A GUIDE TO CUSTOM SCENARIO BUILDING

Version 1.0 – by Shalbatana

Direct contributors:
SDF development team
Warlord
Peaceman
Beamweapons
(if I've forgotten you, remind me)

This document is a work in progress. It contains a compilation of all the posts made on the IFH forums about how to script and build custom missions and scenarios for the space simulator. There is an overview of scripting in general, and even some detailed sections on specific topics. It also includes reference sheets of all known commands to date. Finally, as we are all learning at the same time. Some of the information posted here may be sketchy or may even be inaccurate. This will only improve with time, so keep checking back for updates.

New in this version: Includes information from all posts through 2/01/06. New Layout. Reformatted function reference sheet. Tons of new information. Various other corrections and clarifications.

I. INTRODUCTION

At the moment, there is no GUI for creating scenarios for IFH. Therefore creating a custom mission in requires the use of scripting. Scripting basically means you use a programming language to write out a series of events for your mission. The scripting language for IFH is based on C++, but is actually a custom system designed for the game. However, it does not matter so much that you have programming experience in order to create a custom mission. It is more important that you can think logically and in a detailed manner. So while it is recommended that you have at least basic knowledge of how to program in some language or another, it is not essential. You can learn the theory as you go.

This document is was originally compiled using information gathered from forum discussions, the sample missions provided with the "Danger & Opportunity" prequel campaign, skirmish missions, player generated missions and experimentation. The document is a work of fans of the game, written as a small thanks for all their hard work, so please don't bother the SDF team about anything written here. Post all comments and suggestions to the forum.

The manual assumes you have a good knowledge of your computer. We will not cover how to use the programs mentioned throughout, nor will we be discussing how to create and copy files. This manual assumes that you know at least how to navigate to the IFH game folder and specify which program to use to open any given file.

CONTENTS

II. GENERAL FORMATTING

Because scripting for IFH is done in plain text characters, all you need to accomplish this task is Notepad or Wordpad. You can use a more advanced program like Word if you like, but it is not recommended… sometimes it's better to keep things simple. Those who wish to go even further in design may require a graphics editing program like Photoshop, Paintshop Pro, or Gimp.

There is a very specific syntax to the programming. It defines the Where, Who, What, When and How of the battle (the "Why" is left out of course, that is your storyline, and is laid out for your players when you write your mission briefings).

I'm sure if you're just learning you really want to see some of the actual code, so you can get a feel for how hard it will be to learn. Well rest assured that while there are some principals to learn, you will pick them up quickly, and you will not need to read this entire manual in order to create your own battles.

Every mission script will take on a virtually identical format. The scripting is divided up into little blocks of relevant information, or "structures" of code. When you are writing code, you denote a structure by enclosing it in brackets "{" and "}". Each structure defines one component of the situation that is to take place. Below is a listing of possible structures and what they do.

STRUCTURES	
STRUCTURE	DESCRIPTION
AIGOAL	For Defining or changing the target(s) and reasoning of any ship's artificial intelligence
ASTEROIDFIELD	For creating a horizon of rocks on the battlefield and describing its properties.
CAMERA	For Defining camera movement, angle and other properties. Used for cutaway scenes.
CUSTOMCUE	Allows the user to define a certain list of commands or triggers under a single executable cue.
DIALOG	For Defining a conversation between one or more characters and which files to access. Differs from "message" in that there is a conversation going on, and audio can be played with it.
EXECUTE	For triggering any game console command (use carefully!
FADEINFADEOUT	For Defining how the mission will handle a fade to or from black at a given time.
HEADER	For laying out the overview of the mission and its location
MESSAGE	For triggering a message to be sent to or from the player or another ship. And what files to access.
MODEL	For calling up a specific model for placement into the scene.
MUSIC	For setting up background music cues
OBJECTIVE	For Defining the target or goal of a ship's actions
PATH	For Defining the route a given object will follow
PROPERTIES	For altering the defined base elements of an object
SHIP	For Defining a ship or object, its contents, side and properties
SINGLESHIP	For creating a ship with its own team, or defining a single ship within a fighter flight
SUBTITLES	For adding and defining subtitles including when and how they appear and the appropriate .str file to access.
TEAM	For defining what parties will be involved in the battle.

You do not need all of these structures for every mission. Others you will need to use many times over, the ship structure will be needed at least once for each ship or group of ships. However, if you are making a battle that is not in or near an asteroid belt, there's no need to include the asteroid structure.

Let's jump right in and get a feel for what a mission script looks like. We'll look at some code and break it down as we go. This will give us an outline of what's needed to get started. I'm going to use a script posted on the forums (by Vash_the_Stampeede).

The header starts the script and introduces the overall framework around which the mission will be built. Here, line for line we are telling the computer that:

Code Sample:	***Structure Explanation:***
HEADER {	*The header section starts here*
rules single	*This is a single player rules mission (singleplayer missions have some different rules than multiplayer maps, even if only one person is playing).*
Environment vathan	*The battle will take place in the Narn V'than system.*
Name Test Mission	*The name for the scenario that IFH will reference in its menus.*
vid_near 6	*Specifies switching distance for graphics*
AmbientMusic EvilAmbient01.mp3	*At the start of the mission, play the music entitled "EvilAmbient01.mp3"*
StartFadedOut 1	*Start on a black screen*
Include0 team_attack.MAP	*Designates that the file "team_attack.MAP" should also be referenced when the computer is looking for instructions for playing this mission.*
Slot00 Ship:"Fighter1" Team:"EA"	*Designates that the ship the player will fly will be "Fighter1" and is on team "EA"*
}	*end the header section*

Each line of the structure carries one "function", "cue", and/or "variable" that defines one facet of the structure, and they go from broad statements to the specific as you go down. When you've laid out all the specifics, it's time to go to another structure.

So what are functions, cues, and variables? Think of functions as giving the computer an order to do or remember something. A Cue can be defined as a trigger of an event, defining what you want to happen, or what should happen at that moment, or tell of something that has already happened. A variable, of course is a value that changes over the course of the mission, and can be used to determine how the game should proceed when it has a choice of going in two different directions.

There is also one more piece of code we should mention. This is the "modifier". The modifier is basically the "adjective" of a command. It is a clarifier used with functions that have the potential to operate on many targets. You would use the modifier to tell the computer which of those targets to focus on. It is similar to a variable, though it cannot be altered to reflect a change. Note that modifiers can change based on situations in the game. If you used the modifier "nearship" for example, it would change based on which ship is actually closest to the player. The difference is the player or scenario builder cannot specifically set the value of the modifier,

where as with the variable they can. Modifiers will get their information from a specific instant in the game, at the time they are called, where as variables can be held over long stretches of time. You can find a sheet listing all known functions, cues and variables in the appendix.

Lets move on to the next structure. In this section we will determine the "who". This includes what ships, or objects are to be included in the area, which side they are on, and how they are likely to act. Though you can place the object definitions in any order, the best format to remember here is to define one side and all its units, then move on to the next.

The section below defines the overall characteristics of team 1.

Code Sample:	***Structure Explanation:***
TEAM {	*Begin a team declaration section*
Name EA	*Names the team for use in scripting*
Color 0 255 0	*Defines how their radar signals will appear to the player. Color is defined in RGB format, so 0,255,0 means use no red, full green, and no blue. 100% is the value 255 because we're talking a palette of 256 colors here.*
Hostiles Enemy	*The bad guys are any ship (or item) on team "enemy" and team EA wants to destroy them*
Neutrals Neutral	*Any ship [or item] from team "Neutral" is non-agressor to all who are on the EA team*
}	*End a team declaration section*

So logically, in the next set of structures we will define what ships or items make up the EA team. Remember that objects like boxes, satellites and stations can also be appropriated to a team, it doesn't have to be just ships.

Code Sample:	***Structure Explanation:***
SHIP {	*Begin a ship declaration section*
Name Skull Squadron	*Assigns a name to a group of ships or flight of fighters (in this case a fighter group of 12 ships)*
Player 1	*This ship is going to be controlled by player 1*
Class sa-23c	*This ship will be a sa-23c starfury class. We're telling the computer which ship the player will fly and which model to use.*
Skin default	*The ship will look like a generic starfury, there are no special markings*
Position 0 0 0	*Position the ship in a grid reference in relation to the system*
Rotation default	*Start the ships point of view rotated in the default direction*
Velocity default	*Set the ships starting velocity to a value (default is zero, or standing still)*
Formation X	*The ships this declaration refers to are in an "x" flying pattern.*

Cargo Pilot	*The ship carries a pilot onboard*
Team EA	*This ship belongs to team EA*
Count 12	*Designates how many ships in this fighter squadron*
ArrivalCue True	*The ship will arrive in the system at the start of the scenario*
ArrivalFrom Jumpgate	*The ship will arrive via jumpgate that is in the system*
AI Elite	*Set the artificial intelligence to "elite" level (obviously in this case AI only operates on the non-player controlled ships in the squadron)*
MinHull 0	*Specifies the minimum number a ship's hit points can fall to. If hull integrity falls to zero, the ship is destroyed.*
MissionCriticalCraft 0	*This ship will not show up in the mission critical list on the hud*
Afterlife 0	*When ship is destroyed, it is disintegrated…nothing remains*
Missiles ZR35	*this ship carries type "ZR35" missiles.*
}	*end the ship declaration section*

It may seem at first glance here that we're only defining one ship. However note the use of the "Count" function. We're actually setting up a flight of twelve fighters here. Of which player 1 will be a part. Later on we will define which ship belongs to the player using the *Singleship* structure.

Now we have to tell the computer how to handle this squad when no player is controlling it. Note that we are not telling it how good the AI should be, but only towards what end it should base its moves on, that is done in the ship declaration section above. Instead we are just going to tell the pilot what his orders are.

Code Sample:	***Structure Explanation:***
AIGOAL {	*Begin Artificial Intelligence goal declaration*
default	*Assign the generic goals to the AI*
}	*End AI assignment declaration*

The default AI goal here will simply by to attack the enemy whenever possible in the most efficient manner its AI setting will allow. At first glance one might think this should just be a single line added to the ship declaration. However you have the ability to add multiple goals, and conditionals here, so that each ships tactics can change depending on the battle situation.

You've probably noticed by now that each structure follows a specific pattern. It tries to define one aspect of the scenario, or one and only one event to take place.

Getting back to our example code here… Now that we've covered one ship in detail, lets breeze through the others, and point out any differences in the functions as they come up. Continue reading each line though until you become familiar with the patterns in the programming. I'm only going to put an explanation for anything we haven't seen before, or a new twist on something we have.

Code Sample:	***Structure Explanation:***
TEAM {	
Name Enemy	*Create a team with the name "Enemy"*
Color 255 0 0	*Appears red on players radar*
Hostiles EA	*This team actively dislikes the EA team*
}	
SHIP {	
Name EAS_Monolith	*use the name "EAS_Monolith" for this ship*
Class nova_md	*Use the nova_md model for this ship*
Skin ncg_monolith	*Use the monolith markings on this model*
Team Enemy	*Belongs to the team named "Enemy"*
AI Elite	*The computer will fly this ship as an elite pilot*
Position 0 0 8100	*Designates starting position in relation to the center of the system where the battle takes place.*
Rotation 0 1 0 D180	*specify starting orientation on ship*
ArrivalCue MissionTime:"" Delay:"14"	*arrive after 14secs have gone past*
MinHull 0	*is destroyed if hull falls to zero or below*
ArrivalFrom Hyperspace	*ship will arrive by making it's own jump vortex*
}	

Note the format of the "ArrivalCue" line. In this case a cue is what you use when you want to set up an event to take place at a certain time. There's the field, then the modifier in double quotes, then the specification of the modifier (when needed) in single quotes. This is the format needed for all cues. Cues often work very closely with variables. We'll talk about that later. For now, just consider know that the cues and variables are the "how" of our mission, as they describe how things are going to happen, and how certain triggers will effect other events.

The next chunk of code is rather long, take your time with it. Note any brand new commands, and how and when they are used. See you in about 5 pages!

Code Sample:	***Function Explanation:***
AIGOAL {	
Default	
}	
SHIP {	
Name EAS_Boreas	*Name this ship "EAS_Boreas"*
Skin ncg_boreas	*put the Boreas markings on it*

Class nova	*use the nova model for this ship*
Team Enemy	*belongs to enemy team*
AI Good	*this ship will not fight as smart as the monolith*
Position 2000 1000 8100	*set the starting location for the ship*
Rotation 0 1 0 D180	*note all enemy ships are facing direction*
ArrivalCue MissionTime:"" Delay:"10"	*arrives after 10secs game time*
MinHull 0	
ArrivalFrom Hyperspace	*arrives via its own jump engines*
}	
AIGOAL {	
Default	
}	
SHIP {	
Name EAS_Schwarzkopf_J	*names the ship "EAS_Schwarzkopf_J"*
Class nova	*use nova model*
Skin ncg_schwarzkopf_j	*use schwarzkopf model*
Team Enemy	*assigns team*
AI Average	*ship fights less intelligently than monolith & Boreas*
Position 2000 -1000 8100	*assign starting location*
Rotation 0 1 0 D180	*assign rotation and course*
ArrivalCue MissionTime:"" Delay:"11"	*Arrive 11 seconds into scenario*
MinHull 0	
ArrivalFrom Hyperspace	*arrives under its own jump engines*
}	
AIGOAL {	
Default	
}	
SHIP {	
Name EAS_Schwarzkopf_S	*2nd ship named "Schwarzkopf", designated "S"*
Class nova	
Skin ncg_schwarzkopf_s	*unique markings of course*
Team Enemy	
AI Average	*fights just as good as its sister ship*

Position -2000 1000 8100	*also in a new location*
Rotation 0 1 0 D180	*ship starts rotated 180 degrees in Y direction*
ArrivalCue MissionTime:"" Delay:"10"	*arrives 10 seconds in*
MinHull 0	
ArrivalFrom Hyperspace	
}	
AIGOAL {	
Default	
}	
SHIP {	
Name EAS_Pluto	*assigns the name "EAS_Pluto"*
Class nova	*This ship is s nova dreadnaught*
Skin default	
Team Enemy	
AI Average	
Position -2000 -1000 8100	*Starting position relative to [player?][playfield?]*
Rotation 0 1 0 D180	*faces player 1 again*
ArrivalCue MissionTime:"" Delay:"11"	*arrives after 10 seconds*
MinHull 0	
ArrivalFrom Hyperspace	
}	
AIGOAL {	
Default	
}	
SHIP {	
Name Fighter3	
Player 0	*This ship is controlled by the computer [why is this necessary sometimes, and not others?]*
Class sa-20	*a starfury*
Skin default	
Position default	
Rotation default	
Velocity default	

Formation Square	*Fly in a square formation.*
Team Enemy	
Count 18	*This ship is not a single fighter, but a squad of 18*
ArrivalCue MissionTime:"" Delay:"35"	*Arrives 35 seconds into mission*
ArrivalFrom Jumpgate	*it's coming in soon after player 1, and behind him!*
Cargo Pilot	
AI Good	*ships not player controlled are "good" fighters*
MinHull 0	
MissionCriticalCraft 0	
Afterlife 0	
Missiles NONE	
}	
AIGOAL {	
Default	
}	
TEAM {	
Name Neutral	*We're onto a 3rd team in this mission*
Color 0 0 255	*They show up as blue on radar*
Neutrals EA Enemy	*will neither help nor hinder EA team or enemy team*
}	
Note that if neutral ships are attacked by either side, they will remain neutral, unless you program a cue to change their allegiance.	
SHIP {	*remember, a ship structure is for objects too!*
Name Jumpgate	
Class jumpgate	*This object is a… (you guessed it)*
Team Neutral	*They do not wish to attack anyone*
AI Elite	
Position 0 0 –4000	*This ship is "behind" the player*
Rotation 0 1 0 d180	*it faces opposing the EA player*
ArrivalCue true	*ship starts in the system*
MinHull 100	*This ship can't even be damaged, let alone destroyed!*
}	

SHIP {	
Name EAS_Agamemnon	
Class Omega	
Skin EAS_Agamemnon	
Team EA	*This ship fights along with the player's squad against the enemy.*
AI Elite	
Position 2100 100 –5000	*arrival position*
Rotation default	*faces same as ea player (0 degrees)*
ArrivalCue MissionTime:"" Delay:"98"	*arrives after enemy*
MinHull 0	
ArrivalFrom Hyperspace	*via its own jump engine*
}	
AIGOAL {	
Default	
}	
SHIP {	
Name EAS_Damocles	
Class Omega	
Skin ocg_Damocles	
Team EA	
AI Elite	
Position -2100 100 –5000	*starts slightly behind and diagonal to Agamemnon*
Rotation default	
ArrivalCue MissionTime:"" Delay:"98"	*arrives same time as Agamemnon*
MinHull 0	
ArrivalFrom Hyperspace	
}	
AIGOAL {	
Default	
}	

Note this next ship has some new fields. These are coordinate systems that will define the position and formation the ship takes relative to player 1.

Code Sample:	***Function Explanation:***
SHIP {	
Name Fighter2	
Player 0	
Class sa-23e	
Skin default	
Position 0 0 0	*same location as player 1*
Row0 1 0 0	*Designates how ships are laid out (see "Rotation Matrix" section)*
Row1 0 1 0	
Row2 0 0 1	
Rotation default	
Velocity default	
Team EA	
Count 12	*there are 12 fighters in squad*
ArrivalCue true	*will start in the system*
Formation Circle	*starts in circle formation with squad*
Cargo Pilot	
AI Elite	
ArrivalCue MissionTime:"" Delay:"97"	*arrives same time as Agamemnon*
MinHull 0	
ArrivalFrom Jumpgate	*comes through jumpgate*
MissionCriticalCraft 0	*This does not show up in the critical list*
Afterlife 0	*The hull does not remain after ship is destroyed*
Missiles NONE	*this ship cannot launch missiles*
}	
AIGOAL {	
Default	
}	
SHIP {	
Name EAS_Perseus	
Player 0	
Class omega	
Skin ocg_perseus	

Position default	
Rotation default	
Velocity default	
Team EA	
Count 1	*There's one ship in this group (Assigning a group to one ship is useful because you can later target this ship with the group command)*
ArrivalCue true	*is in the system at beginning*
Cargo NoCargo	*If scanned, player will read that the ship has no valuable freight*
AI Elite	
MissionCriticalCraft 1	*This ship will show up in the mission critical list on the hud (you must take this ship out)*
Afterlife 1	*When destroyed, ship's hull will float around*
ArrivalCue MissionTime:"" Delay:"102"	
MinHull 0	
ArrivalFrom Jumpgate	
Missiles default	
}	

Initially the AI goals for all ships have been set to "default". However, we now are going to assign orders to each of these ships. They are grouped here, rather than interspersed within each individual ship above. This is mainly for programming convenience, but note that sometimes the order in which commands and goals are given can effect how the AI will react. You should always assign an initial AI when you first create a ship structure. By assigning the default values (in many cases, this is essentially a "null" command?), and then changing them later, you can regroup the AI structures so they are all in one spot in your code. That way if in troubleshooting your mission you decide some ships are misbehaving, you know where to look. Consider the AI goals as the "What" of the scene, they describe what is going to take place.

Code Sample:	***Function Explanation:***
AIGoal {	
OrderFor EAS_Perseus	*specifies the ship to receive orders*
Cue True	*ship has already been given orders at mission start*
AIType Pilot	*ship will interpret orders like a pilot [as opposed to?]*
GoalType Attack	*this ship will head into battle with the enemy*
Allegiance Hostile	*ship is an active aggressor to the enemy*
ShipType warship	*ship will handle as a warship*
Priority 80	*Sets the priority of this goal to 80% (compared to others this specific ship may have)*
}	

AIGoal {	
OrderFor EAS_Damocles	
Cue True	
AIType Pilot	
GoalType Attack	
Allegiance Hostile	
ShipType warship	
Priority 80	
}	
AIGoal {	
OrderFor EAS_Agamemnon	
Cue True	
AIType Pilot	
GoalType Attack	
; Anything after semicolon is ignored	*Just FYI, you can leave comments within your code simply by placing a semicolon on the line. Anything after it will by passed over by the computer.*
Allegiance Hostile	
ShipType warship	
Priority 80	
}	

Well, there's a complete script. It might seem to you that some things are missing, and that there should be a lot more involved. Remember though that the game itself handles all the fine details. With scripting, we just have to tell the computer that we want ship "A" to go from point 1 to point 2 and what to do if something comes up along the way. The IFH game itself will figure out how to make it happen technically. It already has the models and artificial intelligence built in, so there's no need to worry about that. Your only goal in scripting is to tell it which models and AI to use. The sample script above should be considered of moderate length for a simple battle. Of course the length of scripts will vary greatly depending on the scenario you wish to play out, and the complexity will vary greatly.

If you're new to scripting, then right now you're probably thinking that this is all too much. I know there's a lot of information to take in at once, programming is not easy, but don't get discouraged yet. It is not necessary to learn every line given above and commit it to memory. There are plenty of reference charts at the end of this manual if you're looking for a specific piece of information. With time the information will grow on you. What you should understand at this point is that writing a scenario follows a specific structure that is flexible in some places, and firm in others. It's like writing a television script in that you have to tell the characters, locations and beginning, middle and end of your scenario. Of course it differs in that you have multiple potential endings. You should have an idea of the elements that need definition in your scenarios, such as environment, teams, objects, goals, etc. And finally you need to be the

director, adding sound, camera angles and all the rest as needed. You should also know that there are specialized commands that must be used to get your code to do what you want. Sometimes leaving out something like a quotation mark, or capitalizing a letter that should be lowercase will mean the difference between a script that blows the player's mind, and one that won't even load.

That said, at the end of the day your ability to realize the mission you envision will be determined strictly by how well you know how the coding, and how detailed you get with the fine points of your programming. There is a lot more to designing than just laying out two sides and telling them to fight. Although it seems like a lot so far, we've only scratched the surface here.

III. VARIABLES, CUES AND LOGIC STATES

Throughout the mission events will change depending on players actions and how the battle is going. You will want the computer to recognize these changes and react accordingly. This is done through the use of variables and cues.

As stated earlier, variables are simply items that have states that change, and cues are triggers for programming commands. A cue can be triggered directly by your programming (like using the "arrival" cue at the beginning of a battle to make a ship appear), or indirectly through a variable (like telling a ship to arrive only if another ship has been destroyed).

Variables in their purest form can be broken down into categories by mathematics and logic. The simplest is a binary variable, meaning the event is true or has happened (so it's value, or "flag" is set to "1", meaning "yes" "true" or "on"); or it is false or has not happened yet (so it's flag is set to "0", meaning "no", "false", or "off"). Note that all values are set to zero by default, but programming itself will obviously change that. If you write a script and place a ship in a system, then obviously a variable that checked there were any objects on the playfield would be set to "1".

The next variable one needs is trinary. These are useful for determining outcomes. Here a value of zero is used to mean that the outcome is undecided, and that the action is still in progress. A value of one could then mean that the outcome was positive or successful. If an outcome was negative, or unsuccessful however, then we would set the value to "-1". Using this method you could set the value for a "kill" variable to "0" to say the player is battling a ship. If the player destroys that ship, the computer should set the value to "1", and this would trigger a happy ending to play. On the other hand if the enemy ship destroys a player, the computer knows to play a sad ending because the value was changed to "-1" the minute the player failed his mission.

The last type of variable would contain anything of four to an infinite number of states. These are good for queries like "how far is one ship from another?", to which the variable would be set to the distance between them. Another variable could be how many ships are left on the map, and if that value falls below 50% then the rest surrender. This type of variable can be an integer, a percentage or a decimal as needed.

All cues are programmed into the IFH system, though it is also possible to create custom cues. Custom cues are useful for creating and referencing a series of checks for a multitude of factors with just a single line. For more information on creating custom cues, check out Chapter V.

Cues can be referenced and activated directly or you can set them to activate when a condition is met (or stops being met), and finally use them as conditionals for structures. When choosing to use a specific cue for a situation, think very carefully about how events are really triggered, how they interact and how they can be avoided. It will help prevent bugs.

Before we get to that though, know that you will undoubtedly eventually come across a situation where you want to combine and/or modify cues and variables. This is done with logic expressions such as "AND", "NOT" and "OR". These are basic mathematical expressions that will become invaluable when scripting more complex situations. You can find plenty of information on how and why each of these work in any basic electronics manual. Here we will restrict our discussion to how they work for programming IFH.

If you were to say that a mission has been completed for better or worse you might say:

Code Sample:
Cue Completed “mainmission” NOT:”0”.

If you wanted to say that a ship was on the battlefield at one time, but is no longer there for whatever reason, you could say:

Code Sample:
((ARRIVED:"xxx") and (DESTROYED:"xxx")) or (GONE:"xxx")

Note that if you add *not:"1"* to a cue then it will invert the output of the cue. If the normal output of a cue would be true, it will return false and vice versa.

Code Sample:	
Cue Arrived	*would return false*
cue arrived not:”1”	*would return true*

If you want an objective that fails when you stray from the path just:
Objective

Code Sample:
failurecue proximity:"player" object:"path" distance:"?" not:"1"

Finally it can negate a cue. If you wanted to set ship xxx to a value of having never arrived in a system, you would say:

Code Sample:
ARRIVED:"XXX" NOT: "1"

This particular example is kind of silly of course because if the ship does not start out in the system, then arrived is “0” anyway, and once it is there, arrived would remain “1” even if the ship departed (as if to say “the ship had arrived at some point in the past regardless of if it is present or not now”). So this particular example might only be used for one of those vintage Star Trek moments, where the alien waves his hand and the enterprise disappears, and everyone thinks it had never existed. Nevertheless “NOT” in all its forms is extremely powerful, and you will eventually find a need for negating a cue.

The actual cue can be based on virtually any piece of relevant information. Some cues are based on conditions of a ship or group, others are based on a general battle situation or event, still more can be based on the actions of the player. There is a list of cues in the appendix section, however I’m going to go through them here in greater detail. Since these are the driving force of change in your scenarios, it is important that you understand what they do and how to use them.

SHIP / GROUP STATUS CUES

These cues help you determine actions for the game based on the status of a ship or group of ships. They can be useful for things like "if a raider comes within 1000 km of a freighter, it will start to attack.

All these CUES can be applied to flight groups and individual ships:

Destroyed
Applies to: objects
Meaning: returns "True" if the ship or the group of ships has been destroyed (not flown away in hyperspace)

Code Sample:
ActivateCue Destroyed:Beta

Gone
Applies to: objects
Meaning: returns True if the ship or the group of ships has left the battlefield. It does not matter if it has been destroyed OR flown away in hyperspace.

Code Sample:
OBJECTIVE {
Name KillTransport
Description custom_simple03_Obj2
Class Primary
Cue Destroyed:"Columbia"
FailureCue Gone:"Alpha"
}

Arrived
Applies to: objects
Meaning: returns TRUE when the specified Ship/Group appears on the battlefield

Code Sample:
Arrived:"Alpha"

returns TRUE if all ships from Alpha group have arrived

Scanned
Applies to: objects
Meaning: returns True if the ship or the group of ships has known Cargo (is scanned). That means, either the ship was declared with default "NotScanned" variable, or the ship was scanned by player when "NotScanned 1" was specified.

Code Sample:
OBJECTIVE {
Name Scan
Description custom_simple03_Obj1
Class Primary
ActivateCue True
Code Sample 2
Cue Scanned:"Columbia"
}

Scanned:"Beta" Percent:"50"
;returns TRUE if 50% of Beta group have been scanned

Completed
Applies to: Objectives
Meaning: returns true if specified Objective was successfully completed, otherwise returns False.

Code Sample:
ActivateCue Completed:"KillTransport"

Failed
Applies to: Objectives
Meaning: returns true if specified Objective was not achieved, otherwise returns False

Code Sample:
ArrivalCue Failed:"ProtectMonolith"

NotInProgress
Applies to: Objectives
Meaning: returns true if specified Objective was failed OR completed, otherwise returns False

Code Sample:
NotInProgress:"KillEveryone"

Proximity
Applies to: objects
Meaning: returns True if the ship(s) are at or within the specified distance to other ship(s).
Usage: Proximity:"FirstObject" Object:"SecondObject" Distance:"Distance value"

Code Sample:
AIGoal {
OrderFor Beta
Cue Proximity:"Beta" Object:"Columbia" Distance:"2100"
AIType Pilot
GoalType Attack
Allegiance Hostile
ShipType Transport
Priority 60

Docked
Applies to: All ships
Meaning: returns TRUE when the specified Ship/Group finishes docking

Code Sample:
NotInProgress:"KillEveryone"

EnteredHangar
Applies to: All ships
Meaning: returns TRUE when the specified Ship/Group enters hangar on ship or station

Code Sample:
EnteredHangar:"Gamma 3" Not:"1"

;returns TRUE if Gamma 3 has NOT entered hangar

HULL STATE CUES

These cues are based on the condition of a ship. It can help you do things like making a raider quit fighting and run if its hull drops below 30%.

HullBelow
returns TRUE when the specified Ship's hull drops below specified value(%).

HullAbove
returns TRUE when the specified Ship's hull stays specified value(%).

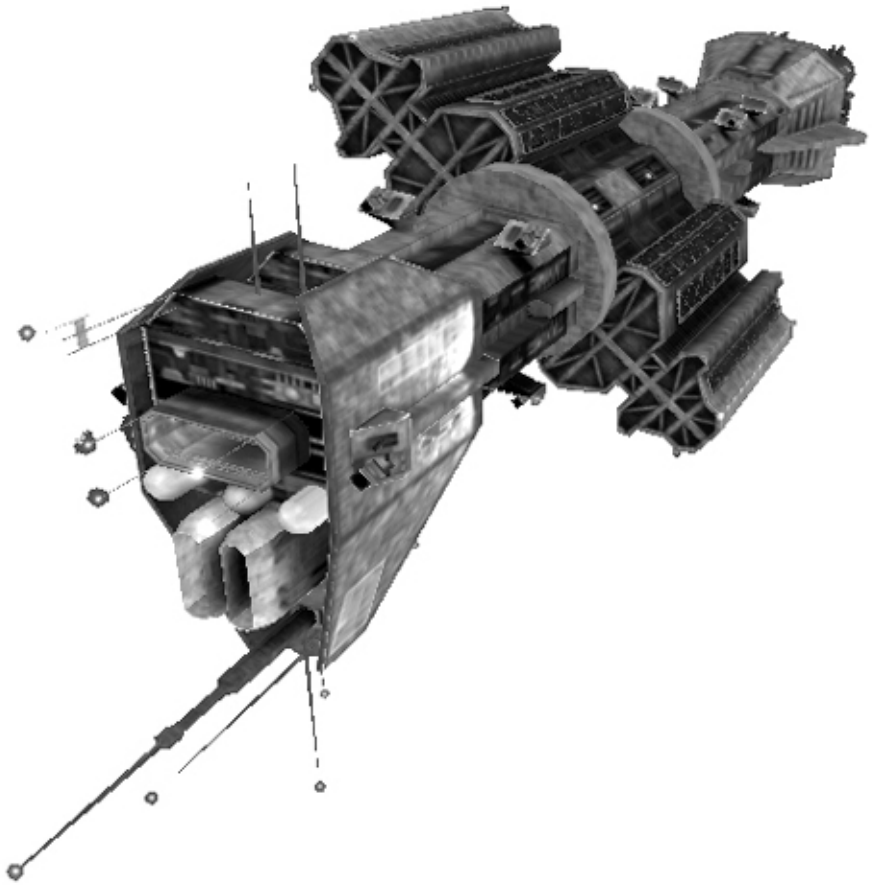
HullBelowOrEqual
returns TRUE when the specified Ship's hull drops below or equal to the specified value(%).

HullAboveOrEqual
returns TRUE when the specified Ship's hull stays above or equal to the specified value(%).

Code Sample
HullBelowOrEqual:"Alpha 1" Percent:"10"

;returns TRUE if Alpha 1's hull dropped to 10 percent or less

There are more cues that you will come across as you go. Generally they all follow a format similar to one of the cues outlined above. Be aware of cues that have similar meanings. They will give similar results in some circumstances, but not in others. “Gone” is similar to “destroyed”, and in many cases will return the same answer, but they are not interchangeable. Although a ship that is destroyed is always gone, a ship that is gone is not always destroyed. Logic quirks like these can lead to some bugs becoming hard reveal.



IV. SCRIPTING A MISSION

DON'T START FROM SCRATCH

When scripting a custom mission, there are a few main folders you will need to access regularly. These locations house almost all the information relevant to a specific mission or scenario. They are:

..\DATA\MISSIONS\
..\DATA\SCRIPTS\
..\DATA\SOUNDS\
..\DATA\MAPS
..\DATA\SKINS **(check these)**

They will be located in your IFH folder, which is wherever you chose to install it, most likely in C:\program files. If these folders do not yet exist, you can create them manually. In truth, you can create any folders for organization, as long as they all reside in the "data" folder, and are referenced properly in your scripts. This means you need to specify any subfolders they are in (using the "data" folder as your starting point), and are case sensitive with all names. So now that we know where to put all of our files, let's get started making a script.

The good news is that a lot of the typing has been done for you already, so there's no need to start with a completely blank page. Instead of starting from scratch, you can either copy a sample mission (located in the "missions" folder) and rename it or generate a skirmish mission in the game, and then rename it. The former will have varied AI's and goals laid out for you to work off of. The latter has the advantage of having virtually all the ships you want already created with all necessary information. So if you have a minimal amount of ships in your scenario, work off a sample mission. If however, you're just going for one big brawl with lots of ships, use the skirmish approach.

For a real timesaver, generate a skirmish mission with the same amount of each type of ship that are to appear in your script and save it. You can then go in and customize them, with all the essentials already laid out. If you plan on having more ships then than the skirmish will allow, you can just make two skirmishes and combine them. Then paste in any parts you want from other custom missions as well.

Remember too that there's also a bundle of sample scripts towards the back that were created to achieve a specific purpose. You can lift these samples right out of the manual and modify them for your own scripts. If someone else's mission performs a function similar to what yours requires, you can lift pieces from them and modify them to your needs. There are plenty of missions on the forums to lift from.

If you use a large chunk as-is, it is customary to give a line of thanks to the original author in your file notes. This is not by any means mandatory however, and you should feel free to do it or not at your own discretion.

Using this method, it is possible to create an entire mission and have done very little typing at all. Your efforts may even be limited to changing team names and AI Goals. The more unique, creative and grand your mission is however, the more you will probably have to do.

So now let's get in to some of the specific details about the sections of coding. We'll cover a little bit of every major element of the game. Of course you do not need to use all of these in every mission, only the one's you need.

CHOOSE A LOCATION

Most environments have little effect on gameplay other than looks. Hyperspace of course has some properties the others don't. Simple will just leave you in the middle of random space with no planets or nebulas around. Scheffer belt is set up more for an asteroid field. Many of the others have planets, nebulas, moons, or some other scenery that can make a good backdrop to a battle. Also note that some of the systems have the option of displaying either 2d or 3d planets.

Really the only other things your chosen location will influence are continuity things like the amount of beacons available when jumping into hyperspace. In the end though it mostly comes down to a matter of preference. Any system is as good as another for a space battle.

Check the environments chart in the appendix for a list of all known systems and their general appearance.

DETERMINING WHICH SHIPS TO USE

As with environments, determining which ship classes and types to use is really more of a design choice that a logical one. Choose the ships and ship types based on what you want your scenario to be. Be aware however that the limitations of certain ships may force some compromises in your choices. Hyperions, are more window dressing/ cannon fodder/ a place to land starfurys right now than anything else, as they currently have no weapons systems built into them. Omegas are required if you want beam weapons, and thunderbolts will not dock on most ships. Keep these things in mind when building your teams.

Check the Ship Classes chart in the appendix for a list of all known ships, their strength and quirks.

ADDING MUSIC

Music can be played once through or looped continuously until told to stop. You can also use cues to change the music based on the action taking place. Although a list of the included music options is included in the appendix, I would recommend just going straight to the music folder and playing each piece until you find the ones you want.

[Adding more music options may be as simple as dropping your new .mp3 file into the music folder, this is unconfirmed however and may not work.]

ADDING SOUNDS AND AUDIO

In addition to having your dialog and messages show up as subtitles on the screen or in the communications window, you can also add voices and sound effects to your missions. To do this you should use the "sound" function at the same time you put up the text.

Code Sample:
MESSAGE {
From Instructor
Ghost 1 ;Set to "1" if no such object in the scene
Text Inst_t03_034
Sound Speech/Inst_t03_034.mp3
Cue Destroyed:"Crimson" Percent:"50" Delay:"3"
}

This Checks the "Speech" folder for a file named "Inst_t03_034.mp3" and plays it when the instructor sends the text message with the same name (minus ".mp3" of course). If you use this command in a structure without a text cue, you have a way to make special effects. For example, play a loud knock while jolting the player's ship diagonally down and they were instantly hit by something.

All sounds must be placed in the IFH\DATA\SOUND folder. It is also a good idea to place them in a subfolder named "SFX" or "speech" so they are all in one place. Weather you want to keep all your dialog in one folder and your effects in another is up to you, as long as they are both in the DATA\SOUND folder to start with.

Look at the sample missions, and at the missions constructed by forum members for examples of implementation of these features.

ADDING SUBTITLES FOR DIALOG AND COMMUNICATIONS:

Subtitles must be laid out in an ".str" file which is once again, a simple text file with some coding embedded in it. You call on this file using the 'Text" function. It is also a good idea to use the "Ghost1", which we'll get in to in a minute.

There are two different types of communications, messages and dialogs. You can use a "message" to send a communication to a ship, like passing on an order to the player from a lead ship. However using "dialog" allows full conversations to take place. This is more useful for cut scenes and plot development. Messages will go straight to your comm. System and should be considered an inter-ship email of sorts. Thus these messages are usually not accompanied by

audio, and since you are already reading it, there's little need for subtitles either. Dialog on the other hand comes in over your headset, and can consist of both incoming and outgoing conversation. Subtitles are best used here to clarify the dialog that's taking place.

When posting communications, one must provide many pieces of information: The sender, the receiver(s), the location of the sound file to be used (unless using a persona, you must provide your own sound recordings), the text file to use for the subtitle(s), and the text to place in the comm. System. You may not need every one of these in every circumstance, but consider them all each time and use what is appropriate.

There are three other factors that may or may not effect communications. The first is the above mentioned "persona". The game comes with several characters that were used in the prequel campaign. It is possible to reference this character information (including the recorded voices). However, because the information is embedded in the game resource file, you cannot manipulate them as you would your own recordings, they are what they are. Personas should be assigned in the ship structure using the following method:

Code Sample:

Persona PoloStation

The second is timing. Let's say you program that at the time the player destroys a zephyr, Delta 1 will say "great, now head back to the barn". However, when the time comes, you find that the zephyr has already destroyed Delta 1. Well to prevent Delta 1 from talking from the land of the dead, you can use the "Ghost1" command. Inserting this will tell the computer that if your speaker is destroyed, the dialog should come from some other valid source.

The third influential factor is that you may want to include subtitles as a means of setting (without dialog or communications). At the start of the mission you might want to say "2248: the V'than system". Then you may simply use the text command. Use this for short subtitles. That way you can insert the text into the script without making an entirely separate text file.

Look at the sample missions, and at the missions constructed by forum members for examples of implementation of these features.

CAMERA OPERATIONS:

Camera operations for cut scenes in IFH are also set by giving a few general pieces of information and letting the computer do the rest. To set up a shot you need to tell the computer where to put the camera, where to point it, how to frame, hand how if at all any of these change.

First you should name your cameras. Usually it's best to given them an order based on the sequence in which you will use them. So we'll name our first camera "camera1". Next we'll place the camera with the usual coordinate system X Y and Z. If you want the camera to physically move in a certain direction then you set the velocity of travel for each axis. If you do not want the camera to move you can set everything to zero or leave the line out entirely.

Now just because we've placed our camera and told it to move in a certain direction, you are not limited to pointing the camera in that direction of movement. Changing the rotation of the camera will allow you to move in one direction, while looking in another.

The last general piece of information we can give is the zoom factor or "FOV". Low values stands for high zoom factor (you're really in there), and high values means your camera is very wide. The default zoom is about 50, which puts you right in the middle

So now that our camera is set in its initial course, we should make it point at an object, and follow it to another. If our camera is pointing at IO and we want it to pan over to see a ship when it comes out of hyperspace. To do this we could use the "LookAt" command and the "LookAtCue" set to trigger when the ship arrives. These commands carry a number with them because IFH allows you to have a camera set up with multiple shots, and cut to them one at a time. So, now we have our cameraman pan over to the Agamemnon just arriving and heading to a station. He will stay on that shot, and follow the ship no mater where it goes (without moving the camera other than what we've told it to do) until the end of the scenario, or until a release cue is given. Once again we must tell the camera where to center its attention as it leaves the ship. If we do not set a "ReleaseTo", then the camera will simply stop panning and let the ship drift out of sight. If a ReleaseTo is specified, the camera will latch on to the new target as it passes it. The Final ReleaseTo should be to the game (no target) or to another camera, which you call up by name.

So to recap, you must tell the camera what to shoot, how to shoot it, and for how long. To end the shot you tell the computer where to go next.

Other features on cameras to be aware of s adding a motion blur effect, which adds a sort of streaking on the trailing edge of moving objects, The amount of streak is determined by the speed of the object multiplied by the amount of blur added (a number between 0 and 1).

SPECIFYING FLYING FORMATIONS:

When flying your fighters, you may need to line them up in a formation at times. This is especially important when they are very close, like when first exiting you're a jumpgate. If you do not specify a flying formation, they may end up on top of each other, then die horribly when they start to move. Specifying fighter formations is as easy as adding the function to your ship structure.

Code Sample:
Formation Wedge

You may also include the "FormationSpacing" on the next line to specify how far apart the ships should be. Note also that formations can also be used on a group of cap ships. You form a group of 3 Omegas with "Count 3" in the ship structure, then assign them a formation using the code above.

At Current, it is unknown if/how to get different types of cap ships to fly in formation.
See the reference chart at the back for a list of all known flying formations.

USING A ROTATION MATRIX:

So what the hell is a rotation matrix you're asking. Well, the following definition is lifted from Wikipedia, the free encyclopedia (www.wikipedia.com):

Quote:
ROTATION MATRIX: The set of all rotations about a given axis, together with the operation of composition, form a continuous group. The matrices discussed in this article then provide a representation of the group. Any rotation matrix is orthogonal, that is, the inverse of a rotation matrix is its transpose. In particular, the dot product of each row with itself is equal to 1, and the dot product of different rows is zero.

A Rotation matrix in IFH is a way of designating how a fighter or group of fighters will fly as a whole to perform rotation maneuvers (think "Epiphanies" when the starfury groups are performing and launching fireworks). They can be used in conjunction with, and sometimes instead of the flying formation command. Add this to your squadron SHIP structure. They are useful to when you want a fighter group to rotate around a central axis in some way or another. Even more complex maneuvers can be achieved by using rows.

Sample of a Row Declaration Section:
Row0 1 0 0
Row1 0 1 0
Row2 0 0 1

Code Sample:
SHIP {
Name Sigma 1
Class sa-23c
Team EA
Position 0 0 0
Row0 1 0 0
Row1 0 1 0
Row2 0 0 1
ArrivalCue True
}

[The above code should align your ships to rotate in formation a in a 45 degree line.???]

The syntax for a rotation matrix is:
Rotation X Y Z A

where
X Y Z - is an axis of rotation,
A - is an angle of rotation in radians, (or in degrees if prefixed with D)

Code Sample:	***Explanation:***
Rotation 0 1 0 D90	This tells the computer to rotate your craft 90 degrees around yaw axis (positive should be to the right, and negative to the left).

Code Sample:	***Explanation:***
Rotation 1 0 0 D45	This tells the computer to rotate your craft 45 degrees around pitch axis.

Code Sample:	***Explanation:***
Rotation 0 0 1 D180	This tells the computer to rotate your craft 180 degrees around roll axis

More information on this topic will appear in a future update. In the meantime, look at the sample missions, and at the missions constructed by forum members for examples of implementation of these features.

AI GOALS:

Sending a ship into a battle is more than just placing them and telling them to fire. Tactics and strategies come into play. Some ships will even have a special mission to accomplish, or a specific target to attack. Defining tactics for ships is simulated by programming *AIGOAL*s for each ship. Ships can have one or multiple goals, either consecutively or concurrently. If working on the former, you must be sure to specify which goal is in effect when. If your ship has concurrent goals however, it is vital to tell the order of importance for each based on what you want to ship to achieve. If you want it to reach a destination at all costs, then it's path should be a higher priority. If instead you want it to fly towards a point, but take out as many other ships as it can along the way, then the attack goal should take priority over the path. Setting priorities for goals is done with percentages and balancing them effectively will take some trial and error.

AI goals must state which ship or ships they should effect, when they should be put into play, and what the ends of the goal is; be it to proceed to a destination, attack a ship, defend a ship or whatever.

Note that AI goals are different from objectives. You should use AI goals to get the ship to do what you want from moment to moment during the battle. Objectives are your overall victory conditions. You should use the success or failure of AI goals to help determine the success or failure of objectives, but the objectives will ultimately determine whether or not the mission was

ultimately successful for the player, if they will move on to the next mission, and how the following mission will proceed.

Check the aigoals reference chart in the appendix for a list of available goals.

CREATING AN ASTEROID FIELD:

Asteroid fields can be placed into any of the environments you desire, and can be as big or as small as you like. Fields are built using pre-constructed models of various sizes provided with the game.

The fields are divided into 2 sections, stationary and moving. Stationary asteroids can be placed either in fields or in individual positions. They have no inertia and no mass to speak of. In physical terms, this should be a negligable factor because they are so big and massive you couldn't possibly move them anyway. In the example below, we are going to place 30 copies of asteroid02.flm around postion 0 0 15000 in a random formation, spaced 10000m apart. These are the big rocks you see from a distance. We can create them using the *MODEL* structure.

Code Sample:
Model {
Name Asteroid
Class asteroid02.flm
Position 0 0 15000
Row0 1.00 0.00 0.00
Row1 0.00 1.00 0.00
Row2 0.00 0.00 1.00
Cue true
Count 30
Formation Random
Spacing 10000
Collision good
}

Everything in the above code should seem familiar to you by now with the exception of "collision". Collision good means that it uses more accurately defined collision detection for speed. Setting this factor to "rough" instead will give you a less accurate collision detection. This is useful for saving processing power as well as diferentiating key areas of the asteroid system that may be an influence on the battle from those that are less important.

To make asteroids that drift or move, you would use the *AsteroidField* structure. Asteroids created with this structure have mass and inertia for collisions. In defining the asteroids, you can

provide an overall mass and momentum. Since velocity is momentum divided mass, you can adjust these two values to increase or decrease the overall speed of travel. You should always define some type of mass, however if you set the momentum to zero, then these objects will also be stationary.

The code below creates an *asteroidfield* structure that produces all the tiny asteriods that fill our area. Even though the field is set to "random," asteroids seem to be placed rather regularly. They will as a whole react to outside influence. If you hit one, another will fly by later (maybe they cycle back at the edge of the radius).

Code Sample:
AsteroidField {
Name Asteroid
Class tinyast_01.flm
Position 0 0 0
Row0 1.00 0.00 0.00
Row1 0.00 1.00 0.00
Row2 0.00 0.00 1.00
Cue true
Count 100
Formation Random
FormationX 1600
FormationY 1600
FormationZ 1600
RandomRotation 3
Collision ROUGH
Mass 780
Momentum 3433320
Radius 500
FieldRadius 75000
}

"Radius" refers to how far away the small asteroids have to be before you can't see them anymore. "Fieldradius" defines the radius of the entire asteroid field. Asteroids in motion that reach the end of this boundary are reflected back by this border.

There is no reason why you couldn't put big asteroids as moving and the tiny asteroids as stationary. This is up to you to decide. You may also want to put all moving or all stationary, the

choice is yours as long as you take into account how the ones with mass will react as opposed to the ones without. If you shoot at the big asteroids, they will accelerate depending on mass (more mass slower acceleration). In this case, the momentum stated in the code becomes the maximum achievable momentum they can attain.

RandomRotation XX will randomly select the initial rotational spin of the asteroids. A higher number means faster maximum rotation. Finally, *FormationX* is the size of the field in X-direction from the designated central point, so if you set X-position to 0 and *FormationX* 20000 you have an asteroid field from -10000 to 10000. Similar definitions should be used for the other Y and Z axis.

Asteroid models available are: tinyast_01.flm, asteroid02.flm. In theory however, you could make a debris field of any model with the .flm extension. One could make a boneyard for dead ships for example provided you could find the model.

Be aware that when playing the game, flying towards asteroids will not trigger a collision warning for players, so it will be a lot harder to fly close around them without colliding.

Currently there is no list of available .flm models. This may appear in the future.

OBJECTIVES AND ENDING THE MISSION:

As stated before, objectives use the outcome of AI goals and cues to determine whether or not the mission is a success for the player. By no means does a mission solely have to succeed or fail strictly on if the enemy is completely destroyed. What makes the mission a success is up to your imagination and storyline. You can have primary objectives and secondary objectives. Primary of course are the ends that must be achieved by the player in order to move on. Secondary objectives are the icing on the cake. It is up to you to decide if the player needs the secondary objective to move on, but the idea of having multiple objectives gives you the option of playing multiple messages to the player at the end based somewhat on how they do. You may have a mission to destroy the defense stations around an enemy base, so your cap ships can move in and take the base with ease. If the player destroys the satellite, but not all the enemy fighters then the mission should still be a victory as the primary objective has been completed. If the player also succeeds in taking out all the fighters and or the base itself, then you may want to provide a bonus “reward” video or message. Perhaps destroying the base now even allows the player to skip the next mission in your story, or changes the amount of enemies in the following mission. The choice is yours. (note that a lot of this is just theory, as chaining together multiple missions is not officially supported at the current time – though it is somewhat possible.)

Each objective must specify what triggers the end of the mission. If you just have an objective without a trigger, the mission will test for victory right at the outset, and result in a failure even before the player gets to fly. The *activatecue* should define a condition that will trigger the victory check. One common activation cue is *noenemies*, meaning the computer should check for victory if there are no longer any bad guys on the map.

Once activated, the objective must resolve the cues or AI goal names that need to be true (successful) in order to be victorious. You can add multiple victory conditions to a single

objective using a custom Cue or using the *and*, *or* and *not* conjunctions in your cue line. Objectives should also define what happens if the player does not succeed. This is defined by the *failurecue* line. For each different ending you offer, you need to define and resolve a complete ending sequence of some sort, even if it's just a fade to black.

CHAINING MULTIPLE MISSIONS:

One final consideration is whether or not to tell your story across multiple missions. For the moment however this feature has not been implemented in the current release. There are however two workarounds to this problem in the meantime.

The first method is simply to create your missions and zip them for distribution. Then all you need is to tell the player which mission to play first. For the victory condition of each battle, you would put up a subtitle telling the player which mission to go to next. This simple “manual” method is beneficial in its implementation but has some obvious drawbacks. It relies heavily on the player to play the correct map at the correct time, and not play them out of order.

The second method is more of a true chining of missions, but is harder to implement and requires you to manipulate the player’s config file (or have them do it themselves). Some people may not want to do this, though and might shy away from your mission because of it. If you feel this is the route you want to take however, here is what’s necessary.

Open the “user.cfg” file and add "runmainloop 0" to the end.
After that line, execute your own script.

User.cfg script addition example:
Runmainloop 0
exec campaigns/myscript.ini

What you are doing here is telling the system to look in the "DATA/campaigns" folder for the script file named *myscript.ini*. You are essentially tacking on additional information to the user.cfg file. Save the changes and close the file.

Next you need to create the file called *myscript.ini. Go to the DATA/campaigns* folder and make a new text document. Rename it *myscript.ini* and open it for editing. Inside that file you can tell the system what order to play the maps in. Use the code below as an example.

Code Sample:
run mymission1.map
if {last_mission_result == 0} {run mymission2.map} else {quit}
if {last_mission_result == 0} {run mymission3.map} else {quit}

Continue on listing all the missions in your script until you are done, then save the file and close. Remember that all names are case sensitive and you should not use spaces. If you have done

everything correctly, playing the first mission should result in you moving onto the next if you are successful.

V. CUSTOMIZING

Aside from laying out what ships will be in your battles and who is going to attack whom, there are many ways you can change the look and feel of the mission. Though IFH contains many elements of the Babylon 5 universe, it is very flexible, allowing you to expand on that universe, while still staying within the confines of the system.

CUSTOM SKINS

This is a fancy way of saying “put a name and surface on the side of your ship”. This is a way of making ships your own, similar to detailing a car. Skins consist of textures and texture maps. You can create your own or use the skins already included in the IFH D&O game. You do not need to play with generic omegas for example, each one can have a specific name. To use one that is provided, simply enter its name into the skin field in your ship declaration. See the reference sheet at the back for available names, and on the IFH forum you can find pre-designed skins for all official Babylon 5 Omega names.

Note that the Thunderbolt and its prototype are special case when it comes to skins. If you do not put a "skin" line on the XA-23, you get a fighter that has black markings. If you put "skin default" you will get the red shark's teeth thunderbolt. Finally, If you want the black omega markings on the SA-23 put "skin blackomega".

Other ships may have similar reactions with skins, but this has not been researched yet.
Feel free to try it yourself and let us know.

CREATING YOUR OWN SKINS

Why stop at using only the ships and ship names built into IFH? Why not name a ship after your dog? Use the supplied templates (available on the IFH forum) in your graphics editing program to enter the markings you want to appear on your ship. The default font for Earthforce names, by the way, is Aerial. Be sure not to resize the image in dimensions or resolution, and do not move the boxes around. Also remember that anything outside the boxes on the template will not show up on the ship in the game. Be sure to save the file as a .jpg when your done, as that is the type of file IFH looks for. [are other formats supported?]

When done, place the file in the “Data\Maps” folder. (check this path)

Now that you’ve added your new skin, you also need to tell the game that you have created the new image and what it is used for. To do this we will make a small script.
Open up a new document in Notepad or Wordpad. Copy the following example (or use one in the IFH folders) and paste it into the document.

Here is an example of the Agamemnon's skin entry:

Code Sample:
Subst {
DefaultTexture omega_skin.jpg
NewTexture omega_skin_agamemnon.jpg
}

Where "omega_skin.jpg" is, replace it with the name of the ship class you want your new skin to go on. Where "omega_skin_Agamemnon.jpg" is, replace the "omega" with the ship type you entered on the previous line, and the "Agamemnon.jpg" part with the name of your graphic file. Save the document with the format "[class abbreviation]_[yourshipname]. (Do not use the .txt extension). Place the file in the "Data\Skins directory". (check this path)

Sample skin declaration fields:
ocg_Alexander
ngc_LoneRanger

Example of making a skin script:

- I made a skin for a Nova with the name "Lone Ranger".
- I save the "LoneRanger.jpg" file to My IFH\Data\Maps folder.

- I then go into the IFH\Data\skins directory and right-click on the "ngc_boreas" file and copy it. I immediately paste it and rename it "LoneRanger.jpg" (caps doesn't matter, as long as you're consistant).

- Right-clicking again I open it in notepad and modify it to read the following:

  Subst {
  DefaultTexture nova_skin.jpg
  NewTexture nova_skin_LoneRanger.jpg
  }

- Now I save and close.

Remember now to assign this skin to a ship in your mission in the ship declaration section. If you are having problems you can always modify the file later by right-clicking on it and selecting "open with…Notepad"

FLYING OTHER TYPES OF FIGHTERS IN SKIRMISH MODE

By default, the skirmish missions assign the player to a sa-23c starfury. However it is possible to play skirmish missions while flying a different fighter. This is not officially supported, but is very easy to accomplish.

INSTRUCTIONS:
1. open a skirmish example mission
2. scroll down to

```
SHIP {
Name  Sigma 1
Class sa-23c
Team  EA
Position 0 0 0
Row0 1 0 0
Row1 0 1 0
Row2 0 0 1
ArrivalCue True
}
```

3. change sa-23c to minflyer (or any other fighter listed in the reference sheet). You cannot have the player fly a capitol ship. Also know that at some ships are not “complete” in that they currently do not have unique HUD’s, or may be missing some other unique feature… or even parts of the structure! So although you may be flying a Nial, you may still see the scanning and flight system of a starfury.

OVERIDING THE RESOURCE.WAR FILES

The Resource.war file is the main “filing cabinet” of sorts for the IFH campaign. It contains all the default data on the objects, locations and graphics used in the D&O campaign. Currently there is no way to modify this file. Further, the files in resource.war have priority over those in Data folder, thus preventing you from changing the game campaign. So, the only way to change things is to find a way to work around that restriction.

The key here is the “Include” function, which lets you tack on external script files for use in a mission. When you “include” a file, that file takes precedence over the base file, meaning you get everything in the main file plus what is in the included file, with the latter taking precedence.

To Override the resource.war file, simply “include” a file with your new info. For example, include one of the campaign maps and create another ship with the same name as the player's ship. (note that the header is not included, which means you have to set the slot00 ship and the environment – if hyperspace you have to manually include hyperspace.map and set hyperspace to 1.) Of course, this could have some interesting side effects, like flying the hyperspace missions with no hyperspace currents.

With this method, you cannot program new attributes, but you can change the existing properties of objects in the mission with the PROPERTIES structure. You could give yourself nomissilelock against major Rotaine, or give the Monolith harmless AI (or -1 Hull). You could put in new AIGoals for the enemy, give an AI to your old self, or even have another cruiser jump in and engage things. The one drawback of course is that this is all purely recreational and wouldn't work in the actual campaign, since the game still treats it as a custom mission.

VI. NOTES & TIPS

- When writing your code, do not use the tab key to separate functions from modifiers. You must use spaces or you will get processing errors.

- As they were not needed for the current IFH storyline, the Hyperions, and a few other classes have no subsystems. They can fly but not shoot, etc. Subsystems may be added later. Meantime you can use them for window dressing, cannon fodder or docking.

- Decoding the "resource.war" file:
  At present there is no way to decode the ".war" file for modification. If you want to try and crack it, here is what is known so far:
  - the war files are NOT related to Java 2E
  - It is neither a Konqueror file, not a web archive (Unix)
  - The game is based on the "Homeplanet" engine. Knowing how to mod that game may help.
  - Check the customizing section for info on how to bypass some aspects of the resource file.

- Range of AI visibility – ships outside of a 35000m range will not be "visible" to any AI goals. They should be considered outside the "sphere of influence" of your starting point. If there are no viable targets within this range when a goal is called, the goal will be ignored. (This can be worked around by setting the trigger for the AIGoal to only start when called on by the "Proximity" variable.)

- Jumping in - When coming out of hyperspace, the location point you specify is the origin point of the vortex. Ships coming out of a vortex will coast away from that point due to their length and drift, in the direction of their heading. You need to compensate by the following amounts if you want your placement exact:
  - Omega: 6200m
  - Nova / Nova_md: 4500m
  - Fighters: 100m

- Weapons Ranges – Weapons have a maximum effective range. The ranges given in the reference charts are for stationary ships. Actual range will be effected due to velocity and direction of relative movement.

- Note that the Omega's lasers will fire if its target is inside the 35000m "visible" range, but the lasers will dissipate at 20600m and will never reach their target.

- Ship Hitpoints – Comparatively, the Nova is about 85% as strong as the omega in terms of taking damage.

- AI Quirk: If two capships are targeted, and both are nearly in a straight line to the attacker, the attacker AI will not fire unless BOTH ships are in range.

- When setting your attacks, you can use *Group* instead of *Object* to restrict your targets. So, if you have 3 squads, and tell Alpha squad to attack *Group* Beta, they will attack only

Beta squad ignoring Gamma. Remember to specify your groups in the ship declaration section(s).

VII. FAQ

Q. Is there any way to chain custom missions together into a custom campaign?

A. Chaining custom missions together is not officially supported, however it is possible with some manual editing of the *config* file.

Q. Is there any variable that shows the Objective is in progress. Because (Completed:"Mission") or (Failed:"Mission") returns False even if the Objective has not yet started.

A. It may be possible to use the “NotInProgress” cue. If it is set to 1, then the mission is currently not being persued. Logically “NotInProgress” returning “0”, or “not 1” should mean the mission is currently active. [This is un-confirmed.]

Q. Need more data about Proximity trigger. Proximity:"Columbia" Object:"Beta" it doesn't seem to work.

A. Be sure to include the distance value to tell how far the proximity range should be. Also, be sure one of your objects is designated as “Beta” and that it is spelled correctly in both the cue and the ship declaration. Finally, make sure your using the cue inside a structure that is useful. For example, you wouldn’t place the proximity trigger inside the header or ship declaration structures.

Q. Why do the Nova and Hyperion class NOT attack other capships when they get the order to do so. The Omega class attacks other capships without a problem.

A. Because they do not currently posess any Gun hardpoints. They currently have no subsystems (see notes section for more info).

Q. Is there any way of actually launching fighters from an Omega class?

A. You can adjust cue-s for arrival of a squadron and then set the place they are coming from (hyperspace or a cruiser)

Q. The Hyperion class seems invulnerable, is there any way to fix it?

A. It was made invulnreable because of Training Mission 3 having the dummy cannons. They actually worked like real cannons, but the objects were made with “MinHull 10”, making them uindestructable (minimum hull cannot fall below 10%). The only way is to wait for more update

packs with more "working" ships...and there has been no official promise of this. [Update: there may be a way using the “Include” method mentioned in Chapter V. This is un-confirmed.]

VII. CODING SAMPLES:

The following are samples of code that achieve a specific purpose. You modify and build off them for use in your own missions.

GETTING A SHIP TO DEPART FOR HYPERSPACE

Code Sample:
AIGoal {
OrderFor Alpha
Cue True
AIType Pilot
GoalType depart
Priority 100
Object hyperspace
}

To get this ship to exit through a jumpgate, you must specify the gate as your object instead of hyperspace (obviously you have created a gate and placed it into your paying field). You must also assign a the jumpgate to a team, and give it an AI level it's ship structure or it will not activate. (It appears that level of AI may influence which ships can use gate, but this may just be a glitch). For Capitol ships departing without a jumpgate, use a "followpath" command followed by "GoalType Depart". If you would like to have your player activate the jumpgate via the use of the comm. system, then place the "CallAllowed 1" function in the jumpgate's ship structure. To activate, the player must then contact the gate, similar to the way one docks at a station.

GETTING A SHIP TO ARRIVE FROM HYPERSPACE

Code Sample:
ArrivalCue MissionTime:"" Delay:"98"
ArrivalFrom Hyperspace

A ship starting the mission already in the system would not need an arrival cue, only a starting position. If However you have the "ArrivalCue" set to "true", then you will open the scene with the ship coming out of the vortex. Finally you can base the ArrivalCue on a variable or trigger. "MissionTime" is a popular way to delay an arrival, as is when a certain ship is destroyed. Note that, if you use the "ArrivalFrom" statement, you don't need to use Position or Rotation, as they will both default to that of the source object. Remember though that the arriving ships will cruise out past the exact point of arrival (jumpgate horizon or hanger bay) by at least it's length.

HOW TO MAKE BEAM LINES SHOW UP IN HYPERSPACE

Code Sample:
Environment Hyperspace
Hyperspace 1
Include0 Hyperspace.map

Also add the beacon name in the ship structure:

Code Sample:
Position x y z
Relative to Io

To make a route, define a PATH. A nav point is a PATH with one point01, “ghost” must not be set to “1” (“ghost 1” makes it invisible), and set the “Cue True” function. The LAST point defined in PATH will become visible if targeted as NAV point. [If this confuses you, it’s because I’m still confused myself. I’ll refine it later in better detail].

DOCKING PROCEEDURES

To dock with a capitol ship or station, use a structure similar to the following.

Code Sample:
AIGoal {
OrderFor Mu
Cue Arrived:"Mu"
AIType Pilot
GoalType EnterHangar
Object EASTA_Ceti
Priority 20
}

You must also specify the target object as an object open for docking. To do this, you must add "CallAllowed 1" to the destination object’s SHIP structure. Then, in the game, the player must use in game use communication to contact the station or ship and request permission. Note: the thunderbolt is too big to fit in a Nova’s hanger. This is why Nova’s never carry them, and thus why you can never dock on a Nova in one.

GETTING A COMPUTER AI TO FIRE ON ANOTHER NOVA (OR CAPITOL SHIP)

Code Sample:
AIGoal {
OrderFor Nova1
Cue True
AIType Pilot
GoalType Attack
Allegiance Hostile
ShipType Warship
Object Nova2
Priority 50
}

** Don't forget to set up AI in SHIP structure!

The above code works for getting the two capitol ships to fire at each other, and warship class vessels on the opposing teams side. However, that script will not get them to fire at enemy fighters.

GETTING A COMPUTER AI TO FIRE ON BOTH ENEMY CAP SHIPS & FIGHTERS

Below is an example for getting ships to fire on one another and fighters. Remember that to do so, you need to establish the teams and assign them as hostile enemies.

Code Sample:
TEAM {
Name EA
Color 0 255 0
Hostiles Enemy
}
TEAM {
Name Enemy
Color 255 0 0
Hostiles EA
}
SHIP {

Name EAS Achilles
Class omega
Team EA
Count 1
Hull 0
Player 0
AI Elite
Position -3000 0 5000
Rotation 0 0 0
ArrivalCue MissionTime:"" Delay:"3"
ArrivalFrom Hyperspace
MinHull 0
}
SHIP {
Name Piranha
Class nova
Team Enemy
Count 1
Hull 0
Player 0
AI Elite
Position 3000 0 15000
Rotation 0 1 0 D180
;Rotation: Y, X, Z
ArrivalCue MissionTime:"" Delay:"10"
ArrivalFrom Hyperspace
MinHull 0
}
AIGoal {
OrderFor EAS Achilles
Cue True
AIType Pilot
GoalType Attack
Allegiance Hostile
ShipType warship

```
Priority 100
}

AIGoal {
OrderFor   Piranha
Cue        True
AIType     Pilot
GoalType   Attack
Allegiance Hostile
ShipType   Warship
Object     EAS Achilles
Priority   100
}
```

Note that just because two ships are told to not like one another, they may not actively seek engagement with each other. Two ships on completely opposite sides of the battlefield will engage the closer enemy ships before taking on one another. The key is the distance...so if you give the ships an additional goal, to go to a certain waypoint AND attack hostile warships, that helps in forcing engagements.

MAKING A CAPSHIP FOLLOW A PATH AND ATTACK OTHER CAPSHIPS ALONG THE WAY

A way that works (though there may be other ways too) is to make one AiGoal for attacking other capships, and then one aigoal of higher Priority for Followpath who is delayed min. 1 second to the Attack-AiGoal

Code Sample:

```
AIGOAL {
OrderFor Agamemnon
Cue Proximity:"Damocles" Object:"Agamemnon" Distance:"10000"
AIType Pilot
GoalType Attack
Allegiance Hostile
Shiptype Warship
Priority 20
}

AIGOAL {
```

OrderFor Agamemnon
Cue Proximity:"Damocles" Object:"Agamemnon" Distance:"10000" Delay:"1"
AIType Pilot
GoalType FollowPath
Object Pathf11
Speed 10
Priority 25
}
PATH {
Name Pathf11
Point01 0 500 0
Cue True
Ghost 1
}

Note that the enemy capships must not be away by more than 13 degrees of the line between the capship and the waypoint. Otherwise the followpath-aigoal cannot proceed.

TO HAVE ALL SHIPS CONCENTRATE ON ONE TARGET

To focus all your fire on one particular capship, use code similar to the example below.

Code Sample:
Allegiance Hostile
Shiptype Warship
Object TARGETCAPSHIP

TO SIMULATE A SHIP BECOMING DISABLED

For disabling a craft you can use the following:

Code Sample:
Properties {
Cue HullBelow:"SHIP" Percent:"50"
Object SHIP
Team XXX
}

The idea is that once the ship's hull integrity falls below 50%, the team of SHIP is changed - like it surrendered. You can also try changing the AI to Transport – this would allow you to potentially disable weapons, and yet allow the ship to remain on the same team.TO SET AN OBJECT TO RAM ANOTHER OBJECT

Code Sample:
AIGoal {
OrderFor assalt_2
Cue {cue}
AIType Pilot
GoalType Ram
Object {ship to be ramed}
Allegiance Hostile
ShipType Transport
Priority 100
}

You must include all lines must be included for this to work. Currently only fighters and transports have the "ram" goal. Capital ships cannot choose this. As an alternative, you could set a nav point on the other side of the target and set the ramming ship to go to that nav point. If you put the nav point about twice the distance away it will ram at full burn. However if the target ship moves out of the line between the rammer and the nav point, then the collision will fail. There is no way to target a specific part of the ship when ramming.

APPLYING DAMAGE THROUGH A CUE

Weapons hits and collisions are not the only thing that may cause damage to a ship. To trigger damage due to other Nefarious means you use a cue. The following example will trigger the front end of a Nova or Omega or such to blow off 10 seconds after a ship enters its hanger.

Code Sample:
PROPERTIES {
Cue EnteredHangar:"Transport" Delay:"10"
Object EAS Something
Element FrontSection
Hull –1
}

"Element FrontSection" should work for Nova and Omega.

MAKING THE PLAYER AN OBSERVER

You can set up a battle with three teams, so the player can actually watch without being shot at. All you have to do is assign the player to a third team (i.e.) Observer and define that team later on with no enemies.

Code Sample:
HEADER{
Rules single
Environment vathan
Name Test Mission
Description custom_test_desc
vid_near 6
;Define player's ship
Slot00 Ship:"Sigma 1" Team:"Observer"
}
TEAM{
Name Observer
Color 0 0 255
Hostiles
}

APPENDIX: (Quick Reference Charts)

FUNCTIONS AND USAGE			
STRUCTURE	FUNCTION	ARGUMENTS	USAGE EXPLANATION
HEADER			Structure that lays out the overall parameters of the mission
	rules	Single	Specify single or multiplayer rule set
	name	*[user defined]*	assigns a title to the mission
	Description	*[user defined]*	describes the mission (for player info only)
	Vid_near	6 [others?]	sets the switching distangce for models [why change this from default?]
	Environment	***[See chart]***	Specify system map or location for mission
	Include0	name of map file (hyperspace.map, Team_attack.map, etc)	includes elements from the named .map file. remember to include hyperspace.map when you use environment hyperspace.
	Hyperspace	1, 0	if 1 then, objects move under the influence of hyperspace currents.
	Slot00	ship:"shipname" team:"teamname" override:"1"	defines the player's ship. use *Override:"1"* if player will not have control at beginning of mission.
	StartFadeOut		denotes that the mission will start on a black screen
SHIP			Structure used to define an object(s) to appear in mission.
	Name	*[user defined]*	assigns a title to the object being defined (used to referece this object in other functions. also shows up on the HUD.)
	Player	number (within range of group)	specifies which ship in a group is player's ship. (ie 3 for alpha 3) 0 for AI. *[This function is obsolete. use slot00 in header instead]*
	Class	***[See chart]***	Specifies the make of the ship to be displayed.
	Skin	***[See chart]***	assigns a set of textures specific to this ship and model.

	Position	x y z	denotes a specific location on the map, in x y and z coordinates from map center, to place target object(s)
	Rotation	x y z a	rotation of angle a around vector xyz, by default this is radians. if you want degrees prefix the angle with a D instead of A.
	Velocity	x y z	defines how fast a ship will travel at start
	Team	teamname *[user defined]*	Assigns the ship to a specific team or side.
	Count	integer	Amount of objects that this structure is defining. (Sets up an array where a group that normally acts together, and can be referenced all at once by calling on the group name. Similarly, each object can be referenced individually by calling on [groupname][#]
	ArrivalCue	true, false	specifies if the given objective is present on the battlefield at the beginning of scenario
	Cargo	text	Defines what resources the ship is carrying. Displayed on the HUD, shows up if ship has been scanned.
	AI	Elite, Good, Average, Harmless, Transport	Specifies operating intelligence level for the object. MUST give ALL SHIPS(except for the player's ship) an AI, even jumpgates & stations, or they may not work properly.
	MinHull	0 to 100	A minimum level of damage that a ship will not fall below. If this is set to anything other than zero, the ship cannot be destroyed.

	MissionCriticalCraft	1, 0	Denotes a ship who's destruction (or salvation) is critical to the mission. If set to 1, the ship shows up in the mission critical list on the hud.
	Afterlife	1, 0	if 1 broken hull will stay around after ship is destroyed.
	Missiles	none, ZR12, ZR35	This ship is equipped with default number of missiles of specified type.
	Formation	***[See chart]***	Specifies a pattern that a group of ships should fly in, in relation to eachother.
	FormationX	distance in meters	maximum distance of formation in x
	FormationY	distance in meters	maximum distance of formation in y
	FormationZ	distance in meters	maximum distance of formation in z
	FormationSpacing	distance in meters	Sets a minimum distance apart for elements of a group
	Difficulties	easy, medium, hard, ace	Specifies the difficulty levels that this ship appears in. Allowing greater challenge for higher levels through adding ships.
	ArrivalFrom	jumpgatename, hyperspace, capitalshipname, stationname	where ship arrives from when cued.
	RelativeTo	Io, Narn, etc.	makes Position relative to a hyperspace beacon. (works only if hyperspace.map is include0'd) also works for any navpoint[?]
	CallAllowed	1, 0	Allows comm acces. Set to 1 for jumpgates and ships you want to activate or dock with.
	NoMissleLock	1, 0	"1" = missiles cannot get a lock on object. "0" = missiles can lock
	NotScanned	1,0	*notscanned="1"* states that no scans have been performed on an object yet, during or before the mission

	Persona	***[See chart]***	sets audio reactions to comm requests and certain events. alphas are for use on fighters, and the others on capships and stations.
	Hull	-1 to 100	sets starting hull integrity
	Guns	gunname	sets gun type. usually not used.
AIGOAL			Structure that defines a mission goal the ship will need to achieve.
	OrderFor	objectname, groupname	Specifies an object that will receive a priority command
	Cue	*[mission specific]*	triggers the AI goal to become active when the status of the cue is true.
	AIType	pilot, *[others not known yet]*	Specifies that this is a flight command (as opposed to a gunner, etc.)
	GoalType	***[See chart]***	denotes the action for a given order
	Object	objectname, groupname	Specifies a ship, jumpgate, station, etc, that will be modified or targeted by the AI goal. This is specified by "name", not by "class".
	Group	name of a ship structure with a count greater than 1	assigns command to the group with that name in its ship structure.
	Allegiance	hostile, friendly, neutral	if set to friendly then it will attack friendly ships. if hostile, then it will behave normally. (attacking ships on the opposing team.) [verify this]sets how the AI will react when it encounters another team's ship
	ShipType	warship, fighter, transport, jumpgate	Designates the objects strategic category
	ProceedTo	*[user defined]*	tells the object to travel to a certain location
	Speed	speed [m/s?]	sets maximum speed ship should travel when executing AI goal.

	Priority	0 to 100	Used to determine which goals take precedence over others. If the object has a choice between going after two goals equally, it will make its decision based on which has a higher priority
PATH			Structure that defines a route through a series of coordinates.
	Name	*[user defined]*	assigns a title to the path being defined
	Cue	*[mission specific]*	triggers the path to be followed if the status of the cue is true.
	Point01	x y z	specifies a location on the battlefield for the path, in xyz (# = which point you're referencing if more than one). The last will be shown as a nav point.
	Ghost	1, 0	"1" = path points are invisible
TEAM			Structure that defines an individual side of the conflict
	Name	*[user defined]*	For changing the status of an object's hull integrity
	Color	0 0 0 min, 255 255 255 max	sets the RGB values for a given object in regards to the scanners and computer display
	Hostiles	a teamname	Defines who are the active enemy for the team.
	Neutrals	a teamname	Defines who are the non-combatants for the team.
OBJECTIVE			Structure that defines a challenge to be met and overcome
	Name	*[user defined]*	speciifies a title for the objective being defined
	Description	*[user defined]*	describes the mission (for player info only)
	Class	primary, secondary	denotes if this is a critical objective, or one that is meaningful, but not essential.

	Cue	true, false	Triggers the objective to be in progress, if the status of the cue is true.
	FailureCue	*[event]*	resolves an objective as unsuccessful if a given variable returns true
	ActivateCue	true, false	triggers the execution of the objective
CAMERA			Structure that defines camera operations
	Name	*[user defined]*	assigns a title to the camera being defined
	position	x y z	denotes a specific location on the map, in x y and z coordinates from map center, to place the camera
	velocity	x y z	states how fast the camera shold move
	rotation	x y z d	rotation of angle a around vector xyz, by default this is radians. if you want degrees prefix the angle with a D instead of A.
	LookAt*[#]*	object:"name" transition:"number" FOV:"number"	Sets up a camera shot of a given number. The number specifies the shot number.
	LookAtCue*[#]*	True	triggers the camera to take to a shot#
	LookAtReleaseCue0	True	sets up the trigger for the end of a shot
	ReleaseTo	objectname, cameraname	tells the camera where to go for its next shot if none specified.
	MotionBlur	number between 0 (min) and 1 (max)	adds a motion blur effect to the scene. Intensity of effect is specified by number modifier
EXECUTE			Structure that defines a console command to be executed.
	Cue	*[event]*	triggers the execution of the command if the status of the cue is true.

	Command	***[See chart]***	denotes a console instruction to be performed
PROPERTIES			allows you to change the properties of a ship or structure on cue.
	Cue	*[mission specific]*	tells the computer to change the properties of the target object, if the status of the cue is true..
	Object	objectname	Specifies a jumpgate, ship, navpoint, etc. that will be modified or targeted with the properties structure.
	Element	weapons, engines, frontsection	specifies subsystem of a ship.
	Hull	-1 to 100	For changing the status of an object's hull integrity
SUBTITLES			Structure that defines a subtitle sequence.
	Cue	*[mission specific]*	tells the computer to show subtitles, if the status of thecue is true..
	Life	time in seconds	time that subtitle stays onscreen
	X	x screen coordinate	Specifies a position along the X axis to position text from screen ctr
	Y	y screen coordinate	Specifies a position along the Y axis to position text from screen ctr
	Flags	Xcenter, ycenter	Specifies how a font should be positioned in relation to its coordinates regardles of screen size
	Font	File:"serpentine_font.tga" size:"pt"	specifies the font to be used for text by calling a font file.(defaults if font is not present on system).
	Size	24, 30, 32, etc.	specifies the font size of a given piece of text in points

	Color	0 0 0 min, 255 255 255 max	sets the RGB values for a given object in regards to the scanners and computer display
	Text	text or textfilename	desigates a line of text for the subtitles, or a .txt file to get subtitles text from
	Spacing	pts	Tells how far apart letters should be kearned (lead?) in the subtitle text.
	FadeIn	number of seconds	specifies length of fade from black effect
	FadeOut	seconds	specifies length of fade to black effect
MESSAGE			Structure that defines a message to be displayed
	From	objectname	Specifies the origin of a message
	Text	text or textfilename	desigates a line of text for the comm systems, or a .txt file to get text from.
	Sound	speech/mp3filename	play an mp3file from the data/sound/speech folder with the message.
	Cue	*[mission specific]*	tells the computer to display a message, if the status of the cue is true..
	Ghost	1, 0	"1" = message is played, even if specified source is not available.
MUSIC			Structure that defines the music to be played at a given point.
	Cue	*[mission specific]*	tells the computer to play music, if the status of the cue is true..
	File	mp3filename	Tells the computer which file it should access for the needed piece of information
	Loop	"1" = yes, "0" = no	Tells the computer to replay the music selection over and over
DIALOG			Structure that Defines a dialog session to take place.

	Name	*[user defined]*	assigns a title to the dialog sequence being defined
	Cue	*[mission specific]*	tells the computer to play a dialog, if the status of the cue is true..
	Message01	from:"objectname" text:"text / file" delay:"seconds" sound:"speech/mp3filename" lifespan:"seconds"	triggers the playing of a message with the specified attributes. Each message should have its own number.
SINGLESHIP			Structure that defines a specific ship of a group.
	Group	name of a ship structure with a count greater than 1	denotes that ship is part of the group with the given name in ship structure
	Number	number (within range of group)	identifies a specific ship of a group
	Cargo	*[user defined]*	defines what information will be returned if a ship is scanned
	NotScanned	1,0	*notscanned="1"* states that no scans have been performed on an object yet, during or before the mission
	Persona	***[See chart]***	sets audio reactions to comm requests and certain events. alphas are for use on fighters, and the others on capships and stations.
	Missles	none, ZR12, ZR35	This ship is equipped with default number of missiles of specified type.
FADEINFADEOUT			Structure that defines a fade to black or fade from black transition.
	Cue	*[mission specific]*	tells the computer to fade in or out, if the status of the cue is true..
	FadeIn	number of seconds	specifies length of fade from black effect
	FadeOut	number of seconds	specifies length of fade to black effect

MODEL			Structure that places a static object model in the scene
	Name	*[user defined]*	assigns a title to the model being defined
	Class	modelfilename	Denotes model to be used.
	Position	x y z	denotes a specific location on the map, in x y and z coordinates from map center, to place target model(s)
	Rotation	x y z d	rotation of angle a around vector xyz, by default this is radians. if you want degrees prefix the angle with a D instead of A.
	Cue	*[mission specific]*	tells the computer to display a model, if the status of the cue is true..
	Count	number	Specifies amount of object models that will appear.
	Formation	***[See chart]***	Specifies a pattern that a group of ships should fly in, in relation to eachother.
	Spacing	distance on meters	similar to "FormationSpacing". Specifies how far apart two or more objects shold be placed.
	Collision	good, rough	specifies how accurate the collision detecion is.
ASTEROIDFIELD			Structure that places asteriods with actual mass and inertia in the scene
	Name	*[user defined]*	assigns a title to the asteroid field being defined
	Class	modelfilename	Denotes file to use (large, medium or small asteroid size).
	Position	x y z	denotes a specific location on the map, in x y and z coordinates from map center, to place target asteroid field

	Rotation	x y z d	rotation of angle a around vector xyz, by default this is radians. if you want degrees prefix the angle with a D instead of A.
	Cue	*[mission specific]*	tells the computer to display the asteroids, if the status of the cue is true..
	Count	number	Specifies amount of asteroids of a type to appear.
	Formation	***[see chart]***	Specifies a pattern that a group asteroids should be placed in relation to eachother.
	FormationX	distance in meters	maximum distance of formation in x
	FormationY	distance in meters	maximum distance of formation in y
	FormationZ	distance in meters	maximum distance of formation in z
	RandomRotation	number	rate of initial rotation on random axis.
	Collision	good, rough	specifies how accurate the collision detecion is.
	Mass	size [in cubic meters?]	Sets a value of mass of an object for use in collisions for use in collisions
	Momentum	number	for use in collisions represents maximum momentum of any one asteroid. (momentum = mass*velocity)
	Radius	distance in meters	how far away asteriods can be seen
	FieldRadius	distance in meters	Specifies Max width of spread created. Moving objects in this field will bounce off boundary.
CUSTOMCUE			Structure that lets you create a single cue to save you from having to type out a long one multiple times
	Name	*[user defined]*	assigns a title to the custom cue being defined

	Cue	*[mission specific]*	tells the computer what cues are involved in defining the custom cue.
MISC.			Below consists of any cues and variables that do not directly fall into any structure category, but can fit into many of the above structures
	;	*[user defined]*	add a remark to the coding, anything after this mark on the same line is ignored by computer
	Axis	[x],[y],[z]	designates a plane or planes along which to modify a speficic object. Usually used with *Rotation* or when first placing an object
	Default	Objectname, event, etc.	Specifies what will be the default setting for a cue.
	Percent	10%, 50%, etc.	denotes a specific value that will modify a given function
	Row	0 0 0, 1 1 1	Defines the layout of a particular line in a rotation matrix
	ArrivalCue	True, False, event	Set to "true" to trigger the arrival of the ship, or designate an event that will trigger the arrival of a ship.
	Arrived:	obectname	triggers a cue when the specified objects arrival cue turns true
	Cue Completed:	event	Specifies that the structure will be performed if the specified objective resolves successful
	Cue Failed:	event	Specifies that the structure will be performed if the specified objective resolves unsuccessful
	Cue	True, False	Set the value for a specific function to be true or false (or triggers a cue to happen if set to true)
	Cue customcue	Cue customcue:"escape1"	Executes a cue that has been previously defined by the user
	Delay:	number	specifies a length of time to hold on a function before going to the next

	Destroyed:	TRUE	returns true if an object has been obliterated
	Distance:	measurement in meters	returns true if given objects are within a certain range
	FOV:	40	FOV is the zoom when you define a camera - low values stands for high zoom factor - default is about 50.
	Gone:	objectname	Sets the value of the object's "gone" cue to true or false.
	MisionTime:	""	references a running counter that starts when the mission begins & ends when mission ends
	NoEnemies	True, false	Sets the NoEnemies cue to True or false.
	Point[#]	number	specifies a location on the battlefield in xyz (number equals thich point you're referencing if more than one)
	Transition:	3, "4"	executes a video transition of a specified time

CONSOLE COMMANDS	
COMMAND	DESCRIPTION
AccStat "ShipName"	displays acceleration stats to all directions
AiDump	creates a txt file in game folder with data
Assign	
Assignslot	
bind "ControlName" "Key"	Assigns a command to a key. The "binds" are used with the flc_* variables to set your flight controls.
bindoff "ControlName"	disable a key
Bindoffall	disable all keys
bindon "ControlName"	enable a key
Bindonall	enable all keys
bonuses "Number"	[?] default value is 3.0000000
CameraDump	same as Ai only shows coordenates
CameraDump_abs	
closedialog	
clearmapinfo	
CLR	clears console
Connect	
Contoggle	open\close console
Current_advance	
Current_brf	shows briefing click continue to return to game
Current_dbrf_a	
Current_dbrf_b	shows failing debriefing click continue to return to game
Current_mission	restarts mission
Current_Player "name"	Change player Designation e.g Sigma 1
Current_stat	displays "How Many Ships Destroyed (by player\overall)
Current_team "TeamName"	change team if no value was inserted it will show current team default EA
DelatePilot "pilotname"	deletes pilot
Dialog "dialogname"	adds dialog
Dialogex	
Dir	shows map names
Dir_gamedir	data folder path
Dir_gameres	game resource default resource.war
DisConnect	
Dumpvar2html	
exec "FilePath&Name"	executes ini file e.g "scripts\bindon_all.ini" with enables all keys if you do it in training you can ram the instructor at the begining and not fail
exec scripts/bindoff_move.ini	
exec scripts/bindoff_turn.ini	
exec scripts/bindoff_turn.ini	
exec scripts/bindon_turn.ini	
gabbagabbahey	

gamma	dont know but something about the gamma
Goto	Cheat to get to a certain location
greatmaker	“1” = Enable cheats
iddqd	(says “cheater”)
idkfa	(says “cheater”)
Ingamemenu	Opens in game menu(esc)
Ingamemenuclose	
ingamemenuopen	
inputwait	
kick	probably for multyplayer
Loadpilot "pilotname"	loads pilot
macro	do\run\open a macro
macrolist	list of macros
Medals	shows how many medals u have?
missiontime	shows how much time has passed since start of the mission in secs
NewPilot "pilotname"	Create a pilot
objhier "ObjectName"	shows objects hierarchy (meshes etc.)
onmymark	
Pause_game	do i really need to explaine?
Playdemo "name"	plays a recorded demo
Playsoundstream	
Playmusic	
playmmenumusic	
Preload	
Quit	Quits
RecordDemo "Name"	records a demo
reloadadvertexprograms	
Remove	
run	run map type Dir for map names
runex	run map type Dir for map names (same as “run”)
SavePilot "pilot name"	Saves pilot?
say	
SetRule	
Slots	
Srvget	
Srvset	
srvvar	
startcli	start a multy player client (didn’t try might crash)
startlocal	starts a LAN game (didn’t try might crash)
Startsrv	starts a multiplayer server (didn’t try might crash)
Stopsoundstream	
Stopmusic	
sysbind	
Sysbindoff	

Sysbindoffall	
Sysbindon	
Sysbindonall	
Sysunbind	
Sysunbindall	
theeye	Cheat - gives you a view from the ship you name. (theeye instructor)
Unassignall	
unbind "key"	unbinds a key
unbindall	unbinds all keys
userlist	
Version	
Windowcaption	

CONSOLE VARIABLES	
VARIABLE	DESCRIPTION
fl_ai	
fl_alwaysai	
fl_blackouts_redouts	
fl_blur_ab_gain	
fl_blur_ab_time	
fl_blur_collision_gain	
fl_blur_collision_time	
fl_blur_enginewash_gain	
fl_blur_enginewash_time	
fl_blur_hit_gain	
fl_blur_hit_time	
fl_blur_shockwave_gain	
fl_blur_shockwave_time	
fl_boltflares	
fl_callallowedbydefault	
fl_clupdatetime	
fl_cockpit_messages	
fl_colldefimpulsemax	
fl_colldefimpulsemin	
fl_colldmgpermegaj	
fl_collfriction	
fl_collfrimpulsemax	
fl_collfrimpulsemin	
fl_collisiondamage	
fl_collzerothrottle	
fl_defaultfov	

fl_difficulty	
fl_emisensorrange	
fl_emivisualrange	
fl_engine_comp_dead	
fl_engine_comp_dmg	
fl_engine_damage	
fl_flash_speed	
fl_formationspacing	
fl_gibs_collmaxrad	
fl_gibs_collminrad	
fl_gibs_density	
fl_gibs_lifespan	
fl_gibs_rotation	
fl_gibs_speed	
fl_hud_parabolic	
fl_hudalpha	
fl_hudflightcursor	
fl_hypercurrentdoration	maybe how muck time the current stays default 5000
fl_hypercurrentperiod	
fl_hypercurrentspeed	the speed u need to avoid currents default 300
fl_hypersensorrange	the range of your sensors in hyperspace default 0.1
fl_hypervisualrange	
fl_interfacemode	
fl_iris_limit	
fl_iris_speed	
fl_jumpinterval	
fl_jumppoint_default	
fl_lodbias	
fl_lodlevel	
fl_messagedelay	
fl_mute_nonplayer_cannons	
fl_mute_nonplayer_turrets	
fl_nebulatex1	
fl_nebulatex2	
fl_pilot_gforcemax	
fl_pilot_gforcemin	
fl_pilot_gforcespeed	
fl_quitdelay	
fl_sensitivity	
fl_showbackground	
fl_showcockpit	
fl_showcollboxes	
fl_showcollbrushes	
fl_showcollhits	

fl_showcollspheres	
fl_showgibs	
fl_showhud	
fl_showhull	
fl_showintro	
fl_showlensfx	
fl_shownebulas	
fl_showstarfield	
fl_showsthull	
fl_snd_collision	
fl_snd_engines	
fl_snd_enginewash	
fl_snd_hit	
fl_snd_hud	
fl_snd_hudambient	
fl_snd_interface	
fl_snd_message	
fl_snd_messagebg	
fl_snd_misslock	
fl_snd_shockwave	
fl_spacedeb_bounds	
fl_spacedeb_density	
fl_ssfadeoutsound	
fl_starfield_detail	
fl_starfield_names	
fl_stat_objectives	
fl_stoponguidedlnch	
fl_targetingdelay	
fl_throttlespeed	
fl_viewturnlimit	
fl_viewturnspeed	
fl_widescreen	
fl_zoomspeed	
flc_end_mission	"1" = end mission. Also "0" at start of new game = mision ends when loading is finished
flc_grid_absolute	
flc_ingamemenu	toggles in game menu(esc)?
flc_pitch_axis	
flc_quit	either quits or toggles quit option
flc_screenshot	
flc_slide_x_axis	
flc_slide_y_axis	
flc_speed_reverse	
flc_throttle_decrease	

flc_throttle_increase	
flc_wings_toggle	
flc_yaw_axis	
fle_afterburner	
fle_collision	
fle_enginewash	
fle_hit	
fle_primaryfire	
fle_secondaryfire	
fle_shockwave	
joy_autocenter	
joy_deadzone	
joy_ffgain	
joy_nonlinear	
joy_saturation	
mou_deadzone	
mou_nonlinear	
mou_sensitivity	
mou_smoothness	
scr_debugprint	
scr_maxfps	
scr_minfps	
scr_showfps	
snd_channels	
snd_doppler	
snd_is_music_playing	
snd_mixbufferlength	
snd_mixrate	
snd_musicvol	
snd_nodirectsound	
snd_nomusic	
snd_nosound	
snd_sounddriver	
snd_soundvol	
snd_usehardware	
srv_allowassign	
srv_allowpause	
srv_allowunpause	
srv_chksum_ctrl	
srv_fraglimit	
srv_fraglimit_delay	
srv_list	
srv_list_timeout	
srv_maxusers	

srv_name	
srv_pauselimit	
srv_timelimit	
sys_dump_full	
sys_dump_mini	
sys_dump_msgtext	
sys_dump_msgtitle	
sys_dump_showmsg	
sys_dump_timestamp	
vid_colorbits	
vid_depthbits	
vid_far	
vid_fullscreen	
vid_height	
vid_laststencilbits	
vid_near	
vid_point_size	
vid_refresh	
vid_systemfontface	
vid_systemfontfile	
vid_systemfontsize	
vid_width	
vid_windowcaption	
vid_xpos	
vid_ypos	

ENVIRONMENTS	
LOCATION	DESCRIPTION
sk_cooke 2	Gold planet, no nebula
sk_danghor5	Pink/orange planet, pink ring nebula, moonlets (asteroids)
hyperspace	Swirling red haze
scheffer_belt	No planet, blue & green ring nebula
simple	Stars only
sk_scheffer	Blue & green planet with clouds, blue & green ring nebula
sk_sol_io	Near IO (red), Jupiter in distance (orange, white, brown)
sk_vthan	Bluegray & white planet, blue & violet sectional nebula
zhadum	Brown planet, red & yellow spot nebula
* (without the sk, the planets become 3d models, and you can fly around them. (seems to work with some systems))	

SHIP TYPES ("shiptype" field)	
TYPE	DESCRIPTON
Fighter	Single (usually) pilot ship made for speed, maneuverability & dogfighting. Short range.
Transport	No real weapons, designed for long range hauling of crew and/or cargo.
Warship	Large military vessel. Multiple Weapons. Long range, slow and less maneuverable.
Jumpgate	Stationary jumpgate for entering and exiting hyperspace.

SHIP CLASSES ("class" field):			
RACE	SHIP TYPE	CLASS	HIT PTS
Earth Alliance	Omega-class destroyer	omega	
	Nova-class dreadnought	nova	
	Modified Nova-class dreadnought *(EAS Monolith)*	nova_md	
	Hyperion-class cruiser	hyperion	
	Starfury SA-20 Nova	sa-20	110
	Starfury SA-23B Aurora	sa-23b	120
	Starfury SA-23C Aurora	sa-23c	120
	Starfury SA-23E Aurora	sa-23e	120
	Thunderbird Prototype *(verified Easter Egg)*	xa-31	135
	Thunderbird	sa-31	
	EA standard shuttle	ea_crew_shuttle	800
	EA space station	ea_ringstation	
	EA tanker	ea_tanker	3000
	EA freighter	ea_transport	
	EA escape pod	ea_escpod	
	EA maintenance bot	maintbot	
	Psi Corps transport *(may display default texture)*	psi_tr	
Minbari Federation	Nial fighter	nial	150

	Minbari flyer	minflyer	120
Centauri Republic	Sentri fighter	sentry	110
Narn Regime	G'Quon-class heavy cruiser	gquon	
	Frazi fighter	frazi	260
	Delta Wing fighter	zephyr	75
	Narn transport	narntransport	
Drazi Freehold	Sky Serpent fighter	skyserpent	160
Misc	Four-beam jumpgate	jumpgate	
	Three-beam jumpgate	narn_jumpgate	
	Asteroids	asteroid01, asteroid02, asteroid03	
	Containers	container_box	
	Communications Sattellite *(verified Easter Egg)*	comsat	
	Cargo platform	platform_cargo	
	Landing platform	platform_land	

SHIP SKINS ("skin" field):	
SHIP TYPE	SKIN NAME
Hyperion-class cruisers	hcc_athena
	hcc_hyperion
	hcc_lexington
	hcc_midway
	hcc_prometheus
	hcc_trafalgar
Nova-class dreadnoughts	ncg_boreas
	ncg_monolith
	ncg_schwarzkopf_j
	ncg_schwarzkopf_s
Omega-class destroyers	ocg_agamemnon
	ocg_damocles
	ocg_hermes
	ocg_juno

	ocg_medusa
	ocg_nemesis
	ocg_perseus
	ocg_theseus
	ocg_vesta
Starfury (sa-23)	blackomega
	colonel
	derek
	dragon
	puff
	rotane
	spacewitch
	voss
	wolfpack
	Syndicate *(Delta wing)*
SA-31	xa-31
XA-31	[none]* *(If you don't put a skin line, you get a fighter that has black markings)*
	default *(red shark's teeth)*
EA Shuttles	eafr_blue
	eafr_green

GOAL TYPES	
GOAL	DESCRIPTION
Attack	Attack an enemy or target
Defend	Defend a friendly or neutral
Depart	Leave the battlefield (jumpgate or jumpengine)
EnterHanger	Enter the hanger of a ship or station
Followpath	Fly on a specified route
Patrol	Fly around a specific object or location
Protect	Guard a specific object
Ram	Collide ship with another object (fighters & transports only)

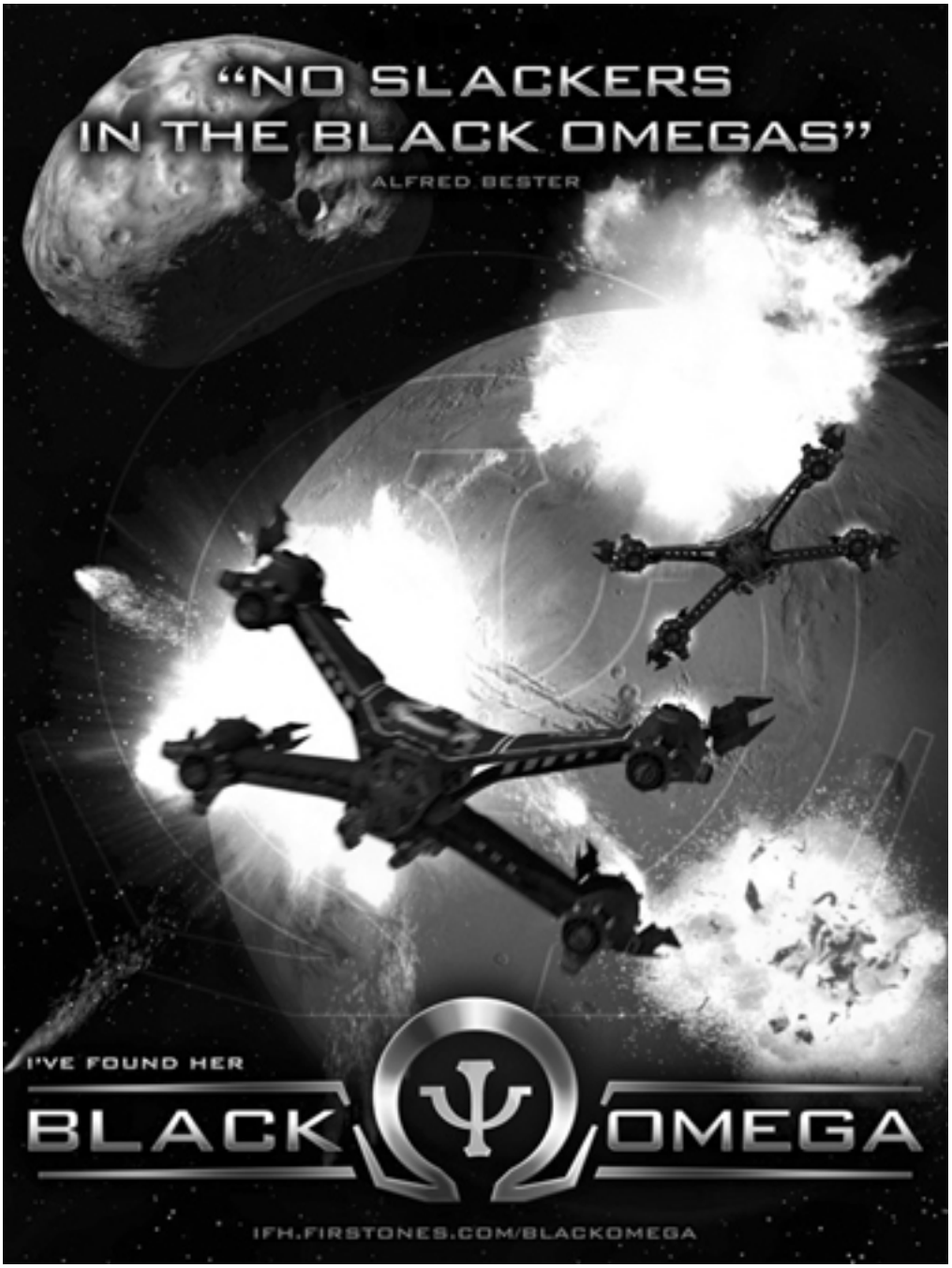
FLYING FORMATIONS ("FORMATION" field):	
LOCATION	DESCRIPTION
Circle	
Claw	
Cloud	
EvenWedge	
Line	
Random	
Square	
Tight	
Wedge	
X	One ship in ctr with others vertically on 4 diagonals
XLine	Horizontal line
YLine	Vertical line
ZLine	Straight line

MISSILE TYPES ("missiles" field)	
TYPE	DESCRIPTON
ZR12	
ZR35	

WEAPONS RANGES		
	PULSE	BEAM
Omega	6100m	20600m
Nova / Nova_md	4900m	
Sa-20	2000m	
Sa23b/c	2400m	
Sa23e	2800m	
Nial	2000m	

PERSONAS	
PERSONA	OBJECT CLASS
Monolith	Nova class destroyer.
Lowe	
PoloStation	Ring station
Alpha1	fighter
Alpha2	fighter
Alpha3	fighter
Alpha4	fighter
Alpha5	fighter

CUES	
CUE	DESCRIPTION
Arrived	Ship has appeared on the battlefield one way or the other
Departed	Ship has left the battlefield by jumping out or docking
Destroyed	The ship has been obliterated
Gone	The ship is no longer on the battlefield for whatever reason
Scanned	The ship's cargo has been actively scanned by the player
Completed	A goal or objective has been finished one way or the other
Failed	A goal or objective is completed, but was not successful, or not completed.
NotInProgress	A goal or objective has ended, or has not begun yet
Proximity	A specified target is within a given range (or not)
Docked	Ship has hard-connected to a station or ship
EnteredHanger	Ship has passed inside the hanger of a ship or station
HullBelow	Check if the integrity of the ship has fallen below a certain level
HullAbove	Check if the integrity of the ship is still above a certain level
HullBelowOrEqual	Check if the integrity of the ship has fallen below or is equal to a certain level
HullAboveOrEqual	Check if the integrity of the ship is still above or is equal to a certain level
MissionTime	Checks if the duration of the mission passed a specific duration or not



COMING SOON

February 08, 2006