FX Alarm Project Proposal

Contents

Minimum Viable Product (MVP not most valuable player but Most Valuable Professional also)

The following MVP should be worked on in the following order of deliverables:

1. Configuration page or pages (website) for diagramed use cases laid out, and including one “happy path” to drive forth the development from the front to the backend, this includes the paired unit test module in the event we have a regression to fix.
2. Data module based on the use case model, this includes the paired unit test module in the event we have a regression to fix.
3. Data harvest module:
    a. One python module based on a python imitation of the data, this includes the paired unit test module in the event we have a regression to fix.
    b. One python module based on a completed, accessible, verified http data pipe time permitting, this includes the paired unit test module in the event we have a regression to fix.

Website Data Source (Data Driven) Strategy Steps:

1. Prepare a PYTHON-BASED reflection of this data model and provided by ONE defined “happy path” from the configuration website pages we will build for this conduit data pipeline via HTTP. This step is to provide a backup mechanism to show my working model during demo day as contingency plan.
2. Website source is via paid subscription ($19.95 per month) requires regular verification this bill is PAID, ACTIVE, that the paypal.com billing agreement is ACTIVE, and verify that it is ACCESSIBLE. This verification grants us personal and non-professional access under those terms so long as the service is running.
3. IFF the preceding steps are verified, then permission will be sought to build the remaining section of this HTTP pipe line later in the MS Project .mpp schedule as changes forth coming.
4. Begin construction of the primary data pipeline based on HTTP get, Xslt, and Xml. The service API does not expose HTTP soap protocol based web services except through the CSS styled Http layer human readable.

Follow-Up Work (Out of Present Scope TODO):

1. Module/Test Module - Trade module configuration page
2. Module - Email template configuration html notice
3. Module/Test Module - Email alert notice generation
4. Module - Trade Execution Program/API
5. Test Module - Trade Execution Program/API

Project Goal:

The goal of this project is to create a data harvest python program that is configurable and viewable through an external website to specifically apply execution settings for the data harvest python program, and be able to view what recently happened on the current trading session for the day for events. The second top level goal for this project is to execute trades in reaction to this financial data through the MetaTrader terminal using the language for that terminal.

FX Alarm Final Project – Specific Module Deliverables for Minimum Viable Product sprint

Module - Interface with FX website Data Source

This module will be a bit easier for me now because some years back explored around how to take and record a precise user interactions and UX (user experiences) as a single use case with a webpage through a tool called a web test recorded in Selenium or the Web Performance Test in Visual Studio. This is meant to take each individual recording use case and produce that recorded interactions set as a specific unit test for a performance test or a load test server side.

For this module I will take such a recording to produce precisely the form post parameters required to:

1) Login to the website I will use as a data source,
2) Use the best means available that the current active session is still active and maintain that session, and
3) Logout when configured to or switched off.

There are two Python importable module able to do this either the bit more light weight requests module, or the Django module/project template itself – this is still to be determined also.

Rubric Use Case – Harvest Configuration and Turn Harvester On/Off Page

A configuration itself consists of the 8 currencies that you see in Figure 1. Each of the currencies are paired off with the other 7 currencies that form 28 currency pairs we trade in these 8 currency groups.

The following C# WPF/XAML user interface mockup on .NET v4.5 was designed by me as a prior approach to solving this problem:

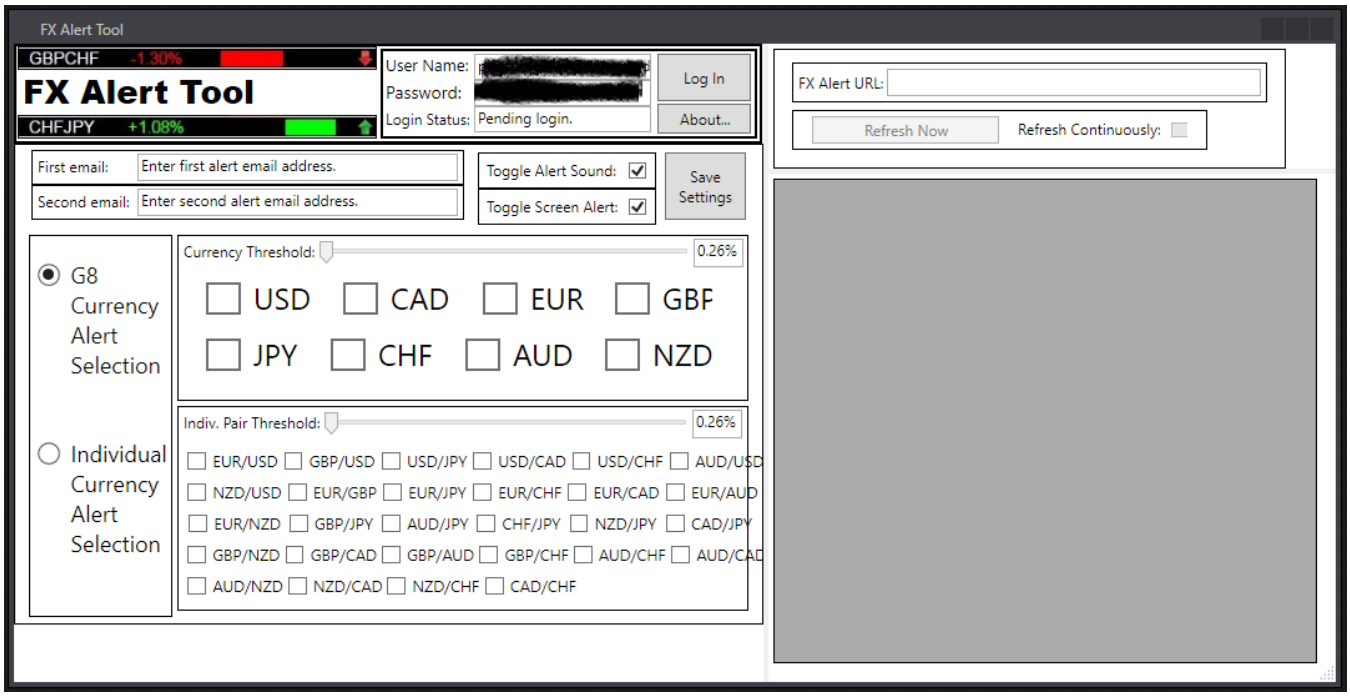

Figure 1 Prior UI Mockup for Configuration Page

As you can see, many elements of this user interface should be reused going into HTML5, CSS3, and JavaScript.

1. For sure we need the login control with the option to save the login.
2. We'll need a drop-down box for selection of the website we reading data from, and has one item for now until later versions.
3. Also let's plan on an Enable/Disable control so we know for sure when the engine is off and not harvesting from the server side execution page.
4. We will need the ability to select a currency group, an individual currency pair, or select pairs in a group. This should later be saved also with the associated credentials and a unique configuration ID for selecting or deleting that configuration for later runs.
5. The data Currency Threshold and Indiv. Pair Threshold slider bars are still the gold standard, and are needed in determining strength or weakness. 0.26% is the minimum acceleration value on a session.
6. The top left graphic will most likely get reused.
7. The email and alert settings bar should be moved to a different place
8. The right side section of this user interface mockup used for viewing a separate URL should be removed, and replaced with an Event Log Page that we'll need per the next section.

Rubric Use Case – View Last Harvest Data (for test assistance) and Event Log Page

This web page will need to be kept simple because it's only purpose will be to function as an Event(s) Log Page starting with the most recent data received on a per-configuration basis separated and highlighted for very clear reading along with all other configuration related events that occurred per session/day. It should have a scroll bar for more easy reading and event line selecting and copying.

The only technical challenge currently foreseeable on this module is if the login or active session logic were to stop working – hence regularly checking this will be important. The website the module is interfacing with may become part of the secure credentials data table later in the data module.

Test Module - Interface with FX website

Testing this module should be straight forward, but actual test criteria for this module is not yet determined as these requirements still may change depending on what module is imported to my website interface module to do this.

Module – Html (well-formed/malformed) data harvest

This module will be responsible for doing the actual heavy lifting of gathering the html parse-able data through the use of the Beautiful Soup XML/HTML python module or similar lxml python module, or language api. My use of actual xslt/html parsing is rusty at best, but I do remember well that certain tools provided by .NET or other open source tool are able to accept an html document of repeated document object model (DOM) embedded sheet data that repeats over and over again in the same pattern – can be gathered from the embedded html repeated markup through tag/element matching and extract the readable data to another form such as an internal data structure we have gone over like List or Dictionary in python. The data could also simply be written out to another source for safe keeping.

The tool I remembered using last was at the command line that accepted a sample XML or HTML marked up language document, and produced the associated best rules-matched stylesheet transform document such as >xsd.exe html_data_sample.html then outputted the document html_data_sample.xlt as the transform rules document.

Data Harvesting Design (Specific Plan)

Thanks to your exceptional first tutorial of the web inspector tool, we have these captures to look forward to:

1. From the http headers request URL ending in index.php?qc=22, this is the main data source we have the following screen capture under the Network→Body Http tabs path (saved file is primary_data_sample.html):

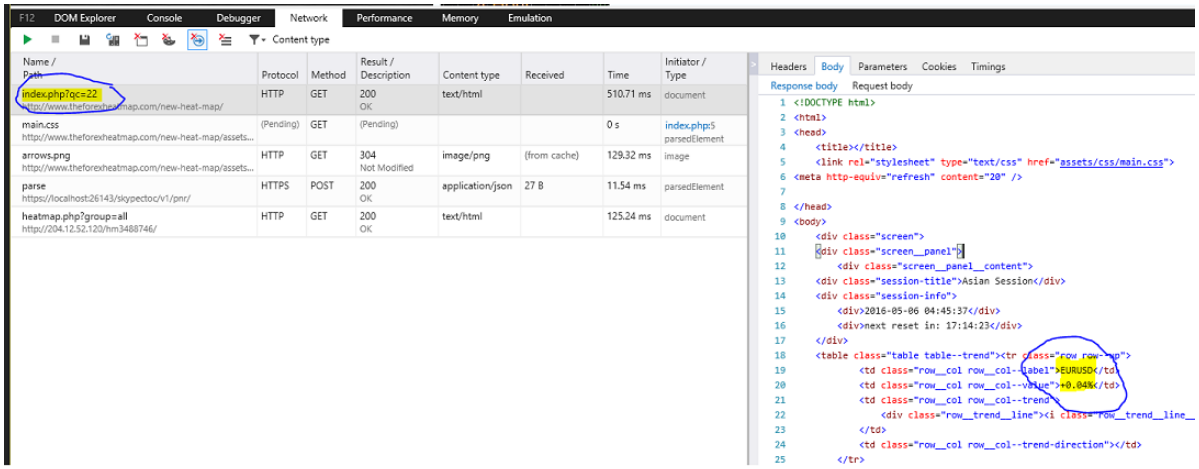

Figure 2 Example of Primary Data Source

2. From the http headers request URL ending in heatmap.php?group=all, this is the backup data source we have the following screen capture under the Network→Body Http tabs path (saved file is backup_data_sample.html):

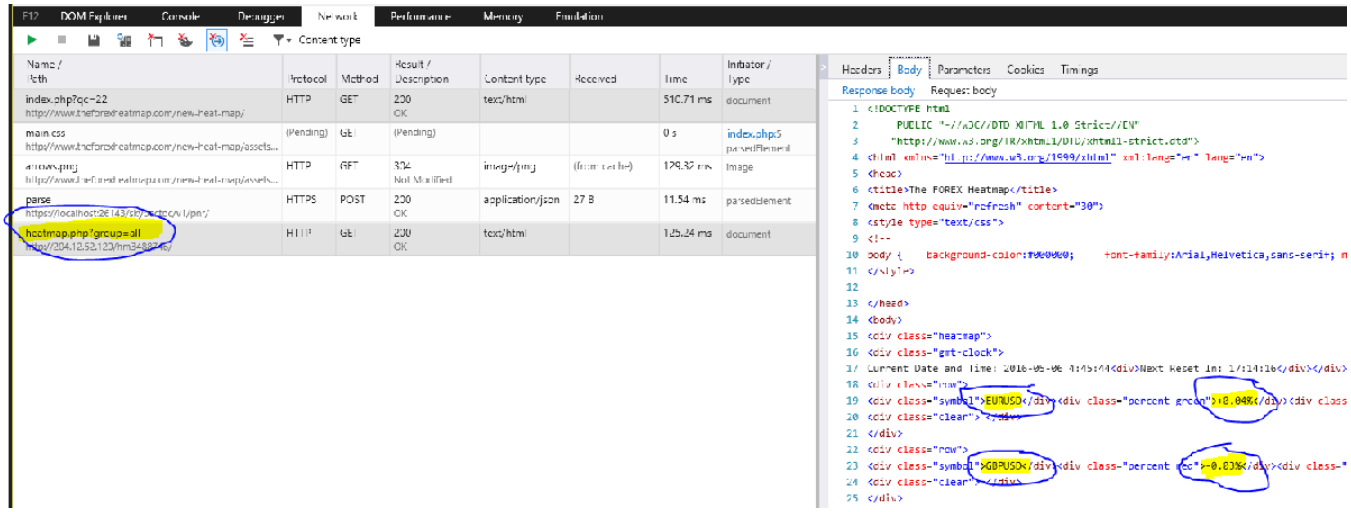

Figure 3 Example of Backup Data Source

Furthermore with the web inspector tool: I made the following observations about the appearance of this data in a live session that may or may not be relevant, and needs help disseminating and updating in further screen/network captures:

With this tool and also the Chrome web inspector tool, I have found that I need to be running a javascript fuction against a session/login cookie only website URL as get_v4.php or index.php?qc=22 with the following java script function code to get the primary data source:

```
<script>
var _x = function(
{
var form = document.getElementById("acForm");
form.target = "v4Heatmap";
form.submit();
form.parentNode.removeChild(form);
var script = document.getElementById("xs");
script.parentNode.removeChild(script);
}();
</script>
```

The second heatmap v2 backup data source has this link as the current 3-month long valid URL http://204.12.52.120/hm9485488/ with the java script function code:

```
<script language="JavaScript">
document.write('<iframe src="http://204.12.52.120/hm9485488/" height="1300"
width="820" frameborder="0" scrolling="auto" target="_blank">
</iframe>');
</script>
```

The technical challenge for me in this module is that transforming the data to a form I can use in python must be solved – how this is to take place is part of this module's challenge. For this challenge – I would still prefer still using Xslt/Xml once I can confirm we have a secure and stable pipe line.

Test Module – Xslt/Xml data harvest module

The testing for this module is pretty clear also – access the website, gather the data based on configuration of what data to look for, and verify the expected data is in a form we can use with python and its associated modules.

As with the last module – this preliminary test plan is subject to change if the requirements or features of this module changes.

Module - Data credentials and configuration

This module for sure will have to interface with Django API data classes for specific storage of:

1. My secure credentials,
2. the website that is the target of receiving data from,
3. the configuration of the data harvesting module about what and how to receive the data, and finally
4. the configuration of what and how the trade execution module will perform trades if that is implemented here.

The only current technical challenges with this module is overcoming any unforeseen problems in actually working with and learning the Django data access API layer through the classes it creates for me to work with.

Data Model for MVP sprint

The beginning data model/schema starts out simple here:

1. Save a configurationID for easy access and modification to and from the UI to modify or delete from the database.
2. Save the credentials from the Login control saved from the UI.

3. Save the selected website from which we're reading data.
4. Save the configuration overlay from the UI of what is currently running.

Test Module - Data credentials and configuration module

The test criteria of this module is if the data persisted to the database is actually there, and can be called on through the Django API data classes at any time.

Module/Test Module - Data harvest configuration web page

This is one of two confirmed web pages of a website I will need in order to allow the user of this system to specifically configure the data harvest execution module. Please the power point slide to show how this interaction should take place. How the configuration is and will take place is still to be determined.

The only foreseeable challenge of doing this webpage module is learning and working with the web layer of Django page construction and piping or hosting.

The test criteria of this module is if a configuration to the data harvest engine succeeds in getting proper data. This test criteria is also subject to change as this criteria for the configuration changes.

Follow-Up Work (Non-deliverables for Minimum Viable Product sprint):

Module/Test Module - Trade module configuration page

This module is the second of two confirmed configuration pages.

It will be used to configure that trade execution engine.

How and what defines a trade execution profile is still to be determined.

The only current foreseeable challenge for this module is the trade execution engine module must be constructed in order to properly test this webpage module.

Rubric Use Case – Trade Configuration and Turn Trading On/Off Page

To be determined.

Rubric Use Case – View Last Trade Event and Trading Event Log Page

To be determined.

The test criteria for this web page module is not yet determined.

Module - Email template configuration html notice

This module is simply a default set of email template html pages that will be used to create the emails notices.

Testing for this module is that the email generated need to look good enough to notice.

Module/Test Module - Email alert notice generation

This module is responsible for sending an alert email when an important event happens for the data that was observed.

Exactly what event are to be looked for, and what defines an event is still to be determined.

The test criteria for this module is to be determined also when event definitions are determined.

Module - Trade Execution Program/API

This module will be responsible for the actual trade execution of the project.

The details of what this module actually will be is the current challenge for this module because the language of MQL4 must be learned in order to do this module.

Another name for program like this are called “expert advisors”, I find that only comforting if I made it.

Test Module - Trade Execution Program/API

The test criteria for this module is if under a certain trade configuration that a trade actually succeed to a paper trade otherwise known as a fake trading account.