The Assignments Scripting Language (ASL) for FSCaptain

Version 1.8.0

Note: All specifications are subject to possible change. Every effort will be made to keep existing facilities unchanged while we add new features, but in some cases, that may not be possible.

December 2017 62 Pages

Contents

This page is intentionally blank – except for this line and the page number below.

Introduction

FSCaptain extends its simulation of the aviation environment to include more task-oriented (as opposed to the procedural-oriented ATP side) by allowing pilots to seek and obtain 'Assignments' that involve accomplishing a goal without having more than minimally necessary procedures for meeting that goal imposed by the system.

This is done by implementing a scripting language that specifies the locations and goals of assignments, and optionally can include detailed structured placement and movement of objects in the sim world that are a part of the assignment.

(Note that we use the word 'Assignment' instead of 'Mission' because we want to avoid confusion with the built-in 'Missions' system and language that is built-in to ESP-based simulators. In FSX they are called Missions and in Prepar3D they are called Scenarios. The FSCaptain system is completely separate from either of these two thus we call ours Assignments, although throughout our documentation you will often see the word 'mission' used as having the same meaning as 'assignment'.)

How Assignments are Implemented in FSCaptain

From a conceptual standpoint, individually scripted assignments are tied to an airport. To obtain a mission to accomplish the pilot uses the regular sim facilities to place themselves at an airport that has assignments provided for it, in an airplane appropriate for the assignments available, and uses the standard FSCaptain FCDU device to call up list of available missions, and selects from that list the mission he or she may want to attempt. Missions are accomplished in 'free flight' mode in the sim, there is no special mode required.

The FSCaptain folder tree contains a special folder named 'Contracts'. Inside that folder, assignments written for a particular airport are written in plain text inside a sub-folder named for the airport's ICAO code. Inside that folder, a plain text file named for the airport's ICAO code and suffixed with the special extension '.asl' contains all the scripts for all the missions available at that airport.

Scripts may make use of auxiliary support files. Examples of these support files include HTML for assignment briefings, and blocks of ASL script that can be included in script text by the ASL COPY facility (described later in this document.) Auxiliary files are kept in the folder for the airport whose scripts they belong to.

Here is an example of creating a new script for an airport that doesn't yet have any scripts written for it. Let's say we want to write a simple assignment to deliver a special cargo of 400 pounds from

airport DABC to airport EXYZ.

The first step is to create a folder in the FSCaptain 'Contracts' folder named DABC. This is the airport where the pilot will see and be able to accept the assignment.

Then, using the plain text editor of your choice (Notepad will work, I use the free Notepad++) you will create a text file inside your new folder named 'DABC.asl' (These names are not case sensitive, and you choose what to capitalize and what not to.)

To properly implement our mission (i.e. assignment, we use the term to mean the same thing interchangeably) the file DABC.asl contains the following in the ASL language:

```
MISSION=101,Offered=DABC,Priority=100,Times=1,VolMin=0,VolMax=2,Name=Deliver Cargo to EXYZ,Score=5
Text=Transport special cargo to airport EXYZ. Return to DABC after delivery.
Target=EXYZ,Lat=47.486537,Long=-122.212437,Meters=10000,MinAGL=0,MaxAGL=0,Action=U
Stage=1,Start=$DABC,Dest=EXYZ,End=$DABC,Type=1,PAX=0,Cargo=400,Time=2,Special=0
```

(Don't worry that at this point you may not understand anything about that script. That's what this document will help you with. Right now, understand that's a script that defines this very simple mission.)

With this script in place, then any pilot that brings up the FCDU and logs in at airport DABC will see an option on the right-hand side of the Preflight page to 'Select MISSION'. (Normally, if no missions are present at an airport, this option says what it used to say all the time; View INFO.)

If the pilot clicks on that option, he will see every appropriate assignment that's written into the DABC.asl file in the DABC folder of the FSCaptain\Contracts folder. He/she can then click on the assignment they are interested in and the process will lead them on as described in the "Assignments.pdf" document. Once the assignment is complete it will be logged in their regular FSCaptain and the time spent will accrue to the log and the score will be factored into the overall score for their career in the same way as an FSCaptain ATP procedural flight.

Why have a "Contracts" folder?

Strictly speaking, Assignments are a type of flight that can be offered individually, or as part of an FSCaptain "Contract." A Contract is a multi-flight arrangement that can be conducted as multiple ATP flights, or as multiple Assignments. So, in the future, a Contract could require you to fly *one million pounds* of cargo from Airport A to Airport B using ATP rules within a time constraint… or another Contract could require you to fly several Fire Retardant Drop Assignments within a time constraint….

Overview and Example of ASL

A mission script consists of two sections: a descriptive part and an animations and effects section. The second is, as we said, optional. Animations (SimObjects that move and interact with the user) and effects (things that emit particles, like smoke or explosions or fires) help to add immersion to a mission. But remember FSCaptain assignments happen in free flight and don't have to have everything that moves or happens to be scripted. FSX/P3D already provides a "living world" and our missions take place in that world. The existing objects and effects are made a part of it as well.

The descriptive section consists of only a few statements that work in a "fill in the bank" type way to decide on a mission pattern and where it will take place. The required statements are only four:

The statements below must be provided in the script in the order shown.

The **Mission** statement: this describes the basic parameters about the assignment to allow the system to decide whether to offer it or not, and where, and what the point reward will be if you complete the mission objective.

The **Text** statement(s): This statement or series of statements provides the description of the mission and any appropriate instructions for each phase of the mission. Only one Text statement is required: the one describing the assignment itself. Separate 'Text' statements are available which can be different for each stage of the assignment.

The **Target** statement: Each mission has only one target zone. This statement defines that zone, and most crucially, it defines the class of mission. The class will determine how the various parameters are treated by the mission tracker, and what the objective of the mission is. The five classes are: delivery, pickup, recon, airdrop, and attack. In a sense these classes define the relation between the three geographical points of the mission described in the stage statement below.

The **Stage** statement: This provides the details on where the mission will take place and what it is to do. The meaning of these parameters is determined by the mission class defined in the Target statement, but the pattern of the parameters of the Stage statement is always the same.

You can define quite challenging missions using these set of four statements alone. Here is an example we will sue to illustrate how this all works:

Example: Simple Delivery Class Mission

```
MISSION=7185,Offered=L20A,Priority=50,Times=9999,VolMin=1,VolMax=1,Name=Deliver to
LS85,Score=5Text=Deliver 1500LB Equipment 2 PAX to LS85 (NDB 291 VOR 108.5)
Target=LS85,Lat=20.447705,Long=104.715762,Meters=10000,MinAGL=0,MaxAGL=0,Action=U
Stage=1,Start=$L20A,Dest=LS85,End=$L20A,Type=1,PAX=2,Cargo=1500,Time=2,Special=0
```

This short script defines a typical Air America mission to deliver 1500 pounds of freight from Lima Site 20A to Lima Site 85. This sounds about as simple as it gets. But the challenge in this mission isn't what and how – it's the fact that Lima Site 85 is an 800-foot dirt airstrip on top of a mountain with a steep drop from all but one side. There is zero room for error. And if you are flying with real weather as I always do, the mountains of northern Laos in that area are frequently shrouded in mist and clouds. And there is no ATC available to help. Plus, as we described in the combat section above, it is (optionally) possible that you would encounter random hostile fire that can damage or destroy your aircraft anywhere over those mountains. The challenge of a mission doesn't have to come from the complexity of its design. A simple task in aviation can be absolutely terrifying and require amazing piloting skills, like this one does.

The syntax of the language is described in detail in the next section of this book. Here we will be pointing out the important parts.

The `MISSION=` statement parameters say this mission is offered at airport L20A, it has a priority of 50 (half the time it comes up it will be offered, and half not), it can be run 9999 times, it's only available to a volume class 1 airplane (class one is “small”, the classes are 0=Tiny, 1=Small, 2=Medium, 3=Large, and 5=Huge.) Its name is “Equipment to LS85”, and if you meet the mission objectives you will receive 5 points on top of your base score. For a starting pilot the base is 85, so 85+5 = 90, which is an A- score. There is one optional parameter. `Model=` can specify an exact model of aircraft required for a mission, but we don't have it in this example.

The `TEXT=` statement (the ASL language is not case sensitive) just provides a description which the pilot will see on the FCDU. You should provide any details necessary, such as here the VOR radial and distance to location LS85 is provided. This is the simple version of `TEXT=`. You can also include, in addition to this, TEXT1= through TEXT5= statements. Each contains text that will displayed in the load, outbound, engaged, inbound, and complete stages are active, respectively.

The Target and Stage statements are the meat and potatoes of the mission.

`TARGET=` defines the target and names it as “LS85”. This is also the ICAO of the airport in this case, but it doesn't have to be, a target can be any point on the earth, and this is the name you give it. The actual point is the Lat and Long parameters following the name. Meters defines the size of the target zone. This means that here our target zone is anywhere with a circle 10,000 meters (that's 10km of course) of that point defined by the Lat and Long. This relates to the phases. The pilot will be OUTBOUND until they get within 10km of the defined target. Then they will

transition to the ENGAGED phase. As long as they stay within that 10km zone they will stay engaged for a Delivery class mission. Once they leave that zone they will become INBOUND. Inbound where? That will be defined on the `Stage` statement next. The next two parameters allow you to specify a vertical element to the target zone – two altitudes AGL for the floor and ceiling of the zone. leaving them zero means there is no floor or ceiling, as in this case. And last but not least, the Action parameter crucially defines the mission class. The classes codes are **U=Unload** (Delivery class), **L=Load** (Pickup class), **R=Recon** (Reconnaissance, or Survey or Loiter class), **D=Airdrop** (Airdrop class) and **A=Attack** (attack class.) Remember this determines what the parameters on the `Stage` statement mean, and what's expected of the pilot on the mission.

`STAGE=` Defines the parameters of the mission itself. In our example, the three crucial geographical points are defined as:

`Start=` This says the airport (or object, see the "animation section below for what that is) that the pilot must be "at" to start the mission. This may or may not be the airport where the mission was offered. In this case it is. If this is an airport, it's specified as an ICAO with a $ in front of it. If there is no $, it is assumed to be an object that's defined later in the animation statements.

`Dest=` Specifies the target. This is the point of the center of the target zone. In this example it is LS85 which is the name of the single target. It could be an airport using the $ rule above, or it also can be the name of an object. In order to meet any objective, the pilot must at least enter the target zone – failure to do so will result in no points (from the `MISSION` statement above being awarded.)

`End=` Specifies where the mission will end. In order to fully complete any mission, the pilot must land and be fully stopped at or near the point defined by this END parameter. In our example this is $L20A – the L20A airport. So, after delivering these supplies you must fly back there. But this could also be any other airport, or the name of an object defined in the animation statements.

The above three parameters define the three geographical points of the mission: where it starts, where the pilot must be to meet the objective, and where it must end. These points can be any fixed point, the name of an airport, or the name of an object – and remember objects can move.

There are two basic proximity rules to remember: If the point is defined as an object, the pilot must be within 40 meters of that object plus the wingspan of their airplane to meet the condition of being "near" it. If it is not an object (an airport or target) then the user aircraft must be within 5NM of it to be considered "at" it. Remember loading, unloading, and mission completion operations also require the airplane to be still (but the engines can be running.)

The rest of the Stage parameters are about what you are carrying. `Type` is the type of flight where 1=Cargo, 2=passenger, and 3=ferry and doesn't matter much on an Assignment. `PAX` and `Cargo` define the number of PAX to be flown and the pounds of cargo. These are only relevant on Delivery (U) class flights, or on Pickup class flights (L), or Airdrop class (D). Recon flights and

Attack flights do not have hauling things as their objective. The `Time` and `Special` parameters are required but not used on Assignments at this time – they are relevant to the Contracts system which uses the same ASL syntax. (`Time` is intended to set a time cap, and `Special` defines whether an FSCaptain flight has a combination of the Priority, Live, and or Fragile attributes.)

Optional Statements

There are several optional statements allowed in the description section. Optional statements must also follow the Stage statement but otherwise they can be in any order. They are:

`Timed=` This allows you to specify a time limit by which the mission must be completed, or the objective met points are forfeited, or a time limit that the pilot must stay in the target zone of a Recon class mission. In fact, both can be specified.

`Requirement=` This allows you to prevent your mission from being offered to a pilot based on a wide set of criteria. We already saw that on the MISSION statement we can limit the volume class and even specify an exact model. Here we allow checking the FSX/P3D environment for things are required for the mission to successfully execute. See the syntax section for details. Much of this is devoted to being sure than any animations or effects are executing in an appropriate environment. For example, if your mission calls for meeting a truck on the south ramp of an airport, you will need to specify exactly where that is. But a particular user may have a custom version of that airport installed, which might make your carefully programmed object placement and movement happen in the wrong places. There are other requirements, say to check for external add-on products we support like our own FSCAI (FS Combat AI) and VRS Tacpack and or the freeware FSX@War package. Most of this is related to the extensive combat support. You don't want a combat mission offered if the user doesn't have the required add-ons installed and running to support it.

`Item=` As you may know, when you are carrying freight FSCaptain will automatically construct a manifest of exactly what you are carrying for documentation purposes. Normally it uses its own default Items configuration file for this, but this file has items typical of civil aviation. You are allowed to specify your own item list for the manifest generator here, and if you do, your manifest will be constructed entirely from the items you give in the mission script, except for PAX, which as automatically handled. That way you can be sure you are truly delivering what you intend. But this is optional. You can consider cargo as just weight, and not care about the specifics, which don't really matter to the completion of your mission anyway.

Here is our example fleshed out with some optional statements:

```
MISSION=57185,Offered=L20A,Priority=50,Times=99,VolMin=1,VolMax=1,Name=Dlvry LS85,Score=5,Model=PC-6C
Text=Deliver 1500LB Equipment 2 PAX to LS85 (NDB 291 VOR 108.5).
Target=LS85,Lat=20.447705,Long=104.715762,Meters=10000,MinAGL=0,MaxAGL=0,Action=U
Stage=1,Start=$L20A,Dest=LS85,End=$L20A,Type=1,PAX=2,Cargo=1500,Time=2,Special=0
Timed=MAXMINUTES=60
Requirement=FSCAI
Requirement=SOURCE=L20A,L20A_ADEX_JG.BGL
Requirement=FILE=SimObjects/Misc/FSX@WarPack1_Algeco_Small/model01_ops/model.cfg
Requirement=DOCUMENT=FSX@War/Packs/Vietnam_files/Generic/AAM_HQ.plg
Item=L20A,Box,es,Electronic Equipment,500,500
Item=L20A,Box,es,Tools,250,250
```

We've changed it to a timed mission – you must get back to L20A after the delivery within 60 minutes. Requirements are now quite restrictive. Note on the MISSION statement you have to be in a PC-6C model aircraft – the only one besides a helicopter than can do this mission because of the short runway. (Note that HELICOPTER is also a possible requirement to stand in for any rotary-wing aircraft.) It also requires the FSCAI program to be running. This will provide hostile ground fire from any weaponized object. It requires that the L20A airport be a specific version from a specific BGL file that comes with the Vietnam War Project freeware. And it requires a certain SimObject exist - to check that the FSX@War product is installed. Without that, no hostile fire could happen. It also then checks to be sure that the Vietnam War pack for FSX@War is also installed. And finally, it lists the detailed items we are taking to LS85 - "Electronic Equipment" and "Tools".

All this is optional. But it insures that to be offered this mission the user has to have the freeware add-on FSX@War installed on his system, have the freeware Vietnam War Project scenery pack installed, have a specific FSX@War pack installed to create the combat environment of Laos in the 1960s, and furthermore have the FSCaptain program named FSCAI not just installed but running and connected to the simulator so the FSX@War objects can shoot at the user airplane. Also, he must be in an aircraft with an ATC_MODEL in the aircraft.cfg of PC-6C and this aircraft must be configured in the FSCaptain system as a volume class 1 aircraft.

This is a lot. But all of it together creates an authentic look and feel to this typical Air America mission. It goes from a trivial seeming delivery of some supplies to a difficult and dangerous assignment for any pilot.

In just a relatively few lines of code, plus a collection of add-ons, most of them freeware.

To see other Air America missions, of all classes, look at the file L20A.cfg in the FSCaptain\Contracts folder. (You can also read the *Air America Guide* in the FSCaptain\Documents folder.)

Animations and Effects

Placing specific SimObjects in the world for a particular mission can add atmosphere and immersion, especially if there are some effects involved. But objects can do much more than this: they can profoundly affect the way the mission plays out.

For instance, recall that the three geographical points of the mission pattern can be airports, but they can also be points in space or the name of objects. Why make any of them an object? Well in the case of the mission target (the DEST= parameter) it makes sense to target an object – an ambulance, where ever it may be, that is waiting for you to deliver a patient to it. Or maybe a certain moving vehicle with known terrorists in it you are assigned to terminate with extreme prejudice.

But why should you allow the starting and ending points to be objects?

Imagine you want to design a mission where the pilot must take off from an aircraft carrier, strike a target, and return to the carrier. By making a carrier object you provide them a place to take off and land. But carriers are rarely anchored – they move. As mission designer you decide where they go, in fact. Where you return will not be were you took off. It is a moving start and end point of a mission.

But wait, there's more! Every SimObject you assign to a mission can have a route assigned to it. The primary role of the route is to map out where the unit will move. But also, coded instructions can be placed in the route also. These instructions can provide conditions for the route to continue, or provide actions for how the SimObject will behave and what it will do. They can even change the points awarded based on SimObject actions. For example. Your mission could be to prevent an object from getting to a destination. You fail if it gets there.

All animation statements must follow the last descriptive statement, the Stage statement, and be after any optional descriptive statements. The animation statements most also be specified in a particular order.

There are two required and two optional animation statements. The purpose is to create SimObjects in the simulated world, and to control their actions. They may have no action but just sit there. That's why two statements are optional.

The two required ones for even the simplest object are `POINT` and `OBJECT`.

The `POINT` statement defines a specific point on the earth for later use from other statements. All point statements must be listed before any other animation statements. Each point must have a unique name assigned by you, up to 15 characters in length. You provide the latitude and longitude of the point, and also optionally an altitude and heading. You can also provide an optional speed

parameter for use by the route statement documented below. At a minimum you must provide latitude and longitude. (We have plans to enhance the point statement to include computed point locations, but this is not implemented yet.) *Important*: for all objects that are staying on the ground, the altitude of all points they travel on their route must be zero! Zero means "on the ground" in a POINT statement.

The OBJECT defines a SimObject, and specifies its location and possible action. Every object has a unique named assigned by you, and the title of an actual SimObject you have installed in your SimObjects folder. This can be an airplane, boat, ground vehicle, or "misc" object. The title you provide must match the title in the airplane.cfg or sim.cfg file – NOT the folder name. Or, and this is typical of most missions, the name of the object can be a "Symbolic Name" which is defined in either the SimObjects_STD.cfg or the SimObjects_USER.cfg files in the FSCaptain Config folder. The concept of using Symbolic Names is treated in more detail later.

After the name and title/symbolic name, you list its type, GROUND, AIR, SEA, MISC, DROP or CONTROL. CONTROL is a special virtual object that does not actually exist (its title parameter should be NONE) but is present in order to control other objects. (DROP is a special object carried on the pilot's aircraft on airdrop type missions. It has a different set of parameters.) Then for a standard object (not DROP) provide the already defined point name where the object will be initially spawned in the point= parameter. Then an Odds= parameter to provide a random factor on whether the object will be spawned or not. 100 means always, 50 means 50% of the time. Note: Objects that contain important instructions must of course be defined as Odds=100. There is also a Role parameter that can potentially give special properties to an object, but normally this is just 0. And last, and very important, the Route= parameter if the object is to move or to control the environment or other objects. Route is optional if its role is just to sit there and look pretty.

Here is an example of just such a "sit there and look pretty" object:

```
Point=GREENCAR01,Lat=46.155841,Long=-123.885488,Altitude=19,Heading=153
Object=GREENCAR,Title=Veh_Car_Mini2_DarkGreen_sm,Type=GROUND,Point=GREENCAR01,Odds=100,Role=0
```

This places a generic green car outside the westernmost hanger at the KAST airport, pointing south. It's always going to be there (Odds=100). It has no special role (Role=0).

Wait! How do I know where that is in latitude/longitude/altitude/heading terms? I see all those numbers on the point statement.

Well, the hardest way to do so (but the simplest way to explain!) is that you bring up the sim, place it in Slew mode, place your airplane where you want it to be, and copy the red text info at the top of the screen to your Point statement. That's tedious and error-prone but it works if you are careful and patient. In order to see decimal coordinates, you will need to modify your FSX or Prepar3d.cfg file, adding these two lines to the [Main] section:

```
LatLonFormat=Degrees
FractionalLatLonDigits=6
```

A much easier way is the Route Builder function built into the FCDU and accessed from the FLTSIM option when you login and press the MENU key. The Route Builder is described in more detail later. (As we said, ultimately, we will have a Windows app called Mission Maker than will allow you to generate a script by filling in blanks on the screen and it will interface with the sim running to obtain coordinates at the press of a simple button. Not yet. Route Builder doesn't solve the problem, but it takes the tedium and silly typing errors out of it.)

Wait! What if the user has a custom KAST installed? Won't that location probably change? Yes, it will. That's why we have the `Requirement=STOCK=KAST` statement available. It will check to see if KAST is the stock version and if not, the user won't see any mission that has that requirement in it. What if you want to make mission animations for a custom airport layout? That's why we have the `Requirement=SOURCE=WHATEVER.BGL` statement. You can require any particular custom airport BGL for a mission!

All `OBJECT` statements must follow all `POINT` statements.

Notice that in the example above we specified the exact title of a physical SimObject when we wrote '`Veh_Car_Mini2_DarkGreen_sm'` in the title parameter. Generally, it's a better practice to use Symbolic names when possible. In the standard SimObjects_STD.cfg file, a symbolic name 'DarkGreenCar' is available and it is linked to the actual '`Veh_Car_Mini2_DarkGreen_sm'` in the sim. Using Symbolic names adds safety and flexibility. If you are using a sim and in the future a new release changes all the names of the SimObjects, you won't have to go and change every script you ever wrote – you only change the link between the symbolic name 'DarkGreenCar' and the actual object that it references once, in the configuration file, and all scripts will follow along automatically. There are other advantages to symbolic names, described later.

Now the optional statements are where it gets really interesting. They are `ROUTE` and `EFFECT`.

In its most basic form the `Route` statement gives each route a unique name, and lists all the points in the route. When an object is assigned to that route by name, it will automatically drive (or fly in the case of AIR points) to each point in sequence until the route ends, when it will sit still (or fly back to the starting point) because it has nothing more to do. Note it will take a direct path between points – it does not follow roads or take terrain into consideration at all. Told to drive between two points on either side of a lake, the ground vehicle will not attempt to drive around

the lake but right over the water. Take this into consideration as you design routes.

However the real power of the scripting language comes in the conditions and actions that can be embedded in a route. There are many such and they are placed in the route in place of a point by putting the instruction in [brackets].

Here is a plain route statement with no instructions:

```
Route=DRIVEAWAY,Points=GREEN02,GREEN03,GREEN04,GREEN05,GREEN06,GREEN07,GREEN08
```

When this route is assigned to any object, and it can be assigned to multiple objects, it simply means as each object with DRIVEAWAY in the Route parameter of the object statement is spawned it will immediate start driving to each point in the list in sequence. Each separate point is called a "route step". All must be defined points of course. If the object is an aircraft, the altitude will be a target and the airplane will climb or dive. If there is a speed parameter on the point statement, the object will attempt to travel at that speed. Heading is not used except to position the object when it is first spawned.

Now here it is modified with a simple instruction embedded:

```
Route=DRIVEAWAY,Points=[HUC;5000],GREEN02,GREEN03,GREEN04,GREEN05,GREEN06,GREEN07,GREEN08
```

The new [HUC;5000] is the instruction of course, but what does this mean? "HUC" stands for Hold Until Close – an instruction that tells the route to wait until the user aircraft is within a certain distance. The 5000 is that distance, in this case 5000 meters. So, now the object(s) that are assigned to this route will not move until the user aircraft gets within 5KM. Then they will start driving.

There are many such instructions that can accomplish useful things. Any instruction can be in place of any point name. Some are more elaborate than this: [HUOC;REDCAR,100] will Hold Until Object Close, in this case "REDCAR", and when it is within 100 meters of the object the route is controlling that object will advance to the next step in the route - which could be a point name or another instruction. These examples are conditions, but some instructions are imperative immediate actions. For example [EFFUP;FIRE;60] is an instruction to start the Effect object FIRE, and set its duration at 60 seconds. Or a commonly used one is [KILL]. This will remove the current object from the game. Here we kill the green car at the end of its run:

```
Route=DRIVEAWAY,Points=[HUC;5000],GREEN02,GREEN03,GREEN04,GREEN05,GREEN06,GREEN07,GREEN08,[KILL]
```

Objects can even control other objects. [OBJUP;BADGUY] will spawn the object named BADGUY. [OBJDN;BADGUY] will remove it immediately. Here is a route that contains multiple instructions and no point names:

```
Route=HAVOC,Points=[HUC;5000],[OBJUP;BADGUY],[EFFUP;EXPLO1;1],[EFFUP;EXPLO2;1],[OBJDN;GOODGUY]
```

You should be able to see what this does. It waits until the user aircraft is within 5000 meters, then spawns an object BADGUY, kicks off two explosions, and removes the object GOODGUY. This assumes of course that all these objects and effects are defined in the script.

All `OBJECT` statements must appear after all `Route` statements.

The `EFFECT` statement defines an effect like an `OBJECT` statement defines an object. It gives it a name and attributes. Later the effect may be manipulated by instructions in a route or routes, but not necessarily. A permanent effect can simply be defined and let stand for the duration of the mission.

Here is a sample `EFFECT` statement:

```
Effect=FIREFF,Title=AAM_Forest_Fire_HG,Point=FIRESPOT1,Active=1,Duration=3600
```

FSCaptain comes with a set of effect objects that trigger stock effects such as explosions and fires.

The effect FIREEFF is defined as a canned one named `AAM_Forest_Fire_HG`. It will be spawned at FIRESPOT1 which is a point defined with a point statement. It is active when spawned, and it will last for 3,600 1/18th of a second 'tick' before going out on its own. Therefore, in this script, a fire will start burning at point FIRESPOT1 and it will keep on burning for a long time.

This example is from the “Fire Retardant Drop” mission. How do we put it out? Like this in a route statement:

```
Route=FIRESTOP,Points=[IFHIT],[EFFDN;FIREFF],[WAIT;5],[EFFUP;SMALLFIRE;3600]
```

This code starts with an [IFHIT] instruction, that causes the route to hold until any dropped object hits its target. Then the route will start advancing. [EFFDN;FIREFF] turns out the fire. The [WAIT;5] simply waits 5 seconds. Then the [EFFUP;FIREFF;3600] starts another effect up but this time uses SMALLFIRE – a smaller fire. We didn't put it totally out we just reduced its size.

So now you should be able to see how the FSCaptain ASL implements missions. The devil is in the details. We encourage you to try your hand by cutting and pasting existing mission scripts and modifying them into what you want.

In the following sections, all details of all possible statements and parameters are formally defined for reference.

General Rules for All Sections and Statements

As you could see from the example we gave, the pattern for ASL statements is a “fill-in-the-blank” one. A series of named statements (in the example they were 'Mission', 'Text', 'Target' and 'Stage' provide all the parameters needed for the system to create and track the pilot's progress through the execution of the assignment.

All parameters in each statement must be in the order specified in this document. Unless marked optional they are *required* in each statement. Unless otherwise specified here each parameter is defined by the pattern **keyword=value** where **keyword** is the defined name of the parameter, and **value** is the value you are supplying for the parameter. All keyword/value sets are separated by commas. For this reason, commas cannot be present in any parameter value, event “free-form” text. Comments can be included on lines by themselves or after the last parameter on a statement by preceding the comment with a semi-colon.

All quantities are in pounds and times in hours, minutes, or seconds. Distances are either in meters for shorter distances that require more granularity, and in nautical miles for longer ones.

ASL Statements are not case sensitive. Case should be used for clarity in reading.

Standard Naming Rules: All targets, points, objects and effects in the ASL language are given a name so that other statements can use them by reference. All these names follow the same simple rules: they can be from 1 to 16 alphanumeric characters, and they should not contain spaces or any special characters except the underscore or dash. They are not case sensitive: the letters 'A' and 'a' mean the same thing. *All names must be unique within a given mission.*

An ASL script has two distinct parts: The Definition section, and the Animation section. Each section consists of required or optional statements in a particular order.

The Definition section is mandatory. The Animations section is optional. In the example in the first section of this book, the simple example of the cargo from DABC to EXYZ, there was only a Definition section. Many assignments only have this one section because that's all they need.

There is no formal marker separating the two sections. The Definition section ends when the first Animation section statement in encountered, which all always be a POINT statement.

The Definition Section Statements

The minimum required to define a complete assignment is one MISSION statement, one TEXT statement, one TARGET statement, and one STAGE statement, in that order. There can only be one of each statement. Optionally, TEXT1, TEXT2, TEXT3, TEXT4, or TEXT5 statements may be included after the optioning TEXT statement to more precisely provide instructions to the pilot for each stage. Also, following the STAGE statement, a set of **optional** statements can place restrictions on the presentation and execution of the assignment.

MISSION Statement

Mission parameter: A number between 100 and 65534 that uniquely distinguishes this mission from all other missions defined for the same airport. Missions for an airport do not have to be defined in numerical order, although they often are for ease of reference.

Offered parameter: The ICAO code for the airport where this mission will be offered to any pilot seeking missions via the Assignments selection on the FCDU device.

Priority parameter: A number from 0 to 100 which will determine how often this assignment will be presented in the list of available missions. This is a percentage. Zero means the mission will never be presented, 100 means the mission will always be presented, and any number in between determines how likely a pilot seeking missions is to see this one on any particular inquiry via the FCDU. 50 would mean a 50% chance of it being presented. A rare mission could be as low as 1% so the pilot would, on average, only see this mission in 1 out of 100 times he looks at the FCDU for a mission list. Using a zero here is a handy way to temporarily disable a mission.

Times parameter: A number for how many times this mission can be completed before it's never offered again on the available missions list. This is keyed by mission number. The system will count, in the pilot's current FSCaptain log, how many times the mission has been logged for the current airline and the current pilot. If that number exceeds this parameter, the mission won't be presented. The most commonly used numbers here are 1 and 9999. 1 means the mission, once flown, will never be able to be flown again by the same pilot at the same airline, unless he were to delete the log entry for the previous flight completing the mission. 9999 means practically 'no limit' since it's highly unlikely even the most dedicated mission runner will do something 10,000 times in one FSCaptain career.

VolMin parameter: The purpose of this parameter and the next, working together, are to prevent missions from being offered to aircraft that are not of an appropriate size and capacity. Pilots are not interested in seeing missions that aren't right for the size aircraft they are piloting. Therefore,

if the volume class of the airplane that the pilot is currently in is less than this minimum he won't see this mission. Volume class numbers are defined as:

- **0 = Tiny**. These are your 2, 4 and 6-seat single engine airplanes or small helicopters. Most are small General Aviation airplanes such as a Cessna 172. Only a few relatively small boxes can fit as cargo.
- **1 = Small**. This is a larger space but still a small airplane. It could be a single-engine or a twin. A larger box or container, perhaps the size of a casket, could fit in one of these. A Cessna 208 would be a good example of a small airplane.
- **2 = Medium**. These sizes are the larger GA airplanes or smaller commercial aircraft. An example would be a small to medium regional jet, or a DC-3, or a Convair CV-580. Quite a sizable container of cargo can fit, but not a large one.
- **3 = Large.** These are large aircraft designed to carry a heavy load of passengers or cargo. All but the largest size cargo items can fit. A Boeing 737, Airbus A320, DC-6, 7, or 9. A Boeing 707 or DC-8 might be the top end of this class. Any standard cargo container can fit in these but not the largest items, such as a truck or tank.
- **4 = Heavy.** These are the big specialist airplanes, such as the 747 or A380. Almost nothing transportable is too big to fit in these, including trucks or tanks.

VolMax parameter: Used in tandem with the VolMin parameter this helps offer missions only to appropriately sized aircraft. If the volume class of the airplane that the pilot is currently in is greater than this maximum, the pilot won't see this mission.

Name parameter: This is the text the pilot will see in the list of available missions on the FCDU. Because of very limited space on the small screen, this should be limited to no more than 35 characters. Any greater than that will be truncated by the system. Any text can be placed here without restrictions except you can't use a comma because that would indicate the end of the parameter.

Score parameter: This is a number, 0 to 15, that specifies the maximum number of points that can be added to the pilot's score if he successfully completes the mission. This is a *maximum* - failure to complete individual tasks can subtract from this score depending on the mission type.

Model parameter: This parameter is optional and can be omitted from the statement. If it is specified, it limits the offering of this mission to one specific model of aircraft, listed here. The model number is determined by the FSCaptain model type. To limit this mission to only aircraft with a model of 'B737' put that in this parameter. Be careful with the use of this, these model numbers are not standardized or enforced at any level, except that each type of aircraft has to have a unique one. Largely this is left to aircraft simulation developers to specify. For example, for a DC-3 type aircraft you could see 'DC3' or DC-3' or 'C47' or 'C-47' or even 'DC3A' or DC-3C', all basically referring to the same type aircraft. You can only specify one of these types here. The

practical use of this parameter is if you want the mission to apply to a specific aircraft and you know the ATC_MODEL specified in the airplane's aircraft.cfg file.

TEXT Statement(s)

This statement simply consists of a limited amount of free-form text after the = sign. It comes in two variations, the plain TEXT statement and the numbered TEXT statement. Both are limited to 6 lines of 35 characters each, so a total of 210 characters. These lines will be automatically formatted and presented on the FCDU by the system.

Words placed in the plain text statement (which is required in every mission) will be presented to the pilot when he first selects mission from the available missions list. It should simply and clearly describe the mission in general terms. The pilot will use this description to decide if he wants to accept or decline the mission. The plain text will also be presented at every stage of the mission that doesn't have a specific numbered TEXT statement provided for it.

The numbered text statement is just like the plain text, except the word TEXT has a number after it from 1 to 5. Each of these statements is optional, but if provided, they should follow the required plain text in the mission script. The number matches the stage of the mission as it progresses, with 1 being the **Load** stage, 2 being **Outbound**, 3 being **Engaged**, 4 being **Inbound**, and 5 being **Completion**. Like the plain text, each numbered text statement can contain up to 210 characters of text in any form. Each statement will present its text to the pilot only during the specified stage. Thus, in using these statements you should provide information helpful to the pilot at each stage.

TARGET Statement

Every mission has a target, and this statement defines that target precisely. It is required, and it must come after the TEXT statement(s) and before the STATE statement.

Target parameter: This is a name that will be used in other statements to refer to this target and it follows the standard naming rules documented at the start of this section. There are some additional special conditions that apply to this name:

- If it less than 6 characters long and starts with a special symbol $ it is an airport ICAO code.
- If the target name matches the name of an object defined in the Animation section below, then the location of the target will be updated to reflect the location of the object no matter where it moves.

- If the target name matches the name of an object defined in the Animation section below, then for a Pickup or Delivery type mission, the aircraft must be within 50 meters of the object for the load or unload to proceed.

Lat parameter: in tandem with the Long parameter, this specifies in decimal coordinates the exact initial position of the target on the surface of the earth.

Long parameter: in tandem with the Lat parameter, this specifies in decimal coordinates the exact initial position of the target on the surface of the earth.

Meters parameter: this defines the radius of the circle around the target point that is referred to as the "engagement zone". When the user aircraft enters this zone, the mission stage will increment from Outbound to Engaged.

MinAGL parameter: in tandem with the MaxAGL parameter this defines the vertical limits of the zone specified by the Lat, Long, and Meters parameters. Only if the user's aircraft is at or above this altitude (above ground level) then it is in the engagement zone. If a zero is supplied in this parameter, it means there is no minimum.

MaxAGL parameter: in tandem with the MinAGL parameter this defines the vertical limits of the zone specified by the Lat, Long, and Meters parameters. Only if the user's aircraft is at or below this altitude (above ground level) then it is in the engagement zone. If a zero is supplied in this parameter, it means there is no maximum.

Action parameter: This one-character code defines the type of mission associated with this target, and in doing it defines the pattern to be followed as the mission proceeds from stage to stage. There are five possible mission patterns:

- U = Unload (Delivery) Mission Type.
  - Load stage: The pilot is required to open the main door and to take on the amount of weight in pounds defined by the Cargo and PAX parameters of the STAGE statement for this mission. PAX pounds are computed at 200 pounds per person (170 pounds plus 30 baggage.) If the total amount of payload weight required is already loaded in the aircraft, nothing will be loaded, and no weight will be removed. Only enough payload will be added to bring the weight up to the requirement. The Load stage ends when the pilot clicks Authorize Load and the weight is confirmed added.
  - Outbound stage: The pilot is required to fly to the defined engagement zone to transition to the Engaged stage.
  - Engaged stage: The pilot is required to successfully land the aircraft and come to a complete stop in the engagement zone. If the target name matches an object in the animations section, then the pilot is also required to be stopped within 50 meters of that object. When these conditions are met, the pilot will be prompted to open the door and

authorize the unload. Once that is done, the weight is removed from the aircraft and the stage is advanced to the Inbound stage.

- Inbound stage: The pilot is required to leave the engagement zone and land at or near the destination defined in the STAGE statement. That destination will either be an ICAO code, or the name of an object defined in the Animations section. If an ICAO code, in order to advance to completion, the pilot must stop the aircraft on the ground within five nautical miles of the center of that airport. If the name of an object, the pilot must stop his aircraft within 50 meters of that object. When these conditions are met, the stage advances to the final one.
- Completion stage: The mission is complete, and the pilot must click on COMPLETE> to end it, score it, and log it.

- L = Load (Pickup) Mission Type.
  - Load stage: This stage is skipped for a Pickup type mission.
  - Outbound stage: The pilot is required to fly to the defined engagement zone to transition to the Engaged stage.
  - Engaged stage: The pilot is required to successfully land the aircraft and come to a complete stop in the engagement zone. If the target name matches an object in the animations section, then the pilot is also required to be stopped within 50 meters of that object. When these conditions are met, the pilot will be prompted to open the door and authorize the load. Once that is done, an amount of weight in pounds computed by the addition of the cargo weight on the STAGE statement, plus the number of PAX multiplied by 200, is loaded and the stage is advanced to the Inbound stage. If the total amount of payload weight required is already loaded in the aircraft, nothing will be automatically loaded, and no weight will be removed. Only enough payload will be added to bring the weight up to the requirement for the mission.
  - Inbound stage: The pilot is required to leave the engagement zone and land at or near the destination defined in the STAGE statement. That destination will either be an ICAO code, or the name of an object defined in the Animations section. If an ICAO code, in order to advance to completion, the pilot must stop the aircraft on the ground within five nautical miles of the center of that airport. If the name of an object, the pilot must stop his aircraft within 50 meters of that object. When these conditions are met, the pilot will be prompted to open the door and authorize the unload. One this is done the weight defined by the cargo and PAX numbers is removed from the aircraft and the mission advances to the final phase.
  - Completion stage: The mission is complete, and the pilot must click on COMPLETE> to end it, score it, and log it.

- D = Drop (Airdrop) Mission Type.
  - Load stage: The pilot is required to open the main door and to take on the amount of weight in pounds defined by the Cargo and PAX parameters of the STAGE statement for this mission. PAX pounds are computed at 200 pounds per person (170 pounds plus 30 baggage.) If the total amount of payload weight required is already loaded in the aircraft, nothing will be loaded, and no weight will be removed. Only enough payload will be added to bring the weight up to the requirement. The Load stage ends when the pilot clicks Authorize Load and the weight is confirmed added.
  - Outbound stage: The pilot is required to fly to the defined engagement zone to transition to the Engaged stage.
  - Engaged stage: The pilot must open the door and overfly the target zone releasing the cargo by selecting DROP> from the menu or toggling the 'no smoking' switch if the aircraft has one. Once that is done, the weight is removed from the aircraft and the stage is advanced to the Inbound stage. Note: some airdrop missions may drop partial loads and multiple passes may be required.
  - Inbound stage: The pilot is required to leave the engagement zone and land at or near the destination defined in the STAGE statement. That destination will either be an ICAO code, or the name of an object defined in the Animations section. If an ICAO code, in order to advance to completion, the pilot must stop the aircraft on the ground within five nautical miles of the center of that airport. If the name of an object, the pilot must stop his aircraft within 50 meters of that object. When these conditions are met, the stage advances to the final one.
  - Completion stage: The mission is complete, and the pilot must click on COMPLETE> to end it, score it, and log it.
- R = Reconnaissance (or Survey or Loiter) Mission Type.
  - Load stage: This stage is skipped for a Recon type mission.
  - Outbound stage: The pilot is required to fly to the defined engagement zone to transition to the Engaged stage.
  - Engaged stage: The pilot may optionally be required, by the optional TIMED statement, to remain in the engagement zone for a specified amount of time. At the end of this time, if there is one, the pilot may depart the engagement zone to transition to the Inbound phase.
  - Inbound stage: The pilot is required to leave the engagement zone and land at or near the destination defined in the STAGE statement. That destination will either be an ICAO code, or the name of an object defined in the Animations section. If an ICAO code, in order to advance to completion, the pilot must stop the aircraft on the ground within five nautical miles of the center of that airport. If the name of an object, the pilot must stop his aircraft within 50 meters of that object. Once this is done the mission advances to the final phase.
  - Completion stage: The mission is complete, and the pilot must click on COMPLETE> to end it, score it, and log it.

- A = Attack Mission Type.
  - Load stage: The pilot will be prompted to VERIFY LOADOUT by the FCDU. This message is informational only. The Load stage ends when the pilot clicks PROCEED.
  - Outbound stage: The pilot is required to fly to the defined engagement zone to transition to the Engaged stage.
  - Engaged stage: The pilot is expected to find and attack the target given in the text but there are no specific requirements and the pilot may leave the engagement zone at any time to transition to Inbound.
  - Inbound stage: The pilot is required to leave the engagement zone and land at or near the destination defined in the STAGE statement. That destination will either be an ICAO code, or the name of an object defined in the Animations section. If an ICAO code, in order to advance to completion, the pilot must stop the aircraft on the ground within five nautical miles of the center of that airport. If the name of an object, the pilot must stop his aircraft within 50 meters of that object. When these conditions are met, the stage advances to the final one.
  - Completion stage: The mission is complete, and the pilot must click on COMPLETE> to end it, score it, and log it.

Note applying to all mission types: At all times during the mission the FCDU supplies an option to the pilot to ABORT the mission. If this is selected, and confirmed, the mission is discarded (regardless of the current stage and situation) *as if it had never been selected*. Any time the aircraft is on the ground and stopped, the option END will appear also. If this is selected and confirmed, the mission will be scored and logged at that point and terminated regardless of the current situation and stage. If all conditions for success are completed according to the rules above at that point, the score will reflect success, otherwise it will be the base score.

STAGE Statement

There is one STAGE statement and it must follow the TARGET statement and precede any optional or animation section statements. Note: there are plans to expand the system's capabilities and allow up to 99 stages in a single mission, but right now, there is only one allowed.

Stage parameter: In the current implementation this must be a numeric 1.

Start parameter: This determines the location where the mission stage will start. It does not have to be the same location where the mission was offered. This will either be a valid existing airport ICAO, or the name of an object or point defined in the animation section. If it is an ICAO it must be preceded by the symbol $. Whether an airport, point, or object this will resolve to a point defined by a latitude and longitude. If it is an ICAO or a defined point, the airplane must be within

5 nautical miles to be considered “at” the starting location. If it is an object, the airplane must be within 50 meters of the object's current location. Completion of the Load stage and meeting the start conditions described here will allow the transition to the Outbound stage.

Dest parameter: This determines the center point of the engagement zone for the mission. It should be the name of the target defined as the first parameter of the TARGET statement above it. It can also be an ICAO preceded by the $ symbol, but if this is done be sure that the ICAO is near the zone defined by on the TARGET statement. It cannot be the name of an object or point defined in the animation section. If you need to link the target to an animated object, have the TARGET statement use the name of the object, and use the target name here.

End parameter: This determines the location where the mission stage will end. It does not have to be the same location where the mission was offered. This will either be a valid existing airport ICAO, or the name of an object or point defined in the animation section. If it is an ICAO it must be preceded by the symbol $. Whether an airport, point, or object this will resolve to a point defined by a latitude and longitude. If it is an ICAO or a defined point, the airplane must be within 5 nautical miles to be considered “at” the starting location. If it is an object, the airplane must be within 50 meters of the object's current location. When the aircraft is stopped and in range (as defined above) of the end point, the transition to the Completion stage happens.

Type parameter: This is the type of mission in terms of the nature of the primary payload. This is the same as an FSCaptain flight type. 1 means a cargo flight, 2 means a passenger flight, and 3 is a ferry flight with neither. This is the type that will be logged in the FSCaptain log of your flights, however it has little meaning to the Assignments system. In the next two parameters you will specify exactly how much poundage is allocated to cargo, and how much to passengers.

PAX parameter: The number of passengers to be carried on this mission. Their status and how they are carried is not relevant to the Assignments system, the only meaning this number has is that it will add 200 pounds to the weight to be carried for each PAX, this represents a standard figure of 170 pounds for the human and 30 pounds for associated baggage. *None of the many rules applying to a standard FSCaptain passenger flight apply here.* This figure is purely used to determine weight to be loaded or unloaded, and to serve as a number to be logged with the record of the flight.

Cargo parameter: The quantity (in pounds) of cargo to be carried on this mission. *None of the many rules applying to a standard FSCaptain cargo flight apply here.* This figure is purely used to determine weight to be loaded or unloaded, and to serve as a number to be logged with the record of the flight. Note: both cargo and PAX numbers can be, and often are, included on the same mission definition.

Time parameter: this is not currently used but is reserved for the future.

Special parameter: On a regular FSCaptain flight we allow three types of "Special" cargo to be carried, and each type refers to a special requirement and, usually, to a higher score if the cargo is delivered successfully. In an assignment script we allow you to specify the FSCaptain special cargo code, and we log it when the flight is loaded. However, in an assignment, this has no effect on the scoring of the mission now. It is purely for documentation purposes in the log when Contracts are enabled. The special codes are 1 for a "Priority" flight, 2 for a "Fragile" flight, and 4 for a "Live" flight. The codes can be added together, for example, a "Live + Priority" flight would be a code 5. So, the valid numbers here are 0 to 7.

Optional Definition Statements

There are four optional statements that can be entered after the STAGE statement and before the animation section begins. The CAMPAIGN statement can only be used once in a mission if it's used at all. The TIMED statement can only be used twice if it is used at all. There can be an unlimited number of MESSAGE statements. The REQUIREMENT statement has no limit on the number of requirements you can place to make the mission valid. These statements can be in any order, but they must be grouped together between the STAGE statement and the first POINT statement of the Animations section.

CAMPAIGN Statement

This statement defines a name that refers to a file containing a database of information about a "campaign". This facility allows an unlimited number of assignments, even those offered at different airports, to be linked together in a dynamic campaign such that the results of one mission will affect the existence, priority, or contents of other missions.

The syntax is CAMPAIGN=[user-defined campaign name]

The way this works is that all the assignments that contain a CAMPAIGN statement that have the same user-defined name are all linked together. This name will be used to create or maintain a file in the FSCaptain\Contracts\Common folder. As missions are executed, information about the results of the mission are recorded in this file. Every other mission that is a part of that same campaign will read this file and adjust itself accordingly. For example, when any SimObject is destroyed or damaged during any mission, the name of that object is recorded in the mission's campaign file if it has one. Then any other mission that uses that same campaign name will never spawn that object again (or will spawn it in a damaged state.) Once hit it remains hit across missions. There are other facilities (see the REQUIREMENT statement) that allow linkage between missions also. We anticipate the abilities of the CAMPAIGN system will expand dramatically over time as the system develops.

MESSAGE Statement

This optional statement provides a name and a set of text for use by the MSG instruction that can be included in the ROUTE statement of the Animation section. See that definition for more information about how this works. The syntax of this statement is:

MESSAGE=[user-defined-name], TEXT=[user specified text message]

TIMED Statement

This allows you to optionally set a time limit in minutes for the mission, and if that number of minutes is exceeded the mission objectives are not considered met and the mission score bonus defined on the MISSION statement will not be earned. The TIMED statement has a slightly different syntax. It is TIMED=MINMINUTES=n and/or in another statement TIMED=MAXMINUTES=n. If either of these options are supplied, the mission timer that's located on the bottom of the FCDU will be colored to indicate the status of the timers. When the timer is colored green, you have not met the minimum requirement. When it is white, you have met the minimum and not exceeded the maximum. If it is red, you have exceeded the maximum.

MinMinutes parameter: If this parameter is supplied in a TIMED statement, then if the total time of the mission is lower than this number of minutes then the mission is not considered successful.

MaxMinutes parameter: If this parameter is supplied in a TIMED statement, then if the total time of the mission exceeds this number of minutes then the mission is not considered successful.

REQUIREMENT Statement

The purpose of this statement or statements is to set conditions that will prevent the presentation of the assignment on the assignments list. This is useful because almost every mission depends on certain conditions being present in order to make sense. A few of these conditions are already accounted for in the syntax of the preceding ASL statements: the volume class of the airplane must be within the limits set by the MISSION statement, and locations defined by the start, destination, and end parameters of the STAGE statement must actually exist in the sim world. But these optional requirements go far beyond those basic checks.

The basic problem is that every simulator installation is different and has different add-on scenery installed in it. But missions by nature depend on certain scenery to be where we expect it to be, or the whole thing doesn't make sense. So, we provide many ways to check and see if what you need for your assignment script is present in the target simulator. If not, no one there will see your mission on a list of possible assignments to accept.

There may be, and often are, multiple REQUIREMENT statements. The syntax of each one follows this pattern:

REQUIREMENT=[keyword]=[parameter value]

Whether there are one or many REQUIREMENT statements, each one must resolve to a true condition, or the assignment is not considered valid in the current sim environment, and it will not be seen on the list of missions available to the pilot given his current aircraft and environment.

Requirement=DOCUMENT=[name of a file in the user's "My Documents" folder]

This assignment is only valid if the file referenced by the path exists in the user's "My Documents" standard Windows folder. This may also be a path relative to that folder. Many FSX/Prepar3d add-ons place files in "My Documents" so you can check for them with this statement.

Requirement=FILE=[full or relative path to any file on the user's system]

This assignment is only valid if the file referenced by the path exists on the user's system. This may be a full path or a path relative to the current simulator's root folder. For a simple example, 'Requirement=FILE=Prepar3d.exe' would only be valid if the file "Prepar3d.exe is present in the sim's root. (Note: while this seems to distinguish Prepar3d from FSX installations, be cautious, as sometimes an add-on will place a dummy "Prepar3d.exe" in an FSX root, or; a dummy "FSX.exe" in the Prepar3d root!) This is a very useful requirement. It's used extensively in the assignments for Air America to check for a valid FSX@War installation, otherwise, those missions aren't visible as they depend heavily on the scenery and SimObjects supplied by that package.

Requirement=FSCAI

FSCAI is the component of FSCaptain that provides “Combat Intelligence”, that is, the ability of hostile SimObjects to target and shoot at other SimObjects including the user aircraft. Since FSCAI is always built-in to FSCaptain, this requirement doesn't check for it, it guarantees that it will run as a part of any mission that has this requirement statement in it. By default, FSCAI is only invoked on an “Attack” (Action=A in the Target statement) type assignment. But the designer may wish to use it for other missions too – such as a cargo delivery in hostile territory. It's the designer's responsibility in the Animations section (below) to provide the hostile SimObjects in the correct places for the mission, but they won't be able to shoot unless FSCAI is running. You can insure that's available with this statement. It's redundant for Attack type missions, but it won't hurt to include it.

Requirement=HELICOPTER

This assignment is only valid if the pilot is currently in a rotary-wing aircraft, as defined by the FSCaptain aircraft configuration file (not by the engine type.)

Requirement=MISSION=[ICAO],[number]

This assignment is only valid if the current pilot has logged the mission defined by [number] that's offered at the airport [ICAO] in the current airline's log. The mission only need be logged once to quality.

Requirement=MISSIONHITS=[ICAO],[number],[hits]

This assignment is only valid if the current pilot has logged the mission defined by [number] that's offered at the airport [ICAO] in the current airline's log. The mission only need be logged once to quality, plus, on the log entry for that mission, at least [hits] number of enemy objects must have been recorded as being destroyed or damaged.

Requirement=NOTSTOCK=[ICAO],[ICAO],...

This assignment is only valid if all the airports defined by the list of ICAO codes given as the parameters are add-on airports and therefore **not** the “stock” out-of-the-box version.

Requirement=PACK=[name of FSX@War pack]

This assignment is not valid unless the FSX@War pack named as the parameter is installed in the user's system in the “My Documents” folder. When using this statement, you should be careful to verify the exact name of the pack you are checking for, and be sure to include the .fwp extension on the name you give as this parameter.

Requirement=STOCK=[ICAO],[ICAO],...

This assignment is only valid if all of the airports defined by the list of ICAO codes given as the parameters are the "stock" out-of-the-box versions and not superseded by any add-on versions.

Requirement=SOURCE=[ICAO],[name of BGL file]

This assignment is only valid if all of the airport defined by the ICAO code given as the parameter originated in the file named as the second parameter. You should include the .bgl extension. This option allows you to be sure an airport is from a specific version. This is often necessary if you are making animations inside the airport property – you must be sure of the locations of all scenery to be sure your animations don't look stupid. If you want to use the stock airports (which everyone has) you can use the Requirement=STOCK to check for it. The SOURCE options allow you to check for any particular add-on version.

Requirement=TOTALHITS=[ICAO],[hits]

This assignment is only valid if the current pilot has destroyed or damaged at least [hits] number of enemy SimObjects in missions logged as having originated at the airport [ICAO].

The Animation Section Statements

The entire "Animation" section is optional, and only needed if you want to spawn and manipulate SimObjects as a part of your mission (or if the TARGET statement, or the Start or End parameters of the STAGE statement refer to objects, which then obviously must be defined here.)

There are other ways to spawn and move SimObjects other than this animations section that exist outside the entire FSCaptain system. One example is the FSX@War packages that spawn hundreds of SimObject targets that our missions interact with as if we spawned them ourselves. The ASL language is constructed such that it will function with SimObjects that are present in the world but not defined in the mission itself. However, most commonly, assignments use the facilities of the Animations section to assure things happen under strict control. Often the facilities of the Animations section can be used to add to an existing external system. We commonly do that with the FSX@War packs, in the scripts we provide as a part of standard FSCaptain.

The animation section starts with the first POINT statement. In general, after that, statements can be in any order, but certain statements must precede others, such as the POINT statement that defines a point in space must precede any ROUTE or OBJECT or EFFECT statement that references that point.

There are four Animation statements: POINT, ROUTE, OBJECT, and EFFECT.

POINT Statement

All animations involve the placing and perhaps the moving of objects. The indispensable basis of this process are the points where the objects are placed and moved to. The POINT statement defines precisely a point either in two-dimensional or three-dimensional space. Each point is given a name, and that name will be used later to reference the defined point. No animation can be done without points. Points are usually listed first in the Animation section since they are required to be listed before any statement that references them, but they can be placed anywhere. Some script writers choose to place groups of points, routes, objects and effects in sections. This is particularly true when using the COPY facility, described later on.

Points generally fall into two broad categories: two-dimensional (ground) points with an altitude of zero, and three-dimensional (air) ones with an altitude greater than zero. It's the altitude that distinguishes the two types. Ground objects that can't fly should always be assigned to ground-type points, and air units should normally be assigned to air-type units, unless the route being defined by the points is for a taxi.

The most tricky thing about creating scripts is obtaining the exact decimal coordinates of the points you want to define. Ultimately, this should be solved by a visual Assignment design program. But in this version, we have a facility of the FCDU that can build points and routes for you from the position of your aircraft in the sim. For details, consult Appendix B in this document.

Point parameter: This is the name of the point, it will be used in other statements to refer to the point, and it follows standard naming rules.

Lat parameter: in tandem with the Long parameter, this specifies in decimal coordinates the exact initial position of the point on the surface of the earth.

Long parameter: in tandem with the Lat parameter, this specifies in decimal coordinates the exact initial position of the point on the surface of the earth.

Altitude parameter: This parameter is optional, and if omitted it will assumed to be zero. If the altitude of a point is zero, that means is it "on the ground". Points used by any ground-bound vehicles or objects should always be zero. If this parameter is above zero, it is the altitude *above mean seal level.* These altitudes should only be used by aircraft in flight. It is valid for the first point of a route to be in the air – if so, the aircraft will be spawned in flight with its engines running and gear up (if possible) and it will immediately begin flying to the second point on its route. If any point is used by a ground vehicle and the altitude is not zero, it will be assumed to be zero.

Heading parameter: This parameter is optional and if omitted will be assumed to be 360 (true north). This number is a true heading from 1 to 360, and it will only be used when an object is initially spawned. The object, when spawned, will be facing this direction and this will not change unless the object is assigned to a route (see the ROUTE statement) that will cause it to move, and then its heading will reflect the direction it is moving in at all times.

Speed parameter: This parameter is optional, and if omitted will default to 20 knots for objects currently on the ground, and an appropriate cruise speed for airborne aircraft. If supplied, then the object's movement will try to achieve this speed. That may not be possible given the wide variety of environments and situations scripted objects can be involved in, but this speed is certainly a goal. This speed will be applied during a route (it only applies to objects on a route obviously, stationary objects have no speed) when the object reaches the point being defined here and it will continue in effect until changed by another speed parameter on another point along the route or until the route (or the object) ends.

ROUTE Statement

In its simple form a route is a named collection of points that defines a path along which a SimObject can be driven or flown. However, the power and flexibility of the ASL language comes from the many *instructions* that can be embedded into a route to change the behavior of the object being routed. An instruction can be substituted for any point name in any route at any time. Instructions are documented after the ROUTE statement itself is documented.

Route parameter: The is the name of the route, and the name must conform to the standard naming rules defined at the beginning of this specification.

Points parameter. This is a list of point names that make up the path any object assigned to this route will follow. Any point name can be replaced by an instruction which is enclosed in brackets to distinguish it from a route name. All point names in a route must have been defined on POINT statements that precede the ROUTE statement on which they are used. There may be up to 256 points and/or instructions in a single route. There is an instruction that switches from one route to another and this chaining can be done indefinitely, so there is no practical limit to the size of a route.

How routes work: When an object is assigned to a route by name, then it examines the first point on the route. If that's a route name, then the object begins immediately driving (if a ground point) or flying (if the object is an airplane and the point has an altitude) to the point. Once the object arrives at the point it advances to the next point in the list. If it never arrives, for whatever reason, the route is said to be "hung". Once there, the process repeats, it either drives or flies to the next point, and so on, until there are no more points. If it is a ground vehicle, it will simply stop and become static. Since airplanes can't stop, an aircraft will cycle back around and fly to the starting point of the route and continue repeating the route in that manner until something stops it.

If the object encounters an instruction instead of a point name, then what happens depends on the instruction. Often, an instruction will stop the route from advancing to the next point until some condition is met, but it may also take some action and then allow an advance. An instruction can even trigger a conditional or unconditional "jump" to another route. It's not uncommon to see strings of instructions between point names.

ROUTE Instructions

Instructions are the real "language" of ASL. Without them we would just have dumb objects following canned routes. These instructions give objects following routes the power to sense conditions and change their behavior based on what they know about their environment. They can even communicate with the pilot, or change, destroy, or spawn other objects or effects.

Every instruction can only be listed in a route in place of a point name. Some routes are even nothing but instructions and don't have any point names. This is valid. Each instruction is always enclosed in brackets [and] and if it has parameters they are separated with a semicolon. There are never spaces in an instruction. Only one instruction is permitted in each route step, separate instructions should be delimited by commas just as if they were point names.

Because space in route statements is scarce, instructions have short names.

Conditional Instructions

Generally, conditional instructions are waits, they stop the route from advancing until a condition is met. This is why the use of the word "Hold" in many of them.

HUC – Hold Until Close: Do not permit an advance of the route step until the user aircraft is within a number of meters of the object following the route, the number of meters is parameter 1. Example: [HUC;10000] – the route will hold until the user gets within 10 kilometers of the object executing this instruction.

HUF – Hold Until Far: Do not permit an advance of the route step until the user aircraft is a number of meters away from the object following the route, the number of meters is parameter 1. Example: [HUF;1000] – the route will hold until the user gets at least 1000 meters away from the object executing this instruction.

HUU – Hold Until Unload: Hold the advancement of the route until the pilot unloads the aircraft. This instruction has no parameters and only has meaning in Delivery or Pickup missions, because they are the only types that perform an Unload. If executing on any other type of mission this instruction will hang the route.

HUL – Hold Until Load: Hold the advancement of the route until the pilot loads the aircraft. This instruction has no parameters and only has meaning in Delivery, Pickup or Airdrop missions, because they are the only types that perform a Load. If executing on any other type of mission this instruction will hang the route.

HUD – Hold Until Drop: Hold the advancement of the route until the pilot performs an airdrop release. This instruction has no parameters and only has meaning in Airdrop missions, because they are the only type that perform a drop. If executing on any other type of mission this instruction will hang the route.

HUEO – Hold Until Exit Open: Hold the advancement of the route until any exit on the user aircraft is opened. This instruction has no parameters and is valid on any type mission.

HUEC – Hold Until Exits Closed: Hold the advancement of the route until *all* the exits on the user aircraft are closed. This instruction has no parameters and is valid on any type mission.

HUOD – Hold Until Object Destroyed: Hold the advancement of the route as long as the object named in parameter 1 exists in the sim world. That name is the name of an object defined on an OBJECT statement in this mission script. It is *not* the title of an actual SimObject in the sim world, although ultimately the script's object definition will resolve to one of those. Example: [HUOD;ENEMYHQ] this instruction will hold the route until the object named ENEMYHQ is destroyed, then the route's execution will continue.

HUOA – Hold Until Object Active: Hold the advancement of the route as long as the object named in parameter 1 *does not* exist in the sim world. That name is the name of an object defined on an OBJECT statement in this mission script. It is *not* the title of an actual SimObject in the sim world, although ultimately the script's object definition will resolve to one of those.

HUOC – Hold Until Object Close: Do not permit an advance of the route step until the object named in parameter 1 is within a number of meters of the object following the route, the number of meters is parameter 2. Example: [HUOC;T62TANK1;1000] – the route will hold until the object named T62TANK1 gets within 1 kilometer of the object executing this instruction.

HUOF – Hold Until Object Far: Do not permit an advance of the route step until the object named in parameter 1 is at least a number of meters of the object following the route, the number of meters is parameter 2. Example: [HUOF;RESCUEHUEY;1000] – the route will hold until the object named RESCUEHUEY gets at least 1 kilometer away from the object executing this instruction.

IFHIT - Do not permit an advance of the route step until a dropped object hits the target. This obviously only has meaning in Airdrop type missions, on any other mission it will hang the route. Note that if all the objects miss the target because of bad aim by the pilot the route is hung anyway. This instruction has no parameters.

WAIT – This simple instruction simply waits the number of seconds given in the first parameter and when that time has passed, the route will advance to the next step. An example: have a unit pause for one minute in its route at the last point it reached: [WAIT;60]

TOD - Do not permit an advance of the route step until the sim's *local* time of day clock is between the times defined by the first and second parameters. These two times are in 24-hour HHMMSS format with midnight being 000000 and one minute before midnight being 235959. The second parameter can be less than the first in which case 24 hours are added to the second parameter so that in effect it is in the next day. Example: [TOD;010000;023000] – The route will advance any time between 1:00:00 AM and 2:30:00 AM on any day (the day doesn't matter only the time) otherwise it will hold.

Imperative Instructions

These instructions cause things to happen, they are verbs; action commands. They do not hold the route but execute immediately and then advance to the next route step.

KILL – Destroy the object that executes this command immediately. Of course, this terminates everything the object was doing and no other step in the route for this object can be executed because it is now dead. This instruction has no parameters.

OBJUP – Object Up: Spawns the object named in the first parameter, which must be an object name defined on an OBJECT statement in the script for this mission. If the object already exists, the command is rejected.

OBJDN – Object Down: destroys the object named in the first parameter, which must be an object name defined on an OBJECT statement in the script for this mission. If the object does not exist, the command is rejected. Any route being executed by the destroyed object is terminated immediately, of course.

EFFUP – Effect Up: Starts the effect named in the first parameter, which must be an effect name defined on an EFFECT statement in the script for this mission. If the effect is already active, the command is rejected.

EFFDN – Effect Down: Removes the effect named in the first parameter, which must be an effect name defined on an EFFECT statement in the script for this mission. If the effect is not active, the command is rejected.

ROLE – Sets the 'Role' parameter for an object. The object is the name of an object defined on an OBJECT statement in this mission script, and the number role code is the second parameter. Objects are assigned 'Role' codes when they are created, this instruction can change that code on the fly. Example: [ROLE;TRUCKTRAP;22] the object named TRUCKTRAP has its role changed to 22, which is an enemy AA gun. Previously, it may have been assigned a role 20 when created, which is an unarmed enemy vehicle. See the OBJECT statement for a list of role codes and what they mean in the context of script execution.

PTADJ – Adjust Points: This code gives a script route the ability to add to, or subtract from, the points that will be awarded to the pilot upon completion of the mission. The adjustment is a positive or negative number between 1 and 15 and is given in the first and only parameter. Example: [PTADJ;+5] gives the pilot five more points on his score.

BOMB – Bombardment: This very powerful AI instruction gives the object executing it the power to bombard a defined circular zone. Explosion effects will be randomly generated within the defined zone at a rate defined by the parameters.

Parameter 1: name of the point that is the center of the zone.
Parameter 2: meters that define the radius of the zone from the center.
Parameter 3: The total number of explosion effects to spawn.
Parameter 4: the interval in seconds between explosions (in practice this will vary a bit.)

These explosions are not just effects. If they are randomly created near or on top of a SimObject they will damage or destroy the object as if it had been hit or near-missed by a bomb. The same rule applies to the user aircraft! Although the user aircraft can't be destroyed it can be severely wrecked by a direct hit, or damaged randomly by a near miss. Example: [BOMB,ZONE1,1000;200;60] This will bombard a zone up to 1000 meters around the point ZONE1 which must be a point *or object* defined in this mission script. 200 total explosions will be spawned at an average interval of 60 seconds between each one. However, in order to not make the process seem too predictable, the actual interval can vary as much as 25% between explosions.

BOMB commands can be given to ground units to simulate artillery, and to air units to simulate air bombardment. Used in conjunction with a carefully designed air route, this combination can accurately provide the visuals of an air strike, including the destruction of the targets of the strike, or not, based on random factors.

The actual effect started by the BOMB command is selected by an EXPLOSION statement, see that statement definition in this section for more details. EXPLOSION is optional, if not given, a medium potency explosion will default.

Control Transfer Instructions

This set of instructions allows you to transfer control to another route or another point in the current route. The effect of these are immediate and they are unconditional, although they can be preceded by a conditional instruction in the step before the one they are defined in.

XFR – transfer to another route: The current route for this object is terminated and the object immediately begins executing the route defined by the route name in parameter 1. Such route name must be defined in a ROUTE statement in the current mission script. Example: [XFR;RETREAT1] means to stop the current route and start the one named RETREAT1.

JUMP – jump to another route step: The current route begins execution at another step on the route immediately. The number is the first and only parameter. A route is a list of steps, each step being either a point name or an instruction. This command changes the current route step. Example: to start the route over: [JUMP;1] jumps immediately to route step number 1.

Communications Instructions

These allow simple text to be sent to the pilot, and some simple choices to be selected by the pilot from a small menu on the FCDU.

MSG – Send a Message: This instruction will send the named message that's given in parameter 1 to the pilot immediately though the FCDU's standard ACARS messaging system. The name refers to a named message defined in the optional statements of message mission script (see the MESSAGE statement for more details.)

SET – Set a Menu Option: There are three available menu spots on the FCDU's Communications page. This instruction activates the slot named in parameter 2 and places the text supplied by parameter 1 next to that spot. Example: [SET;<JTAC LEFT;1] will show "<JTAC LEFT" next to the first LSK available for menu selections. This is used in tandem with the HUR instruction next to solicit choices from the pilot that can affect the operation of the mission script.

HUR – Hold Until Response: The route will not advance to the next step until the pilot clicks on the LSK menu key on the Comm page that is defined by parameter 1 of this instruction. For an example of how this could be used, the following two instructions will hold a route until the pilot clicks the 2nd of the three LSKs devoted to menu input on the Communications page of the FCDU: [SET;<READY;2],[HUR;2]

OBJECT Statement

This is the statement that defines and creates a SimObject in the sim world to be used by the mission script. Each object must have a unique name that follows the standard naming rules. Being defined on an Object statement means the object will be created at its starting point at the beginning of the script unless the 'Odds' parameter contains a zero. What it does after than depends on if its assigned to a route. If not, it is a static object. If so, the point names and instructions in the route will control the activities of the object.

Because instructions are so useful, it is possible to define *virtual objects* – objects that do not spawn a SimObject in the sim world, but exist only in the script in order to execute instructions that may control other objects or effects. Such virtual objects are often called *controllers* because they themselves don't exist except as a kind of ghost, serving only to execute a route that has no point names in it, only instructions that spawn, or destroy or change other, "real" objects or effects. Using these virtual objects (there is no limit on the number of them you can create) can add incredible power to a script.

Object parameter: This is the name of the object, and this name must conform the standard object naming rules. This name could be used in a TARGET statement, or in an instruction embedded in a route.

Title parameter: This associates the object being defined with a "real" SimObject, or specifies that it is a virtual object. There are three possibilities:

- *Using a Symbolic Name*. This is the name of an object that is listed in the FSCaptain\Config\SimObjects_STD.cfg file, or the FSCaptain\Config\SimObjects_USER.cfg. If the title given in this parameter is in either of those two files, then the definition there will be used to resolve the actual SimObject title to be spawned. For more information about how this works see the SIMOBJECT statement in this documentation.
- *Using the actual title of a SimObject in the sim world*: this is not recommended, we urge you to use Symbolic names as described above, and if the name you want isn't defined in FSCaptain\Config\SimObjects_STD.cfg, define it using a SimObject statement either globally available to all scripts in FSCaptain\Config\SimObjects_USER.cfg, or in a SIMOBJECT statement in this mission script. One of the reasons this isn't recommended is that if for any reason the named object doesn't exist under that exact name, or can't be spawned, it will simply be missing from the script which could have bad effects if it's expected to do anything important and isn't there to do it.
- *The key word NONE*, which means there is no "real" object, this is a virtual object that will be assigned to a route that will execute instructions applying to the whole script.

Type parameter: This defines what type of object this is. Valid choices are:

- GROUND: this is a ground-bound vehicle that can move but can't fly.
- AIR: this is an aircraft capable of flying or moving on the ground.
- SEA: this is a ship or boat that can cruise on or under water.
- MISC: this is just a SimObject that can't do much except sit there, and execute instructions in any route assigned to it, of course, like all objects.
- CONTROL: this is one of those "virtual objects" with a title of NONE.
- DROP: this is a SimObject that will be dropped out of the aircraft on an Airdrop mission. (This type of object has a slightly different parameter set than the others.)

The remainder of the parameters of the OBJECT statement vary based on whether this is a DROP type object, or any of the others. First, we'll list the parameters for all other objects, since they are more common, then list the DROP object parameters in a separate section.

Parameters following the Type parameter for non-DROP type objects:

Point parameter: The name of a point defined by a POINT statement in this mission script that must precede this OBJECT statement. This point is where the object will be initially spawned, and where it will stay unless it is assigned to a route.

Odds parameter: A number from 0 to 100 that is the percentage likelihood that this object will be spawned in this mission. This allows a considerable degree of flexibility in making the same assignment operate differently different times the pilot runs it. Use of zero here allows a simple way to 'disable' an object in the script without having to delete it. Use of 100 means the object will always spawn on every run of the assignment. Any object that is assigned to a route that contains instructions for the mission should be assigned odds of 100, or you cannot be sure it will be there to execute those instructions (although note the instructions HUOA and HUOD allow you to test for the existence of objects and potentially adapt to the absence of an object.) Any number between 1 and 99 means the percentage chance of it spawning, with 1 being as rare as a 1% chance, and 99 having only a 1% chance of *not* being included.

Route parameter: This is the named route, defined by a ROUTE statement in this mission script, which will determine what the object does during the execution of the script. (See the ROUTE statement itself for much more detail.) This parameter is optional, if not supplied the object will remain static and inactive for the entirety of the mission – with one exception: if the FSCAI combat environment is active (either because of a REQUIREMENT=FSCAI statement in the script or because the target mission type is Attack) and this object is assigned as having a weapon then it will use that weapon even though it has no other role in the script.

Role parameter: This two-digit number can be used to assign different prepackaged roles to units in a combat scenario. The parameter is optional, and if not supplied is equivalent to writing Role=0. These are the role numbers currently implemented:

- Role 00: a neutral object belonging to neither side and having no special power.
- Role 10: a friendly object with no special power.
- Role 11: a friendly object capable of a FAC (Forward Air Control) role with only reporting capability.
- Role 12: a friendly object capable of a FAC (Forward Air Control) role with reporting and the ability to mark hostile targets or friendly forces with colored smoke.
- Role 13: a friendly object capable of a JTAC (Joint Tactical Air Control) role with the same features as role 12 but also the ability to mark targets with a laser to serve as a guide for Tacpack implemented laser guided weapons.
- Role 20: a hostile object with no weapon. This object can be seen and marked by the friendly FAC/JTAC.
- Role 21: a hostile object equipped with a gun-type weapon. This object can be seen and marked by the friendly FAC/JTAC.
- Role 22: a hostile object equipped with an AA-type weapon. This object can be seen and marked by the friendly FAC/JTAC.
- Role 23: a hostile object equipped with a flak-type AA weapon. This object can be seen and marked by the friendly FAC/JTAC.
- Role 24: a hostile object equipped with a missile-type AA weapon. This object can be seen and marked by the friendly FAC/JTAC.
- Role 29: a hostile object that is the primary target of the mission. This object can be seen and marked by the friendly FAC/JTAC.

These role codes do not grant the power to shoot guns to hostile objects – that is done solely via the FSCAI program and its configuration files. The purpose of these codes is to implement the FSCaptain FAC/JTAC feature. In a non-combat script, these roles are all pointless and should be set to zero or omitted.

Parameters following the Type parameter for DROP type objects:

Times parameter: This is the number of times that this object must be dropped to complete the Airdrop mission. It can be any number greater than 1. If the Manual parameter is 1 this will determine the number of passes the pilot must make over the target to drop their entire payload. On each drop, the total payload will be divided by this number and the result will be the amount released. If you only want one pass, code 1 here and the entire payload will be dropped on that one pass. If the manual parameter is zero, then the drop of multiple objects will be automatically sequenced by the Delay parameter on a single pass.

Manual parameter: Code a 1 if payloads are to be manually released, and 0 if they are to be automatically released.

Zone parameter: The zone in meters around the center of the target (defined in the TARGET statement) within which the dropped object will be scored as a "hit" if it falls within.

Delay: Delay in seconds between drops. If the Manual parameter is zero and the Times parameter is greater than 1, this determines the delay between drops of objects. This is optional and if omitted will be 1 second, meaning 1 object per second will be dropped until the number in the times parameter is exceeded.

EFFECT Statement

For our purposes, an effect is an invisible temporary object that can emit a defined particle effect. In practical terms, effects are used in the sim to visually represent moving particle-based things like smoke, fire, explosions, and water (as in waterfalls.) FSCaptain provides a number of standard effects that can be used in the scripts, plus, the script writer can also use any effect he or she has in the 'Effects' folder of their sim, or they can design their own effects. However, be aware that for an effect to be used it must obviously be in the 'Effects' folder of whatever sim is running your assignment script – so if you use any effect that isn't a standard part of the sim, or a standard part of FSCaptain, then you must be sure the effect itself is distributed and installed with your script.

Effects can be static, in that they are started by the EFFECT statement itself. It's also common to define an effect with an EFFECT statement but start it as inactive, then use an instruction in the route for an object to turn the effect on or off based on other things that are happening in the sim world.

Effect parameter: This is the name of the effect that can be referred to by other statements or instructions in the Mission script. This name must conform to the standard naming rules.

Title parameter: This is the title of the SimObject that has the effect attached to it, not the name of the effect itself. It can be any SimObject in the user's sim world. FSCaptain contains a list of 24 standard effect objects, listed in an appendix below.

Point parameter: This is the name of a point defined by a POINT statement in this mission script and which precedes this EFFECT statement. *Important: be sure any effect you intend to remain on the ground has a point that is defined with altitude zero.* Effect objects that last longer than a second in the air behave strangely in FSX.

Active parameter: This is a 1 if the effect is immediately active, or a zero if it is defined here but not yet started. Inactive effects can be started, or stopped, by instructions in a route, see the EFFUP and EFFDN instructions in the ROUTE statement definition.

Duration parameter: This is how long the effect will remain once it is activated when it's created. This parameter is only meaningful when the Active parameter is 1. If an effect is started by an EFFUP instruction, then the instruction will specify the duration and that will override any number coded here. This number is in 1/18th of a second "ticks". To have an effect last two seconds, code 36 in this parameter (2 times 18).

Optional Animation Statements

These statements are not required but can, in special circumstances, help out or provide important information the supplements the required statements. These optional statements can be placed anywhere in the Animation section.

SIMOBJECT Statement

This statement links a 'Symbolic Name' for an object to one or more “real” SimObject titles in the user's simulator environment. Symbolic names can be used in the OBJECT statement instead of having to find and code actual titles of SimObjects.

This system has two advantages: 1) it frees the script from dependence on a title that may change, and 2) it allows the system to pick one of a list prioritized list of “real” SimObjects that best fit the script, automatically. For example, a generic symbolic name of “Jeep” could be resolved first to a specific Jeep vehicle that is defined in the FSX@War default pack; but, if that object does not exist, it could fall back on the HumVee defined in the Prepar3d SimObjects list, but, if this script is running under FSX, it would default to the plain HMMV vehicle defined there – all this would happen without the script writer having to worry about it, all he needs is a “Jeep” and the system figures out the rest based on the definitions in the canned configuration files.
SIMOBJECT statements can be coded directly in a mission script but they are more commonly seen in two standard FSCaptain configuration files located in the FSCaptain\Config folder:

- The SimObjects_STD.cfg file: This contains a set of SimObject statements that define vehicles and other objects that any script can use.
- The SimObjects_USER.cfg file. Initially empty, this is a file where the user can put their own local SimObject definitions without having to put them in the STD file, which is subject to being replaced as FSCaptain changes and expands.

The SIMOBJECT statements in these two files are automatically incorporated in any script without having to be referenced directly. It's as if they were magically copied into your script but you don't see them. The contents of these files are shown in an appendix later in this document for your convenience.

The format, parameters, and purpose of the statements in the standard files and the one documented here is exactly the same.

If you do code a SIMOBJECT statement in your script and it is for the same symbolic name as one defined in the standard two files, your statement will override parameters in the standard for your script only.

SimObject parameter: This is the 'Symbolic Name' that can be used in place of a "real" title in the title parameter of the OBJECT statement when linking an internal object name to an external one. When any name is placed in an OBJECT statement the system consults all SimObject statements available to it (firstly by looking at any in the script itself, then looking in the SimObjects_USER.cfg file, then lastly in the SimObjects_STD.cfg file) and if a name is found that matches the name on the OBJECT statement, the system uses the SimObjectTitle parameter of the matching SimObject statement to resolve the symbolic name to an actual SimObject that exists in the user's system. More details on how it does this are in the 'SimObjectTitle' parameter description below.

Type parameter: This is a word that helps classify SimObjects into categories for organizational and reference purposes. It doesn't affect the object's actual characteristics in the sim at all – those are determined by the parameters of the OBJECT statement that refers to the object.

Class parameter: This is a word that helps classify SimObjects into categories for organizational and reference purposes. It doesn't affect the object's actual characteristics in the sim at all – those are determined by the parameters of the OBJECT statement that refers to the object.

SimObjectTitle parameter: This is a list of titles of SimObjects that potentially could exist in the target user's system. They should be written in priority order. The first one of these titles that is found will be the one selected to be included in the mission. If none are found, the object will not spawn, and an error message will be sent to the user's 'Messages.txt' file in the Contracts folder.

Actual titles of SimObjects can take two forms: the simple form, and the complete form. Any SimObject title in the list can be either form, depending on requirements. The simple form takes up less space and is earlier to read, but many SimObjects defined in the system can't be found using this simplified syntax, so we have the complete form as a necessary alternative.

The simple form can be used under these conditions only:

- The SimObject exists in the sim's SimObjects folder.
- It is placed in a sub-folder matching the **Type** parameter of the SIMOBJECT statement:
  - AIR type units are found in the Airplanes folder.
  - GROUND type units are found in the GroundVehicles folder.
  - SEA type vehicles are found in the Boats folder.
  - All other types are found in the Misc folder.
- The SimObject is defined by a 'sim.cfg' file and not an 'aircraft.cfg' file.
- The title of the object in 'sim.cfg' is *exactly* the same as the folder name where it is defined.

If all these conditions are not met, simply using the SimObject's title doesn't provide enough information to locate it. In that case, you must use the complete form.

The simple form is simply the title of the target SimObject. For example:

```
SimObject=US_Ambulance,Type=GROUND,Class=EMS,SimObjectTitle=VEH_AmbulanceUS
```

This SimObjectTitle simple form works because the actual definition of 'VEH_AmbulanceUS' is located in the GroundVehicles folder under SimObjects in the sim's root folder (that's where we look if the type is GROUND), the folder name that contains it is named VEH_AmbulanceUS, inside that folder the object itself is defined in a sim.cfg file, and the title of the object exactly matches the folder name it's defined in: VEH_AmbulanceUS. (This is a default vehicle in both FSX and Prepar3D, and many of those follow that pattern but some don't.)

Most add-on vehicles from third parties, all aircraft, and some default ones aren't so simple. Therefore, we have a more elaborate syntax that contains all the information needed to track down where they are defined:

The first part of a complete form is the path to the file that defines the object, relative to the SimObjects folder; followed by the filename that defines it (this must be either 'sim.cfg' or 'aircraft.cfg'); followed by the $ symbol; and then ended by the actual title of the object in the sim.cfg or aircraft.cfg file.

As an example:

```
SimObjectTitle=Airplanes\C172\aircraft.cfg$Cessna Skyhawk 172SP
```

This points to one of the variations of the default Cessna 172 distributed with FSX (but not Prepar3D) – this airplane is located in the Airplanes folder under SimObjects, and defined in a folder named 'C172' by an aircraft.cfg file, and the title of this particular variation in that file is `Cessna Skyhawk 172SP`. Note there are other variations defined in that same aircraft.cfg, so it's valid to use `Cessna Skyhawk 172SP Paint1` for example if that's the one you want.

There must be at least one, but there may be many more titles listed in this parameter. The system searches for each one on turn and selects as the one to use the first one it finds. In this way, you can list objects in a priority order from most desirable to least, and the system will select the highest one available to you. For example:

```
SimObject=L19,Type=AIR,Class=PLANE,SimObjectTitle=Airplanes\C172\aircraft.cfg$Cessn
a Skyhawk 172SP Paint1,Airplanes\PA28_180\aircraft.cfg$Piper Cherokee
180,Airplanes\FSX@War Pack Southeast Asia_T-28_Trojan\aircraft.cfg$FSX@War Pack
Southeast Asia_T-28D
```

This long SIMOBJECT statement is attempting to define an “L19” which was a military version of the small Cessna 305, used extensively by forward air controllers in the Vietnam conflict. It offers up three possibilities in priority order: 1) the default FSX Cessna 172, 2) the AI Cherokee 180 that's present in both FSX and Prepar3D, and finally, the T-28D aircraft that's distributed with the FSX@War Southeast Asia pack. On an FSX system, the script would probably use the default 172

(unless the user has removed it or changed its name) but on a Prepar3D installation - which doesn't include the default 172 from FSX - the script would use the Cherokee 180 (if the user hasn't removed for renamed it) and if neither of these is found, it will look for the T-28 (another FAC aircraft used in the same time period) in the FSX@War package. If none are found, the object won't spawn at all.

EXPLOSION Statement

This statement is optional, and it has only one purpose: to specify the title of an effect object to be used by the BOMB instruction when creating its bombardment. It's optional because by default a medium level explosion is used. But you may wish to have larger or smaller explosions in a particular bombardment invoked by BOMB.

The placement matters because all BOMB instructions will use the last title presented in an EXPLOSION statement. So be sure to place the title you want before the BOMB instructions that will use it. It will remain in effect until changed by another EXPLOSION statement so be aware of that if you have multiple BOMB instructions that use different explosion effect objects.

The syntax is:

EXPLOSION=[title of effect object]

Example:

```
EXPLOSION=AAM_Exp_Gen_Fire_MD
```

Appendix A: Debugging ASL Scripts

Developing a sophisticated script can be challenging and time consuming and it requires a great deal of testing. Just like any programming, the system interprets your instructions literally not what you intended, and when you try them out, they often require adjustments and re-tests.

We have several facilities to test your scripts by taking many time-saving shortcuts while they execute normally as far as they are concerned.

First, before we get into deep testing tips, be aware of how important the log is. In many situations where you are wondering what just happened, or didn't happen, and why, the log will answer your question right away. The Assignments system keeps its own separate log which is not the same as the detailed FSCaptain logs. It only contains messages pertaining to the Assignments system, and all are in clear English. To access your log, look at the 'Messages.txt' file located in the FSCaptain\Contracts folder.

If you expected to be offered an assignment but it is not showing up, the log will usually be clear about why it wasn't offered. If your objects are not doing what you expect them to do, again the log informs you of what instruction or task each object is doing. Also, all effects are listed with their status as the status changes. The log is incredibly helpful.

The biggest obstacle to testing is the time spent flying. But actually flying is not necessary to execute a mission! You can simply teleport from airport to airport (using the standard “Go To Airport” menu item) and the Assignments system will adapt to these abrupt changes if you are using the external FCDU – as if you had made the flight. (The internal gauge version will not handle teleports well due to timing issues – use the external FCDU.EXE to debug a script if you want to teleport from place to place.

In addition, unlike a standard FSCaptain flight, the Assignment flights will not reset the FCDU if you move in Slew mode. Everything will continue to function normally while you are slewing around. You can even leave an aircraft parked in space. The Assignments system does not care – it is goal oriented not procedural. This works in the internal as well as the external. To test an airdrop class script for instance you can slew over the target, drop the items, then slew to your destination – or even teleport there by using the “Go To Airport” menu item (if you are using the external FCDU.) Time acceleration up to 16X works fine too.

These features will save you an immense amount of time testing scripts.

Appendix B: Using the Route Builder Feature of the FCDU

Writing the many statements required to animate lots of objects is very tedious and error-prone. We have a small function that can be a big help in taking the tedium out of it.

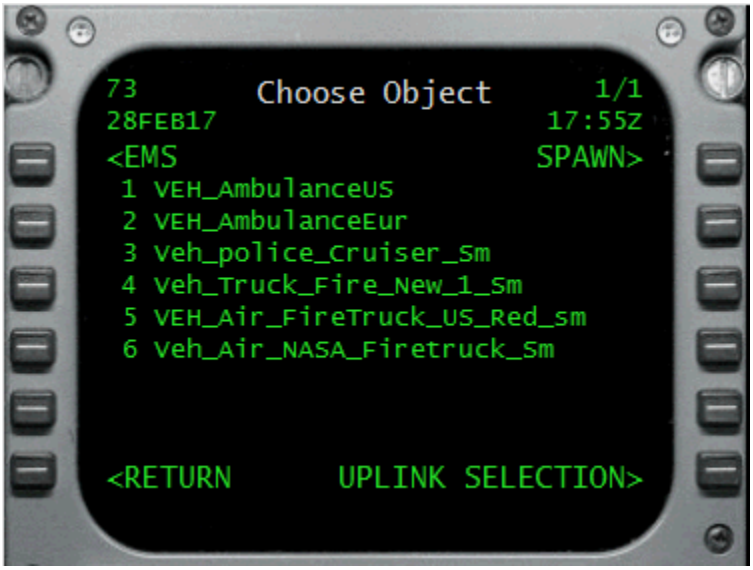
To access the Route Builder, get in an aircraft that's small enough to be used as a marker and guide in finding the right locations for your objects. (Travis favors the FSX trike.) Have the FCDU installed in it, or use the external one. Bring up the FCDU, login, and press the MENU button. On the lower right there is an option FLTSIM. Click on that and you are in the Route Builder.

This page is where you start. You may be tempted to press CREATE but don't yet. A route must have a unique name in a script. Enter the name (up to 10 characters – a route may have up to 15 but we will be appending things to it generating the route so we're limiting you to 10) in the scratchpad first before you select CREATE. That will be the name of the route you are building.

The KAVL.cfg in this example means that the route statements will be appended to the file KAVL.cfg in the FSCaptain\Contacts folder. This is because that's the airport I was at when I logged in to create this example. There is no other option at this time. It will add to any statements there and in any case, you will have to cut the route statements out and place them in your own script.

At the bottom you see your current latitude, longitude, altitude and heading displayed, but this is just for reference, you don't need to know these numbers to use Route Builder.

Once you click on CREATE having entered the name on the Scratchpad, the select page appears:



This is a menu of the possible SimObjects that can be defined for your route. They are organized by type. The first type, commonly used, is EMS type vehicles which you see here. Then there are types BUS, CAR, TRUCK, MISC, MIL, and SHIP. To change the type, click on the line select next to <EMS. It will cycle through all of them and back to EMS. This is so you don't have to page through a long list of SimObjects but can look them up by type.

This information comes from the file "Vehicles.cfg" in the FSCaptain\Contracts folder. You can modify it to add anything you want to it. As distributed it only contains stock SimObjects in FSX Acceleration, and not every one of those. The MIL category is special. These are objects that are weaponized by default via FSCAI. These can shoot at you.

If you aren't sure what a vehicle actually looks like but want to know, enter the vehicles menu number on the scratchpad and press the line select SPAWN>. A copy of the object will spawn in front of your airplane and you can see it. To get rid of it, the SPAWN> will turn into DESPAWN and clicking on it again will remove the object. This feature is just for making visual choices. Stock FSX/P3D contains many similar objects with different paints and looks.

To select the object you want to build the route for, enter its menu number and use UPLINK SELECTION>

This will move you on to the recording page. It will also record the current point your user aircraft is at, its position, altitude, and heading as the starting point of the route where the object will be spawned initially.

You are now in "Recording Mode". Simply slew or taxi your airplane to the next point in the route and press the RECORD> option. Then move to the next point and do it again. Each point is recorded. When you are done press <CLOSE.

The Route Builder will then generate a formatted set of POINT, ROUTE, and OBJECT statements for the object to drive this route. It will write them, in this case, to KAVL.cfg.

If you want a static object that doesn't have a route, just press CLOSE without ever recording a point. In that case, only one POINT and one OBJECT statement will be generated.

This is an example of a simple route that will taxi an airplane from Ramp 10 at KAVL to the hold short point at runway 16:

```
; ---- Route 'SAMPLE' by RouteBuilder ----
Point=SAMPLE01,Lat=35.440069,Long=-82.539503,Altitude=2169,Heading=71
Point=SAMPLE02,Lat=35.440184,Long=-82.539091,Altitude=2169,Heading=71
Point=SAMPLE03,Lat=35.439647,Long=-82.538764,Altitude=2169,Heading=165
Point=SAMPLE04,Lat=35.439124,Long=-82.540172,Altitude=2169,Heading=256
Point=SAMPLE05,Lat=35.438759,Long=-82.541402,Altitude=2169,Heading=250
Point=SAMPLE06,Lat=35.446609,Long=-82.545174,Altitude=2169,Heading=341
Point=SAMPLE07,Lat=35.446524,Long=-82.545660,Altitude=2169,Heading=250
Route=SAMPLE,Points=SAMPLE02,SAMPLE03,SAMPLE04,SAMPLE05,SAMPLE06,SAMPLE07,
Object=SAMPLE-EMS,Title=VEH_AmbulanceUS,Type=GROUND,Point=SAMPLE01,Odds=100,Route=SAMPLE
; ----------- End of Route -----------
```

All I did to make this was follow these instructions using the route name 'SAMPLE' and place my C172 at Ramp 10, then slew it around to each turning pointing in the route, recording each one. It took half a minute. I have it taxiing properly along the taxiways. But it will go where ever I make points. And I control the exact placement.

To use this in an actual script you'd cut and paste the various statements to their correct places – but at least the tedium of typing in coordinates and headings is taken care of for you.

Appendix C: Standard FSCaptain Effects

Here we present a simple list of the standard 'effect objects' you can use in ASL scripts with assurance these are present in every system that has FSCaptain 1.8 or higher installed. Credit: *Many of these effects were developed by the team at FSX at War and are used by permission.*

`AAM_Smoke_Blue` – a column of bright blue smoke.

`AAM_Smoke_Green` – a column of bright green smoke.

`AAM_Smoke_Red` – a column of bright red smoke.

`AAM_Smoke_Orange` – a column of bright orange smoke.

`AAM_Smoke_White` – a column of white smoke.

`AAM_Exp_Air_Burst` – an explosion bursting in the air.

`AAM_Exp_Air_Flak` – a burst of flak in the air.

`AAM_Exp_Gen_Fire_MD` – a medium explosion with a fire effect after it.

`AAM_Exp_Gen_Smoke_SM` – a small explosion with lingering smoke.

`AAM_Exp_Gen_Smoke_MD` – a medium explosion with lingering smoke.

`AAM_Exp_Grd_HG` – a "huge" explosion that leaves a crater image.

`AAM_Exp_Grd_LG` – a large explosion that leaves a crater image.

`AAM_Exp_Grd_MD` – a medium explosion that leaves a crater image (the default "BOMB")

`AAM_Exp_Grd_SM` – a small explosion that leaves a crater image.

`AAM_Muzzle_Flash` – a small flash of fire with smoke representing a gun shooting.

`AAM_Forest_Fire_SM` – a small fire about the size of a bonfire.

`AAM_Forest_Fire_MD` – a medium sized forest fire.

`AAM_Forest_Fire_LG` – a larger sized forest fire.

`AAM_Forest_Fire_HG` – a really pretty big sized forest fire.

`AAM_Oil_Fire_SM` – a small fire putting out dark black smoke.

`AAM_Oil_Fire_MD` – a medium fire putting out dark black smoke.

`AAM_Oil_Fire_HG` – a big fire putting out dark black smoke.

Appendix D: Standard FSCaptain SimObjects

Here we present a simple list of the standard SimObject 'Symbolic Names' you can use in ASL scripts with assurance these are present in every system that has FSCaptain 1.8 or higher installed.

This is a copy of the 'SimObjects_STD.cfg file at the present time.

```
SimObject=Generic_GA_Single,Type=AIR,Class=PLANE,SimObjectTitle=Airplanes\C172\airc
raft.cfg$Cessna Skyhawk 172SP Paint1,Airplanes\PA28_180\aircraft.cfg$Piper Cherokee
180

SimObject=Generic_GA_Twin,Type=AIR,Class=PLANE,SimObjectTitle=Airplanes\beech_baron
_58\aircraft.cfg$Beech Baron 58 Paint1

SimObject=C-130,Type=AIR,Class=PLANE,SimObjectTitle=FSX@War Pack Southeast Asia_C-
130_Hercules

SimObject=C-141,Type=AIR,Class=PLANE,SimObjectTitle=FSX@War Pack Southeast Asia_C-
141A_Starlifter

SimObject=EC-121R,Type=AIR,Class=PLANE,SimObjectTitle=FSX@War Pack Southeast
Asia_EC-121R_Constellation

SimObject=F-4J,Type=AIR,Class=PLANE,SimObjectTitle=Airplanes\FSX@War Pack Southeast
Asia_F-4_Phantom\aircraft.cfg$FSX@War Pack Southeast Asia_F-4J Phantom

SimObject=F-105D,Type=AIR,Class=PLANE,SimObjectTitle=Airplanes\FSX@War Pack
Southeast Asia_F-105D_Thunderchief\aircraft.cfg$FSX@War Pack Southeast Asia_F-
105_Thunderchief

SimObject=T-28D,Type=AIR,Class=PLANE,SimObjectTitle=Airplanes\FSX@War Pack
Southeast Asia_T-28_Trojan\aircraft.cfg$FSX@War Pack Southeast Asia_T-28D,Piper
Cherokee 180,Cessna Skyhawk 172SP Paint1

SimObject=O-1,Type=AIR,Class=PLANE,SimObjectTitle=Airplanes\C172\aircraft.cfg$Cessna Skyhawk
172SP Paint1,Airplanes\PA28_180\aircraft.cfg$Piper Cherokee 180,Airplanes\FSX@War Pack
Southeast Asia_T-28_Trojan\aircraft.cfg$FSX@War Pack Southeast Asia_T-28D


; ACME Vehicles

SimObject=Acme_Pickup,Type=GROUND,Class=TRUCK,SimObjectTitle=VEH_Air_NASA_Pickup_Sm

SimObject=Acme_CateringTruck,Type=GROUND,Class=TRUCK,SimObjectTitle=VEH_Air_NASA_Ca
teringTruck_Sm

SimObject=Acme_Pushback,Type=GROUND,Class=MISC,SimObjectTitle=VEH_Air_PushbackGrey

SimObject=Acme_BagCart,Type=GROUND,Class=MISC,SimObjectTitle=VEH_air_bagcart1

SimObject=Acme_BagLoader,Type=GROUND,Class=MISC,SimObjectTitle=VEH_Air_BagLoaderGre
y

SimObject=Acme_Deicer,Type=GROUND,Class=MISC,SimObjectTitle=GroundVehicles\VEH_Acme
_Iceman\aircraft.cfg$VEH_ACME_deicer

SimObject=Acme_Deicer_Spray,Type=GROUND,Class=MISC,SimObjectTitle=GroundVehicles\VE
H_Acme_Iceman\aircraft.cfg$ACME_deice_spray

SimObject=Acme_Deicer_Antiice,Type=GROUND,Class=MISC,SimObjectTitle=GroundVehicles\
VEH_Acme_Iceman\aircraft.cfg$ACME_antiice_spray
```

```
SimObject=Acme_AirStartUnit,Type=GROUND,Class=MISC,SimObjectTitle=VEH_Air_AirStartU
nit_sm

SimObject=Acme_GroundPowerUnit,Type=GROUND,Class=MISC,SimObjectTitle=VEH_Air_Ground
PowerUnit_sm


; Military Vehicles:

; Generic (used in FSC Scripts) - priority:  FSX@War, Prepar3d, FSX

SimObject=AA_GUN,Type=GROUND,Class=MIL,SimObjectTitle=Misc\FSX@WarPack1_VAB\sim.cfg
$FSX@WarPack1_VAB_BMP2_Savan_mov,BMP-1,VEH_Land_Humvee_Sm

SimObject=AA_TRACER,Type=GROUND,Class=MIL,SimObjectTitle=Misc\FSX@WarPack1_Jeep\sim
.cfg$FSX@WarPack1_Jeep_Pickup01_Desert_mov,Technical_B,VEH_flatbed

SimObject=AA_FLAK,Type=GROUND,Class=MIL,SimObjectTitle=Misc\FSX@WarPack1_Flak\sim.c
fg$FSX@WarPack1_Flak_KS30_Grey_ops,MRAP_x4,VEH_flatbed_rhino

SimObject=AA_SAM,Type=GROUND,Class=MIL,SimObjectTitle=Misc\FSX@WarPack1_SAM_SA13\si
m.cfg$FSX@WarPack1_SAM_SA13_Forest_mov,M901_Patriot,VEH_Flatbed_WithRhino_Sm

SimObject=AA_SIM_GUN,Type=GROUND,Class=MIL,SimObjectTitle=Misc\FSX@WarPack1_VAB\sim
.cfg$FSX@WarPack1_VAB_BMP2_Savan_mov,BMP-1,VEH_SafariTourist_02_Sm

SimObject=AA_SIM_TRACER,Type=GROUND,Class=MIL,SimObjectTitle=Misc\FSX@WarPack1_Jeep
\sim.cfg$FSX@WarPack1_Jeep_Pickup01_Desert_mov,Technical_B,VEH_truck_poacher

SimObject=AA_SIM_FLAK,Type=GROUND,Class=MIL,SimObjectTitle=Misc\FSX@WarPack1_Flak\s
im.cfg$FSX@WarPack1_Flak_KS30_Grey_ops,MRAP_x4,VEH_Land_Pickup_Poacher_Sm

SimObject=AA_SIM_SAM,Type=GROUND,Class=MIL,SimObjectTitle=Misc\FSX@WarPack1_SAM_SA1
3\sim.cfg$FSX@WarPack1_SAM_SA13_Forest_mov,M901_Patriot,VEH_safari_truck

SimObject=BLUE_JEEP,Type=GROUND,Class=MIL,SimObjectTitle=Misc\FSX@WarPack1_VAB\sim.
cfg$FSX@WarPack1_VAB_Humvee_Savan_mov,VEH_Air_PickupUS_Blue_sm

SimObject=BLUE_TRUCK,Type=GROUND,Class=MIL,SimObjectTitle=Misc\FSX@WarPack1_Truck\s
im.cfg$FSX@WarPack1_Truck_U375D_Cover_Forest_mov,Ural375_pickup,VEH_Air_NASA_Cateri
ngTruck_Sm

SimObject=BLUE_APC,Type=GROUND,Class=MIL,SimObjectTitle=Misc\FSX@WarPack1_VAB\sim.c
fg$FSX@WarPack1_VAB_EE11_Savan_mov,BRDM,VEH_Land_Humvee_Sm

SimObject=BLUE_TANK,Type=GROUND,Class=MIL,SimObjectTitle=Misc\FSX@WarPack1_Tank\sim
.cfg$FSX@WarPack1_Tank_Leclerc_Forest_mov,T-72,VEH_Air_NASA_Transbus_Sm

SimObject=BLUE_ARTY,Type=GROUND,Class=MIL,SimObjectTitle=Misc\FSX@WarPack1_Artiller
y\sim.cfg$FSX@WarPack1_Artillery_M109_Salad_Forest_mov,GroundVehicles\M270_Launcher
\sim.cfg$M270_Launcher_Forest,VEH_jetTruck

SimObject=RED_JEEP,Type=GROUND,Class=MIL,SimObjectTitle=Misc\FSX@WarPack1_Jeep\sim.
cfg$FSX@WarPack1_Jeep_UAZ469_Forest_mov,VEH_Air_PickupAsian_grey_sm

SimObject=RED_TRUCK,Type=GROUND,Class=MIL,SimObjectTitle=Misc\FSX@WarPack1_Truck\si
m.cfg$FSX@WarPack1_Truck_Kamaz_Cover_Green_mov,Ural375_pickup,VEH_Air_SupplyTruck_s
m

SimObject=RED_APC,Type=GROUND,Class=MIL,SimObjectTitle=Misc\FSX@WarPack1_VAB\sim.cf
g$FSX@WarPack1_VAB_BMP2_Savan_mov,BMP-1,VEH_Land_Humvee_Sm

SimObject=RED_TANK,Type=GROUND,Class=MIL,SimObjectTitle=Misc\FSX@WarPack1_Tank\sim.
cfg$FSX@WarPack1_Tank_T62_Grey_mov,T-72,VEH_Air_CaterTruck_Euro_sm

SimObject=RED_ARTY,Type=GROUND,Class=MIL,SimObjectTitle=Misc\FSX@WarPack1_Artillery
\sim.cfg$FSX@WarPack1_Artillery_BM21_Forest_mov,M1120_Launcher,VEH_jetTruck
```

```
; FSX@War Specific - with FSX Backups

SimObject=PAVN_ZU-23,Type=GROUND,Class=MIL,SimObjectTitle=Misc\FSX@War Pack
Southeast Asia_AAA\sim.cfg$FSX@War Pack Southeast Asia_ZU-23_ops,VEH_flatbed

SimObject=PAVN_KS-30,Type=GROUND,Class=MIL,SimObjectTitle=Misc\FSX@War Pack
Southeast
Asia_AAA\sim.cfg$FSX@WarPack1_Flak_KS30_Grey_ops,VEH_Land_Pickup_Poacher_Sm


; Prepar3d Stock Military Vehicles - with FSX backups

SimObject=Technical_B,Type=GROUND,Class=MIL,SimObjectTitle=Technical_B,VEH_Land_Hum
vee_Sm

SimObject=BMP-1,Type=GROUND,Class=MIL,SimObjectTitle=BMP-1,VEH_Land_Humvee_Sm

SimObject=BRDM,Type=GROUND,Class=MIL,SimObjectTitle=BRDM,VEH_Land_Humvee_Sm

SimObject=BTR60,Type=GROUND,Class=MIL,SimObjectTitle=GroundVehicles\BTR60\Sim.cfg$B
TR-60,VEH_Land_Humvee_Sm

SimObject=DES-500,Type=GROUND,Class=MIL,SimObjectTitle=DES-500,VEH_Land_Humvee_Sm

SimObject=HMMWV_Support,Type=GROUND,Class=MIL,SimObjectTitle=HMMWV_Support,VEH_Air_
PickupAsian_grey_sm

SimObject=M270_Launcher,Type=GROUND,Class=MIL,SimObjectTitle=GroundVehicles\M270_La
uncher\Sim.cfg$M270_Launcher_Forest,VEH_jetTruck

SimObject=M901_Patriot,Type=GROUND,Class=MIL,SimObjectTitle=M901_Patriot,VEH_Air_Su
pplyTruck_sm

SimObject=M1120_Launcher,Type=GROUND,Class=MIL,SimObjectTitle=M1120_Launcher,VEH_Ai
r_SupplyTruck_sm

SimObject=M1126_Stryker,Type=GROUND,Class=MIL,SimObjectTitle=M1126_Stryker,VEH_Air_
SupplyTruck_sm

SimObject=MRAP_x4,Type=GROUND,Class=MIL,SimObjectTitle=MRAP_x4,VEH_flatbed

SimObject=MRAP_x6,Type=GROUND,Class=MIL,SimObjectTitle=GroundVehicles\MRAP_x6\Sim.c
fg$MRAP_6,VEH_flatbed

SimObject=Odd_Pair,Type=GROUND,Class=MIL,SimObjectTitle=Odd_Pair,VEH_flatbed

SimObject=Parol-4,Type=GROUND,Class=MIL,SimObjectTitle=Parol-4,VEH_flatbed

SimObject=SA-2_Guideline,Type=GROUND,Class=MIL,SimObjectTitle=SA-
2_Guideline,VEH_flatbed

SimObject=T-72,Type=GROUND,Class=MIL,SimObjectTitle=T-72,VEH_flatbed

SimObject=Ural375_pickup,Type=GROUND,Class=MIL,SimObjectTitle=Ural375_pickup,VEH_fl
atbed

SimObject=Ural375_SA2_Loader,Type=GROUND,Class=MIL,SimObjectTitle=Ural375_SA2_Loade
r,VEH_flatbed

SimObject=Fan_Song_Radar,Type=GROUND,Class=MIL,SimObjectTitle=Fan_Song_Radar,VEH_fl
atbed

SimObject=Spoon_Rest_Radar,Type=GROUND,Class=MIL,SimObjectTitle=GroundVehicles\Spoo
n_Rest_Radar\Sim.cfg$spoon_rest_radar_green,VEH_flatbed

SimObject=AN_TPY_2_Radar,Type=GROUND,Class=MIL,SimObjectTitle=AN_TPY_2_Radar,VEH_fl
atbed
```

```
; EMS Vehicles
SimObject=US_Ambulance,Type=GROUND,Class=EMS,SimObjectTitle=VEH_AmbulanceUS
SimObject=EU_Ambulance,Type=GROUND,Class=EMS,SimObjectTitle=VEH_AmbulanceEur
SimObject=PoliceCruiser,Type=GROUND,Class=EMS,SimObjectTitle=VEH_police_Cruiser_Sm
SimObject=FireTruck,Type=GROUND,Class=EMS,SimObjectTitle=VEH_Truck_Fire_New_1_Sm

; Trucks
SimObject=LandmarkTruck,Type=GROUND,Class=TRUCK,SimObjectTitle=VEH_Air_SupplyTruck_
sm
SimObject=SewageTruck,Type=GROUND,Class=TRUCK,SimObjectTitle=VEH_Air_SewageTruck_sm
SimObject=GreyPickup,Type=GROUND,Class=TRUCK,SimObjectTitle=VEH_Air_PickupAsian_gre
y_sm
SimObject=WhitePickup,Type=GROUND,Class=TRUCK,SimObjectTitle=VEH_Air_PickupAsian_wh
ite_sm
SimObject=BluePickup,Type=GROUND,Class=TRUCK,SimObjectTitle=VEH_Air_PickupUS_Blue_s
m
SimObject=NASAPickup,Type=GROUND,Class=TRUCK,SimObjectTitle=VEH_Air_NASA_Pickup_Sm
SimObject=FlameTruck,Type=GROUND,Class=TRUCK,SimObjectTitle=VEH_jetTruck
SimObject=BlueWhitePickup,Type=GROUND,Class=TRUCK,SimObjectTitle=VEH_Air_PickupUS_B
lue_sm
SimObject=FuelTruckGeneric,Type=GROUND,Class=TRUCK,SimObjectTitle=VEH_Air_Fueltruck
_2_sm
SimObject=FuelTruckJet,Type=GROUND,Class=TRUCK,SimObjectTitle=VEH_air_FuelTruck_Jet
_Square
SimObject=FuelTruckRegular,Type=GROUND,Class=TRUCK,SimObjectTitle=VEH_air_FuelTruck
_Regular
SimObject=FuelTruckStandard,Type=GROUND,Class=TRUCK,SimObjectTitle=VEH_air_fueltruc
k01_sm
SimObject=CateringGeneric,Type=GROUND,Class=TRUCK,SimObjectTitle=VEH_Air_CateringTr
uckGrey_sm
SimObject=CateringEuro,Type=GROUND,Class=TRUCK,SimObjectTitle=VEH_Air_CaterTruck_Eu
ro_sm
SimObject=CateringNASA,Type=GROUND,Class=TRUCK,SimObjectTitle=VEH_Air_NASA_Catering
Truck_Sm

; Busses
SimObject=SilverAirTranBus,Type=GROUND,Class=BUS,SimObjectTitle=VEH_Air_TransBus_E_
Sil_sm
SimObject=WhiteAirTranBus,Type=GROUND,Class=BUS,SimObjectTitle=VEH_Air_TransBus_E_W
h_sm
SimObject=NASAAirTranBus,Type=GROUND,Class=BUS,SimObjectTitle=VEH_Air_NASA_Transbus
_Sm
SimObject=RedDoubleBus,Type=GROUND,Class=BUS,SimObjectTitle=VEH_RedDoubleBus_sm
SimObject=RedLuxuryBus,Type=GROUND,Class=BUS,SimObjectTitle=VEH_RedLuxBus_sm
```

```
; Cars
SimObject=Automobile,Type=GROUND,Class=CAR,SimObjectTitle=Automobile,VEH_Car_Mini1_
Black_sm
SimObject=BlackCar,Type=GROUND,Class=CAR,SimObjectTitle=VEH_Car_Mini1_Black_sm
SimObject=BlueCar,Type=GROUND,Class=CAR,SimObjectTitle=VEH_Car_Mini1_Blue_sm
SimObject=LightGreyCar,Type=GROUND,Class=CAR,SimObjectTitle=VEH_Car_Mini1_LightGrey
_sm
SimObject=RedCar,Type=GROUND,Class=CAR,SimObjectTitle=VEH_Car_Mini1_Red_sm
SimObject=WhiteCar,Type=GROUND,Class=CAR,SimObjectTitle=VEH_Car_Mini1_White_sm
SimObject=DarkGreenCar,Type=GROUND,Class=CAR,SimObjectTitle=VEH_Car_Mini2_DarkGreen
_sm
SimObject=BrownCar,Type=GROUND,Class=CAR,SimObjectTitle=VEH_Car_Mini3_Brown_sm
SimObject=JunkCar,Type=GROUND,Class=CAR,SimObjectTitle=VEH_Car_Mini3_Junker_sm
SimObject=BlueWhiteSUV,Type=GROUND,Class=CAR,SimObjectTitle=VEH_Land_SUV_Govt_Sm


; Misc Ground Vehicles
SimObject=AirStartUnit,Type=GROUND,Class=MISC,SimObjectTitle=VEH_Air_AirStartUnit_s
m
SimObject=GroundPowerUnit,Type=GROUND,Class=MISC,SimObjectTitle=VEH_Air_GroundPower
Unit_sm
SimObject=Snowplow,Type=GROUND,Class=MISC,SimObjectTitle=VEH_snowplow
SimObject=FarmTractor,Type=GROUND,Class=MISC,SimObjectTitle=VEH_tractorCab
SimObject=Snowcat,Type=GROUND,Class=MISC,SimObjectTitle=VEH_Snowcat_Sm
SimObject=NASAForklift,Type=GROUND,Class=MISC,SimObjectTitle=VEH_Air_NASA_Forklift_
Sm
SimObject=BlueForklift,Type=GROUND,Class=MISC,SimObjectTitle=VEH_Air_Forklift_blue_
sm
SimObject=GreyForklift,Type=GROUND,Class=MISC,SimObjectTitle=VEH_Air_Forklift_Grey_
sm
SimObject=WhiteForklift,Type=GROUND,Class=MISC,SimObjectTitle=VEH_Air_Forklift_Whit
e_sm
SimObject=YellowForklift,Type=GROUND,Class=MISC,SimObjectTitle=VEH_Air_Forklift_Yel
low_sm
SimObject=GenericBagCart,Type=GROUND,Class=MISC,SimObjectTitle=VEH_air_bagcart1
SimObject=GreyBagCart,Type=GROUND,Class=MISC,SimObjectTitle=VEH_air_bagcart_FlatNos
e_Grey_sm
SimObject=FlatnoseBagCart,Type=GROUND,Class=MISC,SimObjectTitle=VEH_air_bagcart_Fla
tNosed
SimObject=BlueBagLoader,Type=GROUND,Class=MISC,SimObjectTitle=VEH_Air_BagLoaderBlue
SimObject=GenericPushback,Type=GROUND,Class=MISC,SimObjectTitle=VEH_Air_PushbackGre
y
SimObject=GreyBagLoader,Type=GROUND,Class=MISC,SimObjectTitle=VEH_Air_BagLoaderGrey
SimObject=GreyEUBagTractor,Type=GROUND,Class=MISC,SimObjectTitle=VEH_Air_BagTractor
_Euro_Gray_sm
```

```
SimObject=WhiteEUBagTractor,Type=GROUND,Class=MISC,SimObjectTitle=VEH_Air_BagTracto
r_Euro_White_sm
SimObject=GrayUSBagTractor,Type=GROUND,Class=MISC,SimObjectTitle=VEH_Air_BagTractor
_US_Gray_sm
SimObject=WhiteUSBagTractor,Type=GROUND,Class=MISC,SimObjectTitle=VEH_Air_BagTracto
r_US_White_sm


; Ships/Boats
SimObject=GenericCarrier,Type=SEA,Class=SHIP,SimObjectTitle=VEH_carrier01_high_deta
il_sm
SimObject=GenericCruiser,Type=SEA,Class=SHIP,SimObjectTitle=VEH_cruiser01
SimObject=GenericDestroyer,Type=SEA,Class=SHIP,SimObjectTitle=VEH_destroyer01
SimObject=GenericCutter,Type=SEA,Class=SHIP,SimObjectTitle=VEH_USCG_Cutter_Sm
SimObject=GenericYacht,Type=SEA,Class=SHIP,SimObjectTitle=VEH_water_yacht_280ft_sm
SimObject=GenericFerry,Type=SEA,Class=SHIP,SimObjectTitle=wash_ferry_sm
SimObject=CigaretteBoat,Type=SEA,Class=SHIP,SimObjectTitle=VEH_CigaretteBoat_Sm
SimObject=FishingBoat1,Type=SEA,Class=SHIP,SimObjectTitle=VEH_water_fishingboatlg01
_sm
SimObject=FishingBoat2,Type=SEA,Class=SHIP,SimObjectTitle=VEH_water_fishingboatlg02
_sm
SimObject=WaterTanker1,Type=SEA,Class=SHIP,SimObjectTitle=VEH_water_tanker01_sm
SimObject=WaterTanker2,Type=SEA,Class=SHIP,SimObjectTitle=VEH_water_tanker02_sm
SimObject=WaterTanker3,Type=SEA,Class=SHIP,SimObjectTitle=VEH_water_tanker03_sm
SimObject=Sailboat1,Type=SEA,Class=SHIP,SimObjectTitle=VEH_water_sailboat1_sm
SimObject=Sailboat2,Type=SEA,Class=SHIP,SimObjectTitle=VEH_water_sailboat2_sm
SimObject=Sailboat3,Type=SEA,Class=SHIP,SimObjectTitle=VEH_water_sailboat3_sm
SimObject=Sailboat4,Type=SEA,Class=SHIP,SimObjectTitle=VEH_water_sailboat4_sm
SimObject=SmallBoat1,Type=SEA,Class=SHIP,SimObjectTitle=VEH_water_smallboat1_sm
SimObject=SmallBoat2,Type=SEA,Class=SHIP,SimObjectTitle=VEH_water_smallboat2_sm
SimObject=SmallBoat3,Type=SEA,Class=SHIP,SimObjectTitle=VEH_water_smallboat3_sm
SimObject=SmallBoat4,Type=SEA,Class=SHIP,SimObjectTitle=VEH_water_smallboat4_sm
SimObject=CargoShip1,Type=SEA,Class=SHIP,SimObjectTitle=VEH_water_cargoC_sm
SimObject=CargoShip2,Type=SEA,Class=SHIP,SimObjectTitle=VEH_water_cargoD_sm
SimObject=CargoShip3,Type=SEA,Class=SHIP,SimObjectTitle=VEH_water_cargoE_sm
SimObject=CargoShip4,Type=SEA,Class=SHIP,SimObjectTitle=VEH_water_cargoF_sm
SimObject=CargoShip5,Type=SEA,Class=SHIP,SimObjectTitle=VEH_water_cargoG_sm


; Containers
SimObject=Bucket,Type=MISC,Class=STATIC,SimObjectTitle=Bucket
SimObject=Generator,Type=MISC,Class=STATIC,SimObjectTitle=Generator_sm
SimObject=FoodPallet,Type=MISC,Class=STATIC,SimObjectTitle=Food_Pallet
```

```
SimObject=ShippingContainer,Type=MISC,Class=STATIC,SimObjectTitle=Shipping_Containe
r_sm

SimObject=ShippingCrate,Type=MISC,Class=STATIC,SimObjectTitle=TopSecret_Crate_sm

SimObject=CargoPod,Type=MISC,Class=STATIC,SimObjectTitle=Cargo_Pod

SimObject=SwimmerWaving,Type=MISC,Class=STATIC,SimObjectTitle=PPL_Fisherman1_sm


; Static aircraft (cannot be used as AI)

SimObject=A1_ops,Type=MISC,Class=STATIC,SimObjectTitle=Misc\FSX@War Pack Southeast
Asia_A-1\sim.cfg$FSX@War Pack Southeast Asia_A-1_ops

SimObject=A1_dam,Type=MISC,Class=STATIC,SimObjectTitle=Misc\FSX@War Pack Southeast
Asia_A-1\sim.cfg$FSX@War Pack Southeast Asia_A-1_dam

SimObject=A1_des,Type=MISC,Class=STATIC,SimObjectTitle=Misc\FSX@War Pack Southeast
Asia_A-1\sim.cfg$FSX@War Pack Southeast Asia_A-1_des

SimObject=A7_ops,Type=MISC,Class=STATIC,SimObjectTitle=Misc\FSX@War Pack Southeast
Asia_A-7\sim.cfg$FSX@War Pack Southeast Asia_A-7_ops

SimObject=A7_dam,Type=MISC,Class=STATIC,SimObjectTitle=Misc\FSX@War Pack Southeast
Asia_A-7\sim.cfg$FSX@War Pack Southeast Asia_A-7_dam

SimObject=A7_des,Type=MISC,Class=STATIC,SimObjectTitle=Misc\FSX@War Pack Southeast
Asia_A-7\sim.cfg$FSX@War Pack Southeast Asia_A-7_des

SimObject=C47_ops,Type=MISC,Class=STATIC,SimObjectTitle=Misc\FSX@War Pack Southeast
Asia_C-47\sim.cfg$FSX@War Pack Southeast Asia_C-47_ops

SimObject=C47_dam,Type=MISC,Class=STATIC,SimObjectTitle=Misc\FSX@War Pack Southeast
Asia_C-47\sim.cfg$FSX@War Pack Southeast Asia_C-47_dam

SimObject=C47_des,Type=MISC,Class=STATIC,SimObjectTitle=Misc\FSX@War Pack Southeast
Asia_C-47\sim.cfg$FSX@War Pack Southeast Asia_C-47_des

SimObject=F100_ops,Type=MISC,Class=STATIC,SimObjectTitle=Misc\FSX@War Pack
Southeast Asia_F-100\sim.cfg$FSX@War Pack Southeast Asia_F-100_ops

SimObject=F100_dam,Type=MISC,Class=STATIC,SimObjectTitle=Misc\FSX@War Pack
Southeast Asia_F-100\sim.cfg$FSX@War Pack Southeast Asia_F-100_dam

SimObject=F100_des,Type=MISC,Class=STATIC,SimObjectTitle=Misc\FSX@War Pack
Southeast Asia_F-100\sim.cfg$FSX@War Pack Southeast Asia_F-100_des

SimObject=F4_ops,Type=MISC,Class=STATIC,SimObjectTitle=Misc\FSX@War Pack Southeast
Asia_F-4B\sim.cfg$FSX@War Pack Southeast Asia_F-4B_ops

SimObject=F4_dam,Type=MISC,Class=STATIC,SimObjectTitle=Misc\FSX@War Pack Southeast
Asia_F-4B\sim.cfg$FSX@War Pack Southeast Asia_F-4B_dam

SimObject=F4_des,Type=MISC,Class=STATIC,SimObjectTitle=Misc\FSX@War Pack Southeast
Asia_F-4B\sim.cfg$FSX@War Pack Southeast Asia_F-4B_des

SimObject=UH1_ops,Type=MISC,Class=STATIC,SimObjectTitle=Misc\FSX@War Pack Southeast
Asia_UH-1\sim.cfg$FSX@War Pack Southeast Asia_UH-1_ops

SimObject=UH1_dam,Type=MISC,Class=STATIC,SimObjectTitle=Misc\FSX@War Pack Southeast
Asia_UH-1\sim.cfg$FSX@War Pack Southeast Asia_UH-1_dam

SimObject=UH1_des,Type=MISC,Class=STATIC,SimObjectTitle=Misc\FSX@War Pack Southeast
Asia_UH-1\sim.cfg$FSX@War Pack Southeast Asia_UH-1_des
```

```
; Static Objects from FSX@War

SimObject=FSW_Algeco,Type=MISC,Class=STATIC,SimObjectTitle=Misc\FSX@WarPack1_Algeco
_Small\sim.cfg$FSX@WarPack1_Algeco_Small_01_ops

SimObject=FSW_Barrack_Small,Type=MISC,Class=STATIC,SimObjectTitle=Misc\FSX@WarPack1
_Barrack_Small\sim.cfg$FSX@WarPack1_Barrack_Small_01_ops

SimObject=FSW_Barrel_Single,Type=MISC,Class=STATIC,SimObjectTitle=Misc\FSX@WarPack1
_Barrel_Single\sim.cfg$FSX@WarPack1_Barrel_Single_01_ops

SimObject=FSW_Barrel_Multiple,Type=MISC,Class=STATIC,SimObjectTitle=Misc\FSX@WarPac
k1_Barrel_Multiple\sim.cfg$FSX@WarPack1_Barrel_Multiple_ops

SimObject=FSW_Boxes_Small,Type=MISC,Class=STATIC,SimObjectTitle=Misc\FSX@WarPack1_E
verything_Small\sim.cfg$FSX@WarPack1_Everything_Small_Boxes01_ops

SimObject=FSW_Building_Big,Type=MISC,Class=STATIC,SimObjectTitle=Misc\FSX@WarPack1_
Building_Big\sim.cfg$FSX@WarPack1_Building_Big_Brick01_ops

SimObject=FSW_Building_Medium_Mil,Type=MISC,Class=STATIC,SimObjectTitle=Misc\FSX@Wa
rPack1_Building_Medium\sim.cfg$FSX@WarPack1_Building_Medium_Military01_dam

SimObject=FSW_Building_Medium_Civil,Type=MISC,Class=STATIC,SimObjectTitle=Misc\FSX@
WarPack1_Building_Medium\sim.cfg$FSX@WarPack1_Building_Medium_Civil01_ops

SimObject=FSW_Building_Medium_Office,Type=MISC,Class=STATIC,SimObjectTitle=Misc\FSX
@WarPack1_Building_Medium\sim.cfg$FSX@WarPack1_Building_Medium_Office01_ops

SimObject=FSW_Building_Medium_Antenna,Type=MISC,Class=STATIC,SimObjectTitle=Misc\FS
X@WarPack1_Building_Medium\sim.cfg$FSX@WarPack1_Building_Medium_Antenna01_ops

SimObject=FSW_Building_Medium_Radar,Type=MISC,Class=STATIC,SimObjectTitle=Misc\FSX@
WarPack1_Building_Medium\sim.cfg$FSX@WarPack1_Building_Medium_RadarSphere01_ops

SimObject=FSW_Building_Small_Civil,Type=MISC,Class=STATIC,SimObjectTitle=Misc\FSX@W
arPack1_Building_Small\sim.cfg$FSX@WarPack1_Building_Small_Civil01_ops

SimObject=FSW_Building_Small_Office,Type=MISC,Class=STATIC,SimObjectTitle=Misc\FSX@
WarPack1_Building_Small\sim.cfg$FSX@WarPack1_Building_Small_Office01_ops

SimObject=FSW_Container_Single,Type=MISC,Class=STATIC,SimObjectTitle=Misc\FSX@WarPa
ck1_Container_Single\sim.cfg$FSX@WarPack1_Container_Single_01_ops

SimObject=FSW_Container_Multiple,Type=MISC,Class=STATIC,SimObjectTitle=Misc\FSX@War
Pack1_Container_Multiple\sim.cfg$FSX@WarPack1_Container_Multiple_01_ops

SimObject=FSW_Factory_Small,Type=MISC,Class=STATIC,SimObjectTitle=Misc\FSX@WarPack1
_Factory_Small\sim.cfg$FSX@WarPack1_Factory_Small_Type01_ops

SimObject=FSW_Flak_KS30,Type=MISC,Class=STATIC,SimObjectTitle=Misc\FSX@WarPack1_Fla
k\sim.cfg$FSX@WarPack1_Flak_KS30_Grey_ops

SimObject=FSW_FuelTank_Big,Type=MISC,Class=STATIC,SimObjectTitle=Misc\FSX@WarPack1_
FuelTank_Big\sim.cfg$FSX@WarPack1_FuelTank_Big_Flat01_ops

SimObject=FSW_FuelTank_Medium,Type=MISC,Class=STATIC,SimObjectTitle=Misc\FSX@WarPac
k1_FuelTank_Medium\sim.cfg$FSX@WarPack1_FuelTank_Medium_Flat01_ops

SimObject=FSW_FuelTank_Small,Type=MISC,Class=STATIC,SimObjectTitle=Misc\FSX@WarPack
1_FuelTank_Small\sim.cfg$SXatWarPack1_FuelTank_Small_Flat01_ops

SimObject=FSW_GasTank_Medium,Type=MISC,Class=STATIC,SimObjectTitle=Misc\FSX@WarPack
1_GasTank_Medium\sim.cfg$FSX@WarPack1_GasTank_Medium_Sphere01_ops

SimObject=FSW_GasTank_Small,Type=MISC,Class=STATIC,SimObjectTitle=Misc\FSX@WarPack1
_GasTank_Small\sim.cfg$FSX@WarPack1_GasTank_Small_Cylinder01_ops

SimObject=FSW_Jeep,Type=MISC,Class=STATIC,SimObjectTitle=Misc\FSX@WarPack1_Jeep\sim
.cfg$FSX@WarPack1_Jeep_UAZ469_Forest_ops
```

SimObject=FSW_Tank,Type=MISC,Class=STATIC,SimObjectTitle=Misc\FSX@WarPack1_Tank\sim.cfg$FSX@WarPack1_Tank_T62_Grey_ops

SimObject=FSW_Tent_Small,Type=MISC,Class=STATIC,SimObjectTitle=Misc\FSX@WarPack1_Tent_Small\sim.cfg$FSX@WarPack1_Tent_Small_01_ops

SimObject=FSW_Truck,Type=MISC,Class=STATIC,SimObjectTitle=Misc\FSX@WarPack1_Truck\sim.cfg$FSX@WarPack1_Truck_U375D_Cover_Forest_ops

SimObject=FSW_VAB,Type=MISC,Class=STATIC,SimObjectTitle=Misc\FSX@WarPack1_VAB\sim.cfg$FSX@WarPack1_VAB_VBCI_Forest_ops

SimObject=FSW_WorkMachine,Type=MISC,Class=STATIC,SimObjectTitle=Misc\FSX@WarPack1_WorkMachine\sim.cfg$FSX@WarPack1_WorkMachine_Backhoe_Forest_ops

SimObject=FSW_Hut_Long,Type=MISC,Class=STATIC,SimObjectTitle=Misc\FSX@War Pack Southeast Asia_Hut\sim.cfg$FSX@War Pack Southeast Asia_Hut_Long_ops

SimObject=FSW_Hut_Open,Type=MISC,Class=STATIC,SimObjectTitle=Misc\FSX@War Pack Southeast Asia_Hut\sim.cfg$FSX@War Pack Southeast Asia_Hut_Open_ops

SimObject=FSW_Hut_Round,Type=MISC,Class=STATIC,SimObjectTitle=Misc\FSX@War Pack Southeast Asia_Hut\sim.cfg$FSX@War Pack Southeast Asia_Hut_Round_ops

SimObject=FSW_Hut_Stilt,Type=MISC,Class=STATIC,SimObjectTitle=Misc\FSX@War Pack Southeast Asia_Hut\sim.cfg$FSX@War Pack Southeast Asia_Hut_Stilt_ops

SimObject=FSW_Hut_Stilt,Type=MISC,Class=STATIC,SimObjectTitle=Misc\FSX@War Pack Southeast Asia_Hut\sim.cfg$FSX@War Pack Southeast Asia_Hut_Stilt_ops

SimObject=FSW_Boat_Cargo,Type=MISC,Class=STATIC,SimObjectTitle=Misc\FSX@War Pack Southeast Asia_Boats\sim.cfg$FSX@War Pack Southeast Asia_Cargo01_ops

SimObject=FSW_Boat_Fishing,Type=MISC,Class=STATIC,SimObjectTitle=Misc\FSX@War Pack Southeast Asia_Boats\sim.cfg$FSX@War Pack Southeast Asia_FishingBoat_ops

SimObject=FSW_AAA_ZU23,Type=MISC,Class=STATIC,SimObjectTitle=Misc\FSX@War Pack Southeast Asia_AAA\sim.cfg$FSX@War Pack Southeast Asia_ZU-23_ops

; Bipeds (Prepar3d versions 3 & 4 only)

SimObject=CivilianMale,Type=MISC,Class=STATIC,SimObjectTitle=Avatars\Civilian_Male_WhiteShirt_BlueJeans\sim.cfg$Civilian Male White Shirt Blue Jeans

SimObject=SoldierDesert,Type=MISC,Class=STATIC,SimObjectTitle=Avatars\Soldier\sim.cfg$Soldier Desert

SimObject=SoldierJungle,Type=MISC,Class=STATIC,SimObjectTitle=Avatars\Soldier\sim.cfg$Soldier Jungle

SimObject=SoldierSpecOps,Type=MISC,Class=STATIC,SimObjectTitle=Avatars\Soldier\sim.cfg$Soldier Spec Ops

SimObject=SoldierSwat,Type=MISC,Class=STATIC,SimObjectTitle=Avatars\Soldier\sim.cfg$Soldier Swat

SimObject=SoldierWinter,Type=MISC,Class=STATIC,SimObjectTitle=Avatars\Soldier\sim.cfg$Soldier Winter

SimObject=SoldierDesertM4,Type=MISC,Class=STATIC,SimObjectTitle=Avatars\Soldier\sim.cfg$Soldier Desert M4 Carbine

SimObject=SoldierJungleM4,Type=MISC,Class=STATIC,SimObjectTitle=Avatars\Soldier\sim.cfg$Soldier Jungle M4 Carbine

SimObject=SoldierSpecOpsM4,Type=MISC,Class=STATIC,SimObjectTitle=Avatars\Soldier\sim.cfg$Soldier Spec Ops M4 Carbine

SimObject=SoldierSwatM4,Type=MISC,Class=STATIC,SimObjectTitle=Avatars\Soldier\sim.c

fg$Soldier Swat M4 Carbine

SimObject=SoldierWinterM4,Type=MISC,Class=STATIC,SimObjectTitle=Avatars\Soldier\sim
.cfg$Soldier Winter M4 Carbine

SimObject=ArmyInfantry,Type=MISC,Class=STATIC,SimObjectTitle=Avatars\Army_Infantrym
an\sim.cfg$Army Infantry

SimObject=ArmyInfantryM4,Type=MISC,Class=STATIC,SimObjectTitle=Avatars\Army_Infantr
yman\sim.cfg$Army Infantry M4 Carbine

SimObject=NavyMaleBlue,Type=MISC,Class=STATIC,SimObjectTitle=Avatars\Navy_Male_Blue
Camouflage\sim.cfg$Navy Male Blue Camouflage

SimObject=RebelMale,Type=MISC,Class=STATIC,SimObjectTitle=Avatars\Rebel
Male\sim.cfg$Rebel Male

SimObject=RebelMaleAK47,Type=MISC,Class=STATIC,SimObjectTitle=Avatars\Rebel
Male\sim.cfg$Rebel Male AK-47

SimObject=RebelMale2,Type=MISC,Class=STATIC,SimObjectTitle=Avatars\Rebel Male
2\sim.cfg$Rebel Male 2

SimObject=RebelMale2RPG,Type=MISC,Class=STATIC,SimObjectTitle=Avatars\Rebel Male
2\sim.cfg$Rebel Male 2 RPG

SimObject=ArabMaleCasual,Type=MISC,Class=STATIC,SimObjectTitle=Avatars\Arab Male
Casual\sim.cfg$Arab Male Casual

SimObject=ArabMaleCasual2,Type=MISC,Class=STATIC,SimObjectTitle=Avatars\Arab Male
Casual 2\sim.cfg$Arab Male Casual 2

SimObject=ArabMaleCasual3,Type=MISC,Class=STATIC,SimObjectTitle=Avatars\Arab Male
Casual 3\sim.cfg$Arab Male Casual 3