRow=sheet.getLastRow();
      var row = [doc.getName()];
      var columnCategory;
      var found = [];
      var didFind = false;
      var total = 0;
      for (var i = 2; i < sheet.getLastColumn(); i++){ //removed - 1
        columnCategory = sheet.getRange(1,i).getValues()[0][0];
        Logger.log("columnCatgory = " + columnCategory);
        if(columnCategory == "Total"){break;}
        var j;
        didFind = false;
        for (j = 0; j < cats.length; j++){
          if (columnCategory == cats[j]){
            found.push(cats[j]);
            didFind = true;
            if (grades[j]){
              row.push(grades[j]);
            } else{
              if(gradingSyntax=="Numbers"){
        row.push("0");
        }
        else{
          Logger.log("push-1");
          row.push("");
        }
            }
            total+=grades[j];
            break;
          }
        }
        if (!didFind){
          Logger.log("push0");
          row.push("");
        }
      }
      for (var i = 0; i < cats.length; i++){
        didFind = false;
        for (var j = 0; j < found.length; j++){
          if (found[j] == cats[i]){
            didFind = true;
            break;
          }
        }
        if (didFind == false){
          Logger.log("Didn't find the thing!");
          sheet.insertColumnBefore(sheet.getLastColumn() - 1);
          Logger.log("Inserted us a new column!");
          var cell = sheet.getRange(1,sheet.getLastColumn()-2);
          cell.setValue(cats[i]);
          cell.setBackground(getRubricColors().getColor(cats[i]));
          Logger.log("Put a value in the place!");
```

```
        if (grades[i] != 0 && grades[i] != null){
          row.push(grades[i]);
        } else{
          if(gradingSyntax=="Numbers"){
            row.push("0");
          }
          else{
            Logger.log("push1");
            row.push("");
          }
        }
        total+=grades[i];
      }
    }
    var formattedDate = Utilities.formatDate(new Date(), "GMT", "yyyy-MM-dd");
    if(totalFlag=="true"){
    if (total == 0){total = "0";}

    if(sheet.getRange(1,cats.length + 2).getValues()[0][0] != "Total"){
        sheet.insertColumnBefore(sheet.getLastColumn());
        var cell = sheet.getRange(1,sheet.getLastColumn()-1);
        cell.setValue("Total");
        cell.setBackground('white');
      }

    row.push(total);
    }
    else if(sheet.getRange(1,cats.length + 2).getValues()[0][0] == "Total"){
      Logger.log("push2");
    row.push(" ");
    }
    row.push(formattedDate);



    sheet.appendRow(row);
  } else { //Folder exists, spreadsheet does not.
    var spreadsheet = SpreadsheetApp.create(spreadsheetName);
    while (!DriveApp.getFilesByName(spreadsheetName).hasNext());
    file = DriveApp.getFilesByName(spreadsheetName).next();
    docAssistFolder.addFile(file);
    spreadsheet = SpreadsheetApp.open(file);
    sheet = spreadsheet.getActiveSheet();
    oldLastRow=sheet.getLastRow();
    var row = ["Document Name"];
    var row2 = [doc.getName()];
    var total = 0;
    for (var i = 0; i < cats.length; i++){
      row.push(cats[i]);
      if (grades[i]){
        row2.push(grades[i]);
      } else{
        if(gradingSyntax=="Numbers"){
        row2.push("0");
      }
      else{
        row2.push("");
      }
      }
      total += grades[i];
```

```
      }
      if(totalFlag=="true"){
      row.push("Total");
      }
      row.push("Date");
      var formattedDate = Utilities.formatDate(new Date(), "GMT", "yyyy-MM-dd");
      if(totalFlag=="true"){
      if (total == 0){total = "0";}
      row2.push(total);
      }
      else if(sheet.getRange(1,cats.length + 2).getValues()[0][0] == "Total"){
      row2.push(" ");
      }
      row2.push(formattedDate);
      sheet.appendRow(row);
      for (var i = 0; i < cats.length; i++){
        var cell = sheet.getRange(1, 2 + i);
        var color = getRubricColors().getColor(cats[i]);
        cell.setBackground(color);

      }
      sheet.appendRow(row2);
      DriveApp.removeFile(file);
    }
    resizeColumns(sheet);
    generateGrading(grades);
    var newLastRow = sheet.getLastRow();
    if(newLastRow > oldLastRow){
      return file.getUrl();
    }
  }
}


function resizeColumns(sheet){
  for(var i = 1; i <=sheet.getLastColumn(); i++){
    sheet.setColumnWidth(i, 110);
  }
}

/**
function confirmGrading(assignmentName, grades){
   var cats = getRubric().getKeys();
   var docAssistFolder = DriveApp.getFoldersByName("GradeBook").next();
    var spreadsheetName = assignmentName;
    var spreadsheet;
    var check = "confirmed";
    if (docAssistFolder.getFilesByName(spreadsheetName).hasNext()){
      spreadsheet = docAssistFolder.getFilesByName(spreadsheetName).next();
      file = DriveApp.getFilesByName(spreadsheetName).next();
      spreadsheet = SpreadsheetApp.open(file);
      var sheet = spreadsheet.getActiveSheet();
      var lastRow = sheet.getLastRow();
      for (var i = 2; i < sheet.getLastColumn()-1; i++){
        var cellValue = sheet.getRange(lastRow, i).getValues()[0][0];
        DocumentApp.getUi().alert("Cell Value = " + cellValue + " grade = " + grades[i]);
        if(cellValue != grades[i]){
        check="invalid";
        }
      }
```

```
      DocumentApp.getUi().alert(check);
      return check;
    }

}
*/
```

Languages.gs

```
function translate(phrase, authObj) {

  //For now, translate is not implemented
  //return phrase;


  var language;
  if (!authObj || (authObj && authObj.authMode != ScriptApp.AuthMode.NONE)){
    language = PropertiesService.getUserProperties().getProperty("docASSISTLanguage");
  }
  if(language == null){
     language = Session.getActiveUserLocale();
  }
  switch(language){
    case "en":
      return phrase;
    case "es":
      switch(phrase){
        case "Rubric":
          return "Escala";
        case "Review":
          return "Revisar";
        case "Revise":
          return "Repasar";
        case "Rubric Setup":
          return "Preparar Escala";
        case "Select Rubric":
          return "Seleccionar Escala";
        case "Show Reviewers' Identity":
          return "Identificar Revisores";
        case "Options":
          return "Opciones";
        case "Default Rubric":
          return "Escala Base";
        case "Deleting Feedback":
          return "Borrar Comentario";
        case "Category":
          return "Categoría";
        case "Delete Feedback":
          return "Borrar Comentario";
        case "Require Explanation":
          return "Requerir Explicación";
        case "You do not need to highlight text for this type of feedback.":
          return "Usted no necesita resaltar texto para hacer este tipo de comentario.";
        case "Add Feedback":
          return "Hacer Comentario";
```

```
        case "Summarize & Grade":
          return "Resumir y Poner Nota";
        case "Email Report":
          return "Enviar Email";
        case "Gradesheet Name":
          return "Hoja de Notas";
        case "Save":
          return "Guardar";
        case 'Grades saved. Your gradesheet can be found in "docAssist Gradebook" in your
Google Drive.':
          return 'Notas guardadas. Usted puede encontrar sus notas en "docAssist Gradebook"
en su Google Drive.';
        case "Grade":
          return "Poner Notas";
        case "Your Feedback Notes":
          return "Sus Comentarios";
        case "You don't have any feedback notes yet":
          return "Usted todavía no tiene comentarios.";
        case "Update Summary":
          return "Actualizar Resumen";
        case "Optional Explanation":
          return "Explicación Opcional";
        case "note":
          return "nota";
        case "Required Explanation":
          return "Explanación Requerido";
        case "Your teacher requires you to write an explanation for this resolution.":
          return "Su profesor requiere que usted escriba una explicaión para esta
resolución.";
        case "You do not need to write an explanation for this resolution.":
          return "Usted no necesita escribir una explicaión para esta resolución.";
        case "Reply":
          return "Responder";
        case "Resolve":
          return "Resolver";
        case "Reviewer Emails":
          return "Emails de los Revisores";
        case "Author email addresses":
          return "Email del Autor";
        case "Reviewer":
          return "Revisor";
        case "Close":
          return "Cerrar";
        case "Options":
          return "Opciones";
        case "Grading":
          return "Poner Notas";
        case "By text":
          return "Con texto";
        case "By numbers":
          return "Con números";
        case "Compute total in gradesheet":
          return "Totalizar";
        case "Language":
          return "Idioma";
        case "English":
          return "Inglés";
        case "Spanish":
          return "Español";
        case "French":
```

```
          return "Francés";
        case "Arabic":
          return "Árabe";
        case "Cancel":
          return "Cancelar";
        case "":
          return "";
        case "":
          return "";
        case "":
          return "";

        default:
          return LanguageApp.translate(phrase, "en", "es");
      }
    case "ar":
      switch(phrase){
        case "Rubric":
          return "";
        case "Review":
          return "";
        case "Revise":
          return "";
        case "Rubric Setup":
          return "";
        case "Select Rubric":
          return "";
        case "Show Reviewers' Identity":
          return "";
        case "Options":
          return "";
        case "Default Rubric":
          return "";
        case "Deleting Feedback":
          return "";
        case "Category":
          return "";
        case "Delete Feedback":
          return "";
        case "Require Explanation":
          return "";
        case "You do not need to highlight text for this type of feedback.":
          return "";
        case "Add Feedback":
          return "";
        case "Summarize & Grade":
          return "";
        case "Email Report":
          return "";
        case "Gradesheet Name":
          return "";
        case "Save":
          return "";
        case 'Grades saved. Your gradesheet can be found in "docAssist Gradebook" in your
Google Drive.':
          return "";
        case "Grade":
          return "";
        case "Your Feedback Notes":
          return "";
```

```
        case "You don't have any feedback notes yet":
          return "";
        case "Update Summary":
          return "";
        case "Optional Explanation":
          return "";
        case "note":
          return "";
        case "Required Explanation":
          return "";
        case "Your teacher requires you to write an explanation for this resolution.":
          return "";
        case "You do not need to write an explanation for this resolution.":
          return "";
        case "Reply":
          return "";
        case "Resolve":
          return "";
        case "Reviewer Emails":
          return "";
        case "Author email addresses":
          return "";
        case "Reviewer":
          return "";
        case "Close":
          return "";
        case "Options":
          return "";
        case "Grading":
          return "";
        case "By text":
          return "";
        case "By numbers":
          return "";
        case "Compute total in gradesheet":
          return "";
        case "Language":
          return "";
        case "English":
          return "";
        case "Spanish":
          return "";
        case "French":
          return "";
        case "Arabic":
          return "";
        case "cancel":
          return "";
        case "":
          return "";
        case "":
          return "";
        case "":
          return "";

        default:
          return phrase;
      }
    case "fr":
      switch(phrase){
```

```
        case "Rubric":
          return "Échelle";
        case "Review":
          return "Réviser";
        case "Revise":
          return "Revoir";
        case "Rubric Setup":
          return "Préparer l'échelle";
        case "Select Rubric":
          return "Sélectionner l'échelle";
        case "Show Reviewers' Identity":
          return "Identifier les reviseurs";
        case "Options":
          return "Options";
        case "Default Rubric":
          return "Échelle par défaut";
        case "Deleting Feedback":
          return "Supprimer les commentaires";
        case "Category":
          return "Catégorie";
        case "Delete Feedback":
          return "Supprimer les commentaires";
        case "Require Explanation":
          return "Demander une explication";
        case "You do not need to highlight text for this type of feedback.":
          return "Vous n'avez pas besoin de sélectionner (surligner) le texte pour ce type
des commentaires";
        case "Add Feedback":
          return "Ajouter une commentaire";
        case "Summarize & Grade":
          return "Résumer et noter";
        case "Email Report":
          return "Envoyer le rapport par email";
        case "Gradesheet Name":
          return "Feuille de notes";
        case "Save":
          return "Sauvegarder";
        case 'Grades saved. Your gradesheet can be found in "docAssist Gradebook" in your
Google Drive.':
          return 'Notes sauvegardées. Vous pouvez trouver votre feuille de notes dans le
dossier "docAssist Gradebook" dans votre Google Drive';
        case "Grade":
          return "Noter";
        case "Your Feedback Notes":
          return "Votre commentaires";
        case "You don't have any feedback notes yet":
          return "Vous n'avez pas des commentaires encore";
        case "Update Summary":
          return "Actualiser le résumé";
        case "Optional Explanation":
          return "Explication optionnelle";
        case "note":
          return "note";
        case "Required Explanation":
          return "Explication obligatoire";
        case "Your teacher requires you to write an explanation for this resolution.":
          return "Votre professeur demande que vous écriviez une explication pour cette
solution";
        case "You do not need to write an explanation for this resolution.":
          return "Vous ne devez pas écrire une explication pour cette solution";
```

```
        case "Reply":
          return "Répondre";
        case "Resolve":
          return "Résoudre";
        case "Reviewer Emails":
          return "Les emails des reviseurs";
        case "Author email addresses":
          return "L'email de l'auteur";
        case "Reviewer":
          return "Reviseur";
        case "Close":
          return "Fermer";
        case "Options":
          return "Options";
        case "Grading":
          return "Noter";
        case "By text":
          return "Par texte";
        case "By numbers":
          return "Par chiffres";
        case "Compute total in gradesheet":
          return "Totaliser";
        case "Language":
          return "Langue";
        case "English":
          return "Anglais";
        case "Spanish":
          return "Espagnol";
        case "French":
          return "Français";
        case "Arabic":
          return "Arabe";
        case "Cancel":
          return "Annuler";
        case "":
          return "";
        case "":
          return "";
        case "":
          return "";

        default:
          return phrase;
      }
    default:
      return phrase;
  }


}
```

RubricPreview.gs

```
var PREVIEWKEY = "Rubric Preview";

function generateRubricPreview() {

  var body = DocumentApp.getActiveDocument().getBody();
  var rubricName =
PropertiesService.getDocumentProperties().getProperty("ASSISTmentsRubricName");
  var nextElement;

  //If we find the title of the rubric and a table, don't add the preview
  if (nextElement = body.findText("Rubric: \"" + rubricName + "\"") || (rubricName ==
"Default Rubric" && (nextElement = body.findText(rubricName)))){
    if (body.findElement(DocumentApp.ElementType.TABLE)){
      return;
    }
  }

  var id = PropertiesService.getDocumentProperties().getProperty(PREVIEWKEY);

  removeRange(id);

  //Range builder to avoid duplication of the preview
  var rangeBuilder = DocumentApp.getActiveDocument().newRange();

  var body = DocumentApp.getActiveDocument().getBody();
  var pagebreak = body.appendPageBreak();

  rangeBuilder.addElement(pagebreak);

  if ((PropertiesService.getDocumentProperties().getProperty("ASSISTmentsRubricName")) ==
"Default Rubric") {
    var h4 = body.appendParagraph("Default Rubric");
  } else {
    var h4 = body.appendParagraph(translate("Rubric") + ": \"" +
PropertiesService.getDocumentProperties().getProperty("ASSISTmentsRubricName") + "\"");

  }
  h4.setHeading(DocumentApp.ParagraphHeading.HEADING1);
  rangeBuilder.addElement(h4);
  var cats = getRubric();
  var sections = cats.getKeys();
  var tableArray = [];

  for (var i = 0; i < sections.length; i++){

      input = sections[i];

     //var highlightstyle = {};
     //highlightstyle[DocumentApp.Attribute.BOLD] = true;
     //input.setAttributes(highlightstyle);


      var subCategories = cats[sections[i]];
      input = input.concat(":"+"");
      for (var j = 0; j < subCategories.length; j++){
        input = input.concat("\n");
        k = j+1;
        input = input.concat("    "+ k+". "+subCategories[j]);
      }
      //input = input.concat("    "+subCategories.length+".
```

```
"+subCategories[subCategories.length-1]);
      tableArray.push(["", input]);
    }

   var table = body.appendTable(tableArray);
   rangeBuilder.addElement(table);

  for (var j = 0; j < sections.length; j++){
    var headerCell = table.getCell(j,0);
    headerCell.setBackgroundColor(getRubricColors().getColor(sections[j]));
  }

  for (var j = 0; j < sections.length; j++){
    var secondCell = table.getCell(j,1);
    var Text = secondCell.editAsText();
    var boldStyle = {};
    boldStyle[DocumentApp.Attribute.BOLD]=true;
    var boldOffset = sections[j].length;
    Text.setAttributes(0, boldOffset, boldStyle);

   // Text.setHeading(DocumentApp.ParagraphHeading.HEADING3);
  }
  var tableStyle = {};
  //tableStyle[DocumentApp.Attribute.BOLD] = true;
  table.setAttributes(tableStyle);
  table.setColumnWidth(0, 1);
  //showSidebar();

  var id = DocumentApp.getActiveDocument().addNamedRange("Rubric Preview",
rangeBuilder.build()).getId();

  PropertiesService.getDocumentProperties().setProperty(PREVIEWKEY, id);
}
```

saveOptions.gs

```
function saveOptions(numbers, total, customAudio, explanationAudio, languageSelected){
//readd public,  languageSelected
  if (numbers){
      PropertiesService.getUserProperties().setProperty("GradingSyntax", "Numbers");
      if (total){
        PropertiesService.getUserProperties().setProperty("GradingTotal", "true");
      } else{
        PropertiesService.getUserProperties().setProperty("GradingTotal", "false");
      }
    } else{
      PropertiesService.getUserProperties().setProperty("GradingSyntax", "Text");
      PropertiesService.getUserProperties().setProperty("GradingTotal", "false");
    }
  Logger.log(JSON.stringify(PropertiesService.getUserProperties().setProperties({
    CustomAudio:customAudio,
    ExplanationAudio:explanationAudio,
  }).getProperties()));
   // if (public){
    //  PropertiesService.getUserProperties().setProperty("PrivacyDefault", "public");
```

```
    //}else{
     // PropertiesService.getUserProperties().setProperty("PrivacyDefault", "private");
    //}
  PropertiesService.getUserProperties().setProperty("docASSISTLanguage", languageSelected);
  onOpen();
  var current = PropertiesService.getDocumentProperties().getProperty("currentSidebar");
  switch(current){
    case "Review":
      showSidebar();
      break;
    case "Revise":
      showPeerReview();
      break;
    case "Grade":
      showGrading();
      break;
  }
}
```

grading.html

```
<!DOCTYPE html>
<html>
<link rel="stylesheet" href="https://ssl.gstatic.com/docs/script/css/add-ons.css">
<script src="https://ajax.googleapis.com/ajax/libs/jquery/2.1.3/jquery.min.js"></script>
<script
src="https://ajax.googleapis.com/ajax/libs/jqueryui/1.11.3/jquery-ui.min.js"></script>
<img id="loading-icon" src="https://www.assistments.org/images/indicator.gif"
class="loading" hidden>
<div id="div-wrapper">

<div class="titleName">
    <? if (PropertiesService.getDocumentProperties().getProperty("ASSISTmentsRubricName") ==
"Default Rubric"){ ?>
        <b>Default Rubric </b>
    <?} else{?>
    <b> Rubric: "<?=
PropertiesService.getDocumentProperties().getProperty("ASSISTmentsRubricName") ?>" <?}?></b>
</div>

<script>
$(document).ready(function() {
  <?

     if(DriveApp.getFoldersByName("GradeBook").hasNext() == true){

     var array = [];
     var files = DriveApp.getFoldersByName("GradeBook").next().getFiles()
     var sheet;
     var recent;
     if (files.hasNext()){
       recent = files.next();
       array.push(recent.getName());
     }
     while(files.hasNext()){
```

```
        sheet = files.next();
        if (recent.getLastUpdated() < sheet.getLastUpdated()){
          recent = sheet;
        }
        array.push(sheet.getName());
     };
     ?>
   var recentName = "";
   var array2 = [];
  <? for(var i = 0; i < array.length; i++){ ?>
        array2.push("<?= array[i] ?>");
  <? }
  if (recent != null){ ?>
    recentName = "<?= recent.getName() ?>";
   <? }}
  //document.getElementById("assignmentName").value = "bro";
  ?>
  setDefaultValue(recentName);
  $("#assignmentName").autocomplete({
    source:array2,
  });

});

</script>



<style type="text/css">
.titleName {
  font-size: 14px;
}

.loading {
  position:fixed;
  top: 5px;
  right: 5px;
  height:25px;
  width:25px;
}

.ui-helper-hidden-accessible {
  display:none;
}

.ui-autocomplete {
  position: fixed;
  background:#ffffff;
  height: 75px;
  width:175px;
  border: solid 1px;
  overflow-x:hidden;
  overflow-y:auto;
}

ul.ui-autocomplete {
    list-style: none;
    padding: 0px;
    margin: 0px;
```

```
}
.color-box {
    position: fixed;
    float: right;
    width: 10px;
    height: 10px;
    display: inline-block;
}

ol#menu {
  width: 255px;
  margin-top: 10px;
  padding: 0;
}


ol#menu ol {
  display: none;
  margin: 0;
  border-radius:7px;
}

ol#menu li,
  ol#menu a {
  font-family: arial, arial;
  font-size: 14px;

  margin-bottom: 10px;
  margin-top: 10px;
}

ol#menu li {
  line-height: 15px;
}

ol#menu ol li {
  border-bottom: none;
}


ul#menu ol li:before {
  content: "";
}

ol#menu a:hover {
  font-weight: bold;
  text-decoration: none;
}

ol#menu a.active {
  font-weight: bold;
}

.bottom-logo {
  position:fixed;
  bottom:0px;
  left: 60px;
}

#div-wrapper {
```

```
  overflow:auto;
  position:absolute;
  left: 10px;
  right: 0px;
  top: 0px;
  padding-top: 5px;
  border-bottom: 1px solid black;
  height: calc(100% - 80px);
}
</style>
<body>

<ol id = "menu" style="list-style-type: none;">
  <? var sections = categories.getKeys(); ?>
  <? var gradeBoxes = [] ; ?>
  <? for (var i = 0; i < sections.length; i++) { ?>
       <li>
       <svg width="10" height="10"> <rect width="10" height="10" style="fill:<?=
getRubricColors().getColor(sections[i]); ?>"/></svg>
       <input type="<?= isNumeric ? "number" : "text" ?>" id="GradeBox" name="GradeBox"
style="width:50px; position: relative;" placeholder="<?= isNumeric ? "0" : "" ?>">
       <text><?= sections[i]?></text>
       </li>
       <? } ?>
</ol>
<b style="font-size:14px"><?=translate("Gradesheet Name");?></b><br>
<input type = "text" id = "assignmentName" name = "assignmentName" style="width:200px;"
placeholder="Gradesheet for rubric <?=
PropertiesService.getDocumentProperties().getProperty("ASSISTmentsRubricName") ?>">
<br>
<?=translate("Note: a new name will create a new Gradesheet");?>
<br>
<input id="grading-button" class = "action" type="button" value= "<?=translate('Save');?>"
onclick="grade();">
<input id="generate-summary-button" type="button" value="<?=translate('Review');?>"
onclick="backToReview();">
<br>
<span id = "gradingConfirm" style = "color:green"  hidden><?=translate('Grades saved. Your
gradesheet can be found in "GradeBook" in your Google Drive.');?></span>
<br>
<br>
<br>
</body>
</div>
<a href="http://docassist.assistments.org/" target="_blank" class="bottom-logo">
<img
src="https://sites.google.com/site/assistmentsfeedbacktool/resources/docASSIST-sidebar.png"
align="bottom" width="175">
</a>
<script language="javascript">

 function grade(){

   var grades = [];
   var gradeBoxes = document.getElementsByName("GradeBox");
   var assignmentName = document.getElementById("assignmentName").value;
   var confirmMessage = document.getElementById("gradingConfirm");
   for (var i = 0; i < gradeBoxes.length; i++){
     var grade = gradeBoxes[i].value;
```

```
      <? if (isNumeric) { ?>
      grade = parseFloat(grade);
      <? } ?>

      grades.push(grade);
      if (grades[i] == null){
        grades[i] = 0;
      }
    }

    var button = document.getElementById("grading-button");
    button.disabled = true;
    showLoading("loading-icon");
    google.script.run
                .withSuccessHandler(function(check){
                  hideLoading("loading-icon");
                  button.disabled = false;
                  if(check){
                    var gradingConfirm = document.getElementById("gradingConfirm");
                    var confirmString = "Grades saved. Your gradesheet can be found in
\"GradeBook\" in your Google Drive";
                    gradingConfirm.innerHTML = confirmString.replace("gradesheet", "<a
target=\"_blank\" href=\"" + check + "\">gradesheet</a>");
                    gradingConfirm.hidden = false;
                  } else {
                    alert("Grading was unsuccessful. Please try again");
                  }
                })
                .withFailureHandler(function(error){ hideLoading("loading-icon");
alert(error.message); button.disabled = false;})
                .listFiles(assignmentName, grades);
}

function backToReview(){

 showLoading("loading-icon");
 google.script.run
    .withSuccessHandler(function(){
      hideLoading("loading-icon");
    })
    .withFailureHandler(function(error){
      hideLoading("loading-icon");
      alert(error.message);
    })
    .showSidebar();

}

/**
  * Shows the loading icon.
  */
 function showLoading(id){
   var icon = document.getElementById(id);
   icon.hidden = false;
 }

 function setDefaultValue(text){
   document.getElementById("assignmentName").value = text;
 }
 /**
```

```
  * Hides the loading icon
  */
 function hideLoading(id){
   var icon = document.getElementById(id);
   icon.hidden = true;
 }


</script>


</html>
```

options.html

```
<!DOCTYPE html>
<html>
  <link rel="stylesheet" href="https://ssl.gstatic.com/docs/script/css/add-ons.css">
  <img id="loading-icon" src="https://www.assistments.org/images/indicator.gif"
class="loading" hidden>
  <script>
  <? var totalFlag = PropertiesService.getUserProperties().getProperty("GradingTotal");
     var gradingSyntax =
PropertiesService.getUserProperties().getProperty("GradingSyntax");
     var privacyDefault =
PropertiesService.getUserProperties().getProperty("PrivacyDefault");
     var customAudio = PropertiesService.getUserProperties().getProperty("CustomAudio") ||
"Audio";
     var explanationAudio =
PropertiesService.getUserProperties().getProperty("ExplanationAudio") || "Text";
     var language = PropertiesService.getUserProperties().getProperty("docASSISTLanguage");
     if(language == null){
     language = Session.getActiveUserLocale();
     }
     //var language = "en";
  ?>
  </script>
<style type="text/css">
.bottomButtons {
  position: fixed;
  bottom: 0px;
}

.loading {
  position: fixed;
  top: 125px;
  left: 150px;
}

.audioSelect {
  position: absolute;
  left: 130px;
}
```

```
.audioDiv {
  line-height: 29px;
}

</style>
  <head>
    <base target="_top">
  </head>
  <body>
    <b><?=translate("Grading");?>:</b>
    <br>
    <form action="">
    <input type="radio" name="grade" id="text" onclick="enableSave(); disableTotal();"<?if
(gradingSyntax=="Text"){?>checked<?}?>><label for="text"><?=translate("By
text");?></label><br>
    <input type="radio" name="grade" id="numbers" onclick="enableSave(); enableTotal();"<?if
(gradingSyntax=="Numbers" || !gradingSyntax){?>checked<?}?>><label for="numbers"><?=
translate("By numbers");?>     </label>
    <input type="checkbox" name="grade" id="total" onclick="enableSave();" <?if
(totalFlag=="true"){?>checked<?}?> <?if (gradingSyntax=="Text"){?>disabled<?}?>><label
for="total"><?=translate("Compute total in gradesheet");?></label><br>
    <br>
    </form>
    <!--
    <br>
    <b>Privacy:</b>
    <br>
    <form>
    <input type="radio" name="privacy" id="public" onclick="enableSave();" <?if
(privacyDefault=="public"){?>checked<?}?>> Rubrics are public by default<br>
    <input type="radio" name="privacy" id="private" onclick="enableSave();" <?if
(privacyDefault=="private"){?>checked<?}?>> Rubrics are private by default<br><br>
    </form>

    --->
    <br>
    <b>Audio Preferences:</b>
    <div class="audioDiv">Custom Feedback:
      <select class="audioSelect" id="custom-audio" onchange="enableSave()">
        <option <? if (customAudio == "Audio") {?>selected<?}?>>Audio</option>
        <option <? if (customAudio == "Text") {?>selected<?}?>>Text</option>
      </select>
    </div>
    <br>
    <div class="audioDiv">Explanations:
      <select class="audioSelect" id="explanation-audio" onchange="enableSave()">
        <option <? if (explanationAudio == "Audio") {?>selected<?}?>>Audio</option>
        <option <? if (explanationAudio == "Text") {?>selected<?}?>>Text</option>
      </select>
    </div>

    <br>
    <b><?=translate("Language");?>:</b>
    <br>
    <select name="docASSISTLanguage" id="languageSelection" onchange="enableSave();">
    <option value="en" <?if
(language=="en"){?>selected<?}?>><?=translate("English");?></option>
    <option value="es" <?if
(language=="es"){?>selected<?}?>><?=translate("Spanish");?></option>
    <option value="fr" <?if
```

```
(language=="fr"){?>selected<?}?>><?=translate("French");?></option>
    <option value="ar" <?if
(language=="ar"){?>selected<?}?>><?=translate("Arabic");?></option>
    </select>

    <div class="bottomButtons">
    <input type="button" value="<?=translate("Save");?>" class="action" id="SaveButton"
onclick="saveOptions();" disabled/>
    <input type="button" value="<?=translate("Cancel");?>"
onclick="showLoading('loading-icon-save'); google.script.host.close();">
    <img id="loading-icon-save" src="https://www.assistments.org/images/indicator.gif"
class="loading" hidden>
    </div>

  </body>
  <script language="javascript">
    function enableSave(){
        document.getElementById("SaveButton").disabled=false;
    }

    /**
  * Shows the loading icon.
  */
 function showLoading(id){
   var icon = document.getElementById(id);
   icon.hidden = false;
 }

 function saveOptions(){
    showLoading("loading-icon");
    var numbers = document.getElementById("numbers").checked;
    var total = document.getElementById("total").checked;
    // var public = document.getElementById("public").checked;
    var languageSelection = document.getElementById("languageSelection");
    var languageSelected = languageSelection.options[languageSelection.selectedIndex].value;
    var customAudio = document.getElementById("custom-audio").value;
    var explanationAudio = document.getElementById("explanation-audio").value;

    google.script.run
            .withSuccessHandler(function(){hideLoading("loading-icon");
google.script.host.close();})
            .withFailureHandler(function(error){hideLoading("loading-icon");
alert(error.message)})
            .saveOptions(numbers, total, customAudio, explanationAudio, languageSelected);
//readd public

 }

 var totalChecked;

 function disableTotal(){
   var total = document.getElementById("total");
   total.disabled = true;
   totalChecked = total.checked;
   total.checked = false;
 }

 function enableTotal(){
   var total = document.getElementById("total");
   total.disabled = false;
```

```
    total.checked = totalChecked;
  }

  /**
   * Shows the loading icon.
   */
  function showLoading(id){
    var icon = document.getElementById(id);
    icon.hidden = false;
  }

  /**
   * Hides the loading icon
   */
  function hideLoading(id){
    var icon = document.getElementById(id);
    icon.hidden = true;
  }

   </script>
</html>
```