CREATING A SCENARIO FILE FOR UXAS USING AMASE

This walkthrough provides “quick and dirty” instructions on creating messages that set up a scenario for UxAS and AMASE using AMASE’s Setup Tool.

Both AMASE and UxAS are able to read in LMCP messages from XML files. While AMASE and UxAS are also able to receive LMCP messages from various internal or external software services, it is often more convenient to read in a fixed set of base messages that define a scenario from a file. Such messages generally define things like air vehicle configurations; initial air vehicle states; keep-in and keep-out zones; points, lines, and areas of interest; known tasks that might be performed; and automation requests to perform some set of known tasks. The AMASE Setup Tool makes it easier to create the messages that define a particular scenario.

One difficulty is that AMASE and UxAS have diverged somewhat since the AMASE Setup Tool was originally created. UxAS provides automation capabilities that AMASE does not, and so it uses a slightly different and larger set of LMCP messages to describe tasks to be performed and to make automation requests to perform the tasks. However, the AMASE Setup Tool is still useful since it is far easier to set (lat, long) coordinates for air vehicles, points, lines, and regions on an interactive map than it is to set them through a text editor. The AMASE Setup Tool will therefore get you the “80%” solution for UxAS.

1.0 CREATING A SCENARIO IN THE AMASE SETUP TOOL

In AMASE, messages that initialize a scenario are often collected into a single “scenario file.” While you could create these messages manually, AMASE includes the AMASE Setup Tool to help create these messages. The script to run this tool is SetupTool.sh. Depending on whether you use *nix or Windows, it can be found at

OpenAMASE/OpenAMASE/run/linux/SetupTool.sh

OpenAMASE/OpenAMASE/run/windows/SetupTool.sh

Further instructions assume you’re using the linux version. When you execute this script, an AMASE GUI should load with a Map Display, an Event Editor, and a Scenario Validator. The Map Display has icons along the left that allow you to place aircraft and other generic mobile entities. You can also zoom in on the map to focus on a smaller region.

Let us start by creating some aircraft; point, line, and area searches; and keep-out and keep-in zones. An example is shown below in Figure 1. Aircraft were placed using the blue triangle icon on the left of the map, search tasks with the green icons, the keep-in zone with one of the yellow icons, and keep-out zones with the red icons. **Note that when creating lines and polygons, you have to double-click the last point to indicate that you've finished drawing.**

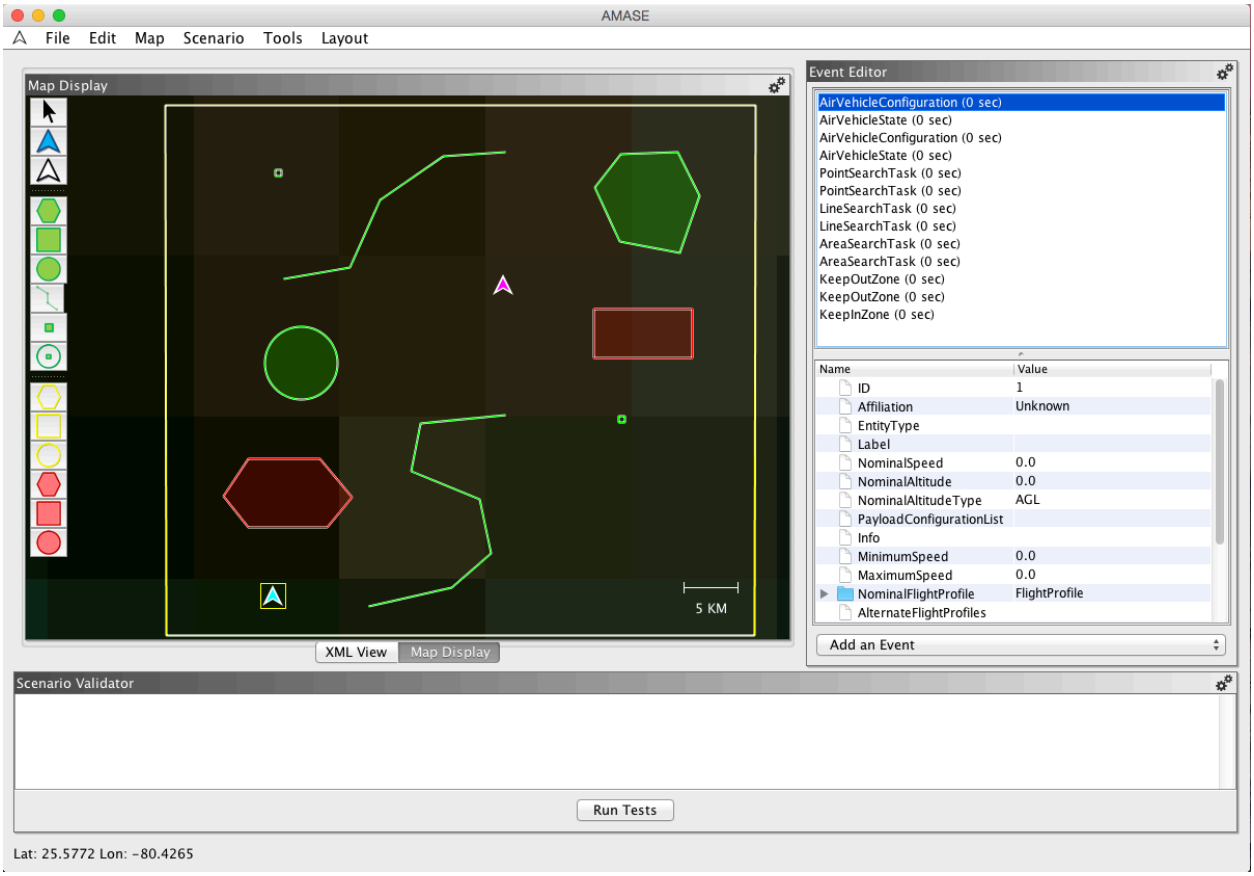

Figure 1: An example scenario with two air vehicles, some keep-out and keep-in zones, and several point, line, and area searches.

Note that every time you create something, an element shows up in the Event Editor. These elements correspond to LMCP messages. If you click one, you can edit its fields manually. If you right-click a message element, you can set the time the message will be sent[1], delete the message, copy or cut it, or have it pop up in a separate editing

[1] UxAS has its own method for setting message time, so this is not generally set in the AMASE Setup Tool.

panel. If you left-click a message element, its contents will appear in lower Event Editor panel. At the bottom of the Event Editor is a drop-down menu that contains all LMCP messages that have been built with AMASE. Note that AMASE generally only interprets messages from the CMASI message set. However, if you've built both AMASE and UxAS with the same LMCP message sets (aka Message Data Models or MDMs), then this drop-down menu will list all messages interpreted by UxAS.

If you run the Scenario Validator by clicking the "Run Tests" button, it will report numerous issues with the AirVehicleConfiguration and AirVehicleState messages. The AMASE Setup Tool does not provide valid default values for these, so to fix errors, you will need to go back and edit these messages. Reasonable values for fields that must be edited for **both** of the AirVehicleConfiguration messages are:

- NominalSpeed: 22.0
- NominalAltitude: 500.0
- MinimumSpeed: 20.0
- MaximumSpeed: 35.0
- NominalFlightProfile/FlightProfile/Airspeed: 22.0
- NominalFlightProfile/FlightProfile/MaxBankAngle: 20.0
- NominalFlightProfile/FlightProfile/EnergyRate: 0.0003

In case you want the vehicles to change altitudes during the scenario, you should also define a FlightProfile under AlternateFlightProfiles for each vehicle. Reasonable values for fields that must be edited for **both** of the AirVehicleConfiguration messages are:

- AlternateFlightProfiles/FlightProfile/Airspeed: 22.0
- AlternateFlightProfiles/FlightProfile/VerticalSpeed: 5.0
- AlternateFlightProfiles/FlightProfile/MaxBankAngle: 20.0
- AlternateFlightProfiles/FlightProfile/EnergyRate: 0.0003

For **both** of the AirVehicleState messages, most of the warnings can be ignored, but you do need to set the energy level and altitude:

- EnergyAvailable: 100.0
- Location/Altitude: 500.0

The remaining warnings can mostly be ignored, except that UxAS needs information about UAV sensor payloads for several tasks of interest. To **both** of the AirVehicleConfiguration messages, you will need to add a GimbalConfiguration and CameraConfiguration by right-clicking the PayloadConfigurationList field and adding one of each of these, as shown in Figure 2. **Note that for any message field that is a**

list, you must choose Add from the right-click menu to add an element, even if the elements are primitive types like integers.

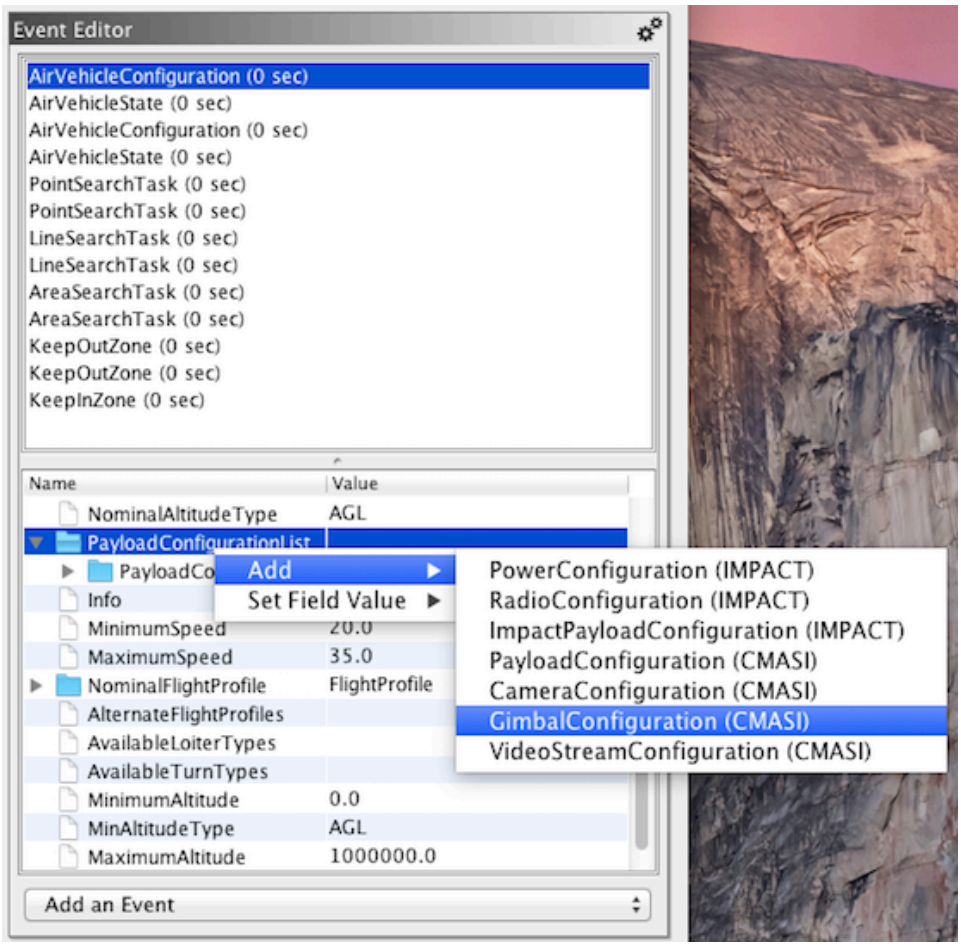

Figure 2: Adding CameraConfiguration and GimbalConfiguration fields to an AirVehicleConfiguration.

Recommended values for fields that need to be changed for the AirVehicleConfiguration messages are:

- CameraConfiguration/PayloadID: 1001 (1002 for vehicle with ID 2)
- CameraConfiguration/FieldOfViewMode: Discrete
- CameraConfiguration/MinHorizontalFieldOfView: 0.11
- CameraConfiguration/MaxHorizontalFieldOfView: 45.0
- CameraConfiguration/DiscreteHorizontalFieldOfViewList:
  [45.0, 22.0, 7.6, 3.7, 0.63, 0.11]
- CameraConfiguration/VideoStreamHorizontalResolution: 1024
- CameraConfiguration/VideoStreamHorizontalResolution: 768
- GimbalConfiguration/PayloadID: 101 (102 for vehicle with ID 2)

- GimbalConfiguration/MinElevation: -130.0
- GimbalConfiguration/MaxElevation: 40.0
- GimbalConfiguration/IsAzimuthClamped: true
- GimbalConfiguration/IsElevationClamped: true
- GimbalConfiguration/MaxAzimuthSlewRate: 115.0
- GimbalConfiguration/MaxElevationSlewRate: 115.0
- GimbalConfiguration/ContainedPayloadList: [1001] ([1002] for vehicle ID 2)

Note that for some values, it's easier to leave them blank and fill them in later with a text editor, potentially copying messages or portions of messages from an existing scenario file. But for completeness, these instructions use the AMASE Setup Tool as much as reasonably possible.

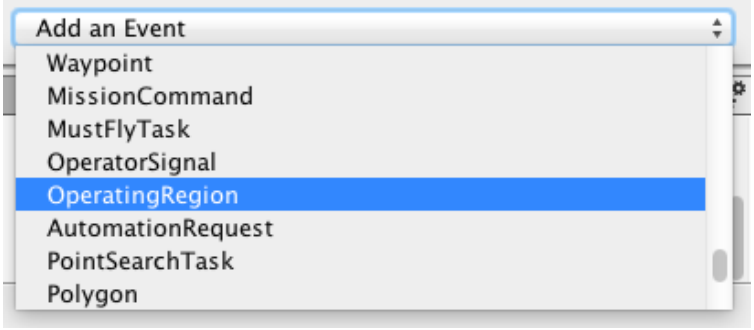

Figure 3: Adding an OperatingRegion in the Event Editor panel.

The next thing you need to add is an OperatingRegion message to tell UxAS where the vehicles are allowed to fly. You can do this by selecting the OperatingRegion element in the "Add an Event" pulldown menu of the Event Editor, as shown in Figure 3. An OperatingRegion contains KeepOutZone and KeepInZone elements. You will need to modify the following fields of the new OperatingRegion message for this scenario:

- ID: 10
- KeepInAreas: [3]
- KeepOutAreas: [1, 2]

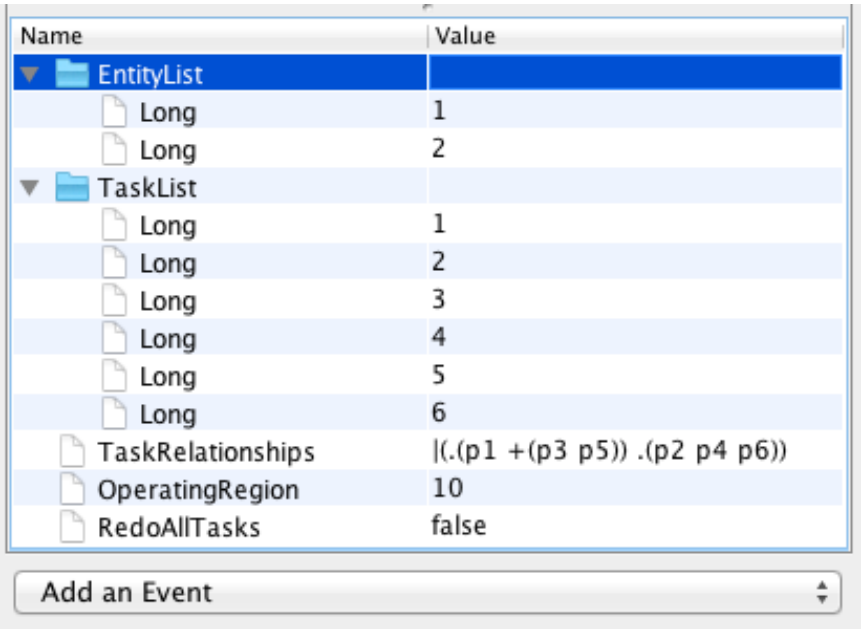

Figure 4: An example AutomationRequest message as added and edited in the Event Editor panel.

The last thing you need to add is an AutomationRequest message to tell UxAS which tasks you want it to plan for. You can do this by selecting the AutomationRequest element in the "Add an Event" pulldown menu of the Event Editor. An AutomationRequest lists which vehicles are eligible to fulfill the request, which tasks are part of the request, what process algebra relationships constrain the tasks, and what OperatingRegion applies. For this example, you will need to modify the following AutomationRequest fields, as shown in Figure 4:

- EntityList: [1, 2]
- TaskList: [1, 2, 3, 4, 5, 6]
- TaskRelationships: |(.(p1 +(p3 p5)) .(p2 p4 p6))
- OperatingRegion: 10

TaskRelationships is a process algebra string that describes constraints on how the tasks should be executed. This string uses a prefix notation, with a "p" prepended to each TaskID. All tasks in the TaskRelationships string should be listed in the TaskList, but tasks in the TaskList need not appear in the TaskRelationships string. Note that RedoAllTasks is currently unused.

You should now have all the messages you need to define an interesting scenario for UxAS. Save the file. The rest of these instructions will assume you've saved the file as `AssignTasks_Scenario.xml`.

2.0 PREPARING OUTPUT FROM THE AMASE SETUP TOOL FOR UXAS

UxAS reads in messages a little differently than AMASE. Rather than reading them in from a single file, UxAS uses its `SendMessagesService` to read in one message per file. This service broadcasts messages to other UxAS services at specified times. By convention, each message is saved in a file that is named according to the message type and some unique identifier, such as vehicle ID, TaskID, ZoneID, etc. The `SendMessagesService` is the configured to send in each message at a slightly different time. The file `AssignTasks_cfg.xml` (**Error! Reference source not found.**) shows an example configuration for UxAS that will work with the specified vehicles, tasks, zones, and automation request.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<UxAS EntityID="100" FormatVersion="1.0" EntityType="Aircraft">
  <Bridge Type="LmcpObjectNetworkTcpBridge" TcpAddress="tcp://127.0.0.1:5555" Server="FALSE">
    <SubscribeToMessage MessageType="afrl.cmasi.MissionCommand" />
    <SubscribeToMessage MessageType="afrl.cmasi.LineSearchTask" />
    <SubscribeToMessage MessageType="afrl.cmasi.VehicleActionCommand" />
  </Bridge>
  <Service Type="TaskManagerService"/>
  <Service Type="AutomationRequestValidatorService"/>
  <Service Type="SensorManagerService"/>
  <Service Type="RouteAggregatorService"/>
  <Service Type="RoutePlannerVisibilityService" MinimumWaypointSeparation_m="50.0"/>
  <Service Type="AssignmentTreeBranchBoundService" NumberNodesMaximum="0"
           CostFunction="MINMAX" />
  <Service Type="PlanBuilderService"/>
  <Service Type="AutomationDiagramDataService"/>
  <Service Type="WaypointPlanManagerService" VehicleID="1" NumberWaypointsToServe="512"
           NumberWaypointsOverlap="5" param.turnType="FlyOver" GimbalPayloadId="101"/>
  <Service Type="WaypointPlanManagerService" VehicleID="2" NumberWaypointsToServe="512"
           NumberWaypointsOverlap="5" param.turnType="FlyOver" GimbalPayloadId="102"/>

  <Service Type="SendMessagesService" PathToMessageFiles="../MessagesToSend/">
    <Message MessageFileName="AirVehicleConfiguration_V1.xml" SendTime_ms="150"/>
    <Message MessageFileName="AirVehicleConfiguration_V2.xml" SendTime_ms="160"/>
    <Message MessageFileName="AirVehicleState_V1.xml" SendTime_ms="170"/>
    <Message MessageFileName="AirVehicleState_V2.xml" SendTime_ms="180"/>
    <Message MessageFileName="KeepOutZone_1.xml" SendTime_ms="200"/>
    <Message MessageFileName="KeepOutZone_2.xml" SendTime_ms="210"/>
    <Message MessageFileName="KeepInZone_3.xml" SendTime_ms="220"/>
    <Message MessageFileName="OperatingRegion_10.xml" SendTime_ms="230"/>
    <Message MessageFileName="PointSearchTask_1.xml" SendTime_ms="300"/>
    <Message MessageFileName="PointSearchTask_2.xml" SendTime_ms="310"/>
    <Message MessageFileName="LineSearchTask_3.xml" SendTime_ms="320"/>
    <Message MessageFileName="LineSearchTask_4.xml" SendTime_ms="330"/>
    <Message MessageFileName="AreaSearchTask_5.xml" SendTime_ms="340"/>
    <Message MessageFileName="AreaSearchTask_6.xml" SendTime_ms="350"/>
    <Message MessageFileName="AutomationRequest_All.xml" SendTime_ms="5000"/>
  </Service>

  <Service Type="MessageLoggerDataService" FilesPerSubDirectory="10000">
    <LogMessage MessageType="uxas" NumberMessagesToSkip="0"/>
    <LogMessage MessageType="afrl" NumberMessagesToSkip="0"/>
    <LogMessage MessageType="eid" NumberMessagesToSkip="0"/>
    <LogMessage MessageType="uxas.messages.task.AssignmentCostMatrix" NumberMessagesToSkip="0"/>
    <LogMessage MessageType="AircraftPathPlanner" NumberMessagesToSkip="0"/>
    <LogMessage MessageType="RoutePlanner" NumberMessagesToSkip="0"/>
  </Service>
</UxAS>
```

Figure 5: The UxAS configuration file for this scenario.

Unfortunately, the current process for saving messages from a file generated by the AMASE Setup Tool to separate files is manual. You can copy and paste each message into its own file, with each file saved in subdirectory `MessagesToSend` and with names as in Figure 5: AirVehicleConfiguration_V1.xml, AirVehicleConfiguration_V2.xml, AirVehicleState_V1.xml, AirVehicleState_V2.xml, KeepOutZone_1.xml, KeepOutZone_2.xml, KeepInZone_3.xml, PointSearchTask_1.xml, PointSearchTask_2.xml, LineSearchTask_3.xml, LineSearchTask_4.xml, AreaSearchTask_5.xml, AreaSearchTask_6.xml, OperatingRegion_10.xml, and AutomationRequest_All.xml.

3.0 RUNNING THIS EXAMPLE

To run this type of example, you want to have two terminal windows ready. In one, you run the script `runAMASE_AssignTasks.sh`, shown in . This will pop-up the AMASE Simulator with the scenario loaded. In the other, you will run the script `runUxAS_AssignTasks.sh`, shown in . As soon as you hit enter to run the UxAS script, you'll want to hit the Play button in the AMASE simulator[2]. The planned paths for the AutomationRequest should show up, and the vehicles should fly them. Note that you can also just run the UxAS script, which will return the planned the paths as sets of waypoints, but then there is no simulation of how the air vehicles actually fly the paths.

```
here=$PWD;

cd ../../../OpenAMASE/OpenAMASE;
java -Xmx2048m -splash:./data/amase_splash.png -classpath ./dist/*:./lib/*
avtas.app.Application --config config/amase --scenario
"../../OpenUxAS/examples/05_AssignTasks/AssignTasks_Scenario.xml";
```

Figure 6: The AMASE shell script for this example.

```
#! /bin/bash

SAVE_DIR=$(pwd)

RM_DATAWORK="rm -R ./datawork"
RM_LOG="rm -R ./log"

BIN="../../../build/uxas"

mkdir -p RUNDIR_AssignTasks
cd RUNDIR_AssignTasks
$RM_DATAWORK
$RM_LOG
$BIN -cfgPath ../AssignTasks_cfg.xml
```

Figure 7: The UxAS shell script for this example.

[2] This is part of the reason that there is a delay of 5 seconds for the AutomationRequest message.

In either case, UxAS will save all the messages it sends and receives in a *.db3 SQLite file in the subdirectory `RUNDIR_AssignTasks/datawork/SavedMessages`. You can see all the messages that are internal to UxAS, and all the messages that are sent to AMASE and received from AMASE. You can also see what messages are received by AMASE in the AMASE Simulator GUI by selecting Scenario > Show Scenario Events in the menu. You should be able to pause AMASE mid-simulation and look at these without interfering with the operating of AMASE and UxAS.