OWL troubleshooting/debugging/testing

Chris Barrieau

For a full test, please see the “Test Plan(Full)” section at the bottom of the document

Here is just kind of what I’ve been doing to test – hopefully this is helpful to others for getting an idea how to test systems. There might be stuff I left out – please ask me questions.

Hopefully this will help people do some testing on all the pieces – everyone should be able to get the back end running, even without IIS, and the database populated.

To set up the system for debugging without IIS, you need to

1. Get the code from Github
2. Run PopulateDB to set up the database
3. Run Visual Studio and open OWLWebsite
4. Go to path\front-end\owl.html and double click.

The system should, at this point, be everything you need to do full testing. IIS testing is more particular and can be done at rarer intervals or by people who have the system set up. The main goal is to test the code with the other components, which this allows you to do.

# Front End

I’m not expert on this piece or testing javascript – probably Scott and Arthur would know much more. For testing I usually put an alert(“”); box in code in order to return messages or variables, with some degree of success.

Code can typically be refreshed after a save (if you make an update to a .js file, you can click refresh in the browser to make it work) the exception is if Visual Studio is running OWLWebsite: Sometimes it freezes a snapshot of the code and won’t do the updates, and the old code will be used. If this is the case, exit visual studio and relaunch, refreshing before you start the program again.

# Back End

For testing this piece, there are a few things which can be done. First, the program can be run in IIS or in the debugger, but not both at the same time. If you want to run the debugger, go to IIS and shut off the web page (click on the globe near OWLWebsite and click stop in the right pane). In Visual Studio, launch the program by going to the green arrow to launch with debugging.

It’s possible you may encounter errors (**I will try to get these fixed by Thursday 11/18/2015)**.

If you aren’t allowed entry at <https://localhost:32297> (or it times out, or gives an error), try using <http://localhost:32296> and see if that returns the list of categories in a browser window. **If this is the case, ClientCommunicationModule.js needs to have the value ‘serveraddress’ updated to the new value.**

The other thing to try is to go into the .vs folder, under config\application config

In the section <bindings> for the site name (note there may be multiple, change all of them), change to:

**<bindings>**

**<binding protocol=”http” bidingInformation=”\*:32296:localhost”/>**

**<binding protocol=”https” bindingInformation=”\*:32297:localhost”/>**

**</bindings>**

Basically, the http and https can sometimes both be pointing to 32297, and I’m not sure what setting in Visual Studio will change that, but it can be updated here.

For testing once it launches, you can put breakpoints (f9 on a line of code) at various places and use the locals window or watch window to check for results while the system is running. Good places to put breakpoints are inside exception catch statements, and any place where DAO is called in Services.cs (DAO.Query, DAO.Insert, DAO.TransferQuestionAns). Single step through the code to see what it does (F10 advances one step, F11 will enter function calls), or press F5 to see if it gets into an exception. Make note of the exception; it will probably involve the SQL database but will say what the issue is.

When we add logging we can also check the logs for successes or failures.

# Admin (insert)

To test this piece, you need to have access to the database and to the back end (unless you use Scott’s echo code). This is standard input testing, but what also can be tested is to go to the database and write a query using the plantid (or some other feature) to make sure it was added to the database. If successfully added, the object should also be ‘echoed’ at the admin page.

# PopulateDB

To test this piece, you need to have access to the database. Run the code and do select on all the tables, making sure they are full. In the case of the categories, make sure the values are valid.

A full test is to run the full integration on the data (see below).

# A full integrated test (basic)

1. After getting everything configured, go to [https://localhost/owl.html](https://localhost:32297)
2. Click on every category and make sure the data is populated. Make sure every picture is working.
3. Click on a category and click submit
4. Validate that results are returned.
5. In the same category, click on a different selection
6. Validate that the results change.
7. Test every selection to make sure that it works. If an entry returns no results, run a query on the database to make sure that it doesn’t.
8. Click reset. Repeat with every category, clicking reset each time.
9. Validate the results window displays pictures, duckduckgo URLS, and duckduckgo ImageURLs.
10. Go to the admin page. Enter in an item and select insert.
11. Do basic input testing on the text boxes –long strings, or leave entries blank. The system should be able to handle these cases.

# Multiple entry select testing

1. Running the C# back end in debug mode, put a breakpoint in READ on the DAO line.
2. Run the front end with a ‘submit’. Make note of your selections.
3. Look at the FloraObj. Make sure the selections you made are recorded by the FloraObj.
4. Let the code run, validate that results are returned to the main window.
5. A similar process can be applied with the admin insert page, but with a breakpoint set on CREATE. It can also be done with the GET statement to get the categories, to make sure it is being called.

# Whatever you can think of

Part of testing is to come up with cases that the developer or one person might not have thought of. Look at the inputs and see if you can break the system. The idea of multiple testers is to have multiple perspectives trying things.

# Test Plan (full):

1. Delete all tables in the database
2. Run populateDB (note: you may need to run the program CreateDB or create a database “OWL” if this is your first time running the software).
3. Do a select top 1000 on all four tables (plant, planttype, locaiton, questionans). Validate the data.
4. Launch TestOWL application in Visual Studio. Build.
5. Launch OWLWebsite application in Visual Studio. Build
6. Stop then start OWLWebsite in IIS
7. Stop then start OWL in IIS
8. Validate <https://localhost:32297> returns the list of categories in JSON.
9. Launch TestOWL in Visual Studio. **You should get 0 errors.** Press enter to close it. If you get errors, log how many failed and where they failed (text should indicate where it failed)
10. Launch <https://localhost/owl.html>
11. Run the full integrated test (see above)
12. Go to <https://localhost/owladmin.html>. Validate you are sent to the login page.
13. Validate the login with “Admin”, “Admin”
14. Logout. Make sure you leave the page
15. Attempt to login with “Admin” “Fake”
16. Make sure the login was rejected
17. Attempt to login with “Fake” “Admin”
18. Make sure the login was rejected
19. Validate the login with “Admin”, “Admin” again
20. Insert several items by filling in values and clicking “Submit”
21. Give several items the same plant ids. Make sure the plant ids returned are incrementing
22. Don’t fill in a plant id. Make sure the first four letters of the name and a number are returned as plant id.
23. Don’t supply a plant id or a name. Make sure an error is returned.
24. Attempt a delete of a plant id that doesn’t exist. Make sure a message is displayed
25. Attempt a delete of a plant id that does exist. Make sure it displays.

# JSON section

To test JSON, there are several things which can be done:

1. Use the TestOWL Program. TestOWL will use JSON.net to convert the objects from json strings into FloraObj. It then checks to make sure the entries match what should be in the FloraObj
2. The other method to test JSON is to go to the front end. The items you should see are the name and the ImageURL, as well as an image search and search on duckduckgo (if the feature is turned on).
3. The OWL\_Log.txt file can also be used to show json being sent into the server (the server does not print out json messages since query results can be very large).