

DECO3801 Test Plan Document

THEM - Typed HTML5 Evaluation Machine

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

Functional Test Plan

Testing Strategy

There are three major testable components of the our 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 our front-end website and database with a suite of computer-run tests. Instead, we wrote up a series of scenarios that we would undertake to 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 we are running on the parser can be found in Appendix A within this document. This gives an indication of the tests we have 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.

Test Case Transcript

Python Parser Tests

Test Number	Test Description
1	Testing that a specific error is being reported correctly, given a particular fragment of HTML as the input.
2	Test that the JSON-RPC server is running and can respond to a remote function call.
3	Testing that the JSON-RPC server is able to handle up to a maximum of 5 concurrent remote function calls.
4	Testing that a 6th concurrent connection (1 connection over the maximum of 5 concurrent connections) times out.
5	Testing the parser’s response to a case where input of an empty string is supplied.
6	Testing the parser’s response when the input string doesn’t contain any valid HTML.
7	Testing that a correctly formed JSON-RPC 2.0 request is handled by the server, which should respond with a JSON-RPC 2.0 response.
8	Testing that a malformed JSON-RPC 2.0 request containing incorrect parameters for a particular function causes the server to respond with an error.
9	Testing that a JSON-RPC 2.0 request calling a function that isn’t registered on the server causes the server to respond with an error.
10	Testing the parser response when a tag with a URL attribute is supplied with a valid relative file path.
11	Testing the parser response when a tag with a URL attribute is supplied with a nonexistent relative file path.
12	Testing the parser response when a tag with a URL attribute is supplied with a nonexistent file in existing directory.
13	Testing the parser response when a tag with a URL attribute is supplied with a valid absolute file path.
14	Testing the parser response when a tag with a URL attribute is supplied with a nonexistent absolute file path.
15	Testing the parser response when a tag with a URL attribute is supplied with nonexistent files in existing directory.

Website and Server Tests

Test Number	Test Description	Inputs
1	View Home page	Go to URL, or click li
2	View Help page	Click link from any pa
3	View Direct Input page	Click link from any pa
4	Validate direct input	The user types their in
5	View Upload File(s) page	Click link from any pa
6	Upload single HTML file	The user selects a file
7	Upload multiple HTML files individually	The user clicks the Ad
8	Upload multiple HTML files together from one dialogue	The user selects multip
9	Upload multiple HTML files, some individually and some from one dialogue box	The user performs a co
10	Upload non-HTML file	The user attempts to c
11	No file selected on upload	The user attempts to c
12	View Upload Zip page	Click link from any pa
13	Upload zip file	Zip file selected on pre
14	View Uploaded Set page	User either uploads m
15	View Uploaded File page	User either selects a fi
16	Remove file after certain period of being untouched in the server	A file should be remov
17	User attempts to access files they have not uploaded	The user manually att

Implications of Functional Testing

Our functional testing highlighted some issues with all aspects of our application.

User Experience Goals

We had a clear user experience in mind while developing this website. Through its ease of use and minimal effort on the part of the user, we have aimed to create a very surgical, ambient, passive experience. The tool should give users immediate insight into the issues with their HTML and websites. This is where the user's experience with our tool ends, for this session. The user now can go and fix their file externally, return to our program and almost instantly receive another assessment of their code's validity. We do not aim to get the user invested in our system. However, we wish to create a reliable and worthwhile experience, brief as it is. The user should not be frustrated by the errors the program reveals, with the focus 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 everything is instantly accessible and requires very few clicks to navigate. We have designed the website to require as few as five clicks to access the primary functionality of the system.

User Testing Plan

User Testing Strategy

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

- 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 our part lead to us getting very few in this category. Ultimately, we got information from ten users - five were undergraduates who had not done DECO1400, one was an undergraduate who had done DECO1400 and four were masters students who had done DECO7140.

We formulated six key scenarios for our 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

We focused on the metrics of time taken to complete each scenario, and, in keeping with our surgical user experience, number of clicks required to complete each scenario. As we also wanted to create an enjoyable environment for the users, we also made note of any particular emotions and reactions of the users as they undertook the scenarios. Our 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 we get started, 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.

User Test Results

Summarise the results of your tests. For each scenario-based test: tabulate your metric results against each task describe or present your users' feedback during / after the test

Implications of User Testing

- in general, users had no trouble navigating the system - the result of the validation (i.e. highlighted tags) was not well understood by users

Close with a general discussion that: summarises the issues raised identifies areas for improvement and design suggestions outlines any redefinition of functional and user test plans for final prototype (We dont care whether your prototype passes the tests. What is important is that your prototype is sufficiently broad to validate your test plan for the final product.)

Appendix A - Python Test Code

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 if __name__ == '__main__':
112     unittest.main()

```

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         self.assertIn(((15, 18), u'incorrect-end-tag-placement-in-head',
           {u'name': u'a'})),
176             self.parser.errors, "Failed to report closing tag which doesn
           't belong in the head section.")
177
178     def test_tags_after_eof(self):
179         """
180         Tests that starting and closing tags occurring after the last
181         instance of a closing HTML tag are reported as an error.
182
183         Input:
184         A HTML fragment with a start and closing tag pair occurring
185         after the start and closing HTML pair.
186
187         Expected Results:
188         An error being thrown for both the start and closing tags

```

```

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

```

```

240
241     self.assertIn(((32, 34), u'unexpected-start-tag-after-body', {u'
242         name': u'a'})),
243         self.parser.errors, "Failed to report misplaced starting tag
244             found after the closing body tag.")
245
246     self.assertIn(((35, 38), u'unexpected-end-tag-after-body', {u'
247         name': u'a'})),
248         self.parser.errors, "Failed to report misplaced closing tag
249             found after the closing body tag.")
250
251 def test_missing_closing_html_tag(self):
252     """
253     Test that a missing closing HTML tag is reported.
254
255     Input:
256     A HTML fragment missing a closing HTML tag.
257
258     Expected Results:
259     An error should be thrown stating that the closing HTML tag is
260         missing.
261     """
262     inputFragment = "<html><head></head><body></body>"
263
264     self.parser.parse(inputFragment)
265
266     self.assertIn((-1, -1), u'no-closing-html-tag', {}),
267         self.parser.errors, "Failed to report missing closing HTML
268             tag.")
269
270 def test_early_termination_before_head(self):
271     """
272     Test that an early closing HTML tag before the head phase
273     is reported as an error.
274
275     Input:
276     A HTML fragment with the head and body sections placed after
277     a closed set of HTML tags.
278
279     Expected Results:
280     An error should be thrown stating that the closing HTML tag
281     has been found before the head phase.
282     """
283     inputFragment = "<html></html><head></head><body></body>"
284
285     self.parser.parse(inputFragment)
286
287     self.assertIn(((6, 12), u'early-termination-before-head', {u'name
288         ': u'html'})),
289         self.parser.errors, "Failed to report early termination
290             before head section.")
291
292 def test_early_termination_in_head(self):
293     """
294     Test that an early closing HTML tag in the head phase

```

```

289         is reported as an error.
290
291     Input:
292     A HTML fragment with the closing HTML tag placed within
293     the set of head tags.
294
295     Expected Results:
296     An error should be thrown stating that the closing HTML tag
297     has been found in the head phase.
298     """
299
300     inputFragment = "<html><head></html></head><body></body>"
301
302     self.parser.parse(inputFragment)
303
304     self.assertIn(((12, 18), u'early-termination-in-head', {u'name':
305         u'html'})),
306         self.parser.errors, "Failed to report early termination
307         before head section.")
308
309 def test_early_termination_before_body(self):
310     """
311     Test that an early closing HTML tag before the body phase
312     is reported as an error.
313
314     Input:
315     A HTML fragment with the closing HTML tag placed before the body
316     section.
317
318     Expected Results:
319     An error should be thrown stating that the closing HTML tag
320     has been found before the body phase.
321     """
322
323     inputFragment = "<html><head></head></html><body></body>"
324
325     self.parser.parse(inputFragment)
326
327     self.assertIn(((19, 25), u'early-termination-before-body', {u'
328         name': u'html'})),
329         self.parser.errors, "Failed to report early termination
330         before head section.")
331
332 def test_early_termination_in_body(self):
333     """
334     Test that an early closing HTML tag in the body phase
335     is reported as an error.
336
337     Input:
338     A HTML fragment with the closing HTML tag placed within
339     the set of body tags.
340
341     Expected Results:
342     An error should be thrown stating that the closing HTML tag
343     has been found in the head phase.
344     """

```



```

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

```

```

392     def test_unknown_doctype(self):
393         """
394         Test that a doctype with an invalid name is reported as being
395         an unknown doctype.
396
397         Input:
398         A HTML fragment containing an invalid doctype name.
399
400         Expected Results:
401         An error should be thrown reporting that the doctype name is
402             invalid.
403         """
404         inputFragment = '<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN'
405             ' "http://www.w3.org/TR/html4/strict.dtd">'
406
407         self.parser.parse(inputFragment)
408
409         self.assertIn(((0, 89), u'unknown-doctype', {}),
410             self.parser.errors, "Failed to report unknown doctype.")
411
412     def test_space_after_doctype(self):
413         """
414         Test that a doctype tag has a space between the doctype
415             declaration
416         and the doctype name.
417
418         Input:
419         A HTML fragment containing a doctype with no space between the
420             doctype
421         declaration and the doctype name.
422
423         Expected Results:
424         An error should be thrown reporting that there is no space
425             between
426         the doctype declaration and the doctype name.
427         """
428         inputFragment = '<!DOCTYPEhtml>'
429
430         self.parser.parse(inputFragment)
431
432         self.assertIn(((0, 8), u'need-space-after-doctype', {}),
433             self.parser.errors, "Failed to report missing space after the
434             doctype declaration.")
435
436 if __name__ == '__main__':
437     unittest.main()

```

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

```



```

299         not_supported.", "code": -32601}, "jsonrpc": "2.0", "id": "
300         A3s23"}'
301
302     self.assertEqual(response.read(), expectedResponse, "Wrong
303         response.")
304
305     def test_invalid_filepath(self):
306         """
307         Tests that the server responds with an error when
308         a client attempts to make a function call for a function
309         which hasn't been registered to the server.
310
311         Input: A JSON-RPC 2.0 request containing a function name
312         which hasn't been registered on the server.
313
314         Expected Result: The returned JSON-RPC 2.0 response
315         string should match the string expectedResponse, which
316         contains a response representing an unsupported method
317         error.
318         """
319         conn = httplib.HTTPConnection("127.0.0.1:8080")
320         fragment = base64.b64encode(b'<img src=../image.jpg><img src=
321         directory2/image2.jpg>')
322         params = [{"files": ["image.jpg", "directory/", "directory/
323         current.html"], "document": fragment, "filename": "directory/
324         current.html"}]
325         request = json.JSONEncoder().encode({"jsonrpc": "2.0", "method":
326         "not_registered",
327         "params": params, "id": "A3s23"})
328         header = {"Content-type": "application/json"}
329
330         conn.request("POST", "", request, header)
331         response = conn.getresponse()
332         conn.close()
333
334         expectedResponse = '{"error": {"message": "Method not_registered
335         not supported.", "code": -32601}, "jsonrpc": "2.0", "id": "
336         A3s23"}'
337
338         self.assertEqual(response.read(), expectedResponse, "Wrong
339         response.")
340
341     def test_invalid_filepath(self):
342         """
343         Tests that the server responds with an error when
344         a client attempts to make a function call for a function
345         which hasn't been registered to the server.
346
347         Input: A JSON-RPC 2.0 request containing a function name
348         which hasn't been registered on the server.
349
350         Expected Result: The returned JSON-RPC 2.0 response
351         string should match the string expectedResponse, which
352         contains a response representing an unsupported method
353         error.
354         """
355         conn = httplib.HTTPConnection("127.0.0.1:8080")

```

```

346     fragment = base64.b64encode(b'<img src=../image.jpg><img src=
347         directory2/image2.jpg>')
348     params = [{"files": ["image.jpg", "directory/", "directory/
349         current.html"], "document": fragment, "filename": "directory/
350         current.html"}]
351     request = json.JSONEncoder().encode({"jsonrpc": "2.0", "method":
352         "not_registered",
353         "params": params, "id": "A3s23"})
354     header = {"Content-type": "application/json"}
355
356     conn.request("POST", "", request, header)
357     response = conn.getresponse()
358     conn.close()
359
360     expectedResponse = '{"error": {"message": "Method not_registered
361         not supported.", "code": -32601}, "jsonrpc": "2.0", "id": "
362         A3s23"}'
363
364     self.assertEqual(response.read(), expectedResponse, "Wrong
365         response.")
366
367 def test_invalid_filepath(self):
368     """
369     Tests that the server responds with an error when
370     a client attempts to make a function call for a function
371     which hasn't been registered to the server.
372
373     Input: A JSON-RPC 2.0 request containing a function name
374     which hasn't been registered on the server.
375
376     Expected Result: The returned JSON-RPC 2.0 response
377     string should match the string expectedResponse, which
378     contains a response representing an unsupported method
379     error.
380     """
381     conn = httplib.HTTPConnection("127.0.0.1:8080")
382     fragment = base64.b64encode(b'<img src=../image.jpg><img src=
383         directory2/image2.jpg>')
384     params = [{"files": ["image.jpg", "directory/", "directory/
385         current.html"], "document": fragment, "filename": "directory/
386         current.html"}]
387     request = json.JSONEncoder().encode({"jsonrpc": "2.0", "method":
388         "not_registered",
389         "params": params, "id": "A3s23"})
390     header = {"Content-type": "application/json"}
391
392     conn.request("POST", "", request, header)
393     response = conn.getresponse()
394     conn.close()
395
396     expectedResponse = '{"error": {"message": "Method not_registered
397         not supported.", "code": -32601}, "jsonrpc": "2.0", "id": "
398         A3s23"}'
399
400     self.assertEqual(response.read(), expectedResponse, "Wrong
401         response.")

```

```

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 = httpplib.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()

```