{intro text}

Basic workflow description

Step-1, General UI Orientation
Step-2, Select endpoint function
Step-3, Load associated WSDL version
Step-4, Configure function Request/Response parameters
Step-5, Configure replay parameters
Step-6, Execute event timeline

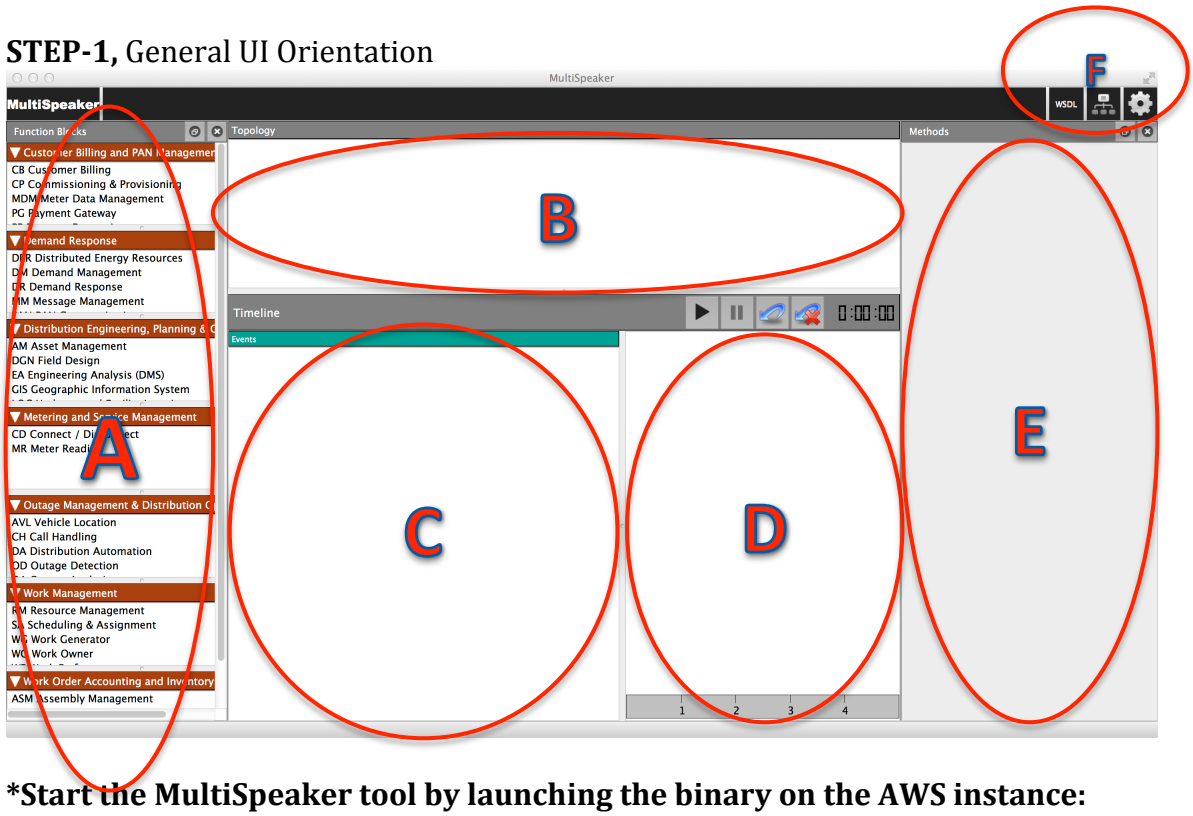
Current capability
{description}

{graphic}
Functional Framework Logical Arrangement

Process Flow
{description}

{graphic}
Process flow

STEP-1, General UI Orientation



***Start the MultiSpeaker tool by launching the binary on the AWS instance:**

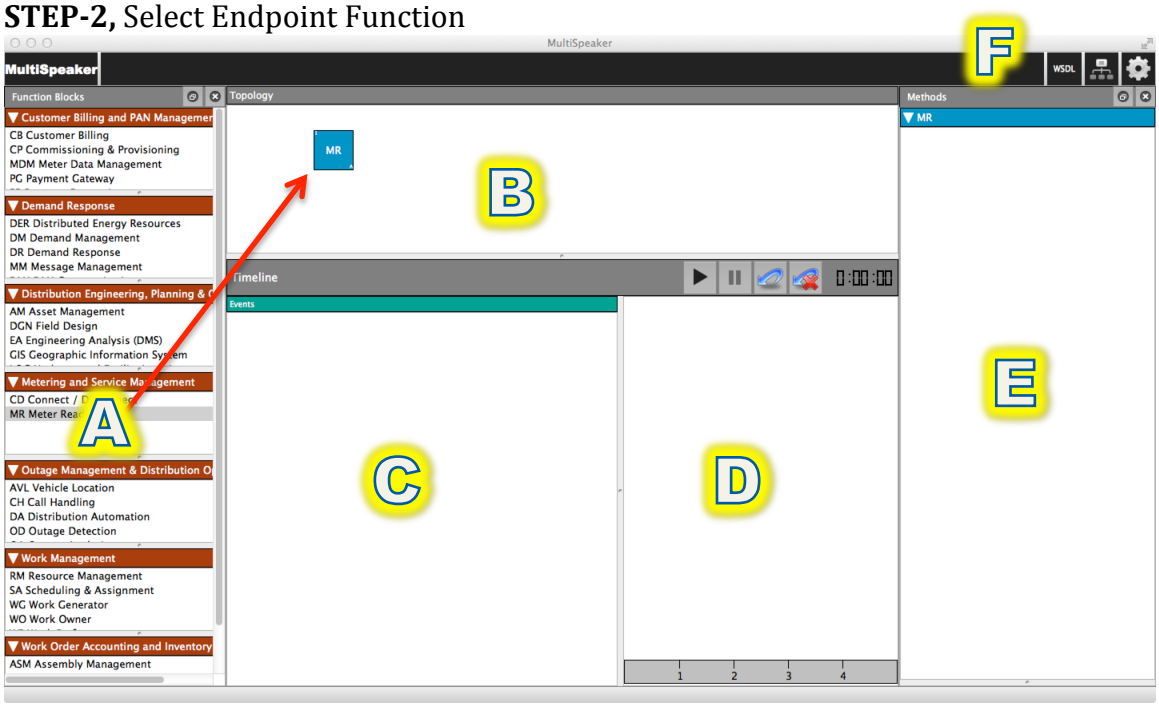
/home/multispeaker/./MultiSpeaker &

I'd put it in the background because you'll need to start the TCP catcher (MultiSpeakerServer) later in the procedure. You don't need to be root.

General orientation to the UI:

A. "Function Blocks" are a list (by functional use) of the endpoints.
B. "Topology" is a region to drag an endpoint from the function block menu (A) to. It will result in adding the selected endpoint. {image}
C. "Timeline" region will list Multispeak generated "events" in order of execution. You can double-click on these to modify attributes of an individual entry, such as payload attributes, or timing of when the event is fired to the replay generator.
D. "Flag" region. There will be a flag {image} for each event in the timeline. (this is relative currently, and isn't operational. Later in the instruction you'll see that you drag an event from region E to D in a time-relevant location to create an event.
E. Multispeak endpoint functions are listed here. You select and drag these to region-D.
F. These are controls for the overall UI.
    a. The WSDL selection tool
    b. The MiniNet SDN configuration tool
    c. Version/Help (help is mostly mouse-over at this point)

STEP-2, Select Endpoint Function

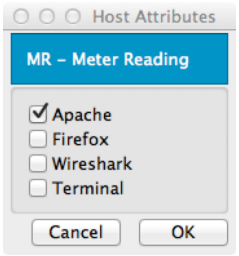


Selecting an endpoint from the Function Block region.

A. Click on a desired endpoint from the list (e.g., MR Meter Reading)
B. Drag the endpoint to the topology region B

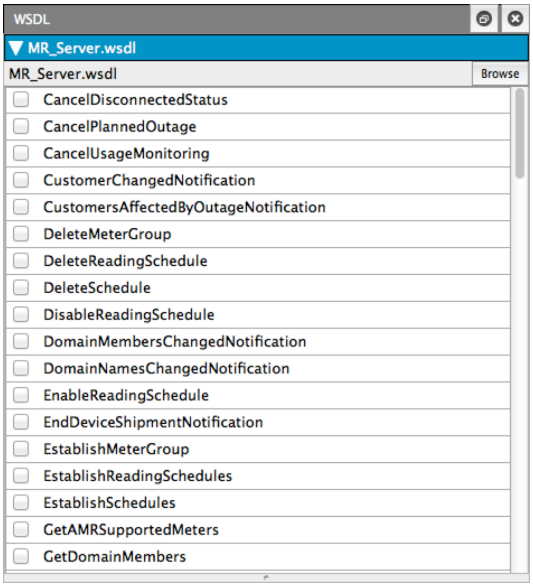
*You'll notice the MR endpoint in region B and an associated MR in the methods list in region E.

*If you double-click on the MR box on the left you'll see a pop-up that will let you select some additional features. *This was intended for the initial test framework and we haven't decided what we're going to do with this yet, so it is an undocumented feature at this point!* It used to direct the MiniNet environment to load additional services for each endpoint as a convenience during our initial testing. (This pop-up looks like this:



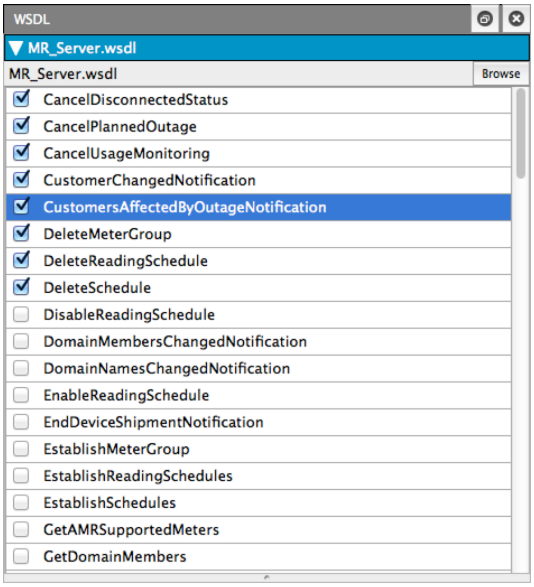
STEP-3, Load associated WSDL version

A. By clicking the "WSDL" button from region F (hard to see but upper right in the UI), you'll navigate to the desired version (3,4,5) of the WSDL for the MR endpoint. This graphic represents the MR Version-3 endpoint functions in the associated Version-3 WSDL.
*I've located the WSDL version sets (all of them) on the AWS directory /home/multispeaker/MultiSpeak-All-Versions-FULL location so start there then drill down into the WSDL version you'd like the endpoint to reference for its' function listing.

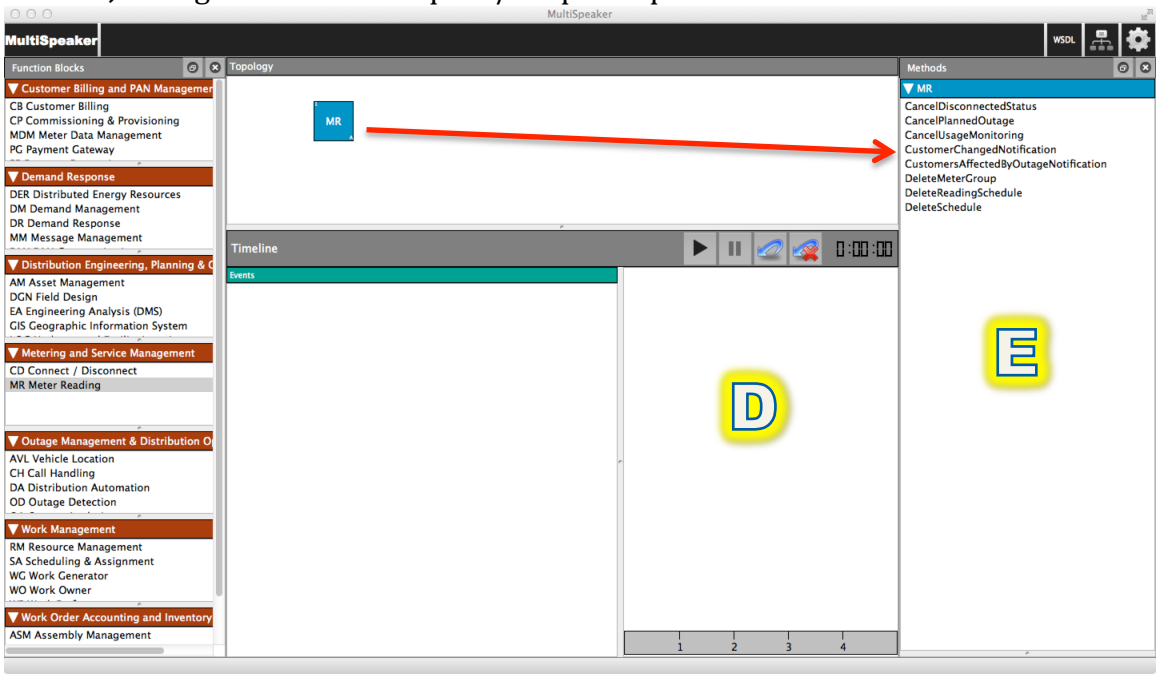


B. Once you've selected the desired reference WSDL you can select the functions that you'll be actively using. You can go back later and select/deselect from the active list in region E if desired, just click the WSDL button again and the pop-up will present itself again.

C. I've selected the first few MR endpoint functions as depicted.



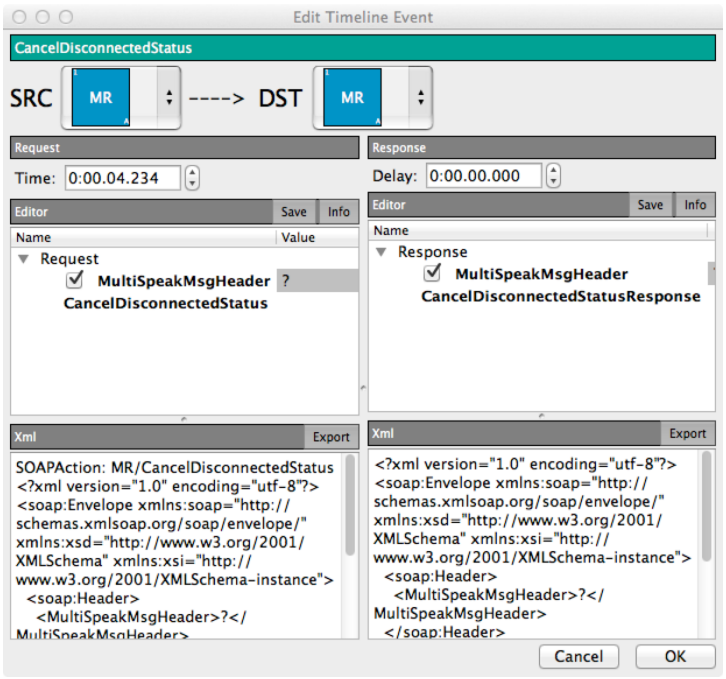
STEP-4, Configure function Request/Response parameters



Step-4 Graphic

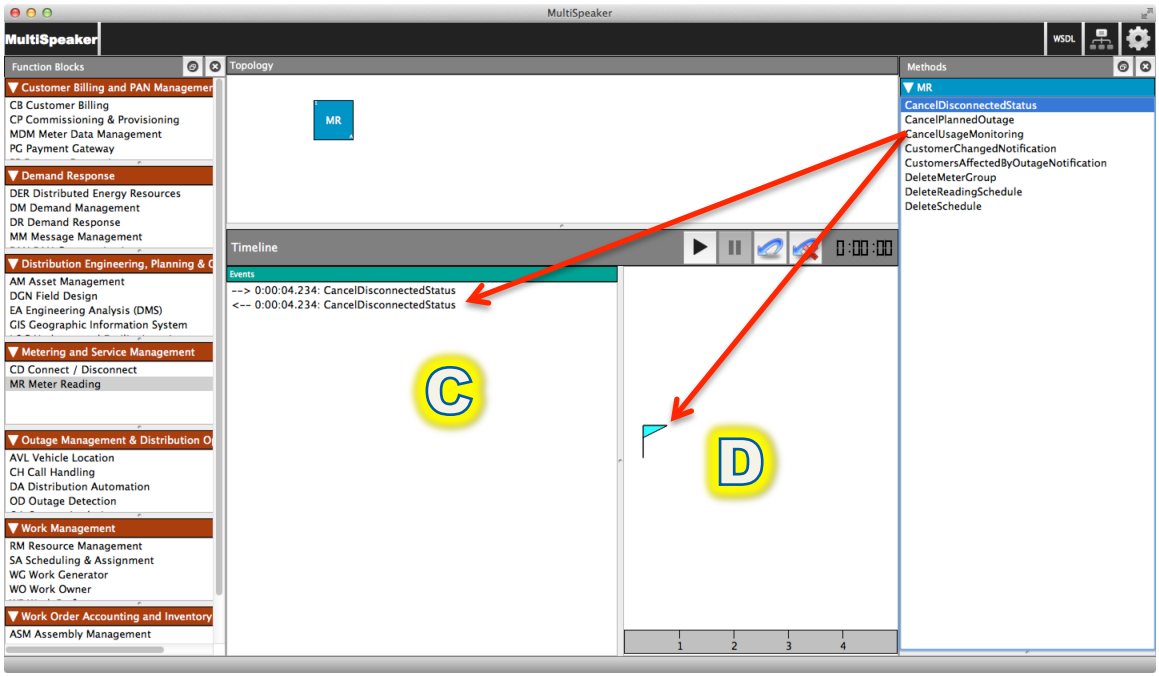
As a result of selecting the WSDL functions you now have a list of them in region E. Now we're going to drag the desired function to the timeline flag region D. Select any of them you like and as many as you like as this will simply create more events. *An "event" consists of a Request/Response pair of communications messages so don't fret about seeing two individual event messages and two flags for each selected event.

I selected the "CancelDisconnectedStatus" Function and dropped it to the left-most area of the flag region D. The *Edit Timeline Event* window popped up. This will allow you to configure **all** of the desired event parameters.
**MultiSpeak Version-3 is a bit of a challenge because there were so many hard-coded constraints and so little payload configurability when compared to Version-5.*



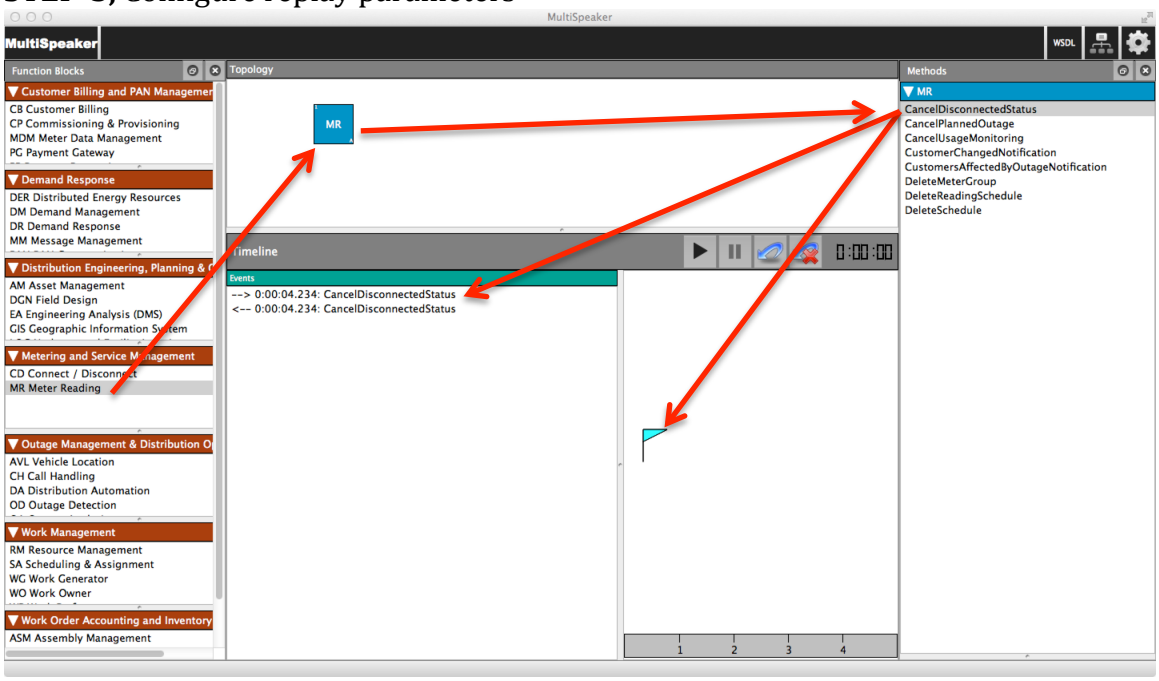
For this simple example, you'll see that the SOAP payload (Version-3) is really only a signal to some application actor (real-life utility application).

Here is the resulting configuration for one event that I have selected.



I dragged out the CancelDisconnectedStatus function to where the flag is (there are two of them and you can't see the second one because it was configured with no delay in the preceding pop-up window) in D and after configuration of the payload the events (request and response) show up in region C.
**You can double-click on the region C event and the Edit Timeline Event window will reappear where you can modify any of the event parameters or payload.*

STEP-5, Configure replay parameters

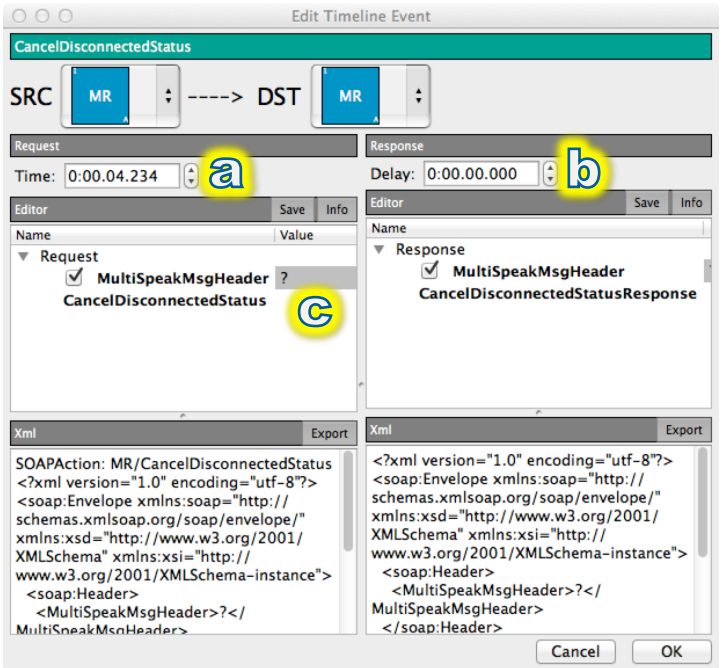


Summary: The UI depicts a single event (with both request and response payloads). From left to right *MR Meter* Reading was dragged and created the *MR* block, then after selecting the desired WSDL and (subsequent functions) you configured a request/response event which resulted in the flags and listings in the timeline region.

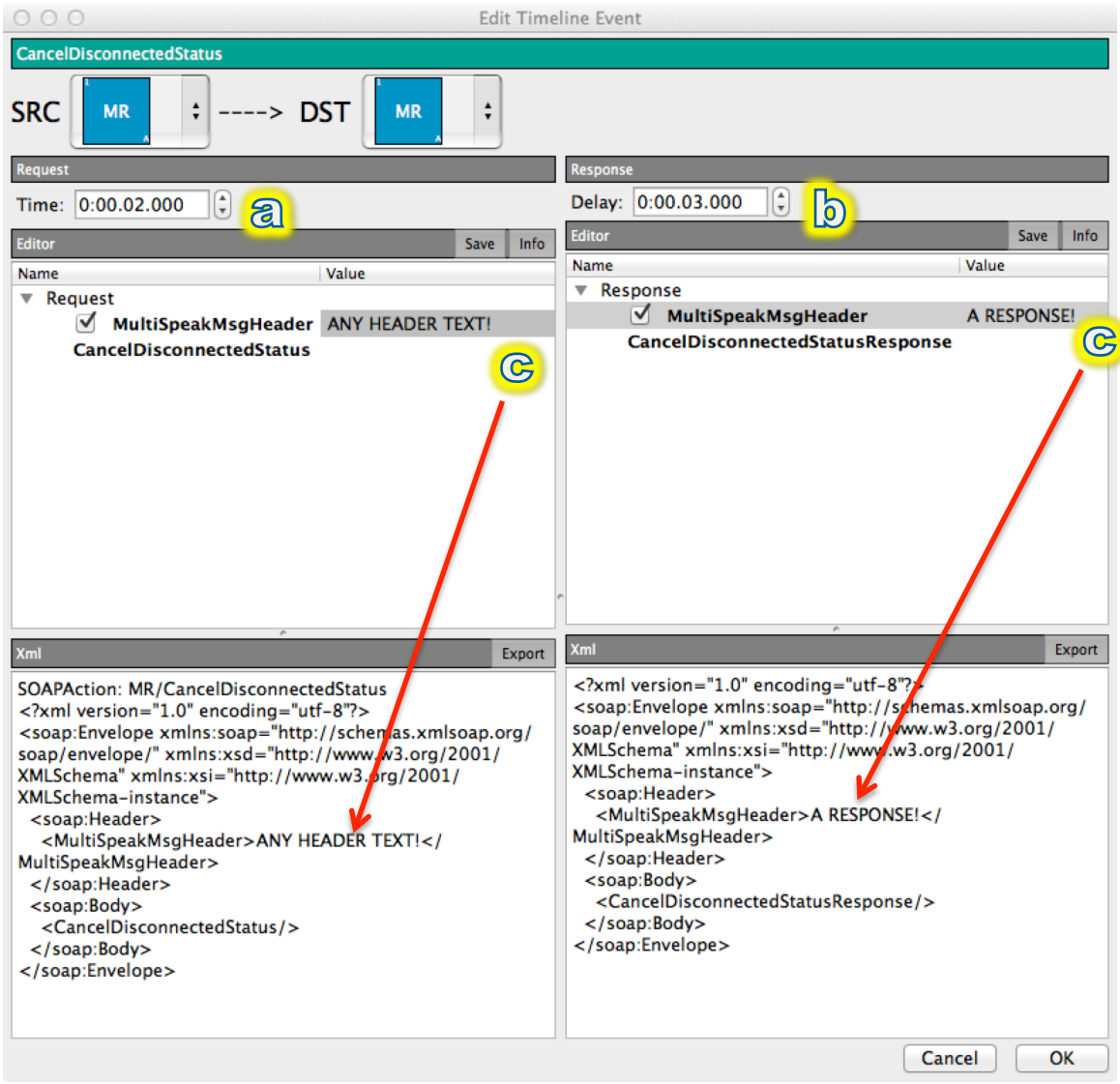
Before I show you how to start the event sequencer I'll introduce a couple of additional features that will help you further define (on confine) the event.

First, double-click the first event (--> 0:00:04.234: CancelDisconnectedStatus). You'll see this pop-up.

We're going to:
a) modify the time that the request msg fires
b) add a delay to the response msg, and
c) add a value to the SOAP payload.



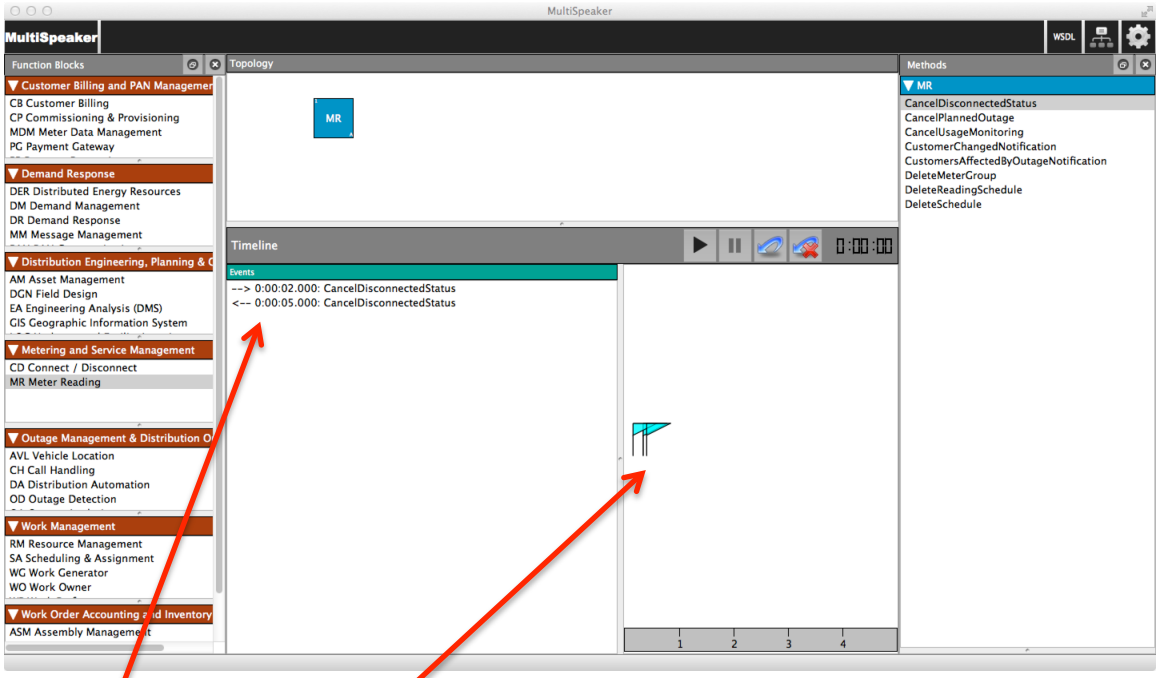
Here are the results of our modifications!



a) I set the event trigger to 2-seconds after start.
b) I set the Delay time to 3-seconds (…after request message fires)
c) I modified the SOAP payload "?" to "ANY HEADER TEXT!"
   I modified the SOAP payload "?" to "A RESPONSE"

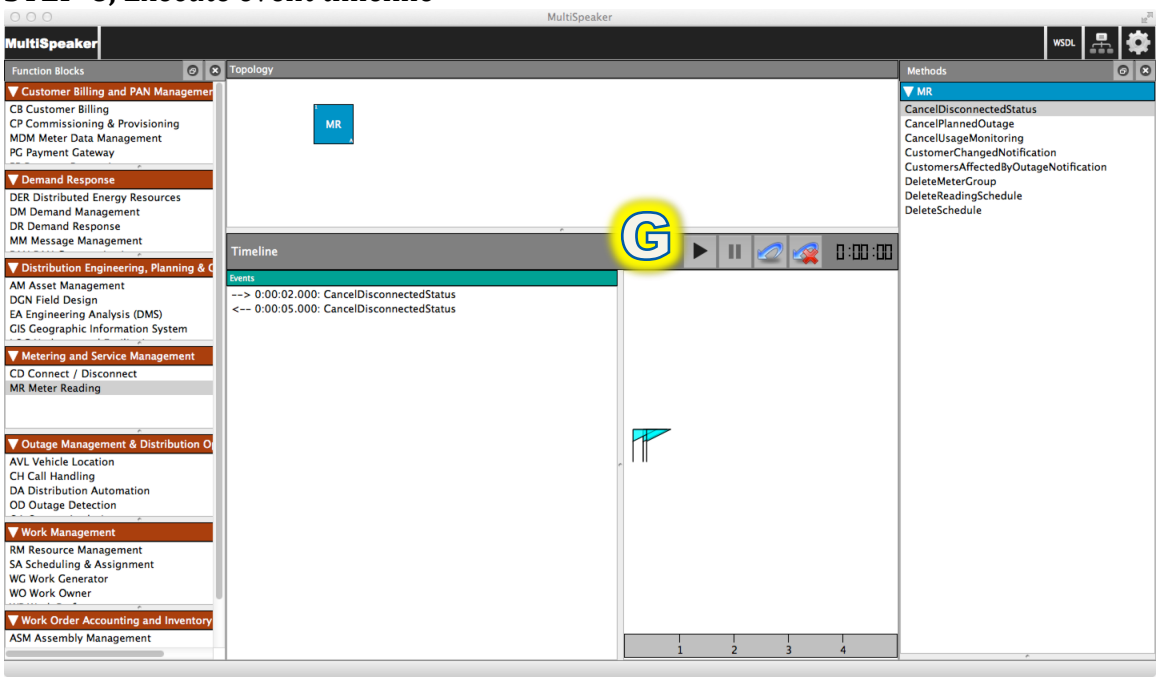
The arrow is pointing to an automatic update to the XML window reflecting the change to the payload.

The UI now reflects the changes and I'm ready to move to the final step enabling the timed event sequence.



Timing updated, and flags offset reflecting relative delay in response msg.

STEP-6, Execute event timeline



Enabling the event sequencer by using the controls in region G (left to right)

- **Play** = start the sequencer
- **Pause** timer = adjust any events that have NOT been fired
- **Reset timer** = begin experiment from beginning
- **Reset Timeline** = (**deletes ALL events!**)

**We haven't programmed the save function yet! Once we do this you won't have to worry about deleting the timeline events that you've meticulously configured!*

IMPORTANT!

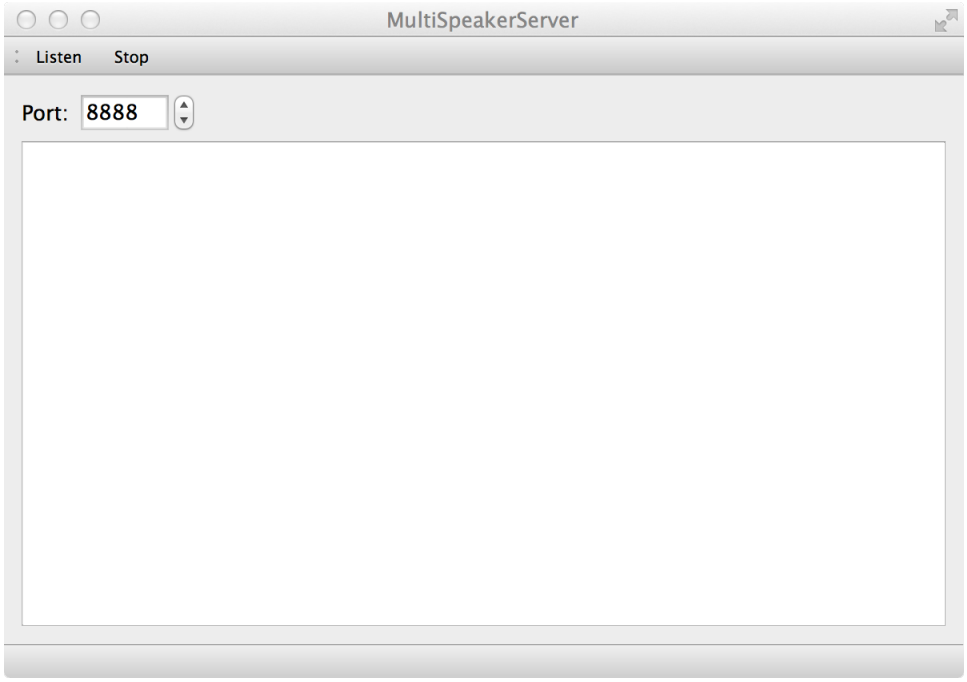
You need to use the TCP tool below configured correctly to catch the event! This tool was created to *test* the test tool. ☺ Will's MiniNet playback framework will be the real target for the TCP connection from the MultiSpeaker tool. Until this is implemented (very soon!) you won't be able to use the capturer to catch, parse and record the MultiSpeak data. (You knew that, but I'm reminding you! Heh)

For now you start the TCP server (catcher):

***Start the MultiSpeakerServer tool by launching the binary on the AWS instance:**

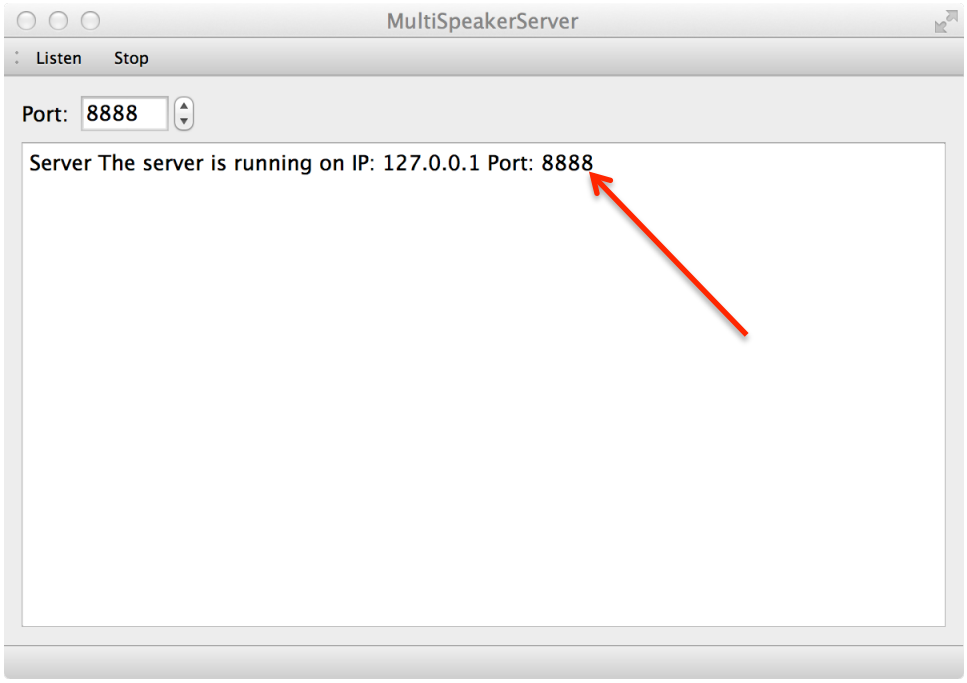
/home/multispeaker/./MultiSpeakerServer &

I'd put it in the background because you'll need to start the TCP catcher (MultiSpeakerServer) later in the procedure. You don't need to be root.



Here is the TCP catcher (MultiSpeakerServer) application. It is very crude on purpose! ;)

Configure the desired port and click on the "Listen" button. You'll now be listening for a TCP connection from MultiSpeaker on that port.

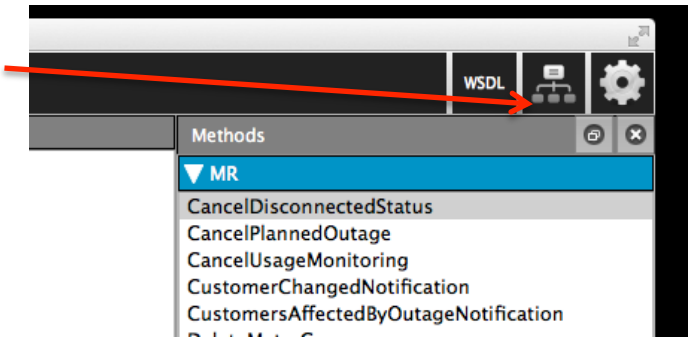


Now configure the MultiSpeaker TCP connection.

In the upper right corner of the UI click on the network icon.

This will display the MiniNet configuration screen.

*It does NOT launch MN at this time!

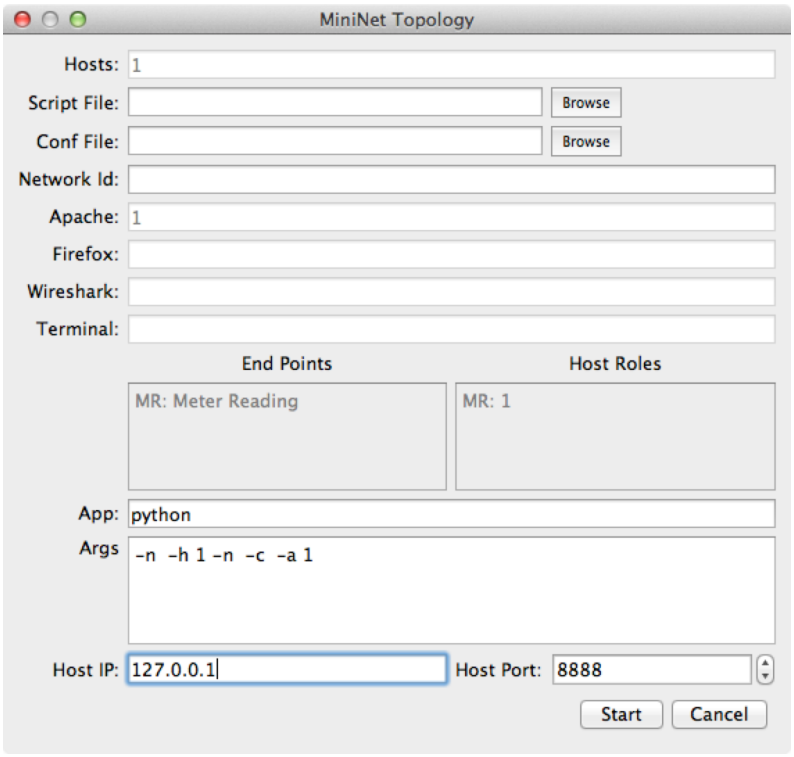


At this time you only need be concerned with configuration of the "HostIP:" setting.

Set the IP to the local loop back 127.0.0.1 for now.

You can actually use real IPs on a physical network if you like! I'm only concerned with the output not the network fidelity at this point.

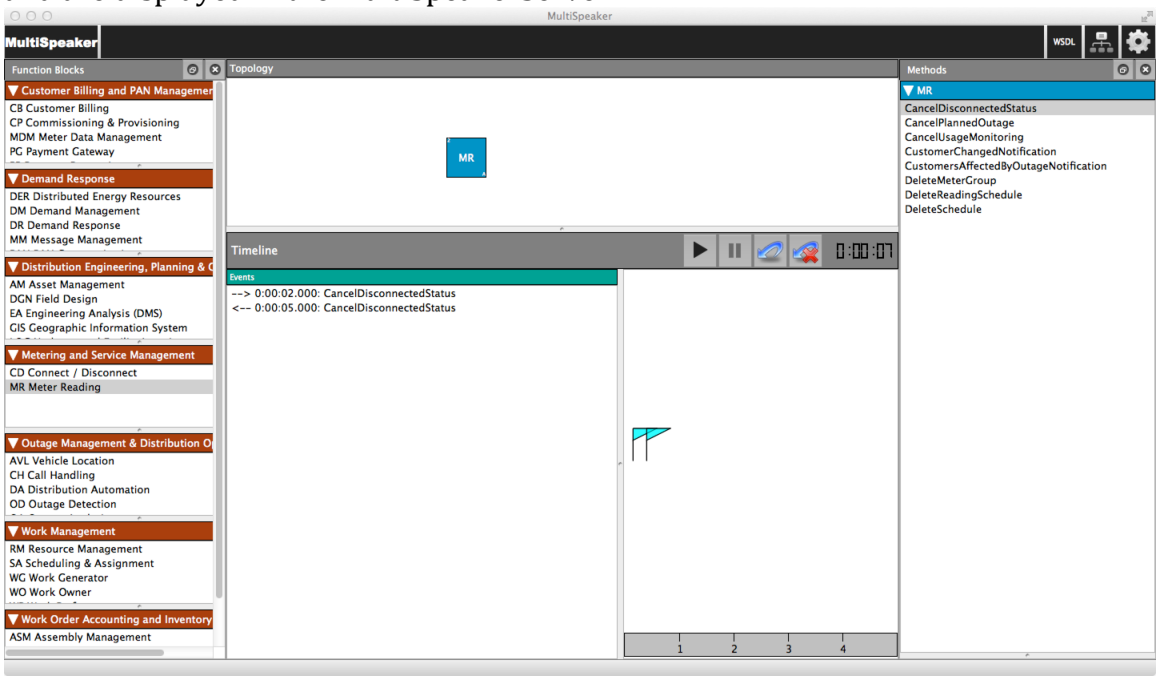
Hit the "Cancel" button (don't worry, it saves your setting!)



Now you are ready to begin!

Hit Play, Pause, change stuff, add endpoints, mix versions…knock yourself out!

After clicking the play button the events fire at their scheduled time on the timeline and are displayed in the MultiSpeakerServer.



The MultiSpeakerServer is a simply TCP capture of the event stream being sent to the MiniNet replayer framework.

**There are a few things we've already discussed on the XML SOAP output. I've identified them in the next page and are a priority for development beginning 12/8.*

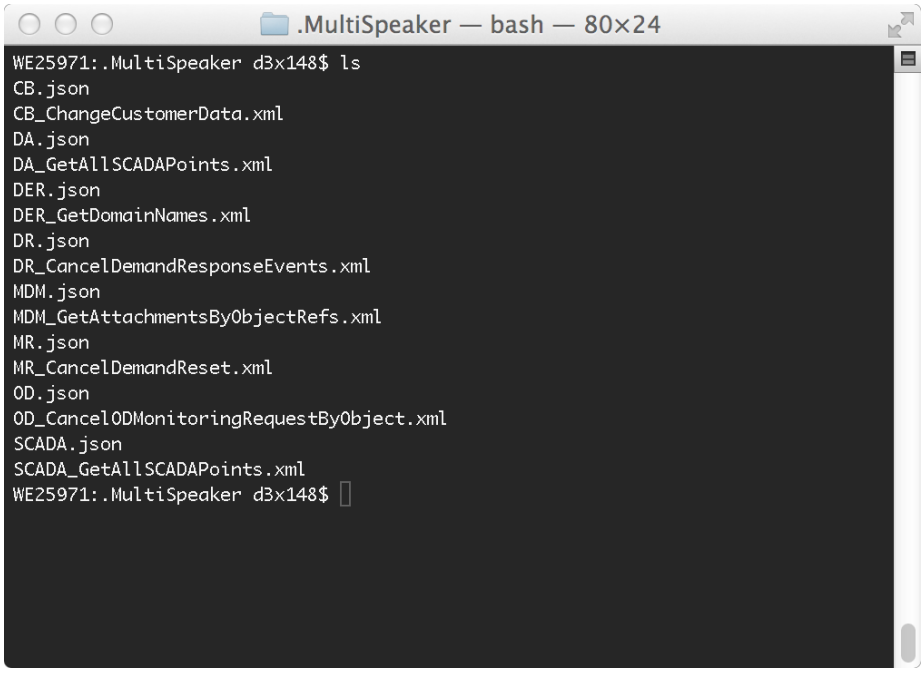
The format for this output is constructed to identify a SRC and DEST node for the message exchange. For our test it is run locally on the 127.0.0.1 interface. I've configured both client and server to run on a physical network IP and it worked fine. I haven't pursued this further because there are a couple of additional XML/SOAP details to work out before the PCAP would have any value to the team (i.e., ingest via the Essense capturer and parsing functions).

We'll add these deficiencies shortly. They are:

a) Add additional MS schema references.
b) Add the XML prefixes throughout the payload
c) Add feature to insert correct MS version reference when the reference WSDL is selected.

After the experiment is complete, there are a few idiosyncrasies we've identified and will be refactored in future revisions.

First, there is a hidden file ".MultiSpeaker". It contains the configurations of the WSDL functions that have been selected. If you want to start over on a specific endpoint you'll need to deleted it manually right now. You can delete all of them if you want a clean configuration. Otherwise, if you've already configured/selected functions from say an MR endpoint, and you drag that back out to the Topology region, it will already contain the functions you'd already previously selected.



This screenshot is from my MAC running MultiSpeaker. The .MultiSpeaker hidded directory on the AWS instance is in the /home/pcraig directory (because that is the user that owns the MultiSpeaker application.

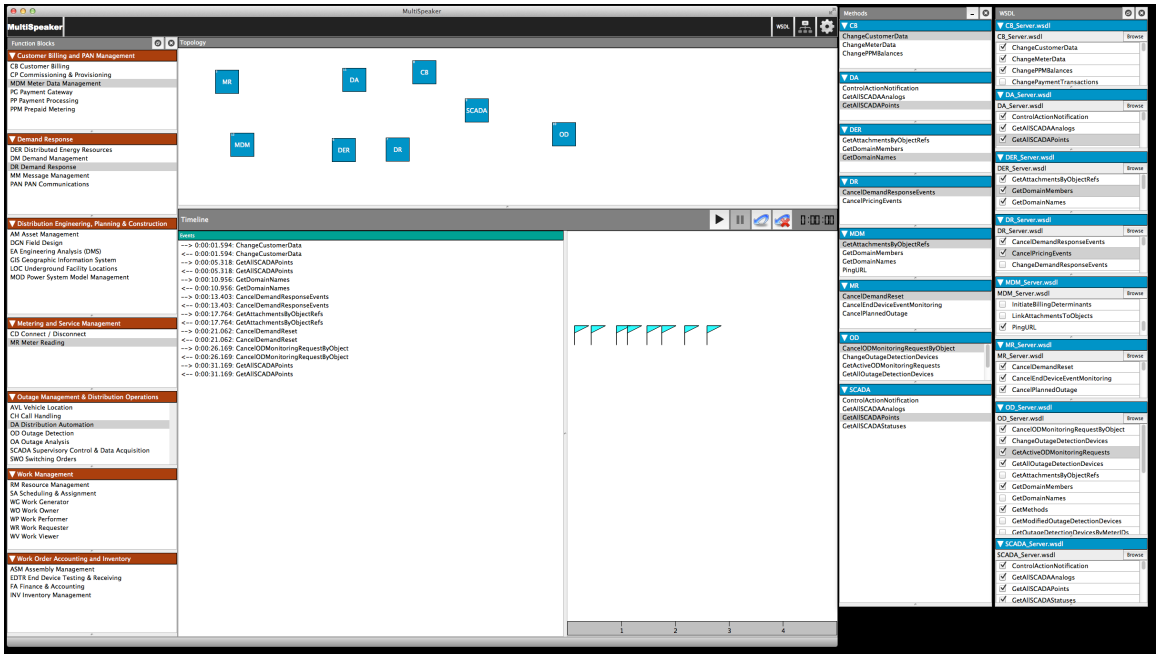
For example, the file **CB.json** contains:

```
{
  "ChangeCustomerData": true,
  "ChangeMeterData": true,
  "ChangePPMBalances": true
}
```

Which indicates that those endpoint functions were selected by the user and are maintained in the json configuration reference file.

Future Revisions

A more complex experiment can be easily created. Below is this configuration contains 8 endpoints and I've selected a single function from each one that has created a pre-scheduled event (request/response message pair).



Notice that I've also undocked the WSDL and methods panes so I could have more room on the operational MultiSpeaker screen to monitor and control the flow of the experiment. These are typical windows UI features and we plan to add more as we progress, but are focused on function not form at the moment.

Immediate coding effort:

a) Add additional MS schema references.
b) Add the XML prefixes throughout the payload
c) Add feature to insert correct MS version reference when the reference WSDL is selected.

Next generation coding effort:

a) Correct minor bugs in current features
    a. Add back in the live-look of the endpoints communicating with each other when an event is fired.
    b. Construct an more effect WSDL/XSD file management capability
    c. Add project save features. (Allows a user to configure a test-sequence offline and then load/run at will.
    d. The function region at the far left will more accurately represent the endpoint version being used. In some cases there may be field installations of multiple MS versions. We understand this and are

planning to adjust the UI for this capability that will be easy for the user to configure a multi-version test sequence.