

DECO3801 Test Plan Document

THEM - Typed HTML5 Evaluation Machine

Carl Hattenfels, Scott Heiner, Shen Yong Lau, Robert Meyer, Brendan Miller, David Uebergang

Contents

| | | |
|----------|--|-----------|
| 1 | Functional Test Plan | 2 |
| 1.1 | Testing Strategy | 2 |
| 1.2 | Implications of Functional Testing | 2 |
| 1.3 | Test Case Transcript | 3 |
| 2 | User Experience Goals | 8 |
| 3 | User Testing Plan | 11 |
| 3.1 | User Testing Strategy | 11 |
| 3.2 | Implications of User Testing | 13 |
| 3.3 | User Test Results | 14 |
| 3.4 | User Test Document | 17 |
| 4 | Appendix A - Python Test Code | 19 |
| 4.1 | Syntax Tests | 19 |
| 4.2 | Page Structure Tests | 23 |
| 4.3 | JSON-RPC Server Tests | 39 |

1 Functional Test Plan

1.1 Testing Strategy

There are three major testable components of the web application: the front-end website, back-end parser and database. While it was easy to write Python test cases for the back-end parser, it was more difficult to test the front-end website and database with a suite of computer-run tests. Instead, a series of scenarios were drafted that would ensure that the web application was running correctly and as expected. Clearly, all of these scenario tests can be “implemented” as they are merely actions performed by us. This means that a test fails when some functionality is not yet implemented, or when fixing one error creates another error.

The test cases that are being run on the parser can be found in Appendix A within this document. This gives an indication of the tests that are currently implemented in the system. More tests are being added periodically, as different types of HTML5 errors are added to the parser. Each HTML5 error will have its own associated test. Other parser related tests are also contained in Appendix A, including the JSON-RPC server tests. These test for concurrency (can the parser handle 5 concurrent requests?) as well as correctness of JSON-RPC input and output.

1.2 Implications of Functional Testing

The functional testing highlighted some issues with all aspects of the application during initial development. There were several sections of the functionality which were unimplemented initially that are now implemented. The interesting implications of the testing showed that the code was implemented correctly. An appropriate fix for the case of empty files (Website and Server Tests, 11) was implemented post-testing. Many of the tests relating to errors were unimplemented initially, but all are present in this final solution. The test suite that is now being used for this program is complete and thorough.

1.3 Test Case Transcript

Python Parser Tests

| Test Number | Test Description | Inputs | Expected Output / Resulting Action | Pass / Fail + How to Fix |
|-------------|--|---|---|--------------------------|
| 1 | Testing that a specific error is being reported correctly, given a particular fragment of HTML as the input. Since not all of the error checks have been or can be defined in advance, the implementation of this test case is reactionary and will be continually updated to include new error checks as they are added. The existing tests will have to be run each time a new error check is implemented to ensure that all existing functionality still works as intended. | A tailored fragment of HTML that should cause a specific error to be reported. eg. “<head> <head> </head> </head>” => Testing for the duplicate set of head tags. | Confirmation that the expected error and associated error code is being returned for the given HTML fragment, in the expected character position of the input fragment. | Pass |
| 2 | Test that the JSON-RPC server is running and can respond to a remote function call. | A single client making a function call to the server. | The function call should be processed without an error being raised, implying the server is currently running. | Pass |
| 3 | Testing that the JSON-RPC server is able to handle up to a maximum of 5 concurrent remote function calls. | Five concurrent function calls are made to the server. | The test case records the time that each response is received by each of the client instances. The function being called has an internal sleep delay of 2 seconds, so the recorded time for each client should be slightly over 2 seconds, implying all 5 calls were made and processed at the same time. | Pass |
| 4 | Testing that a 6th concurrent connection (1 connection over the maximum of 5 concurrent connections) to the JSON-RPC server results in a delayed response. | Six concurrent function calls are made to the server. | As above, the time the response is received is compared to the time the call was made. The first 5 connections should behave as above, receiving a response in just over 2 seconds. The 6th call will receive a response after 4 seconds, 2 seconds after the server is able to respond after the first 5 connections have been responded to. | Pass |
| 5 | Testing the parser’s response to a case where input of an empty string is supplied. | An empty string. | The parser should respond with a general error stating that the input is empty, preventing other general errors such as missing closing HTML tags or page structure sections (head, body, footer). | Pass |

| | | | | |
|----|--|--|--|------|
| 6 | Testing the parser's response when the input string doesn't contain any valid HTML. | A garbage string which doesn't contain any HTML tags or tag like elements eg. jblah; | The parser should respond with a general error stating that the input doesn't contain any valid HTML. | Pass |
| 7 | Testing that a correctly formed JSON-RPC 2.0 request is handled by the server, which should respond with the correct response. | A JSON-RPC 2.0 request containing a small HTML code fragment to be parsed. | A JSON-RPC 2.0 response containing an array of errors related to the given request. The response should also contain the same ID value as the one passed to it with the request. | Pass |
| 8 | Testing that a malformed JSON-RPC 2.0 request containing incorrect parameters for a particular function call causes the server to return an error. | A JSON-RPC 2.0 request containing an invalid parameters array for the function call <code>parse.html</code> . | A JSON-RPC 2.0 error response with a message of "Invalid parameters". | Pass |
| 9 | Testing that a JSON-RPC 2.0 request calling a function that isn't registered on the server causes the server to send an error response. | A JSON-RPC 2.0 request containing a function name that hasn't been registered on the server. | A JSON-RPC 2.0 error response with a message of "". | Pass |
| 10 | Testing the parser response when a tag with a URL attribute is supplied with an valid relative file path. | Html fragment: <code></code> . File list: <code>['image.jpg', 'directory/', 'directory/current.html', 'directory/directory2/image2.jpg']</code> . Current file: <code>'directory/current.html'</code> | The parser response should NOT contain an error indicating an invalid file path. | Pass |
| 11 | Testing the parser response when a tag with a URL attribute is supplied with an non-existent relative file path. | Html fragment: <code></code> . File list: <code>['directory/', 'directory/current.html']</code> . Current file: <code>'directory/current.html'</code> | The parser response should contain two errors indicating invalid file paths, associated with the src attributes of the img tags. | Pass |
| 12 | Testing the parser response when a tag with a URL attribute is supplied with an non-existent files in existing relative filepaths. | Html fragment: <code></code> . File list: <code>['directory/', 'directory/current.html', 'directory/directory2/']</code> . Current file: <code>'directory/current.html'</code> | The parser response should contain two errors indicating invalid file paths, associated with the src attributes of the img tags. | Pass |

| | | | | |
|----|--|---|--|------|
| 13 | Testing the parser response when a tag with a URL attribute is supplied with an valid absolute file path. | Html fragment: “”. File list: [“directory/”, “/image.jpg”, “directory/current.html”, “directory/directory2/”, “directory/directory2/image2.jpg”]. Current file: “directory/current.html”. | The parser response should NOT contain an error indicating an invalid file path. | Pass |
| 14 | Testing the parser response when a tag with a URL attribute is supplied with an non-existent absolute file path. | Html fragment: “”. File list: [“directory/”, “directory/current.html”]. Current file: “directory/current.html” | The parser response should contain an error indicating invalid file paths, associated with the src attributes of the img tag. | Pass |
| 15 | Testing the parser response when a tag with a URL attribute is supplied with non-existent files in existing relative file paths. | Html fragment: “”. File list: [“directory/”, “directory/current.html”]. Current file: “directory/current.html”. | The parser response should contain one errors indicating invalid file paths, associated with the src attributes of the img tags. | Pass |

Website and Server Tests

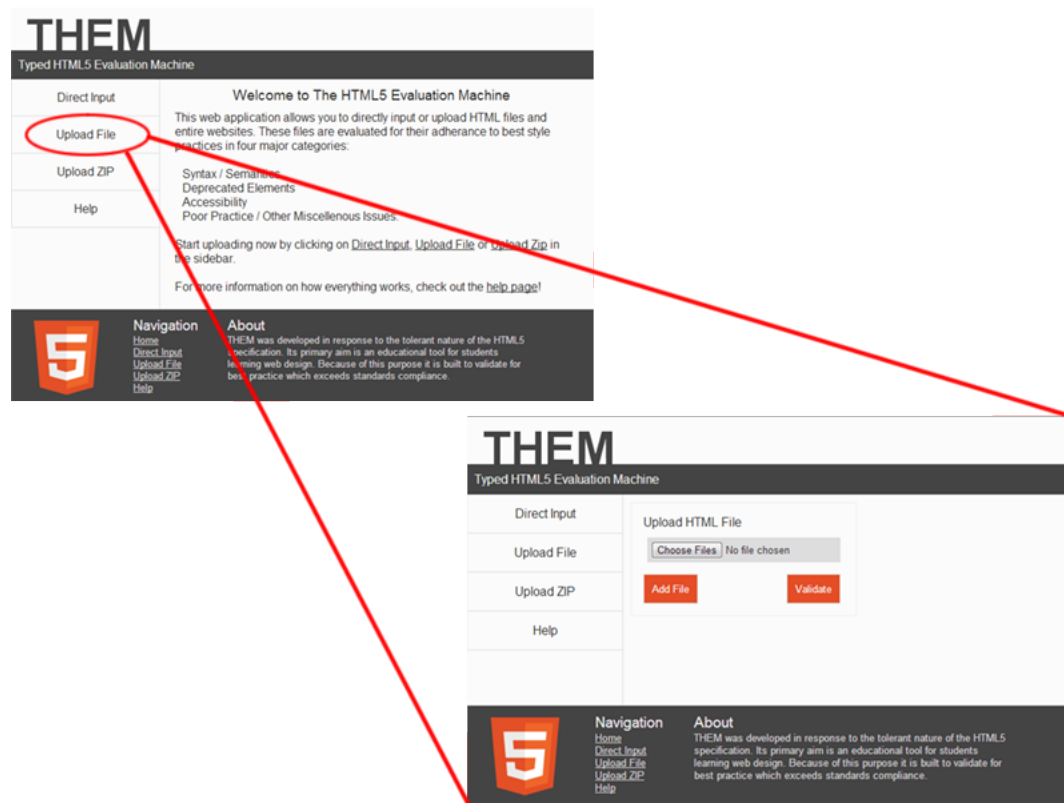
| Test Number | Test Description | Inputs | Expected Output / Resulting Action | Pass / Fail + How to Fix |
|-------------|--|---|--|--------------------------|
| 1 | View Home page | Go to URL, or click link from any page | The home page is displayed. | Pass |
| 2 | View Help page | Click link from any page | The user is sent to the help page. | Pass |
| 3 | View Direct Input page | Click link from any page | The user is sent to the direct input page. | Pass |
| 4 | Validate direct input | The user types their input into the text field on the Direct Input page and clicks Validate. | The input text is saved in a new set with a single file in it. The user is redirected to the Show File page. | Pass |
| 5 | View Upload File(s) page | Click link from any page | The user is sent to the Upload File page. | Pass |
| 6 | Upload single HTML file | The user selects a file and then clicks Validate. | The file is saved in a new set with a single file in it. The user is redirected to the Show File page. | Pass |
| 7 | Upload multiple HTML files individually | The user clicks the Add File button the required number of times, then selects a file for each field. They then click Validate. | Files are saved in a new set, user is redirected to uploaded set page | Pass |
| 8 | Upload multiple HTML files together from one dialogue | The user selects multiple files in the dialogue box, then clicks Validate. | Files are saved in a new set, user is redirected to uploaded set page | Pass |
| 9 | Upload multiple HTML files, some individually and some from one dialogue box | The user performs a combination of multiple Add Files and selecting multiple files in the dialogue boxes. They then click validate. | Files are saved in a new set, user is redirected to uploaded set page | Pass |
| 10 | Upload non-HTML file | The user attempts to upload a file which is not HTML. | The user is redirected to the same page and shown a information box informing them that the file chosen is not a HTML file. | Pass |
| 11 | No file selected on upload | The user attempts to upload a file when no file is selected. | Redirect to upload file page with a helpful error message | Pass |
| 12 | View Upload Zip page | Click link from any page | The user is sent to the Upload Zip page. | Pass |
| 13 | Upload zip file | Zip file selected on previous page, user clicked validate | Zip archive is unpacked, files are saved in a new set, user is redirected to uploaded set page | Pass |
| 14 | View Uploaded Set page | User either uploads multiple files, or uploads a zip archive | The user is shown the list of files uploaded in this set, with corresponding error bars, except in the case of a single file in a set or zip, in which case the user is redirected directly to the show file page. | Pass |

| | | | | |
|----|---|---|--|------|
| 15 | View Uploaded File page | User either selects a file on the Uploaded Set page, or uploads a single file, or validates by direct input | The user is shown their uploaded file, with corresponding error bar, general error information, and uploaded text with error highlighting. | Pass |
| 16 | Remove file after certain period of being untouched in the server | A file should be removed from the server after a period of inactivity. | The files are removed from the database after a time. (3 hours) | Pass |

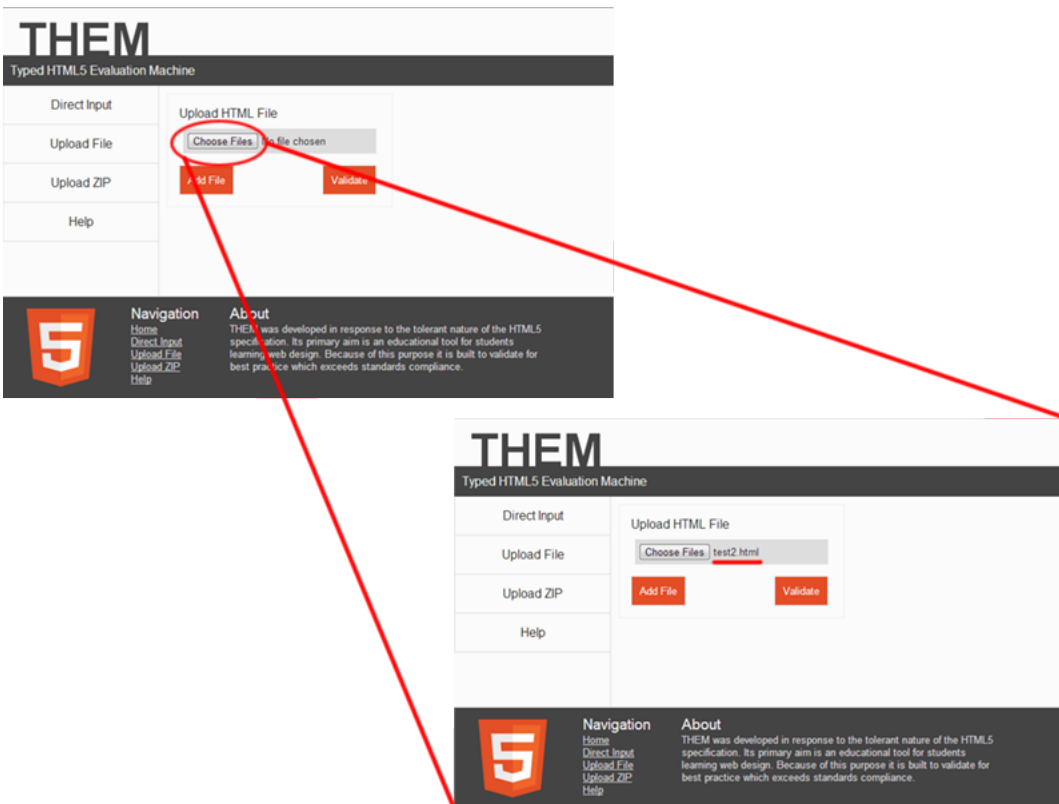
2 User Experience Goals

A clear user experience was in mind while developing this website. Through its ease of use and minimal effort on the part of the user, the application aims to create a very surgical, ambient, and passive experience. The tool should give users almost immediate insight into the issues with their HTML and websites. This is where the user's experience with the tool ends, for this session. The user now can go and fix their file externally, return to the program and almost instantly receive another assessment of their code's validity. There is no aim to get the user invested in the system, and be held on the website for long periods at a time. However, creating a reliable and worthwhile experience is a large focus of this project, brief as this experience with the user is. The user should not be frustrated by the errors the program reveals, with the focus instead on **helping** the user learn and develop better web practices. It is meant to be a program that a user just “touches”, that is, they upload their file they want to check, and then go back and fix it, and then come back to this to validate again, in a cyclic process.

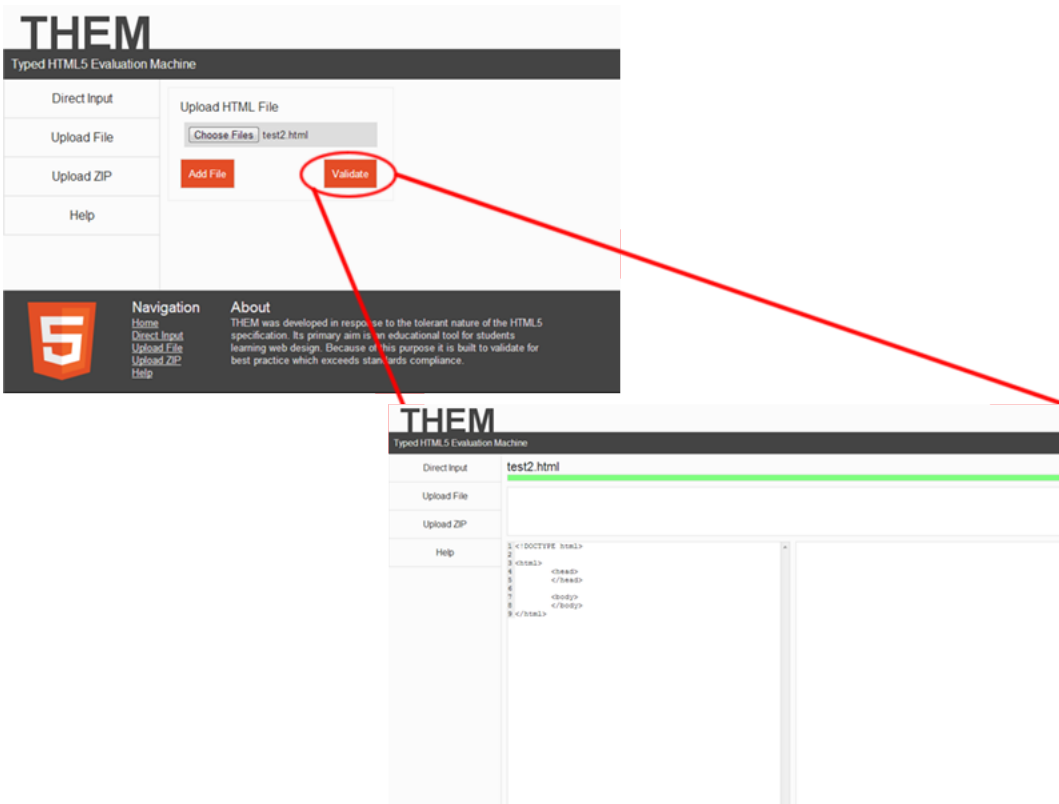
Our priorities are on quick and easy use, which is why all web pages are instantly accessible from all other web pages and users only need a few clicks to navigate. The website has been designed to require as few clicks as possible to access the primary functionality of the system. For example, the following images represent an average user's attempt to verify a file, after brief knowledge of the system's workings.



The first click takes the user to the webpage.



The second click chooses a file to verify. Other clicks may be employed here as the user navigates their file system to find the file they want to upload.



The third click validates the selected file. In this way, the user quickly and easily travels from the opening page to viewing their validated HTML. All throughout the web application, the aim has been to create similar experiences where few clicks are required by the user, and they reach their end goal in minimal time.

Users benefit greatly from this experience. The surgical nature means that they develop a relationship with the website where it is used as an intermediate and reliable tool, much like their text editor or their browser. It is easily inserted as part of their workflow. It is hoped that by using this application, users will experience a great rush of joy when their program finally passes, and that long error bar on the web page turns completely green. Validation, after all, is its own reward.

3 User Testing Plan

3.1 User Testing Strategy

The web application will eventually be utilised by two user groups - students of DECO1400, and students of DECO7140. As such, four major user testing groups were pinpointed:

- Undergraduate students who have already completed DECO1400
- Undergraduate students who have not completed DECO1400 but have worked with computers
- Masters students who have already completed DECO7140; and
- Masters students who have not already completed DECO7140.

However, poor initial consideration due to this highly targeted user base caused us to primarily focus on students who had not done DECO1400, as these users represented students “new” to DECO1400. Focusing also on past DECO1400 students would have allowed us to understand the needs of users who had previously completed the course, and could determine whether the tool would have been worthwhile to them. Poor communication on the part led to us getting very few in this category. Ultimately, information was gathered from nine users - four were undergraduates who had not done DECO1400, one was an undergraduate who had done DECO1400 and four were masters students who had done DECO7140.

Six key scenarios were formulated for the users to undertake. Each scenario was performed on the live prototype at <http://underwaterfall.com>. Storyboards for each of them can be found in the separately submitted Appendix B.

- Getting started by reading the help page (**First Encounter Scenario**)
 - Actor: New User
 - Goal: To understand how the website works, and understand the feedback it provides.
 - Necessity of Scenario: This scenario is required for first time users to understand how to use and interact with the program.
 - Preconditions: User has not previously visited the webpage.
- Validating HTML via Direct Input
 - Actor: User
 - Goal: To check the validity of a piece of copied or typed HTML.
 - Necessity of Scenario: This scenario represents one of the key ways users can get insight into how to program using HTML5.
 - Preconditions: User has a clear understanding of the validation the website provides from the help page.
- Validating HTML via uploading a file
 - Actor: User
 - Goal: To check the validity of a HTML file.
 - Necessity of Scenario: This scenario represents one of the key ways users can get insight into how to program using HTML5.
 - Preconditions: User has a clear understanding of the validation the website provides from the help page.

- Validating websites or multiple HTML files via uploading a zip

| | |
|------------------------|---|
| Actor: | User |
| Goal: | To check the validity of a zip file of either website files or HTML. |
| Necessity of Scenario: | This scenario represents one of the key ways users can get insight into how to program using HTML5. |
| Preconditions: | User has a clear understanding of the validation the website provides from the help page. |

- Fixing a file based on the error suggestions, resubmitting and getting a valid file

| | |
|------------------------|--|
| Actor: | User |
| Goal: | To check the validity of a piece of copied or typed HTML. |
| Necessity of Scenario: | This scenario is the primary point of the application - users learning to correct their HTML5 pages. |
| Preconditions: | User has already uploaded a file and determined the errors relating to their webpage. |

- Attempting to upload a non-HTML file (**Fringe Case Scenario**)

| | |
|------------------------|---|
| Actor: | User |
| Goal: | To check the validity of a non-HTML file. |
| Necessity of Scenario: | Users are fallible and can upload incorrect files. They may also believe the website is capable of evaluating other types of files, like Javascript or CSS. |
| Preconditions: | N/A |

There was a focus in testing of two metrics: time taken to complete each scenario, and, in keeping with the surgical user experience, number of clicks required to complete each scenario. As there was a need to create an enjoyable environment for the users, any particular emotions and reactions of the users as they undertook the scenarios was also noted. the primary strategy for user testing was as follows:

1. Prepare / lay out materials for the participant so that everything is ready.
2. Introduce ourselves to the participant and give them a high-level idea of what they will be doing in their tasks today.
3. Ask participant to fill in and sign consent form. The test conductor will fill in their parts too.
4. Give the participant more detailed instructions about the task they are to do (i.e. access the website, upload file and validate). Ask them to think out loud or to make comments as they work. See if there are any questions from the participants before starting, and answer these where appropriate.
5. When participant is ready, ask the participant to start on the task. Start the timer. Be prepared to count the number of clicks they required to complete the task. Take hand notes as the participant works, according to the arrangements you have worked out amongst the non-participant group. If the participant goes a bit quiet, ask, “what are you thinking now?” or “what are you working on now?”
6. When they complete the first scenario, move them onto the next one, and so on.
7. After completing all six scenarios, ask the participant to fill in the questionnaire. Clarify as necessary.

8. When the participant has finished filling in the questionnaire, check over the responses to make sure that all parts have been filled out, and that the answers are legible.
9. Tell the participant that the session is at an end. Thank the participant for their time.

The results of testing are below, after the “Implications of User Testing” section, as well as the Questionnaire used for testing.

3.2 Implications of User Testing

In general, users had no trouble navigating the system. The average rating for how easy the system was to navigate was 4.78 / 5 on a Likert scale. The user experience goal of the small number of clicks required for each operation was met. No user (from those who had click data registered) needed to click more than ten times. However, the system requires improvement in several key areas. On recommendation from the users, here is a key list of changes that were implemented in the completion of this project:

- Invite the users to click on the error tags. Users often did not realise that they could click on the highlighted text to find out more about the error. Users are now provided with information on the help page about this, as well as told on the file page to “Click on highlighted text for more information.”
- Errors with non-HTML files. As part of the user testing, users were asked to upload a non-HTML file, a common action which could be performed by a user. Many were surprised that the file was actually parsed. The mimetype of the uploaded file is now checked, and the parser is not passed non-HTML code, except in the case of code which is plain-text and could possibly be HTML code that lacks structure. The user is presented with an error on the Upload File screen when they attempt to upload a non-HTML file. The good news is that for the most part, the website did not break, except when a user attempted to upload a Powerpoint file. This is prevented by the previously mentioned fix.
- Add multiple files via Add File button. Users noted that an attempt to add a second file via the Add File button after selecting the first file caused their previously selected file to be forgotten. This needs to be fixed. One user had trouble due to the similarity of the Validate and Add File buttons, so these now differ in colour to highlight the difference functions.
- Blank file uploading. Users choosing no file in a file field of the form, along with some file fields being filled, were parsed as if they were files with no name. As such, they were sent to a set page showing bars with blank file names and error bars. These files are no longer parsed, and if a user clicks Validate with no file selected, they are redirected back to the upload file page and asked to upload a file.

There was no plan to redefine any test plans, but for future testing, working closer together to complete it would be a priority. The differences between the methods of testing used by members of the group was obvious, and it lead to less data than expected. All in all, the web application was well received, with many users who had experience with HTML5 stating they would find the tool beneficial to their studies. It received a rating of 4.6 / 5 on the Likert scale for “How likely would you use this tool to assist you in your study?” among those students who had previous experience with HTML5. The Typed HTML5 Evaluation tool is a fantastic tool for users to evaluate their HTML5.

3.3 User Test Results

Metric Results

| | Get Informed | Direct Input | Upload a File | Fixing a File | Upload Zip | Upload Non-HTML File |
|---------------------|--------------|--|--|--|---------------------|--|
| Tester 1 - DECO1400 | | | | | | |
| Time taken (secs) | | 49.03 | 39.9 | 32.7 | 13.3 | 6.03 |
| Clicks | | | | | | |
| Reaction | | Confused about the highlighting word, unsure it is clickable | Feel confused when add file buton cancel previous upload entry | Learn the error quickly, getting used to the system presentation | Feeling comfortable | Feeling surprised when the result is as expected |
| Tester 2 - DECO7140 | | | | | | |
| Time taken (secs) | | 56 | 22.3 | 27 | 11.2 | 6.5 |
| Clicks | | | | | | |
| Reaction | | Unsure about what the error bar representing | Frustrated as keep mispressing add file instead of validate due to same colour button | Learn the error quickly | Feeling comfortable | Feeling surprised when the result is as expected |
| Tester 3 - DECO7140 | | | | | | |
| Time taken (secs) | | 43.5 | 19.2 | 23.3 | 9.9 | 6.9 |
| Clicks | | | | | | |
| Reaction | | Feeling unimpressed as the error is not validated correctly | Feeling satisfied with the simple way to upload file | Learn the error quickly and fixed it | Feeling comfortable | Feeling surprised when the result is as expected |
| Tester 4 - DECO7140 | | | | | | |
| Time taken (secs) | | 51.2 | 23.1 | 31.78 | 11.67 | 8.3 |
| Clicks | | | | | | |
| Reaction | | Feeling that the presentation of errors is good | Frustrating when trying to upload multiple file, the add file button cancel previous entry | Learn the error quickly, feeling good | Feeling comfortable | Feeling surprised when the result is as expected |
| Tester 5 - DECO7140 | | | | | | |
| Time taken (secs) | | 48.12 | 17 | 20.1 | 10.8 | 5.87 |
| Clicks | | | | | | |
| Reaction | | Unsure about the highlighting words are clickable | Feeling good as it is easy to upload single file | Learn the error quickly | Feeling comfortable | Feeling surprised when the result is as expected |

| | | | | | | |
|--------------------------|---|--|-------------------------------|--|----|--|
| Tester 6 - Undergraduate | | | | | | |
| Time taken (secs) | 7 | 11 | 24.8 | 10 | 8 | N/A |
| Clicks: | 1 | 2 | 6 | 5 | 4 | 5 |
| Reaction | | Didn't realise you could click on errors | | Multiple files added but no files given - still shows the bars | | Powerpoint file uploaded - "max_allowed_packets" error given |
| Tester 7 - Undergraduate | | | | | | |
| Time taken (secs) | 2 | 45 | 29 | 20 | 18 | 20 |
| Clicks: | 1 | 5 | 5 | 3 | 5 | 5 |
| Reaction | | | | | | Note, inf file still was parsed. |
| Tester 8 - Undergraduate | | | | | | |
| Time taken (secs) | 2 | 41 | 14.8 | 63 | 22 | 13 |
| Clicks: | 1 | 3 | 6 | 8 | 7 | 5 |
| Reaction | | | | Backtracked to copy files, didn't get to put in entire tag | | |
| Tester 9 - Undergraduate | | | | | | |
| Time taken (secs) | 2 | 60 | 28 | 211 | 9 | 15 |
| Clicks: | 1 | 5 | 6 | 9 | 4 | 3 |
| Reaction | | | Clicked add file accidentally | Not intuitive to click highlighted text | | |

Questionnaire

| Tester | Q1 | Q2 | Q3 | Q4 | Q5 | Q6 | Q7 | Q8 | Q9 | Q10 |
|----------|--|---|----|----|----|----|----|-----|-----|---|
| Tester 1 | Yes (DECO1400, Programming) | Yes (Java, Javascript, PHP, Python) | 4 | 5 | 4 | 3 | 5 | No | Yes | If you upload a non-zip file to the upload zip no error message is shown. Selecting file then clicking add file clears previous additions. Upload invalid file to upload file (specifically provided zip) shows the file. Adding “<!DOCTYPE html>.” (note full stop) causes errors to appear (raw output). Clicking add file then only uploading 1 file takes you to collections screen instead of the single file screen. Leaving uploads blank causes blank error bars to appear. |
| Tester 2 | Yes (DECO7140) | Yes (Python, Java, Actionscript) | 3 | 2 | 4 | 5 | 4 | No | Yes | |
| Tester 3 | Yes (DECO7140) | No | 4 | 4 | 4 | 4 | 4 | Yes | Yes | |
| Tester 4 | Yes (B InfTech) | Yes (Java, PHP, C#, HTML5, CSS, Javascript, MYSQL, etc) | 5 | 4 | 4 | 3 | 5 | No | No | Confused about the “upload” buttons. Color theme is nice. Single file ->upload file, multiple files ->zip it and upload. Highlight the code in the correction section will be more user friendly (Upload File). General instruction on the UI can be added. Maybe do more market research about existing validation tools. Line number is good to be placed in direct input. |
| Tester 5 | Yes (DECO7140) | Yes (Actionscript, Python) | 3 | 5 | 4 | 5 | 5 | No | No | Simple & clean layout, I would like if I could copy the text and paste the text again to modify it. |
| Tester 6 | No | Yes (Python) | 2 | 4 | 4 | 2 | 5 | Yes | Yes | |
| Tester 7 | No | Yes (Python, Java, Matlab) | 2 | 4 | 4 | 3 | 5 | Yes | No | No way to gauge the effects of the error based on the error message. Didn’t initially realise you can click on highlighted text to see error notes nor which colours associated to errors (thought colour was gauging error intensity.) |
| Tester 8 | No | Yes (Python, Matlab) | 3 | 5 | 4 | 4 | 5 | Yes | No | All good :) |
| Tester 9 | No (though did make website in primary school) | Yes (Python, Matlab) | 1 | 4 | 4 | 5 | 5 | Yes | Yes | It was fun. |

3.4 User Test Document

DECO3801 User Testing Document

Name: _____

Program or Degree: _____

Please wait for your instructions from supervisors before completing these scenarios. Time will be taken between each scenario to write down key information.

Get Informed

Navigate to the help page. Since this is your first time using the software, read up on what the error bars mean here.

Direct Input

Copy some basic HTML text into the Direct Input page, and validate it for errors. This can be from a file you have locally, or you can use the following provided code:

```
<html>
<head></head>
<body></body>
</html>
```

Upload File

Upload a file to the Upload File page, and validate it for errors. If you do not have a file of your own, we can provide you with one.

Fixing a File

Based on the error messages provided, fix your uploaded file and resubmit it.

Upload Zip

Upload a zipped website to the Upload Zip page, and validate it for errors. If you do not have a website of your own, we can provide you with one.

Upload Non-HTML File

Try uploading any file you like. Does the application behave well?

Questionnaire

1. Have you taken DECO1400/DECO7140? If not, do you have any past learning experience in web design (such as HTML4, Javascript, etc).

☐ Yes ☐ No

Past learning experience: _____

2. Do you have any programming background, if so what languages you have been using?

☐ Yes ☐ No

Past learning experience: _____

3. How likely would you use this tool to assist you in your study?

| | | | | | | | | |
|-------------|--|---|--|---|--|---|--|---------------|
| Very likely | | | | | | | | Very unlikely |
| 5 | | 4 | | 3 | | 2 | | 1 |

4. How visually appealing is the website to you?

| | | | | | | | | |
|----------------|--|---|--|---|--|---|--|--------------------|
| Very appealing | | | | | | | | Not very appealing |
| 5 | | 4 | | 3 | | 2 | | 1 |

5. How well did this tool meet your expectations?

| | | | | | | | | |
|-------------------------------|--|---|--|---|--|---|--|------------------------------|
| Met my expectations very well | | | | | | | | Did not meet my expectations |
| 5 | | 4 | | 3 | | 2 | | 1 |

6. Do you feel comfortable with the presentation of the error message(s)?

| | | | | | | | | |
|------------------|--|---|--|---|--|---|--|----------------------|
| Very comfortable | | | | | | | | Not very comfortable |
| 5 | | 4 | | 3 | | 2 | | 1 |

7. How easy did you find the tool to navigate?

| | | | | | | | | |
|-----------|--|---|--|---|--|---|--|----------------|
| Very easy | | | | | | | | Very difficult |
| 5 | | 4 | | 3 | | 2 | | 1 |

8. Is this your first time using HTML5 in developing a website?

☐ Yes ☐ No

9. Would you prefer the option to select multiple files at once (in the explorer window) when uploading?

☐ Yes ☐ No

10. If you have any general feedback/suggestions, feel free to use the space below to tell us.

4 Appendix A - Python Test Code

4.1 Syntax Tests

```
1 from __future__ import absolute_import, division, unicode_literals
2
3 #from . import support
4 import unittest, html5lib
5 from html5lib import treebuilders
6
7 class TestSyntax(unittest.TestCase):
8     """
9     Provides a number of test cases to test the syntax used
10    in the document.
11    """
12
13    def setUp(self):
14        self.parser = html5lib.HTMLParser(tree=treebuilders.
15                                         getTreeBuilder("etree"))
16
17    def test_malformed_tag_name(self):
18        """
19        Test that the tag name isn't an invalid symbol.
20
21        Input:
22        A HTML fragment containing a tag with an invalid tag name.
23
24        Expected Results:
25        An error should be thrown reporting an invalid tag name.
26        """
27
28        inputFragmentEmptyName = "<html>< ></body>"
29        inputFragmentQuestionMark = "<html><?></body>"
30        inputFragmentRightBracket = "<html><></html>"
31
32        self.parser.parse(inputFragmentEmptyName)
33
34        self.assertIn(((6, 6), u'expected-tag-name', {u'data': u' '}),
35                      self.parser.errors, "Failed to report invalid tag name. Get ")
36
37        self.parser.reset()
38        self.parser.parse(inputFragmentQuestionMark)
39
40        self.assertIn(((6, 6), u'expected-tag-name-but-got-question-mark',
41                      {}),
42                      self.parser.errors, "Failed to report valid tag name. Got
43                      question mark instead.")
44
45        self.parser.reset()
46        self.parser.parse(inputFragmentRightBracket)
47
48        self.assertIn(((6, 7), u'expected-tag-name-but-got-right-bracket',
49                      {}),
50                      self.parser.errors, "Failed to report valid tag name. Got
51                      question mark instead.")
```

```

47
48 def test_self_closing_end_tag(self):
49     """
50     Test that a closing tag with a misplaced forwardslash
51     raises an error.
52
53     Input:
54     A HTML fragment containing a closing tag with a misplaced
55         forwardslash.
56
57     Expected Results:
58     An error should be thrown reporting an invalid tag name.
59     """
60     inputFragment = "<html><a></a /></html>"
61
62     self.parser.parse(inputFragment)
63
64     self.assertIn(((9, 14), u'self-closing-flag-on-end-tag', {}),
65                   self.parser.errors, "Failed to report misplaced forwardslash
66                       in closing tag.")
67
68 def test_invalid_self_closing_tag(self):
69     """
70     Test that the use of a self closing tag for a tag
71     which isn't considered a self closing tag returns
72     an error.
73
74     Input:
75     A HTML fragment containing a start tag with a trailing
76         forwardslash
77     (self-closing) for a tag type which isn't a self closing tag.
78
79     Expected Results:
80     An error should be thrown reporting the given tag type isn't a
81         self-closing
82         tag.
83     """
84     inputFragment = "<html><a /></html>"
85
86     self.parser.parse(inputFragment)
87
88     self.assertIn(((6, 10), u'non-void-element-with-trailing-solidus',
89                     {u'name': u'a'}),
90                   self.parser.errors, "Failed to report invalid self-closing
91                       tag.")
92
93 def test_attributes_in_end_tag(self):
94     """
95     Test that attributes occurring in a closing tag are
96     reported as an error.
97
98     Input:
99     A HTML fragment containing a closing tag which contains
100         at least one attribute.

```

```

98     Expected Results:
99     An error should be thrown reporting that the closing tag shouldn'
100         t contain
101     attributes.
102     """
103     inputFragment = '<html><a></a src="blah"></html>'
104
105     self.parser.parse(inputFragment)
106
107     self.assertIn((9, 23), u'attributes-in-end-tag', {}),
108         self.parser.errors, "Failed to report attributes in closing
109             tag.")
110
111     def test_duplicate_h1_tags(self):
112         """
113         Test that any duplicate h1 tags are reported as errors.
114
115         Input:
116         A HTML fragment containing more than one set of h1 tags.
117
118         Expected Results:
119         An error should be thrown reporting that duplicate h1 tags were
120             found in the document.
121         """
122         inputFragment = """
123 <!DOCTYPE html>
124 <html>
125 <head>
126 </head>
127 <body>
128 <h1></h1>
129 <h1></h1>
130 <footer>
131 </footer>
132 </body>
133 </html>
134 """
135
136     self.parser.parse(inputFragment)
137
138     self.assertIn(((56, 59), u'duplicate-h1-element', {u'name': u'h1'
139         })),
140         self.parser.errors, "Failed to report duplicate h1 tags.")
141
142     def test_heading_order(self):
143         """
144         Test that heading elements maintain correct order within the
145             document. The
146             order follows from h1-h6 and they must appear in that order
147             during the document.
148
149         Input:
150         A HTML fragment containing a set of h2 tags appearing before a
151             set of h1 tags.

```

```

148         Expected Results:
149         An error should be thrown reporting that the h2 tags are out of
            order.
150         """
151
152         inputFragment = """
153 <!DOCTYPE html>
154 <html>
155 <head>
156 </head>
157 <body>
158 <h2></h2>
159 <h1></h1>
160 <footer>
161 </footer>
162 </body>
163 </html>
164         """
165
166         self.parser.parse(inputFragment)
167
168         self.assertIn(((46, 49), u'invalid-heading-order', {u'name': u'h2
            ', u'missing': u'h1'})),
169             self.parser.errors, "Failed to report heading tags out of
                order.")
170
171 if __name__ == '__main__':
172     unittest.main()

```

4.2 Page Structure Tests

```
1 from __future__ import absolute_import, division, unicode_literals
2
3 #from . import support
4 import unittest, html5lib
5 from html5lib import treebuilders
6
7 class TestPageStructure(unittest.TestCase):
8     """
9     Provides a number of test cases related to basic page structure
10    for html5 documents.
11    """
12
13    def setUp(self):
14        self.parser = html5lib.HTMLParser(tree=treebuilders.
15                                         getTreeBuilder("etree"))
16
17    def test_singular_tags(self):
18        """
19        Test that the multiple-instance-singular-tag error is thrown
20        for cases where more than one instance of a singular tag block is
21        present.
22
23        Input:
24        Nested blocks of singular tags (html, body, head).
25        eg. <html><html></html></html>
26
27        Output:
28        All three test cases should report a multiple instance of both
29        the start and closing tags for each of the three singular tags.
30        """
31        multipleHTMLInstances = "<html><html></html></html>"
32        multipleHeadInstances = "<html><head><head></head></head><body></body></html>"
33        multipleBodyInstances = "<html><head></head><body><body></body></body></html>"
34
35        self.parser.parse(multipleHTMLInstances)
36
37        self.assertIn(((6, 11), u'multiple-instance-singular-tag', {u'
38            name': u'html'}),
39            self.parser.errors, "Multiple instances of starting HTML tag
40            not reported.")
41
42        self.assertIn(((12, 18), u'incorrect-placement-html-end-tag', {u'
43            name': u'html'}),
44            self.parser.errors, "Multiple instances of closing HTML tag
45            not reported.")
46
47        self.parser.reset()
48        self.parser.parse(multipleHeadInstances)
49
50        self.assertIn(((12, 17), u'multiple-instance-singular-tag', {u'
51            name': u'head'}),
52            self.parser.errors, "Multiple instances of starting HTML tag
53            not reported.")
```

```

47
48     self.assertIn(((25, 31), u'incorrect-placement-singular-end-tag',
49         {u'name': u'head'})),
50         self.parser.errors, "Multiple instances of closing head tag
51             not reported.")
52
53     self.parser.reset()
54     self.parser.parse(multipleBodyInstances)
55
56     self.assertIn(((25, 30), u'multiple-instance-singular-tag', {u'
57         name': u'body'})),
58         self.parser.errors, "Multiple instances of starting HTML tag
59             not reported.")
60
61     self.assertIn(((38, 44), u'unexpected-end-tag-after-body', {u'
62         name': u'body'})),
63         self.parser.errors, "Multiple instances of closing body tag
64             not reported.")
65
66 def test_missing_doctype(self):
67     """
68     Test that the expected-doctype-but-got-start-tag error is thrown
69     for cases where no DOCTYPE is declared.
70
71     Input:
72     Nested blocks of singular tags (html, body, head), all of which
73     are missing the DOCTYPE declaration.
74     eg. <html><html></html></html>
75
76     Expected Results:
77     All test cases should report a missing DOCTYPE declaration.
78     """
79     startTagBeforeDoctype = "<html><html></html></html>"
80     endTagBeforeDoctype = "</head></head>"
81     eofBeforeDoctype = ""
82
83     self.parser.parse(startTagBeforeDoctype)
84
85     self.assertIn(((0, 5), u'expected-doctype-but-got-start-tag', {u'
86         name': u'html'})),
87         self.parser.errors, "Failed to report missing DOCTYPE
88             declaration (start tag before doctype.")
89
90     self.parser.reset()
91     self.parser.parse(endTagBeforeDoctype)
92
93     self.assertIn(((0, 6), u'expected-doctype-but-got-end-tag', {u'
94         name': u'head'})),
95         self.parser.errors, "Failed to report missing DOCTYPE
96             declaration (closing tag before doctype.")
97
98     self.parser.reset()
99     self.parser.parse(eofBeforeDoctype)
100
101     self.assertIn((-1, -1), u'expected-doctype-but-got-eof', {}),
102         self.parser.errors, "Failed to report missing DOCTYPE
103             declaration (EOF before doctype.)"

```



```

93
94
95     def test_closing_html(self):
96         """
97         Test that a missing HTML closing tag is reported when none
98         are present in the document.
99
100        Input:
101        Nested blocks of singular tags (head, body).
102
103        Expected Results:
104        Report whether the the closing HTML tag is present.
105        """
106        multipleHeadInstances = "<head><head></head></head>"
107        multipleBodyInstances = "<body><body></body></body>"
108
109        self.parser.parse(multipleHeadInstances);
110
111        self.assertIn((-1, -1), u'no-closing-html-tag', {}),
112            self.parser.errors, "Failed to report missing closing HTML
113                                tag.")
114
115        self.parser.reset()
116        self.parser.parse(multipleBodyInstances)
117
118        self.assertIn((-1, -1), u'no-closing-html-tag', {}),
119            self.parser.errors, "Failed to report missing closing HTML
120                                tag.")
121
122     def test_misplaced_tags_before_head(self):
123         """
124         Test that both start and closing tags occurring before the head
125         section are reported as being misplaced.
126
127        Input:
128        A number of instances of start and closing tags being placed
129        before
130        the head section.
131
132        Expected Results:
133        Report whether or not the tags preceding the head section are
134        reported
135        as being misplaced.
136        """
137        misplacedHeadTags = "<html><body></body><head></head></html>"
138        misplacedLinkTags = "<html><a></a><head></head><body></body></
139                                html>"
140
141        self.parser.parse(misplacedHeadTags)
142
143        self.assertIn((6, 11), u'incorrect-start-tag-placement-before-
144                                head', {u'name': u'body'}),
145            self.parser.errors, "Failed to report start body tag before
146                                head section.")
147
148        self.assertIn((12, 18), u'incorrect-end-tag-placement-before-
149                                head', {u'name': u'body'}),

```

```

142         self.parser.errors, "Failed to report closing body tag before
           head section.")
143
144     self.parser.reset()
145     self.parser.parse(misplacedLinkTags)
146
147     self.assertIn(((6, 8), u'incorrect-start-tag-placement-before-
           head', {u'name': u'a'})),
148         self.parser.errors, "Failed to report start link (a) tag
           before head section.")
149
150     self.assertIn(((9, 12), u'incorrect-end-tag-placement-before-head
           ', {u'name': u'a'})),
151         self.parser.errors, "Failed to report closing link (a) tag
           before head section.")
152
153     def test_incorrect_tags_in_head(self):
154         """
155         Test that tags which don't belong in the head section
156         are reported as misplaced using the 'incorrect-start-tag-
           placement-in-head'
157         and 'incorrect-end-tag-placement-in-head' errors.
158
159         Input:
160         A HTML fragment with a pair of head tags enclosing a tag
161         pair which doesn't belong in the head phase.
162
163         Expected Results:
164         Inclusion of the 'incorrect-start-tag-placement-in-head'
165         and 'incorrect-end-tag-placement-in-head' errors being reported
166         as part of the returned array of error codes.
167         """
168         inputFragment = "<html><head><a></a></head></html>"
169
170         self.parser.parse(inputFragment)
171
172         self.assertIn(((12, 14), u'incorrect-start-tag-placement-in-head'
           , {u'name': u'a'})),
173             self.parser.errors, "Failed to report starting tag which
           doesn't belong in the head section.")
174
175     def test_tags_after_eof(self):
176         """
177         Tests that starting and closing tags occurring after the last
178         instance of a closing HTML tag are reported as an error.
179
180         Input:
181         A HTML fragment with a start and closing tag pair occurring
182         after the start and closing HTML pair.
183
184         Expected Results:
185         An error being thrown for both the start and closing tags
           occurring
186         after the HTML tags.
187         """
188
189         inputFragment = "<html></html><a></a>"

```

```

190     self.parser.parse(inputFragment)
191
192     self.assertIn(((13, 15), u'expected-eof-but-got-start-tag', {u'
193         name': u'a'})),
194         self.parser.errors, "Failed to report start tag after closing
195             HTML tag.")
196
197     self.assertIn(((16, 19), u'expected-eof-but-got-end-tag', {u'name
198         ': u'a'})),
199         self.parser.errors, "Failed to report closing tag after
200             closing HTML tag.")
201
202     def test_missing_start_tag(self):
203         """
204         Tests that a missing start tag is reported in the case
205         that a closing tag is found without a matching start tag.
206
207         Input:
208         A HTML fragment containing a closing tag without a matching
209         start tag.
210
211         Expected Results:
212         An error being thrown reporting that the matching start tag
213         is missing.
214         """
215         inputFragment = "<html><head></head><body></a></body></html>"
216
217         self.parser.parse(inputFragment)
218
219         self.assertIn(((25, 28), u'unexpected-end-tag', {u'name': u'a'}),
220             self.parser.errors, "Failed to report the lack of a matching
221                 start tag.")
222
223     def test_misplaced_tags_after_body(self):
224         """
225         Tests that any tags occurring after the body phase
226         are reported as being incorrectly placed.
227
228         Input:
229         A HTML fragment with a pair of start and closing tags placed
230         after the closing body tag.
231
232         Expected Results:
233         An error should be thrown for both the start and closing
234         tags found after the closing body tag.
235         """
236         inputFragment = "<html><head></head><body></body><a></a></html>"
237
238         self.parser.parse(inputFragment)
239
240         self.assertIn(((32, 34), u'unexpected-start-tag-after-body', {u'
241             name': u'a'})),
242             self.parser.errors, "Failed to report misplaced starting tag
243                 found after the closing body tag.")

```

```

240
241     self.assertIn(((35, 38), u'unexpected-end-tag-after-body', {u'
242         name': u'a'})),
243         self.parser.errors, "Failed to report misplaced closing tag
244             found after the closing body tag.")
245
246 def test_missing_closing_html_tag(self):
247     """
248     Test that a missing closing HTML tag is reported.
249
250     Input:
251     A HTML fragment missing a closing HTML tag.
252
253     Expected Results:
254     An error should be thrown stating that the closing HTML tag is
255     missing.
256     """
257     inputFragment = "<html><head></head><body></body>"
258
259     self.parser.parse(inputFragment)
260
261     self.assertIn((-1, -1), u'no-closing-html-tag', {}),
262     self.parser.errors, "Failed to report missing closing HTML
263         tag.")
264
265 def test_early_termination_before_head(self):
266     """
267     Test that an early closing HTML tag before the head phase
268     is reported as an error.
269
270     Input:
271     A HTML fragment with the head and body sections placed after
272     a closed set of HTML tags.
273
274     Expected Results:
275     An error should be thrown stating that the closing HTML tag
276     has been found before the head phase.
277     """
278     inputFragment = "<html></html><head></head><body></body>"
279
280     self.parser.parse(inputFragment)
281
282     self.assertIn(((6, 12), u'early-termination-before-head', {u'name
283         ': u'html'})),
284     self.parser.errors, "Failed to report early termination
285         before head section.")
286
287 def test_early_termination_in_head(self):
288     """
289     Test that an early closing HTML tag in the head phase
290     is reported as an error.
291
292     Input:
293     A HTML fragment with the closing HTML tag placed within
294     the set of head tags.

```

```

291
292     Expected Results:
293     An error should be thrown stating that the closing HTML tag
294     has been found in the head phase.
295     """
296
297     inputFragment = "<html><head></html></head><body></body>"
298
299     self.parser.parse(inputFragment)
300
301     self.assertIn(((12, 18), u'early-termination-in-head', {u'name':
302         u'html'})),
303         self.parser.errors, "Failed to report early termination
304         before head section.")
305
306 def test_early_termination_before_body(self):
307     """
308     Test that an early closing HTML tag before the body phase
309     is reported as an error.
310
311     Input:
312     A HTML fragment with the closing HTML tag placed before the body
313     section.
314
315     Expected Results:
316     An error should be thrown stating that the closing HTML tag
317     has been found before the body phase.
318     """
319
320     inputFragment = "<html><head></head></html><body></body>"
321
322     self.parser.parse(inputFragment)
323
324     self.assertIn(((19, 25), u'early-termination-before-body', {u'
325         name': u'html'})),
326         self.parser.errors, "Failed to report early termination
327         before head section.")
328
329 def test_early_termination_in_body(self):
330     """
331     Test that an early closing HTML tag in the body phase
332     is reported as an error.
333
334     Input:
335     A HTML fragment with the closing HTML tag placed within
336     the set of body tags.
337
338     Expected Results:
339     An error should be thrown stating that the closing HTML tag
340     has been found in the head phase.
341     """
342
343     inputFragment = "<html><head></head><body></html></body>"
344
345     self.parser.parse(inputFragment)

```

```

343         self.assertIn(((25, 31), u'early-termination-in-body', {u'name':
344             u'html'})),
345             self.parser.errors, "Failed to report early termination
346                 before head section.")
347
348     def test_tags_between_head_body(self):
349         """
350         Test that a set of tags placed after the head section
351         but before the body section is reported as an error.
352
353         Input:
354         A HTML fragment with a set of tags between the head
355         and body sections.
356
357         Expected Results:
358         An error should be thrown stating that the set of tags
359         can't be placed between the head and body sections.
360         """
361         inputFragment = "<html><head></head><a></a><body></body></html>"
362         self.parser.parse(inputFragment)
363
364         self.assertIn(((19, 21), u'start-tag-before-body-after-head', {u'
365             name': u'a'})),
366             self.parser.errors, "Failed to report start tag after head
367                 phase but before body phase.")
368
369         self.assertIn(((22, 25), u'end-tag-before-body-after-head', {u'
370             name': u'a'})),
371             self.parser.errors, "Failed to report closing tag after head
372                 phase but before body phase.")
373
374     def test_missing_starting_html_tag(self):
375         """
376         Test that a missing starting HTML tag is reported as an error.
377
378         Input:
379         A HTML fragment missing a starting HTML tag.
380
381         Expected Results:
382         An error should be thrown indicating that the fragment doesn't
383         contain a starting HTML tag.
384         """
385         inputFragment = "<head></head><body></body></html>"
386         self.parser.parse(inputFragment)
387
388         self.assertIn((-1, -1), u'no-starting-html-tag', {}),
389             self.parser.errors, "Failed to report missing starting HTML
390                 tag.")
391
392     def test_unknown_doctype(self):
393         """
394         Test that a doctype with an invalid name is reported as being
395         an unknown doctype.

```

```

393
394     Input:
395     A HTML fragment containing an invalid doctype name.
396
397     Expected Results:
398     An error should be thrown reporting that the doctype name is
        invalid.
399     """
400
401     inputFragment = '<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN
        "http://www.w3.org/TR/html4/strict.dtd">'
402
403     self.parser.parse(inputFragment)
404
405     self.assertIn(((0, 89), u'unknown-doctype', {}),
406         self.parser.errors, "Failed to report unknown doctype.")
407
408 def test_space_after_doctype(self):
409     """
410     Test that a doctype tag has a space between the doctype
        declaration
411     and the doctype name.
412
413     Input:
414     A HTML fragment containing a doctype with no space between the
        doctype
415     declaration and the doctype name.
416
417     Expected Results:
418     An error should be thrown reporting that there is no space
        between
419     the doctype declaration and the doctype name.
420     """
421
422     inputFragment = '<!DOCTYPEhtml>'
423
424     self.parser.parse(inputFragment)
425
426     self.assertIn(((0, 8), u'need-space-after-doctype', {}),
427         self.parser.errors, "Failed to report missing space after the
        doctype declaration.")
428
429 def test_end_tag_before_doctype(self):
430     """
431     Test that a closing tag isn't placed before the doctype
        declaration.
432
433     Input:
434     A HTML fragment containing a single closing tag.
435
436     Expected Results:
437     An error should be thrown reporting that a closing tag has been
        placed before the doctype declaration.
438     """
439
440
441     inputFragment = '</html>'
442

```

```

443         self.parser.parse(inputFragment)
444
445         self.assertIn(((0, 6), u'expected-doctype-but-got-end-tag', {u'
            name': u'html'})),
446             self.parser.errors, "Failed to report closing tag before
                doctype declaration.")
447
448     def test_EOF_before_doctype(self):
449         """
450         Test that an error is reported if the document is blank.
451
452         Input:
453         A blank document containing no characters of any kind.
454
455         Expected Results:
456         An error should be thrown reporting that the EOF was reached
            before
457         a doctype was declared.
458         """
459
460         inputFragment = ""
461
462         ""
463
464         self.parser.parse(inputFragment)
465
466         self.assertIn((-1, -1), u'expected-doctype-but-got-eof', {}),
467             self.parser.errors, "Failed to report the EOF occurring
                before the doctype declaration.")
468
469     def test_form_element_not_in_form(self):
470         """
471         Test that form elements must be contained in wrapping form tags.
472
473         Input:
474         A HTML fragment containing form elements which aren't wrapped in
475         form tags.
476
477         Expected Results:
478         An error should be thrown reporting that the form elements aren't
            contained
479         in wrapping form tags.
480         """
481
482         inputFragment = ""
483         <!DOCTYPE html>
484         <html>
485         <head>
486         </head>
487         <body>
488         <datalist></datalist>
489         <fieldset></fieldset>
490         <input></input>
491         <label></label>
492         <output></output>
493         </body>
494         </html>

```



```

495     """
496
497     self.parser.parse(inputFragment)
498
499     self.assertIn(((46, 55), u'form-element-not-in-form', {u'name': u
500         'datalist'})),
501         self.parser.errors, "Failed to report datalist tag outside of
502         wrapping form tags.")
503
504     self.assertIn(((68, 77), u'form-element-not-in-form', {u'name': u
505         'fieldset'})),
506         self.parser.errors, "Failed to report fieldset tag outside of
507         wrapping form tags.")
508
509     self.assertIn(((90, 96), u'form-element-not-in-form', {u'name': u
510         'input'})),
511         self.parser.errors, "Failed to report input tag outside of
512         wrapping form tags.")
513
514     self.assertIn(((106, 112), u'form-element-not-in-form', {u'name':
515         u'label'})),
516         self.parser.errors, "Failed to report label tag outside of
517         wrapping form tags.")
518
519     self.assertIn(((122, 129), u'form-element-not-in-form', {u'name':
520         u'output'})),
521         self.parser.errors, "Failed to report output tag outside of
522         wrapping form tag.")
523
524     def test_duplicate_id_value(self):
525         """
526         Test that the occurrence of duplicate id values is reported as an
527         error.
528
529         Input:
530         A HTML fragment containing 2 elements with the same value for
531         their id attribute.
532
533         Expected Results:
534         An error should be thrown reporting that an id attribute has the
535         same value of a previously declared id
536         attribute.
537         """
538
539         inputFragment = """
540 <!DOCTYPE html>
541 <html>
542 <head>
543 </head>
544 <body>
545 <p id="blah"></p>
546 <p id="blah"></p>
547 </body>
548 </html>
549 """
550
551     self.parser.parse(inputFragment)

```

```

539         self.assertIn(((64, 76), u'duplicate-id-attribute', {u'name': u'p
540             ', u'original': u'p'})),
541             self.parser.errors, "Failed to report duplicate id value
                    usage.")
542
543     def test_duplicate_page_title(self):
544         """
545         Test that a duplicate title element is reported as an error.
546
547         Input:
548         A HTML fragment containing duplicate title elements.
549
550         Expected Results:
551         An error should be thrown reporting that a duplicate instance of
552         the page
553         title has been found.
554         """
555         inputFragment = """
556 <!DOCTYPE html>
557 <html>
558 <head>
559 <title>blah</title>
560 <title>blah</title>
561 </head>
562 <body>
563 </body>
564 </html>
565         """
566
567         self.parser.parse(inputFragment)
568
569         self.assertIn(((51, 57), u'duplicate-title-in-head', {u'name': u'
570             title'})),
571             self.parser.errors, "Failed to report duplicate title element
                    .")
572
573     def test_missing_title_element(self):
574         """
575         Test that a missing title element as part of the head section
576         is reported as missing.
577
578         Input:
579         A HTML fragment containing a basic page structure but missing the
580         required
581         title element in the head section.
582
583         Expected Results:
584         An error should be thrown reporting that the title element is
585         missing from
586         the head sectionself.
587         """
588         inputFragment = """
589 <!DOCTYPE html>
590 <html>

```

```

589 <head>
590 </head>
591 <body>
592 </body>
593 </html>
594     """
595
596     self.parser.parse(inputFragment)
597
598     self.assertIn((-1, -1), u'title-element-missing-from-head', {}),
599         self.parser.errors, "Failed to report missing title element."
600
601     def test_img_missing_alt_attribute(self):
602         """
603         Test that an img tag missing the required alt attribute is
604             reported
605             as an error.
606
607         Input:
608         A HTML fragment containing an img tag missing the required alt
609             attribute.
610
611         Expected Results:
612         An error should be thrown reporting that the alt attribute is
613             missing for the
614             given img tag.
615         """
616
617         inputFragment = """
618 <!DOCTYPE html>
619 <html>
620 <head>
621 </head>
622 <body>
623 <img>
624 </body>
625 </html>
626     """
627
628     self.parser.parse(inputFragment)
629
630     self.assertIn((46, 50), u'img-element-missing-alt-attribute', {u
631         'name': u'img'}),
632         self.parser.errors, "Failed to report missing alt attribute
633             for the given img tag.")
634
635     def test_img_alt_attribute_empty(self):
636         """
637         Test that an img tag's alt attribute, when empty, is reported as
638             an error.
639
640         Input:
641         A HTML fragment containing an img tag with an empty alt attribute
642             .
643
644         Expected Results:

```

```

638         An error should be thrown reporting that the alt attribute is
639         empty.
640         """
641         inputFragment = """
642 <!DOCTYPE html>
643 <html>
644 <head>
645 </head>
646 <body>
647 <img alt="">
648 </body>
649 </html>
650         """
651
652         self.parser.parse(inputFragment)
653
654         self.assertIn(((46, 57), u'img-alt-attribute-empty', {u'attr': u'
655             '}),
656             self.parser.errors, "Failed to report empty alt attriubte for
657             img tag.")
658
659     def test_missing_closing_tag_before_footer(self):
660         """
661         Test that any open tags (missing the closing tag) are reported if
662         the
663         footer section occurs before closing tag.
664
665         Input:
666         A HTML fragment containing an open 'a' tag which is missing the
667         closing tag,
668         followed by the footer section.
669
670         Expected Results:
671         An error should be thrown reporting that the closing tag wasn't
672         found before
673         the footer section.
674         """
675         inputFragment = """
676 <!DOCTYPE html>
677 <html>
678 <head>
679 </head>
680 <body>
681 <a>
682 <footer>
683 </footer>
684 </body>
685 </html>
686         """
687
688         self.parser.parse(inputFragment)
689
690         self.assertIn(((50, 57), u'missing-end-tag-before-footer', {u'
691             name': u'a'})),

```

```

687         self.parser.errors, "Failed to report missing end tag before
        footer section.")
688
689     def test_missing_closing_tags_footer_section(self):
690         """
691         Test that missing closing tags in the footer section are reported
692         .
693         Input:
694         A HTML fragment containing an 'a' tag with a missing closing tag.
695
696         Expected Results:
697         An error should be thrown reporting that the closing tag is
698             missing within the footer
699         section.
700         """
701         inputFragment = """
702 <!DOCTYPE html>
703 <html>
704 <head>
705 </head>
706 <body>
707 <footer>
708 <a>
709 </footer>
710 </body>
711 </html>
712         """
713
714         self.parser.parse(inputFragment)
715
716         self.assertIn((59, 67), u'missing-closing-tags-in-footer', {u'
            name': u'a'}),
717             self.parser.errors, "Failed to report missing closing tag in
            footer section.")
718
719     def test_invalid_tag_name(self):
720         """
721         Test that tags with invalid tag names are reported as errors.
722
723         Input:
724         A HTML fragment containing a tag with an invalid name "blah".
725
726         Expected Results:
727         An error should be thrown reporting that the tag name is invalid.
728         """
729
730         inputFragment = """
731 <!DOCTYPE html>
732 <html>
733 <head>
734 </head>
735 <body>
736 <blah></blah>
737 <footer>
738 </footer>

```

```

739 </body>
740 </html>
741     """
742
743     self.parser.parse(inputFragment)
744
745     self.assertIn(((46, 51), u'invalid-element-name', {u'name': u'
746         blah'})),
747         self.parser.errors, "Failed to report invalid tag name.")
748
749     def test_missing_closing_tag(self):
750         """
751         Test that any missing closing tags are reported as errors.
752
753         Input:
754         A HTML fragment containing an opening 'a' tag with a missing
755             closing tag.
756
757         Expected Results:
758         An error should be thrown reporting that the closing tag is
759             missing.
760         """
761
762         inputFragment = """
763 <!DOCTYPE html>
764 <html>
765 <head>
766 </head>
767 <body>
768 <a>
769 <footer>
770 </footer>
771 </body>
772 </html>
773     """
774
775     self.parser.parse(inputFragment)
776
777     self.assertIn(((46, 48), u'missing-end-tag', {u'name': u'a'})),
778         self.parser.errors, "Failed to report missing closing tag.")
779
780 if __name__ == '__main__':
781     unittest.main()

```

4.3 JSON-RPC Server Tests

```
1 from __future__ import absolute_import, division, unicode_literals
2
3 #from . import support
4 import unittest
5 import time
6 import jsonrpclib
7 from multiprocessing import Pool
8 from jsonrpclib import Server
9 import httplib
10 import simplejson as json
11 import base64
12
13 """
14 Calls the test_concurrency method on the server. Required to be external
15 from the
16 TestJsonServer class as it was causing a "pickling" error when used as a
17 method.
18 """
19
20 def getTime(time):
21     return Server("http://localhost:8080").test_concurrency(time)
22
23 class TestJsonServer(unittest.TestCase):
24     """
25     Provides a number of test cases related to the functionality of the
26     json
27     rpc server.
28
29     These tests require that the server is currently running. The first
30     test checks that the server is running.
31     """
32
33     def setUp(self):
34         self.startTime = time.time()
35
36     def resetCurrentTime(self):
37         self.startTime = time.time()
38
39     def getExecutionTimes(self, numProcesses):
40         """
41         Attempts to call the getTime function with startTime as the
42         only argument in a concurrent manner using numProcesses as the
43         number of concurrent calls to make. The resulting times returned
44         by the remote function, test_concurrency, are added to a list
45         and returned.
46
47         The timeout for each call attempt is currently set to 5 seconds.
48         This will only allow for numProcesses to go up to 10. After that
49         the processing times at the server side will trigger the timeout
50         and result in an exception being thrown.
51         """
52
53         results = []
54         times = []
55
56         pool = Pool(processes=numProcesses)
```

```

53     for i in range(numProcesses):
54         results.append(pool.apply_async(getTime, (self.startTime,)))
55
56     for result in results:
57         times.append(result.get(timeout=5))
58
59     return times
60
61     def test_client(self):
62         """
63         Test that the server is currently running. Required for the
64         remaining server tests to run.
65
66         Input: Attempt to execute a known function on the server.
67
68         Expected Result: No exception being raised, implying that the
69         server
70         is currently running.
71         """
72         exceptionRaised = False;
73         try:
74             getTime(self.startTime)
75         except:
76             exceptionRaised = True
77
78         self.assertFalse(exceptionRaised, "The server isn't running.")
79
80     def test_concurrent_connections(self):
81         """
82         Test that the server can handle the maximum number of concurrent
83         connections while receiving a response in a similar time frame
84         for all requests.
85
86         Input: 5 concurrent function calls to the server.
87
88         Expected Result: The remote function, test_concurrency, contains
89         a 2
90         second sleep call. The sum of the times taken to complete each of
91         the function calls, relative to self.startTime should be between
92         the
93         range 11 > totalTime >= 10.
94         """
95
96         self.resetCurrentTime()
97
98         totalTime = 0
99
100         for time in self.getExecutionTimes(5):
101             totalTime += time
102
103         self.assertTrue(totalTime >= 10 and totalTime < 11, "Failed to "
104             +
105             "execute all 5 concurrent function calls within the expected
106             " +
107             "time frame.")

```



```

105     def test_max_concurrent_connections(self):
106         """
107         Tests that the server processes excess function calls after the
108         initial batch of calls.
109
110         Input: 6 concurrent function calls to the server.
111
112         Expected Result: The remote function, test_concurrency, contains
113             a 2
114             second sleep call. The server has a maximum number of concurrent
115             connections of 5, so the 6th call will take slightly over 4
116             seconds
117             to complete. The sum of the times for all 6 calls should be
118             within
119             the range 15 > totalTime >= 14.
120         """
121
122         self.resetCurrentTime()
123
124         totalTime = 0
125
126         for time in self.getExecutionTimes(6):
127             totalTime += time
128
129         self.assertTrue(totalTime >= 14 and totalTime < 15, "Failed to "
130             +
131             "execute all 6 concurrent function calls within the expected
132             " +
133             "time frame.")
134
135     def test_json_rpc_correct_response(self):
136         """
137         Tests that the server responds as expected to a correctly
138         formed JSON-RPC 2.0 request.
139
140         Input: A correctly formed JSON-RPC 2.0 request containing
141             an empty array of file names, an empty file name (direct
142             input method) and a HTML fragment consisting of '<html></html>'.
143
144         Expected Result: The returned JSON-RPC 2.0 response string
145             should match the string expectedResponse, which contains
146             the expected array of errors.
147         """
148
149         conn = httplib.HTTPConnection("127.0.0.1:8080")
150         fragment = base64.b64encode(b'<html></html>')
151         params = [{"files": [], "document": fragment, "filename": ""}]
152         request = json.JSONEncoder().encode({"jsonrpc": "2.0", "method":
153             "parse_html",
154             "params": params, "id": "A3s23"})
155         header = {"Content-type": "application/json"}
156
157         conn.request("POST", "", request, header)
158         response = conn.getresponse()
159         conn.close()
160
161         expectedResponse = '{"jsonrpc": "2.0", "result": [[1, 0, 5, {"
162             name": "html"}], [25, 6, 12, {"name": "html"}]], "id": "A3s23

```

```

155         "}',
156         self.assertEqual(response.read(), expectedResponse, "Wrong
            response.")
157
158     def test_json_rpc_malformed_parameters(self):
159         """
160         Tests that the server responds with an error when
161         a request contains incorrect parameters.
162
163         Input: A JSON-RPC 2.0 request containing incorrectly
164         formatted parameters to be passed on to the requested
165         function.
166
167         Expected Result: The returned JSON-RPC 2.0 response string
168         should match the string expectedResponse, which contains
169         a response representing an invalid parameters error.
170         """
171         conn = httplib.HTTPConnection("127.0.0.1:8080")
172         fragment = base64.b64encode(b'<html></html>')
173         params = []
174         request = json.JSONEncoder().encode({"jsonrpc": "2.0", "method":
            "parse_html",
175            "params": params, "id": "A3s23"})
176         header = {"Content-type": "application/json"}
177
178         conn.request("POST", "", request, header)
179         response = conn.getresponse()
180         conn.close()
181
182         expectedResponse = '{"error": {"message": "Invalid parameters.",
            "code": -32602}, "jsonrpc": "2.0", "id": "A3s23"}'
183
184         self.assertEqual(response.read(), expectedResponse, "Wrong
            response.")
185
186     def test_json_rpc_unsupported_method(self):
187         """
188         Tests that the server responds with an error when
189         a client attempts to make a function call for a function
190         which hasn't been registered to the server.
191
192         Input: A JSON-RPC 2.0 request containing a function name
193         which hasn't been registered on the server.
194
195         Expected Result: The returned JSON-RPC 2.0 response
196         string should match the string expectedResponse, which
197         contains a response representing an unsupported method
198         error.
199         """
200         conn = httplib.HTTPConnection("127.0.0.1:8080")
201         fragment = base64.b64encode(b'<html></html>')
202         params = [{"files": [], "document": fragment, "filename": ""}]
203         request = json.JSONEncoder().encode({"jsonrpc": "2.0", "method":
            "not_registered",
204            "params": params, "id": "A3s23"})
205         header = {"Content-type": "application/json"}

```

```

206     conn.request("POST", "", request, header)
207     response = conn.getresponse()
208     conn.close()
209
210
211     expectedResponse = '{"error": {"message": "Method not_registered
        not supported.", "code": -32601}, "jsonrpc": "2.0", "id": "
        A3s23"}'
212
213     self.assertEqual(response.read(), expectedResponse, "Wrong
        response.")
214
215     def test_invalid_filepath(self):
216         """
217         Tests that the server responds with an error when
218         a client attempts to make a function call for a function
219         which hasn't been registered to the server.
220
221         Input: A JSON-RPC 2.0 request containing a function name
222         which hasn't been registered on the server.
223
224         Expected Result: The returned JSON-RPC 2.0 response
225         string should match the string expectedResponse, which
226         contains a response representing an unsupported method
227         error.
228         """
229         conn = httplib.HTTPConnection("127.0.0.1:8080")
230         fragment = base64.b64encode(b'<img src=../image.jpg><img src=
            directory2/image2.jpg>')
231         params = [{"files": ["image.jpg", "directory/", "directory/
            current.html"], "document": fragment, "filename": "directory/
            current.html"}]
232         request = json.JSONEncoder().encode({"jsonrpc": "2.0", "method":
            "not_registered",
            "params": params, "id": "A3s23"})
233         header = {"Content-type": "application/json"}
234
235         conn.request("POST", "", request, header)
236         response = conn.getresponse()
237         conn.close()
238
239
240         expectedResponse = '{"error": {"message": "Method not_registered
            not supported.", "code": -32601}, "jsonrpc": "2.0", "id": "
            A3s23"}'
241
242         self.assertEqual(response.read(), expectedResponse, "Wrong
            response.")
243
244     def test_invalid_filepath(self):
245         """
246         Tests that the server responds with an error when
247         a client attempts to make a function call for a function
248         which hasn't been registered to the server.
249
250         Input: A JSON-RPC 2.0 request containing a function name
251         which hasn't been registered on the server.
252

```

```

253     Expected Result: The returned JSON-RPC 2.0 response
254     string should match the string expectedResponse, which
255     contains a response representing an unsupported method
256     error.
257     """
258     conn = httplib.HTTPConnection("127.0.0.1:8080")
259     fragment = base64.b64encode(b'<img src=../image.jpg><img src=
        directory2/image2.jpg>')
260     params = [{"files": ["image.jpg", "directory/", "directory/
        current.html"], "document": fragment, "filename": "directory/
        current.html"}]
261     request = json.JSONEncoder().encode({"jsonrpc": "2.0", "method":
        "not_registered",
262         "params": params, "id": "A3s23"})
263     header = {"Content-type": "application/json"}
264
265     conn.request("POST", "", request, header)
266     response = conn.getresponse()
267     conn.close()
268
269     expectedResponse = '{"error": {"message": "Method not_registered
        not supported.", "code": -32601}, "jsonrpc": "2.0", "id": "
        A3s23"}'
270
271     self.assertEqual(response.read(), expectedResponse, "Wrong
        response.")
272
273     def test_invalid_filepath(self):
274         """
275         Tests that the server responds with an error when
276         a client attempts to make a function call for a function
277         which hasn't been registered to the server.
278
279         Input: A JSON-RPC 2.0 request containing a function name
280         which hasn't been registered on the server.
281
282         Expected Result: The returned JSON-RPC 2.0 response
283         string should match the string expectedResponse, which
284         contains a response representing an unsupported method
285         error.
286         """
287         conn = httplib.HTTPConnection("127.0.0.1:8080")
288         fragment = base64.b64encode(b'<img src=../image.jpg><img src=
            directory2/image2.jpg>')
289         params = [{"files": ["image.jpg", "directory/", "directory/
            current.html"], "document": fragment, "filename": "directory/
            current.html"}]
290         request = json.JSONEncoder().encode({"jsonrpc": "2.0", "method":
            "not_registered",
291             "params": params, "id": "A3s23"})
292         header = {"Content-type": "application/json"}
293
294         conn.request("POST", "", request, header)
295         response = conn.getresponse()
296         conn.close()
297

```

```

298     expectedResponse = '{"error": {"message": "Method not_registered
      not supported.", "code": -32601}, "jsonrpc": "2.0", "id": "
      A3s23"}'
299
300     self.assertEqual(response.read(), expectedResponse, "Wrong
      response.")
301
302     def test_invalid_filepath(self):
303         """
304         Tests that the server responds with an error when
305         a client attempts to make a function call for a function
306         which hasn't been registered to the server.
307
308         Input: A JSON-RPC 2.0 request containing a function name
309         which hasn't been registered on the server.
310
311         Expected Result: The returned JSON-RPC 2.0 response
312         string should match the string expectedResponse, which
313         contains a response representing an unsupported method
314         error.
315         """
316         conn = httpplib.HTTPConnection("127.0.0.1:8080")
317         fragment = base64.b64encode(b'<img src=../image.jpg><img src=
      directory2/image2.jpg>')
318         params = [{"files": ["image.jpg", "directory/", "directory/
      current.html"], "document": fragment, "filename": "directory/
      current.html"}]
319         request = json.JSONEncoder().encode({"jsonrpc": "2.0", "method":
      "not_registered",
320         "params": params, "id": "A3s23"})
321         header = {"Content-type": "application/json"}
322
323         conn.request("POST", "", request, header)
324         response = conn.getresponse()
325         conn.close()
326
327         expectedResponse = '{"error": {"message": "Method not_registered
      not supported.", "code": -32601}, "jsonrpc": "2.0", "id": "
      A3s23"}'
328
329         self.assertEqual(response.read(), expectedResponse, "Wrong
      response.")
330
331     def test_invalid_filepath(self):
332         """
333         Tests that the server responds with an error when
334         a client attempts to make a function call for a function
335         which hasn't been registered to the server.
336
337         Input: A JSON-RPC 2.0 request containing a function name
338         which hasn't been registered on the server.
339
340         Expected Result: The returned JSON-RPC 2.0 response
341         string should match the string expectedResponse, which
342         contains a response representing an unsupported method
343         error.
344         """

```

```

345     conn = httplib.HTTPConnection("127.0.0.1:8080")
346     fragment = base64.b64encode(b'<img src=../image.jpg><img src=
        directory2/image2.jpg>')
347     params = [{"files": ["image.jpg", "directory/", "directory/
        current.html"], "document": fragment, "filename": "directory/
        current.html"}]
348     request = json.JSONEncoder().encode({"jsonrpc": "2.0", "method":
        "not_registered",
349         "params": params, "id": "A3s23"})
350     header = {"Content-type": "application/json"}
351
352     conn.request("POST", "", request, header)
353     response = conn.getresponse()
354     conn.close()
355
356     expectedResponse = '{"error": {"message": "Method not_registered
        not supported.", "code": -32601}, "jsonrpc": "2.0", "id": "
        A3s23"}'
357
358     self.assertEqual(response.read(), expectedResponse, "Wrong
        response.")
359
360     def test_invalid_filepath(self):
361         """
362         Tests that the server responds with an error when
363         a client attempts to make a function call for a function
364         which hasn't been registered to the server.
365
366         Input: A JSON-RPC 2.0 request containing a function name
367         which hasn't been registered on the server.
368
369         Expected Result: The returned JSON-RPC 2.0 response
370         string should match the string expectedResponse, which
371         contains a response representing an unsupported method
372         error.
373         """
374         conn = httplib.HTTPConnection("127.0.0.1:8080")
375         fragment = base64.b64encode(b'<img src=../image.jpg><img src=
            directory2/image2.jpg>')
376         params = [{"files": ["image.jpg", "directory/", "directory/
            current.html"], "document": fragment, "filename": "directory/
            current.html"}]
377         request = json.JSONEncoder().encode({"jsonrpc": "2.0", "method":
            "not_registered",
378             "params": params, "id": "A3s23"})
379         header = {"Content-type": "application/json"}
380
381         conn.request("POST", "", request, header)
382         response = conn.getresponse()
383         conn.close()
384
385         expectedResponse = '{"error": {"message": "Method not_registered
            not supported.", "code": -32601}, "jsonrpc": "2.0", "id": "
            A3s23"}'
386
387         self.assertEqual(response.read(), expectedResponse, "Wrong
            response.")

```

```

388
389     def test_invalid_filepath(self):
390         """
391         Tests that the server responds with an error when
392         a client attempts to make a function call for a function
393         which hasn't been registered to the server.
394
395         Input: A JSON-RPC 2.0 request containing a function name
396         which hasn't been registered on the server.
397
398         Expected Result: The returned JSON-RPC 2.0 response
399         string should match the string expectedResponse, which
400         contains a response representing an unsupported method
401         error.
402         """
403         conn = httplib.HTTPConnection("127.0.0.1:8080")
404         fragment = base64.b64encode(b'<img src=../image.jpg><img src=
            directory2/image2.jpg>')
405         params = [{"files": ["image.jpg", "directory/", "directory/
            current.html"], "document": fragment, "filename": "directory/
            current.html"}]
406         request = json.JSONEncoder().encode({"jsonrpc": "2.0", "method":
            "not_registered",
407            "params": params, "id": "A3s23"})
408         header = {"Content-type": "application/json"}
409
410         conn.request("POST", "", request, header)
411         response = conn.getresponse()
412         conn.close()
413
414         expectedResponse = '{"error": {"message": "Method not_registered
            not supported.", "code": -32601}, "jsonrpc": "2.0", "id": "
            A3s23"}'
415
416         self.assertEqual(response.read(), expectedResponse, "Wrong
            response.")
417
418     if __name__ == '__main__':
419         unittest.main()

```