PROJECT
XED

TITLE
XED User Manual

	Name	Function	Date	Signature
Prepared:	Per Edman	Mission Designer	25 Oct 2004	
	Mattias Paulsson	XEd Lead Programmer	26 Oct 2004	
Checked:	Mattias Paulsson	XEd Lead Programmer	26 Oct 2004	
	Robert Bare	Community Manager	26 Oct 2004	
Authorized:	Martin Walfisz	CEO		

Distribution: All

Office:
Massive Entertainment AB
Box 4297
203 14 MALMÖ
Sweden

Telephone:
+46 (0)40 600 1000

Fax:
+46 (0)40 600 1099

Email:
info@massive.se

SUMMARY

Welcome to the XEd user manual. The first part, the "Quickstart", is a tutorial that lets you create a fully working multiplayer mission in a matter of minutes. The second part is the "XED manual", that explains all details of XEd and gives you hands-on tips and tricks from the level designers of Ground Control II. This part of the manual also contains the script command reference.

DOCUMENT CHANGE RECORD

Issue	Date	Paragraphs affected	Change information
1	2004-10-28	All	New document

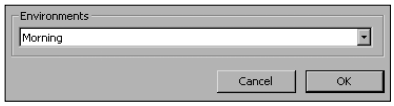
TABLE OF CONTENTS

Part I: Quickstart

Your first multiplayer map

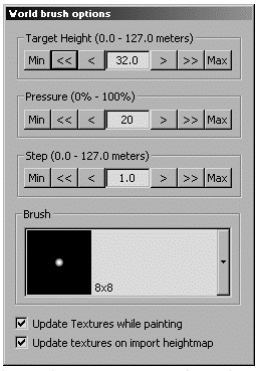
To create your first map, start XEd. Double-click the XEd icon on your desktop. The splash screen will appear. When the Welcome to XEd! dialog has loaded, select the type of Landscape type you would like to use, under “New Mission” at the bottom. Pick “Desert”, and click “New”.

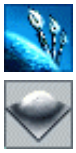
When the map has loaded, the sky will look odd. On the Commands drop-down menu, click **Select Environment**. Environments are connected to the terrain type and desert only has two - one morning and evening. Select “Morning” and click OK.



You should know how to move around. Holding down the right mouse button will allow you to push and pull your way around the map, or W, A, S and D can also be used. Holding down the ALT key will slow this movement by a factor of 10. Holding the right mouse button and the CTRL key lets you look around in the world, while rolling the mouse wheel will raise and lower the camera – this motion, too, can be slowed by holding down the ALT key as well.

We need to raise the whole terrain above water level. The easiest way to do this is **Floodfill**, available under Commands. Water level is just below 19.5 meters, so type in 25 and click OK to raise the ground above this.





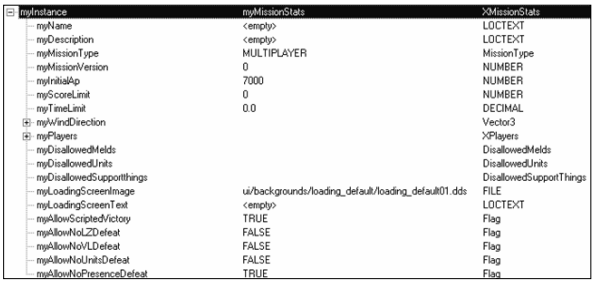
Now that we have flat terrain, let us focus on the terrain tools and brushes. Bring up the world brush options panel. These values are used to control how world brushes function. Now select the **WorldPaintToHeightTool**. Using this tool, terrain can be raised to the Target Height if the terrain is lower, or lowered if it is higher. The tool is useful to flatten large areas to the same height, or to create several hills or mountains of the same average height. Experiment with different brushes, pressures and target heights to get the terrain you want.

It is often useful to detect and re-use a certain height already present in the landscape, using **WorldSetHeightTool**. Clicking the ground with this tool will set the current Target Height to the height underneath the mouse cursor. This tool is essential when expanding a plateau, or when eroding mountains down to the surrounding lowlands.

You will not always want to paint towards a set height. The **WorldPlusMinusTool** can both raise and lower terrain relative to where you paint. Instead of using pressure, this brush tool relies on the “Step” value in the options panel. Just holding down the left mouse button will raise the terrain in the shape of your brush, while holding down the SHIFT key along with the mouse button will lower it instead.

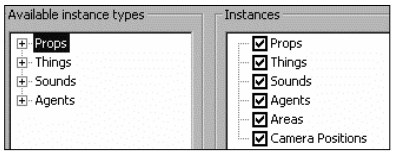
It is also possible to import heightmaps into XEd. Click File->**Import Heightmap** and browse to the file to be imported. The files should be a 513 by 513 pixel grayscale Truevision Targa (.TGA) file.

Once you have created a terrain you are satisfied with it is time to make the map playable. Find and press the **MissionStats** button. This will bring up a Juice Editor for the map settings. Here you can change the amount of AP given to all players at the start of the game, the defeat conditions for the mission, what units to not allow on the map and a lot of other things. Double-click on **myMissionType** and set it to MULTIPLAYER. Then change **myAllowNoPresenceDefeat** to TRUE. Once exported, this map will now be sorted as a Multiplayer mission, and the last remaining player with units or Landing Zones will win. Also fill out a name and a loading screen text for the mission.



Expand myPlayers and go through all eight players available on the map to make sure that both TEAM_1 and TEAM_2 are listed, but no other team. Players may still change their team in the lobby, but if a team is listed among the players in **MissionStats**, players will also be able to choose that team in the lobby - even if that team has no landing zone. Now close the **MissionStats** window.

Click **ViewInstancesDialog** to make the panel visible on your screen, and look at it. Activate the **InstancePlaceTool**. The right tree lists all placed instances, including *areas* and *camera positions*. The checkboxes determine what objects should be visible in the world. Selecting an instance type on the left hand side will allow you to create instances of that type in the world. Double-clicking items in the right tree will allow you to edit existing instances. Now, expand the *Things* branch and then the *GlobalPropTypes* branch. Find the type *LZ_thing* and select it.



Select the **InstancePlaceTool.** Now find two reasonably flat areas on your map where you want dropships to land. Using the **InstancePlaceTool**, click on the ground where you want the *LZ_thing*. This flat white marker is a *thing;* an object that has no collision, but which can be animated. When used to mark a Landing Zone, the marker will change color to reflect whether the zone is neutral, friendly or hostile.

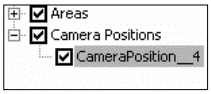
Notice that when you placed the LZ_thing, a new item appeared on the Instances list, under the Things branch. Expand this branch to see the name of the new Thing, which is most likely *LZ_thing__1*. Change this name by selecting it and clicking the Rename button at the bottom of the Instances panel. Do this now, and set the new name to *LZ_1_thing*.

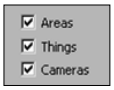
Click the **AreaPlaceTool**. It works a little differently from the **InstancePlaceTool** in that it only places *Areas*, and these require a radius. Therefore, when you select this tool and click on the ground, hold down your mouse button in order to also set the radius of the area before releasing the button to create the area in the place and size you wanted.

If you make a mistake, you can **Undo** the area, or use the **InstanceEditTool**. Clicking on an instance with this tool will select it – surrounding it with a green translucent sphere. Select the area you placed. Now hold down the CTRL key, click and drag the mouse. Normally this would rotate the instance, but because *Areas* are round, it instead allows you to change their size. Holding down ALT instead will allow you to move the instance to another position. When you are done, find the new *Area* in the Instances list and change its name to *LZ_1_area*.

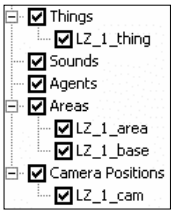
The Landing Zone area and marker is where dropships land, but where do they come from? The *BasePosition* is an *Area* and it needs to be placed quite a distance away from the Landing Zone. In Ground Control II: Operation Exodus, landing zones lay between 750 and 1000 meters from their base position, and most importantly, 500 meters outside the *PlayField*. Red lines cross the visual battlefield where the *PlayField* ends, and it can be changed by pressing the 5th button on the first row, **PlayFieldCoords** (shortcut 3). Do not do that now, but place a second *Area* instance somewhere outside of the red line. If you want to check the distance between them, the **RulerTool** (shortcut TAB) is the 2nd last icon on the second row. If you place too many areas, use the Delete key on your keyboard, or press the **InstanceDeleteTool** (shortcut T) to remove the selected instance. Rename this area *LZ_1_base.*

The last item that needs to be placed in the world is a camera to watch over the Landing Zone. This is the point of view that a player will return to if the W key is pressed while in the game. Placing cameras is fairly easy: Find a good view in XEd, preferably one with good overview of the landing zone, and press the **AddCameraPosition** (shortcut F12) button. You will not see the camera from that perspective, because your view is currently inside of it, but if you move your view, the camera will appear as a small, blue blob in mid-air. To change the camera, move your view to a new position, select the camera in the Instaces list on the **Instances** panel and click the button "*Set current camerainstance to this*" on the right side of the list. When you are done, change the name of the camera to *LZ_1_cam*.





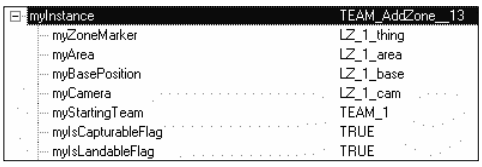
Before we start scripting, it is a good time to review the names we have chosen for the items related to the Landing Zone we are about to create. In the Instances panel, find the top right area named *Text Labels* and activate the ones relevant: Areas, Things and Cameras. This will make the instance names visible in the world, which can be helpful or confusing, depending on the amount of visible instances.



You now have all the components necessary to create a landing zone. Use **ViewScriptsDialog** to make the script panel visible. In the upper left is a list of all existing scripts. As you can see, START is the only script listed, and it is empty. When you have closed that window, click the *New* button in the Script window to create a new script named ZONES.

Select the START script, find TRIGGER_AllPlayersReady under *Available Commands* and double-click it. Change myTriggeredScript to ZONES. This means that once the trigger activates, it will start the ZONES script, and the commands within it will run. In our case, that means the Landing Zones are created once all players are reported ready. Now find the WEATHER_ChangeWeatherEffect command, and double-click it. When the options for this command shows up, select one of the Default_Cloud weathers, as this will add a cloud cover and fog border to your map.

Select the ZONES script. In the list of Available Commands, find **TEAM_AddZone** and double-click it. These are the attributes relating to Zones. First fill in the objects we placed in the world: set myZoneMarker to *LZ_1_thing*, myArea to *LZ_1_area*, myBasePosition to *LZ_1_base* and set myCamera to *LZ_1_cam*. Change myStartingTeam to TEAM_1, then change myIsLandableFlag to TRUE and finally change the myMinimapIconFile to point to the file *ui/skins/lz.dds* rather than *vl.dds*. Leave all other values for now. Once you press close, you can now see the new TEAM_AddZone command listed on the top right hand list when the ZONES script is selected. Use the right-hand



Rename button to the name of the landing zone to LZ_1. Now do it all one more time to create a second landing zone.

This time, name all components something with LZ_2, and tie them all together with a TEAM_AddZone command with myStartingTeam set to TEAM_2. Rename this new zone LZ_2. Once you have done this, all components of the map are in place. Click save to name your map and save it. Once it is saved, press the **Export** button, if you can. Sometimes, it becomes disabled and you will need to move the camera or something in the world for it to realize that the map has been "changed". Wait for the export to be completed, and when it is done you will have a brand new SDF file in your Custom_maps directory – your very first exported and compressed map for Ground Control 2.

Part II: Xed Manual

XEd Overview

The user interface

XEd's main window contains a menu bar, a toolbar, a status bar and the 3D-view, where most of the editing is done. The most commonly used commands are available both on the menu bar and the tool bar, while both also contain some unique commands.

Moving around

To move around in the world, hold down your right mouse button, or RMB, and move the mouse. Use the mouse wheel to change the camera height. To turn the camera, hold down Ctrl + RMB and move the mouse. Holding down ALT at the same will slow down the movement speed, for greater accuracy. The status bar shows information about mouse pointer coordinates, number of placed instances and current instance position and rotation.

Orientation and coordinates

As you can see from the status bar while pointing the mouse pointer at the ground, the coordinate system has the [0:0] point in the top right, north-east corner of the map, and coordinates grown down and left or south and west from there. This affects the placement of instances, wind directions and playfield coordinates among other things. Try to keep this in mind.

X is 0 (zero) along the map's east edge and increases to 3072 toward the west edge of the map. Y is the altitude over zero level on the map. Z is 0 (zero) along the northern edge of the map and increases to 3072 the further south it goes. The ground itself can never be taller than 127 meters, but instances can be placed higher than that.

The menu bar

The commands for New, Open and the Save work much the same as all other programs which edit files. The Export command on the other hand allows you to compile and save the map into a special compressed format (SDF) which can then be loaded into Ground Control II and distributed as the final map.

Some commands are available only from the menu bar, such as "Reset all windows" which can be useful if you accidentally lost a window, or placed it on a monitor no longer connected to your system. The reload commands are also only available on the menu bar. Using these, XEd can be used while different data files are being edited on the same disk as XEd is running on. When certain files have been updated, the "Reload" commands will allow you to update the relevant data and display the result immediately without the need to even restart XEd.

The command Cycle Weathers will allow you to view the map with various weathers active, which would otherwise require you to start the game and wait for the script to activate a weather. Every click on the Cycle Weathers command will change to the next listed weather. Using Reload environment will remove all weather effects.

Menu	Command	Shortcut	Description
File			
	New		Starts a new map. Displays the select landscape dialog.
	Open		Open a mission with the "Open" dialog.
	Save		Saves the mission. If the mission is unsaved, the "Save As" dialog appears.
	Save As		Give mission a new name and save it.

	Export	Shift X	Compiles map and saves to SDF format.
	Batch Export	Ctrl Shift X	Exports multiple missions at once.
	Import Heightmap		Opens a file dialog to select a 513x513 pixel greyscale TGA for a heightmap.
	Exit	Alt F4	Exits XEd.
Edit			
	Undo	Ctrl Z	Undo (up to) the 500 latest commands.
	Preferences	5	Displays the preferences dialog.
View			
	2D Map	Shift F6	Toggles the 2D map.
	World brush options	Shift F2	Toggles the WorldBrushOptions dialog.
	Texture brush options	Shift F3	Toggles the TextureBrushOptions dialog.
	Instances	Shift F4	Toggles the Instances dialog.
	Scripts	Shift F1	Toggles the Scripts dialog.
	AI Hint options	Shift F7	Toggles the AI hint options dialog.
	Reset all windows		Reset all windows to default positions.
Commands			
→ Data	Reload all textures	F9	Reloads all textures from disk.
	Reload all models	F8	Reloads all models (MRB files) from disk.
	Reload all splatscripts	F7	Reloads splat scripts from disk. Re-textures the terrain, destroying changes.
	Reload Mission	F6	Reloads the entire mission from disk.
	Reload environment	F10	Reloads the environment data from disk.
	Regenerate splatting	F5	Regenerates texture “splatting”. This will delete any texture painting.
	Regenerate lightmap	F3	Regenerates the light map. Useful after changing environment.
	Regenerate minimap	F4	Regenerates the mini-map.
	Generate fractal heightmap	6	Displays the Fractal Heightmap Generator dialog.
→ Camera	Add camera position	F12	Adds a camera position instance.
→ Snap	Snap all instances		Snaps all gridded instances to the grid.
	Put on ground		Puts selected instance to ground.
→	Mission stats	1	Displays the mission settings dialog.
	Select environment	2	Displays the select environment dialog.
	Playfield coordinates	3	Displays the playfield size dialog.
	Floodfill	4	Displays the flood fill dialog.

Weather preview			
	Cycle weathers		Cycle between weather effects. The status bar displays the current weather.
Help			
	Help topics	F1	Displays the help file.
	About XEd		Displays the about and credits dialog.

The tool bar

The entire first row of buttons on the tool bar consists of commands available from the menu bar, and so will not be repeated here. However, it can be good to know that the tool bar buttons display a visible “pressed” state when they are activate. This is useful for toggling the different option dialogs on and off. Sometimes, dialogues end up on top of each other, so be sure to move your dialogs looking for that last one.

Group	Button	Shortcut	Description
World tools	Paint to height tool	Z	Selects the paint to height tool. Use this tool to raise or lower terrain towards the target height.
	Paint +/- tool	X	Selects the paint +/- tool. This tool can both raise and lower terrain by the Step value.
	Set height tool	C	Selects the set height tool. This tool picks the height under the cursor for other tools.
	Ramp tool	R	Selects the ramp tool. This tool can create height field ramps using the current brush.
	Inspect tool	J	Selects the inspect tool. This tool displays passable and non-passable areas on the map.
	AI hint tool	H	Selects the AI hints tool. Use this to paint hint maps, describing valuable areas to the AI.
Texture tools	Texture paint tool	V	Selects the texture paint tool. This paints with the current texture, brush and pressure.
	Texture erase tool	B	Selects the texture erase tool. Erases texture painting, reverting terrain to splat script defaults.
Instance tools	Instance edit tool	F	Selects the instance edit tool. This tool is used to select, move, rotate and resize instances.
	Instance delete tool	T	Selects the delete tool. This tool deletes selected instances when it is selected.
	Instance copy tool	Y	Selects the copy tool. This lets you place copies of the selected instances.
	Instance grep tool	U	Selects the grep tool. This tool selects all instances of the same type.
	Instance pick tool	I	Selects the pick tool. This tool sets the selected type to the type of the instance clicked on.
	Instance place tool	Q	Selects the place instance tool. This tool places instances of the currently selected type.
	Area place tool	Ctrl Q	Selects the area place tool. This tool places areas for use in scripts.

	Ruler tool	TAB	Selects the ruler tool. Click the ground and hold down your mouse button to measure distance.
	Choke points tool	K	Selects the choke point tool as the current tool. This tool defines chokepoints in the world.

Other commands

Some commands are listed neither on the menu bar or the toolbar, but has key shortcuts and mouse commands. LMB, RMB and MMB refer to the left, right and middle mouse buttons in the list below, and "MWheel" is the wheel on most mice.

Command	Shortcut	Description
Delete	DEL	Deletes the currently selected instances.
Deselect	F2	Deselects all selected instances.
Move up	W	Moves camera forward.
Move down	S	Moves camera backwards.
Move left	A	Moves camera left.
Move right	D	Moves camera right.
Move around	RMB	Hold this and move the mouse to move around the map.
Draw gestures	MMB	Hold down and move mouse to draw gestures.
Free look	CTRL RMB	Hold down and move mouse to look around.
Slower	ALT	Hold down while looking or moving to move slower.
Change altitude	MWheel	Raise and lower the camera.

Gestures

The middle mouse button, or MMB, is used for gesturing, i.e. invoking commands by drawing a pattern on the screen. Predefined gesture patterns can be assigned to commands in the preferences dialog.

The status bar

The status bar shows information about mouse pointer coordinates, the number of placed instances and the position and rotation of the currently selected instance. When Cycle Weathers has been used, the current weather is also typed out here, on the far left.

Modes

You could say that XEd can work in four distinctly different modes. The first three all deal with the three-dimensional display: World editing, Texture editing and Instance editing. The fourth – or first, depending on tastes – deals with script editing and all kinds of JUICE file editing. It is not a mode in the same way as the first three, but it is a distinctly different area to work in.

World edit mode

The first mode, World Editing, is where you use the different tools to raise, lower and sculpture the land according to your plan or inspiration. There is a levelling tool, a raise / lower terrain tool and a ramp tool, which can be immensely useful to creating terrain which agents should be able to move across. It is always a balancing act to create a map which looks realistically rough but which is also even enough that vehicles can drive across it without looking like they're running the slalom.

Texture edit mode

In the second mode, Texture Edit mode, the four or six available landscape textures can be painted onto the surface of the terrain with various pressures. This way, the few available textures can be combined to create at least sixteen distinct combinations. A flat dark rode can have a vague covering paint of cracked ground to make it look rougher, or mountainsides could have a thin paint of dark asphalt to make it look shaded and varied.

Some basic texturing is performed automatically by what is called “splat scripts”, which decide that mountainsides have one default texture, low and even terrain have another texture, high even terrain have a third texture and so on.. but these basic outlines are not suited for all terrains, and will look repetitive unless touched-up using the Texture Paint tool. Use a low pressure and a wide soft brush for easy touches, or raise the pressure to draw sharp distinguishing lines between textures.

Instance edit mode

In this mode, objects are placed, moved, copied, selected and deleted from the mission. This is the mode in which tanks are placed on the ground or moved across it, areas are placed and drawn up to size, ambient sounds are sprinkled throughout the landscape and animated objects are placed and rotated to where they will stand so that the cinematic cameras can see them for cutscenes and the like. Therefore, this mode is closely tied to the final and most complex mode – script editing.

Script edit mode

The script mode is the “programming” of level design. This is where objects become interactive and where the in-game story takes place. A simple multiplayer map does not have a lot of scripts: START is always where the map begins, and then there is a script for sounds, one for zones and maybe one for neutral gun emplacements if you want them. Using various commands these scripts are activated when they should be. A “Trigger” waits for all players to connect to the game before creating landing zones, letting the map begin. A more complex multiplayer map may contain script events for taking over bases, stealing a flag, destroying important buildings or vehicles, entering a minefield, winning the game on a special condition or any one of a thousand possibilities.

A single-player mission can have several dozen scripts, sometimes as many as fifty or sixty, with sweeping camera angles and cinematic cutscenes, complex mission objectives with multiple actions that need to come together. It sounds difficult, and it is difficult, but it needs only be as difficult as you want it to be. It is up to the designer to decide how complex the map should be.

This mode will be described in the chapter The Basics of Scripting and then in greater detail in the Script Reference.

Tools

What follows is a listing and descriptions of the tools available to you in the World, Texture and Instance edit modes. All tools will activate their relevant options dialog as soon as they are selected. These dialogs can also be toggled from their corresponding tool bar buttons.

World edit tools

These are the commands available in World edit mode. As soon as either of these commands are selected, the World brush options dialog becomes visible.

Name & key	**WorldPaintToHeightTool – Z**
Undoable	Yes
Description	This tool uses the currently selected **Brush** from the world brush options dialog, and the **Pressure** setting to change the height under the cursor towards the target height. The higher the **pressure**, the faster the change.
LMB	Paints world up or down toward **Target Height** using **Pressure** with the current **Brush**.

Shift MWHEEL	Changes the current **Brush**.
Name & key	**WorldPlusMinusTool – X**
Undoable	Yes
Description	This tool uses the currently selected brush to increase or decrease the height of the heightmap with the number of meters of the **Step** value.
LMB	Increases heightmap **Step** meters using the current **Brush**.
Shift LMB	Decreases heightmap **Step** meters using the current **Brush**.
Shift MWHEEL	Changes the current **Brush**.
Name & key	**WorldSetHeightTool – C**
Undoable	No
Description	This tool picks the current height of the heightmap under the brush and sets it as the current Target Height. With a large brush, the average height of all points under the brush is used.
LMB	Sets **Target Height** to the height currently under the mouse pointer
Shift LMB	Same as above but rounds the height off to the nearest half meter.
Shift MWHEEL	Changes the current **Brush**.
Name & key	**WorldRampTool – R**
Undoable	Yes
Description	Click and drag with this tool to create a line which represents the ramp. When mouse button is released the ramp is created. Ramp is effected by **Pressure**.
LMB	Press the mouse button where you want the ramp to start, and hold your button down while moving to the position where you want the ramp to end. Release the mouse button to create the ramp using the selected brush.
Shift MWHEEL	Changes the current **Brush**.
Name & key	**WorldInspectTool – J**
Undoable	No. This tool only displays data, it alters nothing.
Description	This tool displays a 32x32 tile overlay that shows passable and non passable tiles
LMB	Hold down to show the land path map, for wheeled and tracked vehicles.
Shift LMB	Hold down to show land-and-water path map, for hover movers.
Ctrl LMB	Hold down to show land-and-woods path map, for infantry movers.
Name & key	**WorldAiHintsTool – H**
Undoable	No
Description	Use this tool to change the values of the tiles in the various AI hint maps.
LMB	Changes the value of the tile in the current “hintmap to edit” to the “Strength” value.
Name	**ChokePointTool**
Undoable	Partially
Description	This tool paints tiles in the chokepoint map, and associates a group of tiles to a chokepoint instance. **Note**: This tool is unfinished and buggy.

LMB	Click and drag to paint in the choke point map. When LMB is released, an instance editor is displayed and you can manually assign area instances that represent AI objectives to the left and right sides of the chokepoint
Shift + LMB	Remove tiles from the choke point map

Texture edit tools

Name & key	**TexturePaintTool – V**
Undoable	Yes
Description	Use this tool to paint textures on your current heightmap.
LMB	Paints the current **Texture** using the current **Brush** and **Pressure** value
Shift WHEEL	Changes the current **Brush**.
Ctrl Shift MWheel	Changes the current **Texture**.
Name	**TextureEraseTool – B**
Undoable	Yes
Description	This tool reverses any texturepaining back to splatscript default.
LMB	Erase the texturepainting using the current **Brush**.
Shift MWHEEL	Changes the current **Brush**.
Ctrl Shift MWheel	Changes the current **Texture**.

Instance edit tools

Name & key	**InstanceEditTool – F**
Undoable	Yes (selection, No)
Description	This tool is used to manipulate (select, move, rotate, and resizing (areas)) instances.
LMB	Click to select an instance, drag to select multiple instances.
Shift LMB	Adds a new selection to the current selection
Ctrl LMB	Rotates the selected instance or instances. Gridded instances can not be rotated when more than one instance is selected.
Shift Ctrl LMB	Rotates the selected instance(s) and snaps to SNAPVALUE degrees
Alt LMB	Moves the selected instance or instances.
Shift MWHEEL	Changes the height of the selected instance(s)
Name & key	**InstanceDeleteTool – T, DEL**
Undoable	Yes
Description	This tool deletes the selected instance(s) when activated.
Name & key	**InstanceCopyTool**
Undoable	Yes
Description	This tool copies the selected instance(s), attaching the copies to the pointer.
LMB	Click to place the copies
Name & key	**InstanceGrepTool**
Undoable	No

Description	This tool selects all instances of one type
LMB	Click to select all instances of that type
Name & key	**InstancePickTool**
Undoable	No
Description	This tool sets the current instance type to that of the picked one.
LMB	Click to select the current instance type
Name & key	**InstancePlaceTool – Q**
Undoable	Yes
Description	This tool places instances (props, things, sounds, agents) in the world.
LMB	Click to place the current instance type.
Ctrl LMB	Rotate the instance being placed
Shift Ctrl LMB	As above but snaps to ROTATION SNAP VALUE degrees
Name & key	**InstancePlaceAreaTool – CTRL Q**
Undoable	Yes
Description	This tool places areas (used in the scripts) in the world
LMB	Click and drag to place an area
Name & key	**RulerTool – TAB**
Undoable	No
Description	This tool measures distances
LMB	Click and drag to measure distance (meters in world space)
Name & key	**Wizard Manager - L**
Undoable	Depends on the wizard
Description	This tool invokes the wizard manager, which helps you with some tedious tasks.
LMB	Depends on the wizard

World editing

Landscapes and environments

The first choice you will be asked to make when creating a new map is what Landscape you want to use. There are five usable landscape types in Ground Control II, and one that does not make good-looking maps, but is very nice to make maps on. The type of landscape you choose decides what textures are available, what "environments" you can choose – normally day, night and either twilight time – and perhaps most importantly, what things and props you can use.

Battleground

"Battleground" is the bluish, rubble-filled landscape of something that used to be a city. It comes with five different textures, four for flat ground and one for vertical walls, both east-to-west and south-to-north. Among these textures you will find rough dark concrete, broken bright stony or rubble-strewn ground and two different thicknesses of garbage-filled ground. One of these textures has "auto-props", small objects sticking out of it, in this case steel girders of fallen buildings.

The props for this landscape consists of buildings and houses both broken and whole, lots of different kinds and sizes of rubble, concrete bases to put buildings on, sidewalks, harbour equipment and other post-apocalyptic city paraphernalia. There are also some NSA base building blocks. For the environments, “Space day” is very blue, the “space morning” environment is yellow and the “space evening” environment is very, very red. The “day” seems darker than the other two, but it’s a trick of the blue light. “Morning” is the brightest of the lot.

Desert

The dry yellow-reddish sands and sandstone of a desiccated alien landscape. This terrain has six distinct textures, although two of them are only usable on more or less sheer cliff sides. The east-to-west cliff side texture is at least different from the north-to-south texture, which increases variety. Among the other textures you will find dry hard-packed sand dunes, rough darker gravel, sandstone just barely covering shimmering metal ore and one rougher, stonier texture useful for regions mountainous without being vertical. This is a texture set where combinations can lead to a great many varieties, as most of these textures work very well together.

Desert comes with the expected props: Rocks, stones, rocks, dry bones and strange alien vegetation, alien relics and small buildings of Imperial and Viron designs. There are two different environments, one for an orange evening and a morning with a lighter fog, darker ground and a very bright western sky on an alien planet.

Farmland

This landscape was once something like farmlands, but due to the passage of time it has since turned into a swampy verdant jungle. Everything is green, except in the “FL_Night” environment light, where everything turns a sandy dark orange colour for some reason. There are only four textures, two of which are more or less covered in a tall grass, one uneven grey-black rocky surface and one pattern of swampy trampled grass and vegetation.

The available props are consists of many kinds of vegetation, alien trees, giant flowers, xenofacts, strange broken tree trunks and overgrown old buildings. For the base-builder, there are many different kinds of building platforms and ramps, drills, industrial equipment and energy barriers. Several of the vegetation props are so-called “blocks”, single objects which cover a large surface area and look like many small trees standing close together. Combined with “break” objects, to break the monotony, these can be used to quickly cover large areas in realistic-looking vegetation.

There are many environments available in this landscape, such as the FL_morning, FL_evening, FL_sunset and FL_noon which is the brightest of the green environments. The orange Night setting is confusing, but looks good and has a very dark night sky overhead.

Icevolcano

Combining a black volcanic surface with very white snow dunes results in a very versatile landscape type. There are five textures – the two vertical textures are slightly different, but similar enough – and among these five we find one very black basalt / cooling lava texture and one very bright blue-white fairly smooth snow texture, as well as two textures of blue and white glacier ice, where one is has rounded cracked pieces and the other is more chaotic.

Environments are the expected Evening, Morning and Noon. Yellowish morning light casts a golden sheen over the ice, trailing blue shadows across the darkening terrain. The Evening environment is cast in purple light and very faded blue shadows while the noonday sun high in the sky makes the snow glow very white with short but very dark shadows.

For this landscape to truly have any volcanoes, props need to be used. There are several crater props, smoke belching objects and even lava rivers, along with trees and great chunks of ice. This landscape also features many different kinds of NSA base equipment covered in a thin layer of snow.

Islands

This is a very versatile subtropical landscape featuring sand, grass, dark rock or concrete, and mountainous rock, as well as both types of vertical, rocky textures for cliffs and mountains. “Islands” is the perfect landscape for recreating the white cliffs of Dover or most European inland settings, or beachhead missions. Available props range from all kinds of trees and bushes to many facilities and buildings for both the NSA and Terran Empire.

The available environments are morning, noon, evening and night, all of which are fairly neutral. The evening light is bluish while the night has a sound measure of purple light from a beautiful setting sun on the horizon.

Orange

While all the other landscapes are intended for actual use when creating missions for games, the Orange landscape type was only created in an attempt to increase visibility while editing the heightmap. The slope map is immediately visible in red textures slopes and all textures contain a grid, making the shape of the ground more easily visible. There is also a green texture, useful for doodling roads and highlighting strategically important areas of the map while planning how you want the map to be played.

Use this landscape to create your map with clear visual cues, then import it into your “real” map with your desired landscape. Actually creating a map based on this landscape might not even work in the game. Warning: Not all environments listed for the Orange map work. Some might even crash XEd, destroying your unsaved work.

MissionStats

The mission stats (for “statistics”) juice editor is available both as a button on the tool bar and on the Command drop-down menu. One very important part about the mission stats is that any map which is going to be played by AI’s – which means most multiplayer maps for skirmishing, and all single player maps – need to have an AI player defined under myPlayers. This is in order for XEd to export the full data necessary for AIs to play the map. If there is no AI player properly configured for the map, the map will crash if it is started with AI players present.

Initial options

This is where individual map settings are stored, such as the name and description of the map, whether it should be sorted under Multiplayer or Single player maps, how many Acquisition Points each player starts with, and many other settings.

The settings for name and description are fairly obvious. Normally, the name is kept short and the description is written to explain what needs to be done to complete the level, how large the map is and what the general objectives are. This space is limited, however, so keep it short.

The Mission “Type” only really defines whether the map will be sorted as a multiplayer map and show up on menus for on-line play and LAN, or if the map should be sorted as a single-player map under Custom Missions. There is no other difference between the two types of map.

When the map begins, all players should have some currency to play with and order their first units to the ground, or upgrade some critical system on the drop ship. This starting wealth is defined in the myInitialAP value. This is how many AP each new player will get when he or she connects to the game. There is nothing stopping you from setting this value insanely high. Go right ahead. 7000 AP is the default.

Further down, we also find myWindDirection. This value controls the behaviour of smoke and clouds, and ripples on water. The wind direction is measured as a three-dimensional vector which should end up at around 10 total. The default value is an X and Z of 4, and a Y of zero. This means a wind coming down from the north-east corner of the map, blowing toward the south-west corner of the map.

There are also settings below the middle called myLoadingScreenImage and myLoadingScreenText. If you have ever started a mission, you know what a loading screen normally looks like, and that the text appears in the designated top half of the screen. If you

just want a pretty picture, feel free to leave the text field empty. Loading screens need to be in the DDS format, and will be resized to fit the screen while the mission loads. There is no restriction on where loading screens must be loaded from, but there are several screens available in the default directory UI / backgrounds and its subdirectories.

Player settings

After wind direction, the players are defined. The values stored in myPlayer are absolute for Single-player maps, but can be changed in multiplayer games. For example, when an AI plays a multiplayer map, its settings are loaded from a completely different file.

The myPlayers branch contains data on all eight players, most notably whether they are scripted, AI or human, what faction they belong. Valid faction names for Ground Control II are NSA, “Terran Empire” or Viron, but if more factions have been added, this is where the exact faction name should be entered to have the player use those instead.

After myFaction we see their team membership, then a setting for what colour it should have. In multiplayer, colours can be selected, but that selection must be made from what colours were attributed to the eight players under myPlayer for the mission. Stick to the defaults here and you’ll be fine. Then comes the player’s name. Again, this is irrelevant in multiplayer but in Single-player, this is often how the different players are identified. This name appears over units belonging to that player when a mouse pointer is held over them.

The setting myAIConfigFile is important only for players of myType AI, and is not used at all for SCRIPT and HUMAN players. For AI players, the contents of this file – ICE files found in the CommanderAI directory – determines most of the behaviours of that particular AI.

myBasePosition is rarely used, or at least it shouldn’t be used because every Landing Zone already has a base position defined for it. This value is the “default” base position, where drop ships come from, if the player does not have a current Landing Zone. Because you cannot call a drop ship unless you have a Landing Zone, you should be able to safely ignore this value.

The two lists of Drop ship levels – myDropShipStartLevels and myDropShipMaxLevels – together with the three other myDropship values, all control the behaviour and capabilities of the drop ship. “Start levels” are the starting levels for all six types of upgrades, and the “Max levels” are the maximum possible levels for each statistic, combined with the value “Max Total Attribute Levels”, which controls how many “points” can be spent on the drop ship in total on this map. If this value is set to 15, a player could increase three statistics by five points, and then be unable to raise any further statistics, even if the player could afford it. This was never used on either of the multiplayer maps because there was no suitable feedback to let players know what was happening. It might still be a useful setting, however.

If a drop ship is shot down, the “Respawn Time” is how many seconds will pass before a new drop ship becomes available. The default value is 300 seconds, or five minutes. This may not seem like such a long time, but when you really need new troops, five minutes plus the time it takes to fly to the Landing Zone in a drop ship with standard engines can be an eternity.

Drop ships can also be turned off completely by changing “Drop Ship Is Available Flag” from TRUE to FALSE. That means no drop ship will be available to the player for the entire duration of the map.

Allowed and disallowed

By default, all agents and support weapons are allowed on all missions. Virons are also allowed, on all missions, to meld any unit into a new form. However, this behaviour can be changed using the three “Disallowed” lists. By selecting an agent on the DisallowedUnits list, that unit can no longer be requisitioned using Acquisition Points. By selecting a type of support from the DisallowedSupportThings list, that type of support will no longer even be visible on the support menu. Finally, by adding certain melded Viron agents to the DisallowedMelds list, the Viron can no longer create that agent through melding. The reason this is different from DisallowedUnits is that Virons cannot requisition these agents anyway, and so there needed to be a special way to stop melding. However, by blocking melding in this way, players may become confused that they can no longer meld units that are normally able to meld. Use this with discretion, or thoroughly inform the players that they will not be

able to use those agents on this map. To select more than one item from these lists, hold down CTRL while clicking on them, or dragselect in the list.

Victory conditions

If you want a simple way to win, just set a score limit or time limit. Both of these are just default values for the map, and can by changed when the map is started. If myScoreLimit is higher than 0 (zero), the team or player who first reaches that high a score will win the game. Please note that Score is not exactly AP, but must be gathered during the course of the game by destroying enemy agents and taking zones.

When myTimeLimit is higher than 0.0, the game ends when the time runs out and the player with the highest score wins. Please note that myTimeLimit is stored in **seconds**, not minutes or hours, so 300 would be a five minute mission and 3600 would be an hour-long mission.

At the bottom of myMissionStats we find five values that “Allow” different four kinds of “defeat” and one kind of victory. “Allow scripted victory” is TRUE by default, and means that the script command SCRIPT_SucceedMission can be used to give victory to a team. This is how Single-player missions are won, and so by changing this setting, a single-player mission can be easily converted into a multiplayer map by disabling “Winning through following the story” so to speak. Playing a single-player map with this setting would normally mean it never ends, but that’s where you could either add a time limit, a score limit, or activate the other four “defeat” conditions: You do not win because you did something fantastic.. you win because your enemies have all been defeated.

To allow “No LZ” defeat means that if a team loses all of their Landing Zone, they are defeated. In a two-team match, that means the other team wins. This setting is FALSE by default, since losing an LZ does not necessarily mean you cannot get it back.

Allowing “No VL” defeat is similar to the above but deals with Victory Locations. Using this setting is also similar to the old Ground Control game type called Flag Zones. If one team controlled all the Flag Zones, that team won the game. This is exactly it. If one team controls all Victory Locations, this setting will mark all other teams as defeated. This setting is TRUE by default, because most missions are intended to be won by taking all VLs. That’s why we call them VICTORY LOCATIONS.

“No units” defeat is self-explanatory. If this type of defeat is set to TRUE, any player who has lost all of his units will be considered defeated. Please note that not all types of agents are included in “no units”. If all you have left is a hovering radar support thing, you have still lost because a little tiny radar is not going to win your war. If you do not approve, make sure the “Allow no units defeat” setting is set to FALSE, as is the default.

The “No Presence” defeat is a little more complicated. It is a combination of “No LZ” and “No Units”, basically, in that a team will not until they have no units AND no way of getting new units. Normally this means they have lost their last LZ, their last usable unit, and they have no way of getting new ones. This type of defeat is allowed by default.

Heightmap creation and editing

In World Edit mode, all tools either analyse or manipulate the three-dimensional height map which constitutes the battlefield terrain. The six tools related to this mode all have names that begin with “World” and end with “Tool”. The five first are, from left to right on the tool bar: **Paint To Height, Plus-Minus, Set Height, Ramp** and **Inspect** There is a sixth tool, **AI Hint**, but it works a little differently and will be explained further on.

Floodfill

Before you begin creating your own height map, it might be good to know what “floodfill” means. When you start an all-new map, it will be covered in water and have no distinguishing features. The reason is that the terrain starts at level 0 (zero) out of 127 meters, and the water level is somewhere between 19.0 and 19.5 meters above that level. Using flood fill, on the Command menu, you can enter a number value between 0 and 127 and that value will become the new lowest terrain height. Any point lower than this value will be filled with new ground. If the flood fill value was above 19.5 meters, no water will be left visible on the entire map.

This is a simple way of getting a starting situation where you can both raise and lower land, instead of just raising islands out of the water the time. Just a tip.

World Brush Options

As soon as any one of these tools (except AI Hint) is selected, the World brush options dialog panel will appear, if it has not been toggled to visible already. This dialog lists three different values, one Brush selection box and two option checkboxes.

Target Height

The first value is **Target Height**, used by the **Paint To Height** tool. This value is the height in meters and half meters above the absolute bottom of the map. Sea level is normally just below 20 meters. This value signifies the height towards which all brush strokes will strive. If the painted terrain was higher than this value, it will be lowered, and vice versa if the painted terrain was lower. More or less eventually, drawing repeatedly on a map with this tool will create a flat map of this altitude. The **Paint To Height** tool is therefore a useful way of creating plateaus, flat roads, mesas or mountain chains of the same average height.

Pressure

The second value is Pressure, in percent from 0% to 100%. This is how much each “brush stroke” should contribute towards the target height. Setting it too low will have no effect at all while painting, because no single brush stroke will make enough of a difference. Setting this value too high will make a single brush stroke set the terrain instantly to the target height. Change this value to make your brush strokes have a more or less drastic effect on the terrain.

Step

This value alters how the Plus-Minus tool affects the terrain. Every click with the tool will change the terrain by this amount of meters either up or down. This can be useful for creating step-like areas, but too high values will almost definitely create unreal-looking terrain and textures will look stretched.

Brush

A brush is a small greyscale image which is used to make an impression on the terrain or textures of the world. The default setting is a round, smooth brush which is more intense in the middle, and weaker out towards the edges. Painting with this brush can get you rounded hills, or cauldron-shaped valleys. There are several both smaller and larger brushes of the same type, as well as square brushes with sharp edges, brushes that look more random and a few odd brushes for sharp corner walls. The whiter the brush, the sharper the impression on the map.

Combined with high **Pressure** or **Step** values, a single click with a brush will imprint the exact look of the brush on the terrain. Keep the pressure and step low and move your mouse around while painting to get real brushstrokes and rolling, extended swathes of altered terrain. Experiment until you get the hang of it.

Update Textures while painting

This option is on by default, meaning that the “splat script” textures will be automatically regenerated while you change the altitude of your terrain. While first trying out a terrain, this is okay, but once you have used the Texture edit mode to alter the look of the terrain surface, you might want to disable this option or the automatic retexturing will destroy any texture changes you have made.

Default texturing can always be recreated by either of the menu options **Reload all splatscripts** and **Regenerate Splatting**, both of which will reset all texturing, but it can also be done using the TextureErase tool which uses brushes to restore only desired areas on the map to default texturing.

Update textures on import heightmap

This option works much the same as **Update Textures while painting**, except it works on imported height maps. There are times when you want to edit a height map in another tool such as Adobe® Photoshop™ or Terragen® and then re-import the altered height map into

XEd. If this option is checked, all textures will be reset to the splat script standards. If the option is unchecked, the terrain will keep its current texturing even after the new height map has been imported. If large changes have been made to the terrain, it may be best to keep this option enabled, but if only minor changes have been made to altitudes and the shape of hills and valleys, keep it disabled.

Again, if you only wish to retexture some areas of the map, the **Texture Erase** tool may do the job better.

Paint to Height tool

The **Paint To Height** tool is the default tool for world editing. Use it in conjunction with the Set Height tool to create terraces, calderas, plateaus, flat beaches, wind-eroded Texan cliffs and Scandinavian raukar. Use it in long drawn-out lines to create rivers and oceans. This tool can raise mountains that end up having flat tops and sharp sides, whereas the Plus-Minus tool would end up raising the original terrain with all its unevenness. Drawing the terrain with a high pressure is a good way to start creating the rough outline of he map, but eventually a lower pressure will create the better-looking map. It is better to use a lower pressure but more repeated strokes. One drawback of the **Paint To Height** tool is that when the painted area reaches the target height, the terrain becomes more and more flat the more brushstrokes are applied, and flat terrain very rarely looks natural.

Set Height tool

The **Set Height** tool is similar to the “colour picker” tool found in many art programs. The height map can be described as a greyscale picture, and this tool can be used to pick a shade of grey so that the **Paint To Height** tool can then paint using that exact shade of grey. The combination of these two tools will allow you to expand areas that already lie at a certain altitude, such as beaches, plateaus or mountains of a certain average height. Switch between the two tools quickly using the C key for **Set Height**, and Z key for **Paint to Height**.

Plus-Minus tool

Because the **Plus-Minus** tool can both raise and lower terrain at the press of a single button – shift – it is a very useful way to alter an existing terrain. Height differences can make a height map look larger and more complex. One benefit of using the **Plus-Minus** tool instead of **Paint To Height** is that the original surface texture, if the terrain was already full of small bumps and dunes, will still be there after the terrain has been raised or lowered. The **Plus-Minus** tool is also useful for creating slopes and is great for making minute changes to a terrain, especially in conjunction with the Inspect tool, to make areas passable or non-passable. The **Ramp** tool is even better at this, though.

Ramp tool

The **Ramp** tool is a quick, easy and even fun way to create passable areas from one part of the map to another. Because the ramp can become perfectly flat but yet tilted, it can also be used to create flat roads in rough landscape, or to extend a riverbed in a straight line. It is also your best bet for evening out bumps and roughness in sloping terrain.

Use the pressure value to affect how powerfully the ramp will affect the underlying terrain features. At 100% the ramp is an absolute line from start to ending, while at 10% the terrain still maintains some semblance of what it looked like without the ramp. Use low pressure settings to create terrain which seems to slope naturally rather than artificial-looking ramps from point to point. Be careful what brush you use for this tool. A square brush might seem like a good idea but is only really useful for north-to-south or east-to-west ramps. Additionally, square brushes will create large flat areas in the starting and ending position. Rounded, smooth brushes are better because they can go in all directions and will not cause too sharp edges.

Fractal terrain generator

The fractal terrain generator can be found on the Command menu, under the Data heading, click the option at the bottom entitled “Generate Fractal Heightmap”. Save your map before you start playing with it.

Fractals can be a quick, easy and even fun way to create good-looking maps. However, these maps are rarely immediately useful because the random nature of fractals will mostly allow you to create a terrain of a general type. It will never be a map with logical places for bases, victory locations and landing zones with roads in between. Despite this, terrains created with the fractal terrain generator can be both beautiful and realistic, and can be excellent starting points for your own ideas.

Do not be afraid to play around with the settings, loading the various presets and try rendering them one by one, but be careful not to set Iteration values too high, as this can take a lot of time to render. The end result will have much more detail, though.

It is also possible to use just the filters to change the look of your current terrain without destroying it altogether. You can do that by selecting only the Filters tab and raise the "Smooth filter" value, and click "Apply current". It is also possible to create a fractal heightmap and then "blend" it with the current terrain so that some existing features remain. Do this by making sure that the "Clear heightmap" option in the lower left of the dialog is DISABLED, and that the "Blend Value" is somewhere on the scale. Left is less, right is more. In early versions of XEd, "Clear heightmap" was the opposite of what was intended – if the box was checked, the height map was cleared before applying fractals, and if the box was empty, the heightmap was destroyed first. The "blend value" can also be used to control how much of an impression each type of fractal calculation will make on the final heightmap.

Occupancy grid and slope maps

Inspect tool

While you work on the height map, use the Inspect tool regularly to see the various "slope" maps. This shows you what areas will be passable by different types of agents without the need to save, export and play the map in the game. Use the tool frequently while moulding the terrain to your desire making sure that the mountainsides you just created are indeed un-scaleable, and that roads are flat enough for vehicles to drive on.

Hold down the left mouse button and the desired modifier keys to display the three different kinds of passable areas. Green means an agent can pass, red means it cannot. There are three different kinds of path map. The first one is "Land", for wheeled and tracked vehicles such as the NSA Engineer and NSA tanks. Water, forests, and buildings are off-limits to these movers. The second map, which is displayed by holding down SHIFT and the left mouse button, is the "land and water" path map which contains all the areas that can be traversed by hovering agents such as Viron and Imperial vehicles. The third path map, displayed when CONTROL is held down, is the land-and-woods path map, which indicates where infantry can go.

Show occupancy grid

When using this tool, it can be very good to also activate "Show occupancy grid" on the Instances dialog panel. To do this, you first have to make the Instances dialog visible. You can do this either by selecting an instance tool, or by pressing the ViewInstanceDialog button on the tool bar.

The Inspect tool displays what effect the shape of the ground has on the ability to pass, while the Occupancy grid shows what areas are blocked and made passable by Props. Red means none can pass, black means infantry can pass and white means that anyone can pass, even if the ground underneath is ragged enough to produce a red "slope" map in the Inspect tool. This is how bridges can cross canyons. The canyon is full of red impassable slope tiles, but the bridge's centre line is all white, and so vehicles can pass over it and over the canyon.

Please note that if a prop surrounded by a white occupancy is placed near a sheer cliff, agents will be able to "walk on the air" around this prop. Be careful!

AI hints

When the **AI Hint** tool is selected, the Hints options dialog appears if it has not been made visible already, and the mouse cursor will be surrounded by a coloured grid indicating the current values of the selected influence map. There are three kinds of influence map: Strength hints (dark green to bright orange) which affects how valuable the AI will consider

the marked spot of land, Support map (pale green to sharp blue) which affects how important the AI will believe it is to use support weapons on the ground and Minefield map (dark purple to lime green) which was intended to affect the AI's idea of where minefields should be created.

The AI hint map editor is based on tiles 6 meters by 6 meters, same as the grid used for occupancy and path maps. Hint values can be changed by selecting a "Strength" from the Hints options dialog and click on the ground in the 3D view, to paint tiles in that strength colour. All tiles have the value 127 by default, which is the middle value. The relative importance of an area can then be controlled by painting it both weaker and stronger according to taste. This is a good way to make the AI stay away from certain areas which would otherwise seem good – such as high ground which is hard to get to – or to prioritize areas without any obvious advantage, such as the flat ground in front of a base defended by the AI.

If choosing between two positions of otherwise equal value with regards to their proximity to the objective, their altitude over zero-level and so on, the AI will always choose the spot which has also been given additional strength on the AI Hint map. The Support and Minefield hint maps work much the same way, but instead affects the likelihood that the AI will use support weapons or mines on the painted areas. Increase support hint strength on natural roads and choke points, where it is good or at least looks good to use support weapons.

Choke points

From the AI Hint Map dialog you also have access to the list of current choke points, which are added by the Choke point tool. Items in this list can be double-clicked and edited, as well as deleted. To add a choke point, select the special Choke Point tool on the toolbar, and hold down your mouse button while drawing over the terrain in natural choke points. White squares will be created as you draw, and all of the selected white squares will become part of the new AI choke point. When you release the mouse button, a juice editor will pop up asking you to define what AI objectives exist on either side of this choke point. Try to list them as logically as you can, as a choke point is defined as the only, or at least the best way to pass between two strategically important areas. Note that chokepoints are not implemented in Ground Control II, and will have no impact on gameplay.

Texturing and splatting

Textures are the images which cover the three-dimensional objects in a game to give their surfaces a realistic look. The terrain, height map in XEd is textured automatically by something called "Splat scripts". "Splatting" is our name for textures that can be painted on the ground in several transparent layers, on top of each other, to create a great number of variations through combination.

Texture Edit tool

The default textures created by the "splat script" are usually a good starting position, but they are repetitive. To create variety, try painting with other textures on a very low pressure and a soft brush. Using thin layers of darker and lighter textures can give repetitive ground textures a much more varied and realistic look.

Try to mark roads in hard rock and concrete, and rough terrain with uneven rocks and vegetation textures. Add rocky textures to hillsides and cliffs, but be careful not to use vertical textures – the last two textures in six-texture landscapes – on flat terrain as they will look horribly stretched. Paint over such stretching with another texture and a narrow brush.

Once you have placed props to create forests and other dense vegetation, it can also help the look of these forests if you apply darker textures underneath the trees so that there is no bright texture shining through the boughs and leaves. In the same way, paint gravel or concrete inside base areas and especially in front of hangars, vehicle bays and landing zones. It just does not make sense for there to still be grass where wheels and hover fields have ripped at the ground.

Texture Erase tool

For those times when the ground looking worse after your work with the Texture Edit tool than it was before, the Texture Erase tool will use your current brush to completely reset any painted area to the default splat script look. This is less destructive than regenerating the entire splat script, but it gets the job done quickly on those difficult vertical mountainsides. Please note that “pressure” makes no difference while un-painting with this tool.

Placing and editing instances

All Instance edit tools deal with placing, selecting, moving, rotating, copying and deleting instances. The seven tools related to this mode all have names that begin with “Instance” and end in “Tool”. They are, in order from left to right on the tool bar: **Edit, Delete, Copy, Grep, Pick, Place** and **PlaceArea**.

All instances have a position in X, Y and Z directions, as well as orientation values for X, Y and Z. This orientation value can be visualised by holding down CTRL while placing or editing an instance, or just clicking on the ground and holding down your left mouse button. Remember that X is east to west, Y is north to south, and so an orientation of Y = 1 is an instance “looking” due south, while Y = -1 is an instance pointed due north. This is all further complicated by the fact that some instances, such as the BG_LightPole, extends in the direction opposite to the orientation it is heading. When oriented due south, the horizontal bar of this street light will be pointed towards the north.

Script components

Because adding Agents and Sounds will also add script components – PLAYER_AddAgent and SCRIPT_AddSound commands respectively – it is good to have the Scripts dialog visible while adding these types of instances. If you have not done so already, create separate scripts for SOUNDS and different types of AGENTS, for example AGENTS_TURRETS, AGENTS_VIRON, AGENTS_NSA and so on.

Agent and Sound script components will be added into the **currently selected script**, and because it is not possible to move contents between scripts, it is best to display the script dialog and select the script where you want these commands added before placing agents and sounds. The Scripts dialog is also useful because Things require THING_AddThing commands, and Cameras are used by TEAM_CameraSpline commands.

Instances dialog

As soon as any one of these tools is selected, the Instances dialog panel will appear, if it has not been made visible already. This dialog contains two lists, four buttons, six checkboxes for “Text labels”, a Rotation snap value combo box and seven more checkboxes for additional options.

Available Instance Types

This list view contains four branches; one each for Props, Things, Sounds and Agents. By selecting items from this list the designer can choose what type of instance to place on the map. Thanks to this list, props, things sounds and agents can all be placed in the world using the same tool: InstancePlaceTool. Only cameras and areas are placed with different methods.

Instances

This list has six categories because it contains all instances that have been placed on your map. The three buttons below this list can be used to rename instances, expand or collapse the categories, and delete instances from the map.

Be aware that changing the name of an agent, sound or thing instance which have corresponding Script components will destroy the link between the script command and instance. This will result in an unresolved reference from the command, which then needs to be connected to the renamed instance.

Double-clicking on instances in this list will let you manually edit the values for that instance. By holding down the SHIFT key while double-clicking on an instance, the main window will move to and focus on the instance that was double-clicked.

Text labels

Check the boxes to make the names of each instance appear next to it in the 3D world window. It can be very helpful to enable text labels for the type of instance you are currently working with, cameras when creating camera splines, agents when giving orders or areas when creating **Enter Area** triggers.

Rotation snap value

Instances can be rotated when using the Instance Edit Tool by holding down CTRL and dragging the mouse. If SHIFT is held down as well, this rotation will "snap", to even multiples of this value. The default is 45 degrees, which means the selected instance can be rotated in eight directions. If the snap value is changed to 90, the instance can face north, east, west and south, and at 180 degrees snap value the instance can only face north or south.

Additional options

A list on the right side of the Instances dialog contains several options which affect the way instances are placed on the map.

Lowest Y in radius

The first of the additional options is "Lowest Y in radius". This option is helpful in making certain that placed things and props are placed so that no part "floats" above the ground. Without this option, "floating trees" are a common occurrence. Enter a percentage value in the box to make XEd look for the lowest altitude in a smaller or larger area.

Example: A prop has a radius of 10 meters. If the box is checked, XEd will find the lowest y value inside that area. Enter 50 into the box and XEd will search within a radius of 5 meters, 50% of 10 meters. Enter 200 and XEd will search within a radius of 20 meters, which is 200% of 10.

Don't select on placing

When a new instance is placed, that instance also becomes the currently selected item, removing any previously made selection. Enabling this option, the new instance will not be selected and your current selection will remain the same.

Random Rotate

All instances placed while this option is enabled will be rotated in a random direction, its X and Z rotation values both randomized between -1 and 1. This option is very useful in placing vegetation, junk and other things which should appear to have been placed more or less at random.

Show occupancy grid

This option will make the occupancy grid generated by "gridded" instances visible. White tiles mean all agents can pass, red means none can pass and black means only infantry can pass. Only props can have occupancy grids, but not all do. A gridded instance can only be rotated at even multiples of 90 degrees. Read more about it under the heading Occupancy grid and slope maps.

Retain instance height

Without this option, any instance that is moved, or which stands on a portion of ground which is raised or lowered, will be moved to the new height or altitude. If the "retain instance height" option is enabled, all instances will stay at their current height in these cases. Regardless of this option, all new instances will be placed on the ground, but can be raised or lowered by using the Instance Edit tool, holding down SHIFT and rolling the mouse wheel up or down.

Use water level for instances

When this option is enabled, no newly placed instance will be placed below water. The minimum Y value for instances will be the same as the water level. Instances can still be lowered underneath the waves using the Instance Edit tool.

Override GetY

When this option is enabled, and a value is entered into the text box next to it, any moved or otherwise edited agent will be moved to this height (Y value, altitude). This is even true for

instances placed on terrain where the height map is changed – the instances will snap to the GetY override height instead of to the new terrain height.

Agents

Before placing agents, remember to bring up the Scripts dialog and select the script where you want the agents to be created. This is most likely **not** the START script, because at the time the START script runs, there are no PLAYERS present who can use these agents. The exception would be agents which are intended to belong to PLAYER_NOPLAYER such as neutral agents, gun emplacements and capturable vehicles.

Note that all placed agents will belong to PLAYER_NOPLAYER, which can be changed by either double-clicking on the PLAYER_AddAgent command listed in the Script dialog. These commands will be automatically named in a way that associates them to the agent instance that was placed: When placing an agent named F1_jeager__99, the PLAYER_AddAgent command could be named F1_jeager__99_add_agent__100 – the beginning of the name is identical to the instance name, followed by "_add_agent" so as not to confuse the instance and the command, and finally a second count, to identify this particular command.

The numbers placed on instances and script commands are a simple count from 0 and upwards, and has no other use than to distinguish commands and instances from one another. Instances and commands can be freely renamed as described in the section on Naming conventions, but when renaming an instance it is very important to remember that this will cause an unresolved reference in the related command. If F1_jeager__99 is renamed to The_Sniper_Guy, then the command F1_jeager__99_add_agent__100 must be changed so that it references this new name. While doing that, it might be appropriate to change the name of the command as well, but remember to correct any references to this command in triggers, Agent commands, AgentGroups and so on.

Agents can be freely placed, rotated, raised or lowered but any movable agent will automatically be reset to their individual altitude once the game begins. Turrets and gun emplacements will not, and can be made to hover or stick to a certain rotation value. **Warning**: Do not place movable agents in positions they could not normally move to, meaning do not place tanks inside forests or agents under water. In a worst case scenario, this can crash the game.

Areas

Place an area in XEd by selecting the InstancePlaceAreaTool (quick key CTRL + Q) and make a dragging click on the ground. With the mouse button held down, you can change the radius of the area. Release the mouse button to complete the area. The area will be given a numbered name, like Area__0 and so on. Rename it to something easy to remember, like "Area_Enemy_Base_Vault", "Area_Gold" or "Eldorado". Note that you cannot press OK on a rename if the name contains spaces.

Positions and areas are the same thing. The AGENT_MoveTo command uses areas as the position the agent should move to, and the agent will move to the centre position inside the area. When TRIGGER_TeamEnterArea uses the same area, it will consider the entire volume of the area to be part of the trigger.

For practical reasons, it is best to place areas before the triggers and command which need them. If you create the triggers first you will have to close the trigger editor, change to the AreaPlaceTool and place the area before going back to double-click on the trigger or the trigger's unresolved reference and point it to the area. It is better to have all relevant areas prepared beforehand. There is no problem at moving or resizing an area after triggers have been connected, as long as you remember that they will be affected. If an area is removed entirely after it has been used by commands, these commands' reference to the area will be unresolved, and must be connected to a new area or they will not work properly, or

Areas appear to be spherical in XEd, but this is merely a visual cue. In reality, all areas are cylinders based on the outer radius of the visible sphere. So do not worry that airborne vehicles could somehow avoid entering the area – as soon as the aircraft passes above the outer perimeter of the area, it is considered to be inside.

Areas can be freely placed anywhere on a map, but there is no point to rotating, raising or lowering them. The former because a circle is always a circle, and the latter because the sphere is actually a cylinder stretching from bottom to top of the world. The exception to free placement would be Landing Zone and Victory Location areas: in these cases the central point of the area must be accessible – white occupancy and green slope map, or the AI may crash the game.

Cameras

Placing cameras is the easiest thing in XEd once you have learned how to move and look around. Just position your view in the position and direction you would like to place a camera, and click the "AddCameraPosition" button on the toolbar, or press F12.

Like props, cameras require no script counterpart in order to function, but neither do they have any function until a script uses them. Cameras are the make-up of TEAM_CameraSpline and many other camera commands. It can be very helpful to name these cameras according to their use or position, such as CAM_overwall_1 or CAM_Horizon_pan_2, so they will be easy to find in the list of all cameras.

Cameras can be freely placed, but moving and rotating can be done differently to all other instances. Cameras can still be moved, raised, lowered and rotated much like all other instances, but for cameras this is less intuitive. Instead, to move a camera, select it in the list of Instances, move your view to represent the desired new position of the camera and click the button "Set current camera instance to this" in the Instances dialog.

Props

Like areas, props require no additional script commands to be added and can therefore not be moved or removed through scripts. They are just there, solid and static in the map from beginning to end.

Props come in two distinct varieties: Gridded and non-gridded. The "grid" is the occupancy grid which determines whether a prop affects the ability for agents to pass over or through it, such as large buildings, clumps of forest and bridges. The resulting grid can be made visible with the "Show occupancy grid" option.

In practical terms this means gridded props can only be rotated in increments of 90 degrees – they can only be placed so that they point north, west south or east and never at any other angles than these. Gridded props also "snap" to the grid in 6 meter increments, which is the size of the grid tiles. The gridded props should not be placed so their occupancy grids overlaps. This will cause odd pathfinding behaviour.

Props with no grid have none of these limitations, but do not affect the movement of agents and so they must be very small so as not to look odd when agents pass right through them. Typically these are small things like signs, grass, rocks, junk and other "cosmetic" props.

Aside from the limitations on gridded props, they can be placed at any height, rotation or position, but also due to occupancy grid behaviour it is unwise to place props with white grids (always passable) too near altitude drops – valleys, cliffs, drops and pits – as agents passing over the white occupancy tiles will "float" at the same altitude as the prop itself. Be careful!

Sounds

A placed sound will create a SCRIPT_AddSound command in the currently selected script, much like agents. This script can be started at any time and does not rely on such things as players being connected, but it is still best to keep sounds in a separate script to keep track of them. Thanks to this script command, sounds can also be removed through scripting while the game is running.

Sounds appear like small blue boxes hovering above the ground, but are actually played in a much larger radius, normally listed as part of the name. The sound Crickets_200 generates a sound of crickets within a radius of 200 meters with the sound strongest in the central area and weaker at a distance. Crickets_400 is the same sound, but with greater range. The actual size of sounds is specified in the file sound/ambientsounds.juice. Try not to place too many large-range sounds close to one another so that they overlap. Too many sounds will overload the sound system so that some sounds are not heard.

There is no point in rotating sounds, but their position is important and placing a sound above or below the ground can have its uses. Remember that while playing, sounds are played relative to the camera, and placing a sound above the ground in such a way that the camera passes very close to it can have unexpected effects. Sounds can be raised, lowered and moved with the Instance Edit tool, and will snap back to the terrain like props, things and agents.

Things

The list of Things which can be places is identical to the list of Props, but the two have several differences. Things cannot have any kind of collision or occupancy grid. Everything – including bullets, agents and the player's camera – will pass through things. The benefit of Things is that they can be added and removed at will, and they can even be animated by the command THING_SetAnimationState.

Although Things require the command THING_AddThing for them to be displayed in the game, this command is not created automatically. First place the Thing instance in the world, and then create a THING_AddThing command in the scripts where you want it to be created, and link this command to the instance you have created. If you wish to remove, move or animate the thing, you must refer to the THING_AddThing by name when doing so. Only the THING_AddThing command refers directly to the Instance name.

Because things lack occupancy, they can be placed anywhere, at any altitude and in any orientation without snapping.

2D Map

The 2D map can be used for a quick overview of your map. Invoke it by pressing the "View2DMap" on your toolbar.

The default mode of the 2D map is the camera mode. In this mode you can move your camera just by clicking with the LMB. The RMB is used to pan around in the map, and the Wheel is used to zoom in and out. This behavour is the same in all three modes.

The zone mode enables you to move around your zones. See the zone wizard chapter below.

Finally, the playfield mode lets you change the playfield very easily. Just click and drag one of the "corner balls" of the playfield (visible only in playfield mode) to change the size of your playfield.

Ruler

The ruler is a very handy tool for measuring distances in the game world. Select the Ruler tool. Then click and hold down your left mouse button on the ground to measure the distance from that point to any other point on the map. When creating multiplayer maps it is very important that the distances between important objectives are approximately the same for all teams so that no team has an advantage over the other.

Wizard manager

The blue wizard can help you with tedious tasks, such as creating scripts and placing zones.

To invoke the wizard manager, click the wizard manager tool. This will bring up the wizard manager, and select the skirmish wizard as default. The actions from mouse events in the 3D view depends on which wizard is selected. See below for details.

Skirmish Wizard

The skirmish wizard creates scripts and settings for a multiplayer or skirmish map with the press of a single button. Enter a name, a description and a maximum number of teams for your map, and press the "Build template" button. The wizard will generate scripts according to the model used in the MapViper tool. Add agents (such as turrets) to the "Turrets" script and add sounds to the "Sounds" script.

Zone Wizard

Use this wizard to place zones on your map. The wizard takes care of everything. Things, areas, cameras, and basepositions are automatically added and tied together in a scriptcommand. For normal use, the only things you have to decide are the zone type (Landing zone or victory location), Initial owner and maybe the camera generation settings.

The default camera position setting will automatically generate a camera position 50meters away from, and 50meters up from your zone, facing south. If you select the "Use current campos" option, the cameraposition that you had when placing the zone will be used.

When the zonewizard is selected you will have a round blue cursor attached to your mousepointer. This cursor symbolizes your zone, and it has the same radius as the finished zone. A click with the LMB will place a zone using the current wizard settings. To move a zone, turn to your 2D map. (Press the View2DMap button in youtr toolbar). Select the "Zones" mapmode. You'll now see all your zones as icons in the 2D map. Click and drag to move them around. To delete a zone, set the 2D map in camera mode, and delete the areas, things, cameras, and commands that make up the zone manually.

Tips, Tricks & Troubles

There are many dangers to watch out for while creating maps in XEd. Turn to this section of the manual if your map crashes and you do not know why.

Zones must be flat

Problem: The center square of any victory location or landing zone must be perfectly passable. You can not place a blocking structure there, and the terrain cannot be so uneven that it turns up as a red impassable block. Landing Zones are especially vulnerable, as units are spawned from dropships here.

Reason: This problem has two reasons. The first is dropships. There needs to be room for the units dropped to actually stand. A unit cannot be deployed in a red impassable area or inside a building, and it can crash the game. The other problem is with the AI, who will calculate the value of zones according to the absolute centre of it. This point cannot be unreachable. If it is, the AI does not understand how it could possibly defend it. It will crash. We apologize for these crashes, but there are good reasons for them.

Solution: Make sure there are no obstructing objects near the centre of your zones, and make the ground flat enough for any unit to stand on it.

Don't delete zones with 2D map in "Zone" mode

Problem: XEd will crash if a zone command or an instance that is referenced by a such command is deleted while the 2D map is in "Zone" mode.

Reason: Sloppy coding by me (Peen). The 2D map is trying to use deleted pointers.

Solution: Make sure the 2D map is not in "Zone" mode when deleteing zone related instances or scriptcommands.

The Basics of Scripting

Welcome to the Ground Control 2 Scripting Manual, the beginner's guide to Massive Entertainment's level editor: *"XEd"* and script language: *"Juice"*. This document aims to give a helpful, if not complete, insight into how the level design team at Massive Entertainment used their tools to create the missions and maps in Ground Control 2: Operation Exodus.

Starting

The first step is always the most difficult. It is good to start with an idea of what you want to create, but there are many things to keep in mind: We want to create a scripting structure that is practical and logical, but flexible enough to handle changes.

START

Whenever you create a new map in XEd, there is only one script, and that is START. This script will always be started as soon as the map is loaded into the game, before all players have connected to the game, so this is our naturally given first step. Therefore, this script must contain commands that need to be running before the players have entered the game, such as weathers, things and sounds. The START script is also where all other scripts must be started, normally with TRIGGER_AllPlayersConnected.

SOUNDS & THINGS

Because sounds and things, like agents, need a scriptcommand to be started, it is a good idea to place all sound and some thing commands in separate scripts. Start these directly from the START script, since they do not rely on any player or agent references.

PLAYERS_CONNECTED

It is useful to have a script act as a platform where you know that all players are in the game. This goes both for single-player missions, cooperative missions and multiplayer missions. The only time when it is not important to know that ALL players have connected, including humans, scripts and AI players, would be a mission with only one player, and how much fun would that be?

From the CONNECTED script, it is good to start UNITS and TRIGGERS scripts, because agents cannot be created for a particular player until that player has connected. Similarily, triggers cannot detect a player or agents belonging to that player, until then. PLAYERS_CONNECTED is also the best place to start the music and cutscenes that should play when the mission begins. If all scripts are started with SCRIPT_ActivateScript at the same time, the game will pause for a significant moment before moving on. If the scripts are spaced out with TRIGGER_Timer, the pause will be less significant, but the game's performance will suffer over a longer period of time.

ZONES

There's really nothing stopping you from creating zones whenever you feel like it, but during the development of Ground Control II, the standard procedure was to keep zones in separate scripts. Often, at least two scripts were used, one for Landing Zones and one for Victory Locations that are needed at the beginning of the map. Some maps have several different scripts for zones that are activated at different points during the course of the mission. On some maps, landing zones and victory locations are added once the player has reached certain goals, or is given new objectives. Victory Locations are just a way to signify a ground objective, after all.

UNITS (or AGENTS)

It can be very convenient to place all the agents that should be on the map when it starts, in a separate script. Making several UNITS scripts can also be helpful, one for each player or team. This script should then be started before any triggers, so that they can be used to monitor the agents created in this script. To save performance, it can be meaningful to start only small groups of agents at a time, especially if the agents are spawned during heavy action. During Ground Control II, agents were only very rarely spawned over the course of the

game for two reasons: There is a performance hit whenever new agents are spawned, and nobody likes enemies who come out of nowhere, unless they are demons from hell.

TRIGGERS

Another practical thing is to sort all the triggers needed at the beginning of a map into a separate script. Preferably this should be started after all players have connected and after all zones and agents have been created. Some typical triggers are timers that remind the player of the map objectives, triggers that check if vital units are destroyed, or when a player controls a certain number of Zones on the map. Triggers for such things would point to individual EVENT scripts.

EVENTS

These are only really useful in a single-player map, or an objective-driven multiplayer map. Events could be things like enemies reacting when the player takes over a zone, or a vehicle complaining when it comes under fire. Events would also control in-game cinematic cutscenes.

```
Script START
{
  WEATHER_SetWeatherEffect set_default_weather
  {
    myNewWeatherType Default_Cloud_1
  }
  SCRIPT_ActivateScript start_ambient_sounds
  {
    myScript myInstances.myScripts.ambient_sounds
  }
  SCRIPT_ActivateScript start_zones
  {
    myScript myInstances.myScripts.zones
  }
  TRIGGER_AllPlayersReady All_Players_Ready
  {
    myTriggeredScript myInstances.myScripts.Players_Ready
  }
}
Script Players_Connected
{
  SCRIPT_ActivateScript Start_Agents
  {
    myScript myInstances.myScripts.AGENTS
  }
  SCRIPT_ActivateScript Start_Triggers
  {
    myScript myInstances.myScripts.TRIGGERS
  }
  SCRIPT_ActivateScript Start_Briefing_Event
  {
    myScript myInstances.myScripts.EV_01_CINEMATIC_1
  }
}
```

In this example, the START script is used to launch only commands which do not rely on agents or players, such as setting the weather, creating sounds and setting up Zones. Then, when all players have connected to the game, the Agents are created, then the Triggers and finally the first cinematic sequence starts, to tell the story to the players.

Please note that when starting a many scripts at one time, using SCRIPT_ActivateScript commands, and especially when creating many PLAYER_AddAgent commands, there will be

a certain “stutter” as your computer allocates memory and creates the agents. This is normal, but it is good to plan the map start in such a way that this stutter does not affect gameplay.

Multiplayer

Creating a multiplayer mission is much easier, script-wise, than creating a single-player map. The map will, most often, not have any agents on the map except the ones a player orders with a dropship, and the map most likely has very few events. The important parts are:

START
ZONES

Victory in a multiplayer map is usually decided by setting the defeat conditions in MissionStats. Deathmatch missions would have AllowNoUnitsDefeat or AllowNoPresenceDefeat enabled, while VL missions would have AllowNoVLDefeat. What these really mean is that “player is defeated when he has no units”, “player is defeated when he has no presence (zones or agents)” and “player is defeated when he has no landing zones (and an enemy has all of them)”. It’s confusing, but there is a very good reason for it to be this way.

The START script waits for all players to connect, and then starts the ZONES script with Landing Zones for everyone, and Victory Locations if you want them. That’s really all that’s absolutely required to create a map. However, if you want to, you can develop the multiplayer map by adding the following:

PLAYERS_CONNECTED
MAP_IS_SYNC_START
MAP_IS_DROP_IN
SOUNDS
TURRETS
WEATHER

After START, run PLAYERS_CONNECTED, and then from there start ZONES, SOUNDS, TURRETS and WEATHER. This is only the most basic way to build, and there are many ways to flesh out the details. Instead of using a single WEATHER script, you can make a structure of RANDOM_ActivateScripts separated by TRIGGER_Timers that randomize between different weathers. One such structure – Weather_Rain_Randomizer – is available to “Import” into your scripts, with a button in the Scripts panel. However, because Juice cannot handle dynamic links, you will have to connect the triggers to their scripts by hand.

The ZONES script can also be made more complex. To have random starting positions, **do not start the ZONES script.** Instead, use the command SCRIPT_RandomAddZone, add all landing zones to the myZones list and let this special command divide the zones among the teams currently playing. You can also make two separate sets of Landing Zones, for example ZONES_Dropin and ZONES_Synch. Then use SCRIPT_EvaluationBranch to detect if myFlag MISSION_OPTION_ALLOW_DROPIN is set to 1. Set myTrueScript to ZONES_Dropin and myFalseScript to ZONES_Synch. This way, you could make the landing zones used in dropin non-capturable.

Turrets are agents. Leave them belonging to PLAYER_NOPLAYER, which is neutral. When a player’s infantry enters the neutral turret, it comes under control of that player automatically. Turrets spawn with full health, but you could use AGENT_Hurt or AGENT_Die so players are forced to repair them before they can be used.

Single player

Maps for single player mode aren’t necessarily different from multiplayer maps, but the missionstats setting myMissionType needs to be set to SINGLEPLAYER to sort the custom map into the correct category.

```
START
SOUNDS
PLAYERS_CONNECTED
ZONES
UNITS
BRIEFING
EVENT_01
EVENT_02
MISSION_COMPLETE
MISSION_FAIL
```

Just like in multiplayer, it is important to wait until all players have connected before starting the real action. Singleplayer maps usually have some kind of briefing or in-game cinematic at the beginning of the map, which should be started after the players have connected, but before zones and agents.

Once the briefing is over, start the zones and the units scripts so that the players can start playing the mission. Remember to place objectives on the map using the OBJ_HumanObjective command, and markers on the minimap with TEAM_AddMinimapMarker.

You might also want to have scripts for mission failure and success, using the command SCRIPT_SucceedMission, which only needs one parameter for which team should win – all other teams lose when this happens. You can then run these scripts when the super-important allied unit is destroyed, or when the top-secret enemy weapon has been retrieved. Take note, however, that SCRIPT_SucceedMission will have no effect if the MissionStats setting NoScriptedVictory is enabled. The idea is that this setting can be used to make single player and cooperative missions into mindless deathmatch games, or timed missions.

Cooperative games

A cooperative mission is built like a single-player mission, but with the thought kept in mind that the friendly team can have more than one human player. Therefore it is important to use TRIGGER_AllPlayersReady

Cutscenes and cameras

```
Script CINE3_ENTER
{
  TEAM_SetCameraWidescreen TEAM_SetCameraWidescreen__0
  {
    myTeam TEAM_1
    myWidescreenFlag TRUE
  }
  SCRIPT_ActivateScript SCRIPT_ActivateScript__CINE3_WHITEOUT_B
  {
    myScript myInstances.myScripts.CINE3_WHITEOUT_B
  }
  TEAM_SaveCameraPosition TEAM_SaveCameraPosition__CINE3
  {
    myTeam TEAM_1
  }
  TRIGGER_PlayerAction CINE3_CutSceneSkipper
  {
    myPlayer PLAYER_PLAYER1
    myAction SKIP_CUTSCENE
    myTriggeredScript myInstances.myScripts.NULL
  }
  TRIGGER_TriggerIsTriggered TRIGGER_TriggerIsTriggered_Skip_CINE3
  {
    myTriggeredScript myInstances.myScripts.CINE3_EXIT
    myNeededNumberOfTriggeredTriggers 1
    myTriggers
    {
      TRIGGER_PlayerAction* CINE3_CutSceneSkipper
myInstances.myScripts.CINE3_ENTER.CINE3_CutSceneSkipper
      TRIGGER_MessageBoxClosed* MessageBoxClosed_Last
myInstances.myScripts.EV_07.MessageBoxClosed_Last
    }
  }
}
```

The code above is a typical script for starting a cutscene. The camera switches to widescreen, a script which causes the “white flash” effect is started, the current camera position is saved, and as you can see the TRIGGER_PlayerAction named CINE3_CutSceneSkipper waits for PLAYER_PLAYER1 to perform the action “SKIP_CUTSCENE”. The trigger itself only points to a script named “null”, which contains nothing, but it is also being monitored by a TRIGGER_TriggerIsTriggered which will activate

the script CINE3_EXIT if either one of the triggers CINE3_CutSceneSkipper or MessageBoxClosed_Last is activated.

In CINE3_EXIT, several commands are collected which change these settings back. Message boxes will stop playing and be removed, another white flash will appear, the widescreen mode will be disabled and the saved camera position will be restored. As you can see, cinematics are a lot of work.

Naming conventions

It does not matter what you kind of naming scheme you decide to use for your scripts, commands, triggers and instances, but it always helps to name them something other than their default names. If you are creating a zone, make sure all items involved contain the name of the zone they contribute to. Naming the camera, base position, thing and area LZ1_cam, LZ1_base, LZ1_thing and LZ1_area is one way of doing it, but TEAM_AddZone__1_LZ_1 is okay too.

Proper renaming will help you keep track of all the things you use and need while scripting, and although it can take a lot of time to rename several dozen agents to something that signifies what corner of the map they are in, it does save you a lot of grief once you start changing the layout of your map or give orders to these agents.

The same thing goes for scripting events and setting up triggers. If a timer counts down from 20 minutes and then launches event number five, calling it Timer_20m_EV05 will let you easily identify it when you want to use SCRIPT_KillTrigger or TRIGGER_TriggerIsTriggered in conjunction with your original timer. The more commands and triggers you have, the more important this is, and because you cannot always know how complex your map is going to be down the road it is best to acquire the renaming habit before you need it.

Tips, Tricks & Troubles

The "Null" script

Create a script called "Null" or "Nothing" or "dummy", and point all your unresolved script references to that script. Do not put anything important in it, but you could place a DEBUG_DebugText command with a simple message like "NULL!" or "The Null Script has been called instead of the unresolved reference it once was". Then, whenever a trigger or a condition is intended to not actually start another script, perhaps because you have an IsTrigger watching it or because that condition option should lead do no action but to remove the Conditional. This will clean up your list of unresolved references so that you can focus on the unresolved references that you really forgot to link to something important.

All Cameras will also have an unresolved reference for myFocusInstance, but do not try to create a dummy instance for it to look at, unless you do want all your cameras looking at the same exact plant. Just ignore the unresolved references from cameras.

Preparation is key

Place the areas, cameras and agents you are going to use in scripting before you start making script commands. That way, once you start scripting you can just connect all the command's options to existing instances. If you do not, you will have to close the command, place the things you need, and then double-click to edit the command after you have done that. This is a cause for many problems when the designer forgets to return and edit the script. Prepare what you want to do before doing it.

Names cannot have spaces

Problem: This was one of the first problems discovered in the Map Editing and Mod Making forums at forum.massive.se. If a map contains an agent or script name with a space in it, the game will crash, and even XEd will refuse to open the file.

Reason: In the Juice code, space is used as a "delimiter", a hard line between a command, a name, and the next name or command. If you add a space to these names, the first part will be interpreted as the actual name, and the rest of the name will be interpreted as code.

Because you rarely name your object after an actual command, most often this will just crash the game.

Solution: Use underscores "_" instead of spaces in names for your agents, scripts and commands. Be especially careful when editing your scripts with an external text editor.

Reference targets must exist

Problem: Triggers that never fire, actions that never take place. Why is the Special_Truck_Agent not detected by my trigger when the trigger is specifically looking for that agent? Why does my new agent never spawn to the player I specified?

Reason: When creating a new agent, the PLAYER who owns it must have connected to the game, or there will be no player to give the agent to, and the agent will not spawn. Similarily, a trigger cannot be created to wait for an agent that hasn't been spawned yet, or whose player was missing, preventing the agent from spawning.

Solution: Be very careful with the order in which you start scripts, or actions. Wait for players to connect, then create agents for those players, and then any actions or triggers that depend on these agents and players. Alternatively, use triggers that wait for PLAYER_ANY or for TEAMS instead of specific players, and use actions that refer to CONTEXT.AGENT or AgentTypes instead of specific agent instances.

Script Reference

Introduction to Juice

By definition, Juice is not a script language. It is a data definition language, much like XML. This chapter focuses on the use of map scripts to control the flow of events in a mission.

Map scripts are stored in the [MapName]Instances.juice file in your map's folder. Inside this file, we find myInstances.myScripts and myInstances.myInstances. myScripts is where commands are executed and triggers are started, whereas myInstances is where the 3D coordinates and types of all things, props cameras, areas and other instances are stored.

Commands and parameters

myScripts is further divided into several scripts. START is the first one, and is always started as soon as the server has loaded the map. The structure of a script is very simple:

```
Script START
{
}
```

Between the two curly brackets, commands are listed. Similarily, a command lists its parameters between curly brackets. Even a command without parameters has these brackets.

```
SCRIPT_ActivateScript SCRIPT_ActivateScript__Sound
{
  myScript myInstances.myScripts.Sound
}
```

The command SCRIPT_ActivateScript above only has one parameter, named myScript. This command's myScript contains the full name of the script that will be started immediately.

References

Once a command is placed in a script, it is instantiated from a type and receives a name and location: **TRIGGER_AllPlayersReady__0** placed inside the START script will be referred to as **myInstances.myScripts.START.TRIGGER_AllPlayersReady__0** if another command references it. The reference itself will be named to what command it points to, so a reference has three values:

```
PLAYER_AddAgent* F1_Marine__1
myInstances.myScripts.UNITS.F1_Marine__1
```

This parameter was found in an AGENT_Hurt command. It signifies a reference to an instance of the type PLAYER_AddAgent, the reference is named F1_Marine__1, the script command is inside the UNITS script and the PLAYER_AddAgent is named F1_Marine__1. The reference does not need to have the same name as its target, but it makes a reference easier to follow for the human reader. By double-clicking on a reference in XEd, the target can be changed but the name of the reference will stay the same the same. This can be very confusing for the level designer, and so it is much preferred to replace the reference with a new one with a new name.

Something that is very important to remember when creating references in a trigger or action, is that the target of the reference **must already exist**. For example, it is obvious that you cannot heal an agent before it has been created. It is not equally obvious that you cannot setup a trigger to wait for an agent, if that agent has not been created yet. Be very, very careful about this.

Context system

The context system can be confusing to learn at first, but it is key to creating many advanced gameplay mechanics. Using “Affectors” and context objects, the level designer does not have to refer to a specific agent, player or team to give Acquisition Points, Victory, new agents, health or damage. Triggers can set TriggerAffector and actions can set ActionAffector to CONTEXT instead of SPECIFIC. With this done the action or trigger will affect whatever agent, player or team is current stored into CONTEXT. See also the chapter on Triggers.

If an agent activates a trigger, that Agent’s identification is saved into CONTEXT. This agent can now be referred to as CONTEXT.AGENT, the player who owns the agent is saved to CONTEXT.PLAYER and the team the player belongs to is now the CONTEXT.TEAM. Depending on the type of Trigger, the Context can be more or less detailed. Some triggers save only to CONTEXT.AGENT, while others save all values and some save no data at all.

Full CONTEXT

The triggers listed below update all context types - CONTEXT.AGENT, CONTEXT.PLAYER and CONTEXT.TEAM - when they are activated. Scripts started by any of the triggers below can be used to use commands on teams, players, agents and types of agent.

TRIGGER_AgentTypeEnterArea
TRIGGER_AgentTypeHaveKilled
TRIGGER_AgentTypeLeaveArea
TRIGGER_AgentTypeUnderAttack
TRIGGER_AgentTypeWasKilled
TRIGGER_AgentsEnterArea
TRIGGER_AgentsHaveKilled
TRIGGER_AgentsLeaveArea
TRIGGER_AgentsUnderAttack
TRIGGER_AgentsWereKilled
TRIGGER_AnyOtherPlayerEnterArea
TRIGGER_AnyOtherTeamEnterArea
TRIGGER_PlayerEnterArea
TRIGGER_PlayerHaveKilled
TRIGGER_PlayerLeaveArea
TRIGGER_PlayerUnderAttack
TRIGGER_TeamEnterArea

PLAYER and TEAM

The following two triggers update the PLAYER and TEAM value of CONTEXT when they are triggered but do not change CONTEXT.AGENT.

TRIGGER_PlayerConnected
TRIGGER_ZoneTakenByPlayer

TEAM only

These two triggers update CONTEXT.TEAM to the team which triggered them, but changes no other CONTEXT values.

TRIGGER_NumberOfZonesTakenByTeam
TRIGGER_ZoneTakenByTeam

No CONTEXT

These triggers do not update the current CONTEXT at all.

TRIGGER_AllPlayersReady
TRIGGER_CampaignFlagSet
TRIGGER_CampaignFlagNotSet
TRIGGER_Timer
TRIGGER_TriggerIsTriggered

What this means is that it is uncertain what the contents of CONTEXT will be in any script started by TRIGGER_Timer and TRIGGER_TriggerIsTriggered. Most likely it will be the last used actual CONTEXT. For example, if a TRIGGER_TriggerIsTriggered depends on two TRIGGER_AgentEnterArea and starts a script named DEFEND, it would be safe to assume that CONTEXT in DEFEND will still be the CONTEXT saved by the AgentEnterArea trigger which fired last. However, since you can never be sure of this, it is best to not assume it.

Context Example

The following example contains the victory conditions for MP08 - Silent Hills. The script CONDITIONS_WIN is started as soon as possible, but not before the ZONES script, since the TRIGGER_ZonesTakenByTeam trigger needs to reference the zones created inside it.

```
Script CONDITIONS_WIN
{
  TRIGGER_ZonesTakenByTeam
TRIGGER_ZonesTakenByTeam_TEAM_ANY_0
  {
    myTriggeredScript myInstances.myScripts.WIN
    myTeam TEAM_ANY
    myAllZonesFlag TRUE
    myZones
    {
      TEAM_AddZone* VL1 myInstances.myScripts.ZONES.VL1
      TEAM_AddZone* VL2 myInstances.myScripts.ZONES.VL2
      TEAM_AddZone* VL3 myInstances.myScripts.ZONES.VL3
      TEAM_AddZone* VL4 myInstances.myScripts.ZONES.VL4
      TEAM_AddZone* VL5 myInstances.myScripts.ZONES.VL5
    }
  }
}
Script WIN
{
  SCRIPT_SucceedMission SCRIPT_SucceedMission_CONTEXT_TEAM_1
    {
      myTeam CONTEXT.TEAM
    }
  }
```

The CONDITIONS_WIN script contains only one trigger, which waits for any team to capture all of the listed zones. When that happens, the trigger fires and starts the script WIN. The team which caused TRIGGER_ZonesTakenByTeam_TEAM_ANY_0 to fire is now saved to CONTEXT.TEAM. Therefore, in the WIN script, a SCRIPT_SucceedMission action can now be used to give victory to the team which has taken all of the Zones on this map.

Script command listing

Common command attributes

These are some of the attribute types that are used by more than one command.

AgentRefList

This parameter is a list of references to agents. It is most commonly used to list a number of agents who you want to give an order, or inspect with a trigger, or blow up, or something like that. These lists can also contain groups, created with the GROUP_AgentGroup command. The agents in the list **must** exist when the reference is created. In other words, if you have a script for agents and one script for triggers, the

TRIGGERS script must be started after the AGENTS script if some triggers depend on agents inside the AGENTS script. Or, if the agents and their orders are in the same script, the AGENT_AddAgent commands should be near the start of the script, and orders or triggers referring to them must be near the end.

ActionAffector: SPECIFIC, CONTEXT

Whenever a parameter of the ActionAffector type exists, the parameter decides whether the action should affect the specific agent / player / team listed in the action – usually in a AgentRefList or PLAYER_AddAgent reference – or if it should treat the current CONTEXT agent / player / team. When context is selected, any list becomes completely unnecessary and the action will instead affect who last caused the current script to start – "CONTEXT".

TriggerAffector: ANY, SPECIFIC, CONTEXT

This parameter decides who can activate this trigger. Either any agent what so ever, or the specific agents listed in the trigger, or the agent currently in the CONTEXT stack – the agent which most recently activated a trigger starting a script.

Flag: TRUE, FALSE

Used by commands that can both activate and deactivate a function. The unimplemented function AGENT_SecondaryMode used this to distinguish between enabling the secondary mode (when myFlag = TRUE) and disabling it (when myFlag = FALSE).

TriggerRefList

A list of references to triggers, this is used by only three different functions: SCRIPT_KillTriggers, TRIGGER_TriggerIsTriggered and CONDITION_IsTriggered.

General commands

DEBUG_DebugText

Outputs a text string to the debug file. Useful for finding out if a specific script was started, or for adding credits to a map.

TEXT myText

A string containing the text to be printed to the debug file. This string is not localized and can therefore be changed at will. It is only saved in the *Instances.juice file.

WEATHER_SetWeatherEffect

Changes the weather. Some weathers require a transition time of 10 seconds before changing the weather again

GUI_HelpEnableWidget

Shows a flashing tutorial GUI (Graphical User Interface) component in the in game GUI. The "Widget" name must exist in the HelpScreen tree in the guis_ingame.juice file. These are usually just blinking boxes overlapping the user interface, for example Slot1 through Slot9 for the order palette. Widget names are case sensitive. This command was only used during the tutorials for the following widgets: *FormationArrow, MegamapArrowDwn, DropshipBuyArrow, DeployArrow, UpkeepArrow, UpgradeArrow, UpgradeCargoArrow, DropshipselectArrow, SupportweaponArrow, Slot3, Slot8, Slot9*. The "MegaMapArrow" is for the Objectives button, which previously called up a large scaled-up map all over the screen.

TEXT myWidgetToEnable

This is the name of the EXP_Plate component from guis\gui_ingame.juice which you want to enable (display).

GUI_HelpDisableAll

This is a special GUI command only used in tutorials. This command hides all currently flashing tutorial GUI components.

SCRIPT commands

Often, these are commands that do not relate to any particular area of the game, or which couldn't be sorted under another category, but also commands that deal with the immediate starting of scripts, ending the game or playing around with Triggers.

RANDOM_ActivateScript

This randomness command executes one script from a weighted list of scripts. The probability for each script is normalized and weighted against the probabilities for other scripts.

RandomOptions myRandomOptions

This is a little complex. A "RandomOptions" item (note the "s") contains several items called "RandomOption", which in turn consist of:

Script myTriggeredScript*

This is a reference to the script that will be triggered if this RandomOption is the lucky one.

NUMBER myWeight

This is a weighted value, the chance of this particular RandomOption coming true. "Weighted" means that a value of 2 is twice as "heavy" as a value of 1, but a value of 4 is twice again as important.

```
RANDOM_ActivateScript RANDOM_ActivateScript__Weathers
{
  myRandomOptions
  {
    RandomOption myScriptclassEntry
    {
      myTriggeredScript myInstances.myScripts.WEATHER_1
      myWeight 60
    }
    RandomOption myScriptclassEntry
    {
      myTriggeredScript myInstances.myScripts.WEATHER_2
      myWeight 30
    }
    RandomOption myScriptclassEntry
    {
      myTriggeredScript myInstances.myScripts.WEATHER_3
      myWeight 10
    }
  }
}
```

In this example, we can add up the weights, 60+30+10 to 100 so that the weights themselves can be considered percentages. Therefore, we can easily see that WEATHER_1 is most likely to start; WEATHER_2 is the second most likely, while WEATHER_3 only has a 10% chance of starting when RANDOM_ActivateScript__Weathers is executed. There is no vital need for Weights to add up to 100, but it makes for easier calculations.

SCRIPT_ActivateScript

This action runs a script instantly, inside the current script. When a script is started with the ActivateScript command, the contents of the myScript target will be executed inside the same script as the ActivateScript command.

Script myScript*

This is the script which will be executed immediately inside the current script.

```
Script DEBUG_ONE_THREE
{
  DEBUG_DebugText DEBUG_DebugText__One
  {
    myText One
  }

  SCRIPT_ActivateScript SCRIPT_ActivateScript__Sound
  {
    myScript myInstances.myScripts.DEBUG_TWO
  }

  DEBUG_DebugText DEBUG_DebugText__Three
  {
    myText Three
  }
}
Script DEBUG_TWO
{
  DEBUG_DebugText DEBUG_DebugText__Two
  {
    myText Two_Inserted
  }

}
```

Because the ActivateScript starts its myScript immediately, when the script above runs, the debug file will produce something like this:

```
DEBUG: One
DEBUG: Two_Inserted
DEBUG: Three
```

Using this command, one script could start two other scripts after one another, and still have triggers or commands after the ActivateScript command that affects objects inside those other scripts. For example, if we inside the script DEBUG_ONE_THREE use ActivateScript to start UNITS and then TRIGGERS, the scripting inside TRIGGERS could affect agents inside the UNITS script.

Because of this, the contents of CONTEXT used in DEBUG_ONE_THREE can be used also in DEBUG_TWO, since DEBUG_TWO takes place inside DEBUG_ONE_THREE.

SCRIPT_AddSound

This command is necessary for the game to play positional sound effects. It is automatically added to the active script each time a sound instance is placed in the Xed 3D view. The name given to this command is the name used to remove the sound later. If you remove this SCRIPT_AddSound command, the sound will no longer be heard, but the sound will still have a position and a soundID attached to it, so that you could create a new SCRIPT_AddSound command to re-start it.

XSoundInstance mySound*

This is the sound instance that has been placed in the world. You can find the sound and its name under Instances in the Instances window, or you could enable Text Labels for sounds and find it in the world. Double-click the instance to see what kind of sound it is, if the name does not explain it. If it helps, you could imagine that AddSound "starts playing" a sound "lying around" in the 3D world.

NUMBER myBaseFrequencyInHz

Normally this number should be 0, to use the default sound frequency, but it can be used to change the pitch of a sound, creating variations on any sound effect. The default should be 22050, but depends on the sound, so normally a value of 44100 would play the sound one octave higher than it was recorded.

```
SCRIPT_AddSound electro_relay_150__3_add_sound__4
{
  mySound myInstances.myInstances.electro_relay_150__3
  myBaseFrequencyInHz 0
}
```

The command above is the part of a positional sound that you can see in the myScripts section of the Instances file. It tells the game engine to start playing the game, which sound to play and what frequency to use, where 0 is the actual frequency of the sound file. The Further down in the Instances file, under the myInstances section, you will find the sound instance itself:

```
XSoundInstance electro_relay_150__3
{
  myPosition
  {
    myX 1583.360718
    myY 68.970139
    myZ 1202.008179
  }
  mySoundID electro_relay_150
}
```

Here you can see where the sound is positioned in the game world, and that it is of the type electro_relay_150, defined in the file sound\ambientsounds.juice. It is entirely possible to start a new SCRIPT_AddSound with a reference to an XSoundInstance which was "Removed", or even start two AddSounds on the same XSoundInstance.

SCRIPT_RemoveSound

Removes a 3D positional sound added by SCRIPT_AddSound. The SCRIPT_AddSound command must have been started before the SCRIPT_RemoveSound can remove the sound. You could see it as if SCRIPT_AddSound starts playback of a sound source (XSoundInstance) placed in the world, and SCRIPT_RemoveSound stops playback of the sound.

SCRIPT_AddSound mySound*

This is a reference to the SCRIPT_AddSound you want to stop playing.

SCRIPT_EndMission

Not used, but it should work. This command ends the mission completely and abruptly, dumping everyone who played it to the after-game screen. This can be confusing because it appears identical to being kicked from the game, or the server crashing. This command takes no parameters.

SCRIPT_EvaluationBranch

This command runs one script if the conditions are true and another script if they are false. The conditions that are available vary from comparing campaign flags to checking if triggers have been triggered. More information about conditions can be found under the TRIGGER_ConditionBranch entry and in the “Conditions” section.

Script myTrueScript & Script* myFalseScript*

These are references to the two scripts which could be started if the Conditions are fulfilled or not fulfilled, respectively.

Conditions myConditions

See TRIGGER_ConditionBranch.

```
SCRIPT_EvaluationBranch SCRIPT_EvaluationBranch__47
{
  myTrueScript myInstances.myScripts.Artillery_Lost
  myFalseScript myInstances.myScripts.Artillery_Survives
  myConditions
  {
    CONDITION_IntCampaignFlagLessThan myScriptclassEntry
    {
      myFlag Artillery_Alive
      myValue 5
    }
  }
}
```

In the above example, only a single condition is used. If the campaign flag “Artillery_Alive” is less than 5, the condition will be True, and the script Artillery_Lost will be started. If not, the script Artillery_Survives is started instead.

It is possible to leave either myTrueScript or myFalseScript empty in order for nothing to happen in either case, but it is better to create a script without content and name it something telling, like DEAD_END or DUMMY or NULL, and then point the “dead” entry to it.

The reason this command is named SCRIPT instead of TRIGGER, is because it does not wait for anything to happen: The command will immediately evaluate if the Conditions are true or false, and will start the TrueScript or FalseScript instantly depending on the results. This is

different from Triggers, which wait for their condition to become true and then activate their target script.

SCRIPT_KillTriggers

This command kills its listed triggers. It is not recommended to kill a trigger from the script that trigger has started, which is like sawing off the tree limb you are sitting on and can cause some serious problems. The triggers must have been created, but not activated, for a Kill to be successful. A killed trigger will never become activated. Killing a trigger that has already been activated has no effect, and the IsTriggered functions will still detect it as triggered. See also the chapter on Triggers.

TriggerRefList myTriggersToKill

This is a list of references to the triggers that should be killed.

SCRIPT_RandomAddZone

This is a special command for randomizing the ownership of zones. The TEAM_AddZone commands you wish to randomize must exist, but not have been started. It is recommended that these TEAM_AddZone commands are placed in a script that **must not** be started by any command. Only the myTeam ownership of each zone is randomized between the teams available on the map, all other values remain the same as before.

By “teams available on the map”, we mean the teams that are listed among the players in MissionStats

```
SCRIPT_RandomAddZone SCRIPT_RandomAddZone__0
{
  myZones
  {
    TEAM_AddZone* LZ_1 myInstances.myScripts.ZONES.LZ_1
    TEAM_AddZone* LZ_2 myInstances.myScripts.ZONES.LZ_2
  }
}
```

When using this command, you do not have to actually start the script that has the zones inside it. It is very special in this respect, because under normal circumstances the commands inside a script that is not started would never be executed. This command is very special in that way.

SCRIPT_SetGameOverTie

Not used. Unknown if it works. Unknown what happens if it does work. The command was intended to end the mission without announcing a winner. The command takes no parameters.

SCRIPT_SucceedMission

This command sets the winning Team. The affected team wins the match and all other teams automatically get the “mission failed” screen, because they lost.

Team myTeam

The team that wins. Keep this between TEAM_1 and TEAM_8. Who knows what will happen if you try TEAM_NOTEAM?

SCRIPT_SlowMotion

Not implemented. Ignore this.

TEAM commands

These commands are all relative to one specific team. The name of the parameter that specifies the team can vary, but myTeam is the most common.

Available teams are TEAM_1 through TEAM_8, TEAM_NOTEAM which is no team, TEAM_ANY which is any team at all, and of course CONTEXT.TEAM. TEAM_ANY but should only be used for triggers, not for commands.

Common attribute

Team myTeam

This is the team affected by the command, or the owner of the object created by the command.

TEAM_AddMinimapMarker

This command adds a marker in the in-game minimap for an entire team. It uses an area for location and scales the size of its marker depending on the size of that area. The marker will blink ("flash") the number of times specified.

LOCTEXT myToolTipText

This is a string of text which will be displayed when the mouse pointer hovers over the marker. Keep this short and sweet.

NUMBER myNumberOfFlashes

This is how many times the marker should flash before becoming solid. For the single player game in Ground Control II this was 1000, to always draw attention to important objectives. This is just a matter of not confusing the player, and having 10 different objectives all blinking at the same time would also be confusing.

FILE myMinimapIconFile

This is the marker that will be seen in the minimap. Use **ui/skins/map_icons.dds** for the default icon used in the single-player game. See also the TEAM_AddZone and its identically-named value for other markers.

XSphereInstance myPosition*

This is a reference to the area which will be used for position and size of the marker. On a max-sized map, a size of 40 is good, but the smaller the map, the smaller the zone should be, because the minimap is always the same size even when the map size changes.

Remember that this command does not place an object in the game world, only on the map. To place a visible marker in the game world, place a "Thing" of the type Objectives_Marker. See notes on the commands THING_AddThing and THING_RemoveThing.

TEAM_RemoveMinimapMarker

This command is used to remove a minimap marker which is no longer necessary, for example when an objective has been completed. There is no myTeam necessary for this command, because it handles TEAM_AddMinimapMarker commands, which are already team-specific.

TEAM_AddMinimapMarker myMarkerToRemove*

This is a reference to the TEAM_AddMinimapMarker to be removed from view. It must have been added before removing it.

TEAM_AddZone

Adds a zone, which could be either a Victory Location (VL) or a Landing Zone (LZ). Also take a look at the commands TEAM_RemoveZone, PLAYER_AiRemoveZone, OBJ_AddObjective_Zone and SCRIPT_RandomAddZone, as well as the triggers NumberOfZonesTakenByTeam, ZonesTakenByTeam and ZoneTakenByPlayer.

Notes on Zone sounds: All sounds reside in the gc2localized.sdf file, in the directory **sound/ ingame/ general_feedback/**. The sounds are sorted by Faction: F1 is the NSA and F3 is Viron. There are no feedback sounds for the Empire.

The problem here is that Zones pay no attention to what faction the player is. Instead, the designer must choose if a zone uses the NSA or Viron voices to declare that a zone has been attacked or changed owners. In the GCII single player campaign, this was never a problem because in the first half the player uses NSA units and in the second half the player uses Viron units. In multiplayer, NSA is the default, and so the F1_ sounds are used.

myZoneMarker

The zone uses a “thing” for the representation in the 3d world. This thing will show different colors depending on if the zone is neutral or belongs to a player. VL_thing, LZ_thing and LZ_UC_thing are normally used. Others could work.

myArea

The area used for the location and size of the zone. Normally this area is 47 distance units in radius or at least as big as the marker. Sometimes this area could be made big enough to represent a larger tactically important area.

myBasePosition

This area is the base position of the drop ship headed to this zone. The standard is to place these 1000 units away from the zone, and at least 500 units outside the visible playfield. The most important thing is to have the same distance between the zone and base position for all LZs on a map.

myCamera

This is a camera position overlooking the Zone. This is the camera you can snap to using the W key in the game.

myStartingTeam

This is the team who owns this zone from the start. This should be TEAM_NOTEAM for a neutral zone, but never TEAM_ANY. CONTEXT.TEAM can be given zones, too. As long as the myCapturableFlag is set to TRUE, another team can take over the zone from whoever owns it first.

myIsCapturableFlag

If this is set to FALSE the zone cannot automatically change team from its starting team. Use for LZNA (not available) and for zones intended to change ownership through script commands.

myIsLandableFlag

If this flag is set to TRUE the zone acts as a Landing Zone (LZ), VLs should be set to FALSE.

myCaptureAPAward

Each time the zone is captured by a team, a sum of AP can be given to that team.

myTimedApAward

This is the amount of AP that the zone will yield each time the myApAwardIntervalsInSeconds timer reaches zero.

myApAwardIntervalsInSeconds

A looping timer that gives myTimedApAward to the team that owns the zone.

myTimeToCaptureInSeconds

The time it takes for the zone to change from neutral to the TEAM that currently occupies the zone.

myTimeToLooseInSeconds

The time it takes for the zone to change from a TEAM to neutral, when the zone is occupied by enemy units. Pardon the spelling error, but it is too late to change now.

myCautionFactor

This special AI variable affects how the AI units ordered to this zone will behave. A low value will make the units suicidal and a high value (max 1) will make the units use their default behaviour.

myMinimapIconFile

This is the marker that will be seen in the minimap. Map icon files reside under the /**ui/skins** folder in the gc2data.sdf. Use **vl.dds** for Victory Locations, **lz.dds** for Landing Zones and **lzna.dds** for non-capturable Landing Zones.

myCaptureSound

This is the sound played when the zone belongs to your team. The defaults are F1_objectives_LZcaptured.wav for Landing Zones and F1_objectives_VLcaptured.wav for Victory Locations.

myLostSound

The sound played when your team loses the Zone and it becomes neutral. The defaults are F1_objective_LZabouttobelost.wav for Landing Zones and F1_objective_VLabouttobelost.wav for Victory Locations.

mySomeoneCapturedItSound

The sound played when another team captures the Zone. The defaults are F1_objectives_LZlost.wav for Landing Zones and F1_objectives_VLlost.wav for Victory Locations.

myAboutToLooseSound

This sound is never used. Its default value is **objective_neutral.wav**, “Location under fire”.

To create a zone, first place all the necessary parts. For a Landing Zone, first choose a suitable location. Place one thing of the type LZ_thing. Rename it to LZ_1_thing. Then place one Area with a radius of 47 on top of the LZ_thing. Rename he area to LZ_1_area. Now place a “base position” area. Rename this area to LZ_1_base. Finally, place one camera overlooking the Zone and rename it LZ_1_cam.

Then create a TEAM_AddZone command in the desired Script. Set myLandableFlag to TRUE, and point the unresolved references to the thing, the areas and the camera you just added. Rename the TEAM_AddZone to LZ_1 and you are done.

To make a Victory Location instead, set myLandableFlag to false and forget about myCamera and myBasePosition.

TEAM_RemoveZone

Removes a TEAM_AddZone created earlier. The Zone will be removed from the game. This can be useful in cases where a zone which was tactically important is no longer important, or if a Landing Zone becomes too dangerous for anyone to land on. Because this command refers to a TEAM_AddZone which is already team specific and might have changed teams since it was created, this command does not require a myTeam value.

TEAM_AddZone myZoneToRemove*

This is a reference to the TEAM_AddZone to be removed. The zone must have been created before it can be removed.

TEAM_ApRewardOrPenalty

Use this command to give Acquisition Points (AP) to a team, or take AP away from a team. The AP is then divided among members of the team based on their current maintenance percentage. Be aware that taking AP away from a team has not been thoroughly tested, and may fail or crash the game. What if the team already has zero AP?

NUMBER myApRewardOrPenalty

This is the amount of AP applied to the team. A positive number will give AP to the team; a negative number will take AP away.

TEAM_CameraPositionTrackAgent

Not used, but it works. The camera’s position will be locked to a spot and will look directly at the agent specified. This creates a “security camera” perspective on the game. Use RestoreCameraPosition or a short CameraSpline to restore camera control to the player.

XCameraInstance myCameraRelativePosition*

This is the position from which the camera will look towards the designated Agent.

DECIMAL myTransportTimeInSeconds

This is the time it will take the team's camera to be transported to the new position, in seconds.

PLAYER_AddAgent myAgent*

This is the Agent which the camera should focus on. The PLAYER_AddAgent command must have been started, and must not have been destroyed or removed, before the camera command.

ActionAffector myAffector

Because this command deals with both the TEAM controlling the camera, and the Agent the camera should focus on, this switch can be used to track a CONTEXT agent instead of the specific agent above, just as in AGENT commands.

TEAM_CameraShake

This command shakes the camera. Values above 1 cause extreme shaking. Some people would expect 1 to mean 1%, others expect 1% to be 0.01 – really an easy mistake.

NUMBER myIntensityPercentage

This is the intensity of the shaking. 1.0 means 100% and this is an integer value.

NUMBER myDurationInSeconds

This is the duration of the shaking. Because shaking is annoying, please do not do it for longer than a few seconds at a time.

TEAM_CameraSpline

This command can be used to create a "spline" that the camera traverses using a list of timed waypoints. A spline is a curved line made up of several points and each point is a camera position. Because of this, it is best to place the camera positions you intend to use for the camera movement **before** you create the TEAM_CameraSpline command to tie them all together.

NUMBER myStartingFovInDegrees

This is the field of vision used at the start of the camera spline. The default is 90.

NUMBER myEndingFovInDegrees

This is the field of vision used at the end of the camera spline. The default is 90. By changing the field of view during a spline, a zoom-like effect can be attained.

FocusInstanceRefList myFocusInstance

Change this if you want the camera to focus on something – a Thing, Agent or Prop – while in motion. Leave it as an unresolved reference if you do not.

WayPoints myWayPoints

Every list of WayPoints contains several items of "WayPoint", and each WayPoint is defined by the following values:

XCameraInstance myCameraInstance*

Each point on a camera spline is a camera position. This links to the camera position you wish to use for this point in the spline movement.

DECIMAL myTransportTimeInSeconds

This is the time it will take the camera to move from its current position to this WayPoint.

DECIMAL myBias

This value changes the amount of curvature on either side of the waypoint. Changing this value up or down between -1 and 1 will make the spline curve more sharply before, or after this waypoint.

DECIMAL myTension

This value defines the tightness of the curve at this point of the spline. If this value is higher than 0.0 the curve becomes sharper. At 1.0, the camera will move in straight lines between waypoints. If the value is lower than 0.0, the curve will be very rounded.

TEAM_ChangeSoundtrack

This command plays a soundtrack for all players in the team. Music and ambience mp3 files are available in gc2data.sdf under the **Music/** directory.

FILE mySoundtrack

This is the mp3 song to start playing. The default is amb_music_01.mp3 from the **Music/** directory. Play silence.mp3 to stop playing music or ambience.

TEAM_ChangeZoneOwner

Never used in GCII, but it should work. This command changes the zone owner to a new team. It could be used for objective-based changing of zone ownership. Instead of a myTeam value, this command has a value named myNewTeam, for clarity's sake.

TEAM_AddZone myZone*

This is a reference to the Zone which should change owners. The TEAM_AddZone command must have been created, and not removed, when this command is run.

TEAM_FadeToColor

The screen fades between the current camera view and a one-coloured screen, using the timer.

XColor myColor

This is the colour value used for fading to or from. It is in the RGB format for Red, Green and Blue. If all three values are zero, the colour is black and if all values are 255, the colour is white.

FadeType myFadeType

This defines if screen should fade INTO a colour or OUT of a colour.

FadeTo myFadeToScreenType

This determines if the screen should end with a WIDESCREEN image, or restore the image to NORMAL, non-widescreen, "pan & scan".

DECIMAL myFadetimeInSeconds

This is how long the fade should take, in seconds.

```
Script CINE1_ENDFADEBEGIN
{
  TEAM_FadeToColor TEAM_FadeToColor__25
  {
    myTeam TEAM_1
    myColor
    {
      myR 255
      myG 255
      myB 255
    }
    myFadeType IN
    myFadeToScreenType NORMAL
    myFadetimeInSeconds 1.500000
  }
  TRIGGER_Timer TRIGGER_Timer__ACTIVATE_FINALFADE
  {
    myTriggeredScript myInstances.myScripts.CINE1_EXIT
    myTimeInSeconds 1.4
  }
}
Script CINE1_EXIT
{
  TEAM_FadeToColor TEAM_FadeToColor__27
  {
    myTeam TEAM_1
    myColor
    {
      myR 255
      myG 255
      myB 255
    }
    myFadeType OUT
    myFadeToScreenType NORMAL
    myFadetimeInSeconds 3.000000
  }
}
```

The above example shows the end of a cinematic cutscene in Ground Control II. In it, we can see that the image first starts to fade from normal into white during one and a half seconds.

After 1.4 seconds a second fade begins and during three seconds the image fades out from white to a normal view. This is the “white flash” effect seen after all cinematic cutscenes in Ground Control II.

TEAM_FlashPositionInMinimap

This command places a blinking marker in the in-game minimap. Uses an area for location and scales the marker depending on the size of that area. Flashes the number of times specified. This command is similar to the command TEAM_AddMinimapMarker, but is easier to use. A flashing position cannot be removed in the way a minimap marker can, but it will vanish once it has flashed as many times as it was intended to.

NUMBER myNumberOfFlashes

This is how many times the marker should flash before being removed. Please note that this is a difference from TEAM_AddMinimapMarker, which becomes solid after flashing. Do not flash this too many times, because it cannot be removed.

FILE myMinimapIconFile

This is the marker that will be seen in the minimap. Map icon files reside under the **/ui/skins** folder in the gc2data.sdf. Besides the markers mentioned for TEAM_AddMinimapMarker, map_icons.dds and ping.dds can be useful markers. The former is a rather ugly bluish and thick circle, while the latter is a nicer-looking pulsating circle useful for highlighting positions or events on the map.

XSphereInstance myPosition*

This is a reference to the area you wish to use for the size and position of the marker.

TEAM_ForceTopDownCamera

This command switches between forcing the top-down RTS camera and allowing the Ground Control type camera. Do note that for some reason this command does not accept a myTeam attribute. Probably, it affects all human teams, forcing them to use the top-down RTS camera. Please do not use it.

Flag myForceFlag

Set this value “true” to force the standard RTS camera. Set it “false” to return control over this setting to the human players.

TEAM_PlaySound

Plays a sound that is heard for all players on that team regardless of where they are at the time. This is useful for cinematic explosions, for example. See also PLAYER_PlaySound.

FILE mySoundFile

This is the sound to be played. One commonly used sound is **sound/interface/newobjective1.mp3**, which is played whenever a new objective marker is placed on the minimap in the Ground Control II single player campaign.

TEAM_SaveCameraPosition

Use this command to saves the current camera view for all players in a team. The command TEAM_RestoreCameraPosition can be used to restore this saved position at a later time. This is very useful for cinematic cutscenes. This command takes no parameters besides myTeam.

TEAM_RestoreCameraPosition

This command restores the camera view previously saved by the “TEAM_SaveCameraPosition” command. This can be used after cinematic cutscenes to restore the view used before the cutscene began. This command takes no parameters besides myTeam.

TEAM_SetCameraOrientation

Uses the current camera position and only modifies the orientation, the direction of the camera. Transition time between the camera orientations can be specified.

TEAM_SetCameraPosition

Uses the current camera orientation and only modifies the position that the camera should move to. Transition time between the camera positions can be specified.

TEAM_SetCameraWidescreen

Switches between wide screen mode and normal combat view. When in wide screen mode the entire GUI will be hidden so that no unit markers or even the mouse arrow will be visible.

TEAM_SetCameraView

Sets the camera to a previously saved camera position. Both the cameras orientation and position will be used. Transition time between the current view and the new view can be specified.

User interface commands

Normally, message boxes become queued, which means that one will play after the other, even if several message box commands are started one after the other without any timers between them. To terminate such a queue, cut it short, there is the TEAM_PurgeMessageQueue command. The TRIGGER_MessageBoxClosed trigger can be used to see if a message box has ended or been closed.

The first three commands will all show a message box on screen. These three have several attributes in common:

LOCTEXT myText

This is the text which will appear in the message box. The text is localized for each country.

TEXT myCharacter

This is the “talking head” that will be visible on the left hand side of the message box. These characters are read from messagebox_characters/characters.juice in the xed.sdf file. If you do not want any talking face at all, NSA_Pause_Picture is a good choice.

FILE myAudioFile

This is the sound file to be played along with the message box, in mp3 format. For the single-player campaign, these messages were stored in **sound/ingame/mission** and its subdirectories based on mission numbers.

A messagebox character and an audio file can be selected. If several message boxes are triggered, they will be placed in a queue and will be played, and closed one after the other. The script command TEAM_PurgeMessageQue will empty this queue. The trigger TRIGGER_MessageBoxClosed will fire when a specific instance of the below message box types has been closed.

TEAM_ShowAutomaticMessageBox

The message box will be visible for as long as the audio file is playing. When the sound file ends, the message box closes automatically, and the next one in the queue starts playing.

TEAM_ShowButtonMessageBox

The message box will be visible until the player clicks on the message box button. Use this to use a button for skipping forward in scripting. Please note that this was never used in Ground Control II and so it might not work.

TEAM_ShowTimedMessageBox

The message box will be visible for a specified duration. After the time has run out, the box will vanish, its sound file will be silenced and the next message box in the queue will start playing. If the timer is longer than the sound file, the talking head will keep chewing air without sound until the time runs out. Do not let that happen.

DECIMAL myTimeToShowInSeconds

This is how long the message box will be visible and audible, in seconds.

TEAM_PurgeMessageQueue

This command stops the current message box, and removes all queued messages. The corresponding sound file will be silenced. Any triggers waiting for these message boxes to be closed will **not** fire. This is very important to understand. Do not use a TRIGGER_MessageBoxClosed as the only way for an event to start, if you are aware that the message could be erased by a PurgeMessageQueue command. This command only requires a myTeam value.

PLAYER commands

These commands act on players. Therefore, all player commands have a parameter for which player to affect.

Common attribute

Player myPlayer

This is the player, from PLAYER_PLAYER1 to PLAYER_PLAYER8, who will be affected by the command. PLAYER_ANY should not be used for commands, but CONTEXT_PLAYER can be. See the subject “Context system” in the “Script Command Reference” chapter.

The three PLAYER_AI commands have a parameter named myAIPlayer instead of myPlayer. It works in the same way but must refer to a player controlled by an AI. There is no code-based check for this, so the designer must keep track or check the MissionStats settings.

PLAYER_FreezeInput

This command only works on AI players. If the AI is “frozen” it will give no orders to its agents until it is unfrozen. Orders that have already been issued will continue to be in effect. In the start of some missions it is recommended to freeze the AI, give objectives, assign units to the different objectives and then unfreeze the AI, especially if you plan on having a cinematic introduction scene with scripted agents. If the AI is not frozen during this cinematic, it will “start playing” while the human players are still watching the introduction.

Flag myFlag

TRUE = Freeze, FALSE = Unfreeze.

PLAYER_AIChangeDecisionTree

This command makes the AI switch purchase trees. These trees are resident in its brain file, located in the CommanderAI directory. Very few of the existing AIs have more than one such purchase tree, so using this command indiscriminately will have no effect. The topmost tree is the default.

FILE myDecisionTreeFile

This is the name of the new buy tree. This name is stored in the myNodeName attribute in the JUICE file for the CommanderAI used by the AI player. The standard name is OvsDBranch.

```
Script Freeze_AI
{
  PLAYER_FreezeInput PLAYER_FreezeInput__0
  {
    myPlayer PLAYER_PLAYER4
    myFlag TRUE
  }
}
Script ACTIVATE_VIRONS
{
  PLAYER_FreezeInput PLAYER_FreezeInput__2
  {
    myPlayer PLAYER_PLAYER4
    myFlag FALSE
  }
  PLAYER_AIChangeDecisionTree PLAYER_AIChangeDecisionTree__3
  {
    myDecisionTreeFile active
    myPlayer PLAYER_PLAYER4
  }
}
```

In the above example, PLAYER_PLAYER4 was “frozen” at the start of the map in the script Freeze_AI. No command was used to set the purchase tree, because the topmost purchase tree in the c1m11_vir.juice AI file is empty. The second purchase tree in this file is named “active”, and as PLAYER_PLAYER4 is “unfrozen” using PLAYER_FreezeInput with the FALSE flag, the command PLAYER_AIChangeDecisionTree is used to switch to the “active” tree. If the level designer runs another AIChangeDecisionTree command with myDecisionTreeFile set to “passive” instead, the AI would stop buying units.

PLAYER_AiChangeEngageSuccessProbability

With an EngageSuccessProbability of 1, the AI will want 100% chance of success before it attempts an attack. The AI considers a force twice as powerful as the enemy to be a guarantee for success. With an EngageSuccessProbability of 0.5, the AI will want a calculated chance of success of 50% before attacking. Therefore, setting very large values here will prevent the AI from ever attacking an enemy. The parameter myCautionFactor can be set on certain objectives to change the Engage Success Probability differently for different objectives.

DECIMAL myNewEngageSuccessProbability

The new engage success probability value for myAIPlayer.

PLAYER_AiRemoveZone

This command removes an existing VL or LZ from an AI’s awareness. The AI will have no knowledge of this zone, but could still accidentaly capture the zone while moving over it. The zone must have been created before the AiRemoveZone command runs, so placing the command instantly after the TEAM_AddZone commands can be practical.

Possibly, the AI must also have connected to the game, so waiting for a TRIGGER_Player_Connected or TRIGGER_AllPlayersReady may be a good idea.

TEAM_AddZone myZoneToRemove*

This is the TEAM_AddZone that should be removed from myAIPlayer's awareness.

```
Script ZONES
{
  TEAM_AddZone VL1
  {
    //removed to save space//
  }
  TEAM_AddZone VL4
  {
    //removed to save space//
  }
  PLAYER_AiRemoveZone PLAYER_AiRemoveZone_VL1
  {
    myAIPlayer PLAYER_PLAYER4
    myZoneToRemove myInstances.myScripts.ZONES.VL1
  }
  PLAYER_AiRemoveZone PLAYER_AiRemoveZone_VL4
  {
    myAIPlayer PLAYER_PLAYER4
    myZoneToRemove myInstances.myScripts.ZONES.VL4
  }
  PLAYER_AiRemoveZone PLAYER_AiRemoveZone_Enemy_Ignore_VL1
  {
    myAIPlayer PLAYER_PLAYER6
    myZoneToRemove myInstances.myScripts.ZONES.VL1
  }
}
```

In the above example, the Victory Locations VL1 and VL4 are created, and then the AI PLAYER_PLAYER4 removes both zones from its awareness, while PLAYER_PLAYER6 only ignores VL1 and will still be aware of VL4.

The biggest benefit of this command is that the AI will completely ignore the zone, and will not fight to capture it. This way, a particular AI can be told to focus on one area of a map and not try to take over the entire map. Making an AI forget zones in this manner can also save performance on maps with many zones, or AIs.

The value myNumberOfActivePlatoons in the CommanderAI files specifies how many platoons the AI can divide its forces into and thus how many active missions the AI can take on. Normally there is one active mission per zone, plus any additional objectives given to the AI. Skirmish and multiplayer AIs have 12 active platoons, while single-player AIs have fewer, sometimes as low as four, or even one. By using AiRemoveZone, a 4-platoon AI can still be a formidable opponent on an 8-zone map, focusing on a few zones and "forgetting" the others.

PLAYER_MoveCameraToLZ

If the player owns an LZ the camera will travel from its current location to the camera location that is bound to the LZ currently in use. The time in seconds will be used for how long the transition will take.

DECIMAL myMoveTimeInSeconds

The number of seconds the camera will take to move from its current position to the position of the LZ's myCamera. Zero is recommended, because the movement could cut straight through objects in the way.

PLAYER_PlaySound

This command plays a sound to a player. The path used has its root in the sound folder. This sound will only be played once, and will be played directly to the player; it will not be placed anywhere in the world as an object. Any sound in the game can be used, if you find it.

FILE mySoundFile

This is the relative path to the WAV sound that should be played. In-game sounds reside in the "sound/ingame" directory.

```
Script CINE3_Explosion
{
  PLAYER_PlaySound PLAYER_PlaySound__BOOOOOM
  {
    myPlayer PLAYER_PLAYER1
    mySoundFile sound/ingame/noise/largeexplosion1.wav
  }

  THING_SetAnimationState THING_SetAnimationState__ImpFIRE
  {
    myThing myInstances.myScripts.things.Prison_0
    my4LetterAnimationState FIRE
  }
}
```

This Example is from the Single-player mission 23, or number 11 in the Viron Campaign. This cinematic event plays an explosion sound for PLAYER_PLAYER1 and changes the animation state of a Prison building to an animation where it explodes. As you can see, the animated Thing was added in a script named "things", and has the name Prison_0. Now that I think about it, this specific explosion should have used a TEAM_PlaySound rather than PLAYER_PlaySound. If it had, ALL cooperative players would have heard the noise.

Agent commands

Most of the AGENT commands only work on script players, with a few exceptions such as Die, Remove, Heal or Hurt.

Common attributes

AgentRefList myAgents

This is a list of (references to) agents you want to affect with the command. Agents in this case are PLAYER_AddAgent, PLAYER_AiAddAgent or GROUP_AgentGroup commands.

ActionAffector myAffector

This is used to decide if the agent command affects the specific agents listed in myAgents, or if it should affect the agent(s) currently stored in the CONTEXT variable. See the Context heading above for more information.

PLAYER_AddAgent

This command adds an agent instance to the game. The name of the created command instance is then used as a reference to that particular unit. Because this command is added automatically whenever an agent is placed in XEd, you should never have to create this yourself. You will need to double-click these commands once they have been created, to change the owner.

Player myPlayer

This is the player who owns the agent. Neutral agents such as gun emplacements and radar buildings belong to PLAYER_NOPLAYER.

XAgentInstance myAgent*

This is the actual instance in the world that will be used for this agent. You can find these in the Instances tree in the Instances window. You could change this reference, but it would only cause confusion. Better to create a new PLAYER_AddAgent for every Instance instead.

PLAYER_AIAddAgent

Similar to PLAYER_AddAgent, this also forces the agent to an objective. Therefore, myPlayer must be an AI player. The agent will be stuck to the objective until it removed. The best way of doing this is to place the agent normally, and then replacing the automatic PLAYER_AddAgent with a PLAYER_AiAddAgent to the same myPlayer and AgentInstance.

ObjectiveRefList myObjective

This is a reference to the OBJ_AddObjective command that the AI Agent will be ordered to follow. The OBJ command must already exist for this to work.

GROUP_AgentGroup

With this you are able to define a list of agents that you later can use as a reference. You only have to make the reference to the agent group. The group has no impact in the game, but can be very useful in scripting. You can send orders, set triggers or use script commands affecting the group as a whole instead of adding the individual units every time. AgentRefLists can link to both AGENTS and GROUPS.

This is very useful during scripting, before you have decided exactly how many agents will be on the map. Instead of listing all the involved agents in every agent command and agent trigger, create a Group of the agents you wish to use, and refer to that group whenever you give orders or create triggers. If you find you have too many agents, or too few agents, just change the agents listed in the group instead of changing every single trigger and command you created.

AgentGroup myAgentGroup

This is a list which can contain references to PLAYER_AddAgent and PLAYER_AiAddAgent script commands.

PLAYER_ChangeAgentsOwner

This command changes the owner (Player) of a list of agents. Turrets should revert back to NOPLAYER.

Player myNewPlayer

This is the new owner of myAgents. It can contain PLAYER_AddAgent, PLAYER_AiAddAgent and GROUP_AgentGroup. Adding a GROUP_AgentGroup is just like adding all the agents it contains.

AGENT_Die

Instantly kills all the listed agents displaying a damage effect of the specified type. Agents must have been created when this command is run, or it will have no effect on the agents once they spawn.

DamageType myDamageType

What type of damage should be used to kill the agents? Most are logically named, except for Alien, which is a little weird. This only changes the type of effect used when the agent is killed – the agent will die in all cases.

AGENT_EnterBuilding

Tells the a list of agents to enter a specified building. The Agents listed in myAgents need to be chosen carefully, as only agents with the **Resident** parasite can enter buildings and other props. XEd lists all agents for this command, making no distinction between those with or without the parasite. Parasites are added to agents in the file **unittypes.juice** in the /units directory available from the xed.sdf.

XPropInstance myBuilding*

This is a reference to the building prop that the agents should enter. Only enterable buildings are listed

Quadrant myEntryQuadrant

East, west, south or north. This decides which side of the building the agents should occupy. This will work regardless of how the building has been rotated – north is always north.

Props are made able of harbouring agents by having a mySlotFile value defined as a valid file with the .slot extension, in its definition. Props are defined partially in the file GlobalPropTypes which resides under the XEd directory, in the xed.sdf file, but also in the file which corresponds to the terrain type used. For example, props in battleground maps are defined in the file battleground.juice, also in the XEd directory. This is the definition for the Prop BG_Block_House_02:

```
XPropType BG_Block_House_02
{
  myVisualModel
landscapes/battleground/buildings/residential/bg_block_house_02/bg_30m_ruin_
02.mrb
  myShadowModel
landscapes/battleground/buildings/residential/bg_block_house_02/bg_30m_ruin_
02_shadow.sdw
  myLightType SHADOWMAP_AND_LIGHT
  myUnitsAreVisibleThroughMeFlag 0
  myGridFile
landscapes/battleground/buildings/residential/bg_block_house_02/bg_30m_ruin_
02_grid.grid
  mySlotFile
landscapes/battleground/buildings/residential/bg_block_house_02/bg_30m_ruin_
02_slotpos.slot
  myGraphicalImportance ALWAYS_VISIBLE
}
```

AGENT_ExitBuilding

Orders the agents listed in myAgents to exit the building they are currently occupying. If they are not inside any building, nothing will happen. This action takes no other parameters.

AGENT_EnterContainer

This action orders a list of agents to enter another agent that is “enterable”. In order for an agent to be capable of entering such an agent, it too must be prepared by having the parasite named **Containable**. Any agent with this parasite can be containedby any other agent which has the **Container** parasite. However, all agents will be listed as available for both AgentRefLists in this command.

Giving an enter order into a non-enterable agent will have no effect, nor will an order given to agents that cannot enter other agents. Enterable agents for infantry are: Turrets, large radar agents, APCs, engineer vehicles, Covinus, Architectus, Great Corruptor and Contaminator Copter. The only vehicle another vehicle can enter is the NSA Transport Copter.

> *PLAYER_AddAgent* myContainer*
>
> This is the “container” the agents are ordered to enter. When selecting, all agents will be listed including agents incapable of entering another agent.

Agents with Containable parasites with myContainerType 0 can only enter Agents with an Container parasite set to the same value. There is no limit to how many different types of Containable and Containers there can be. In Ground Control II, there was only myContainerType 0 for infantry fitting into APCs, and myContainerType 1 for tanks and other vehicles fitting into the NSA Transport Copter.

```
Container APC 8
{
  myCapacity 8
  myClaimableFlag 0
  myContainerType 0
  myHidePassengersFlag 1
}


Containable CanBeTransported
{
  myContainableType 1
}
```

This is part of the myParasites portion of the F1_Assault_APC definition. The agent has a Container parasite intended for type 0 agents, meaning infantry in this case. The myCapacity flag determines how many other units it can carry at the same time, and the myHidePassengersFlag determines if the agents vanish from view (as in the case of APCs) or if they remain visible (as with the Transport Copter, and the Flintstones’ car). The last value, myClaimableFlag, is primarily used for turrets. With this flag enabled the agent will remain neutral until another agent enters it, when it will change teams to that of the agent who entered it.

As you can see, the Assault APC also has a Containable parasite named “CanBeTransported”, for clarity’s sake. It is myContainableType 1, meaning that the NSA Transport Copter could lift it.

AGENT_ExitContainer

Tells the specified agents to exit the container (APCs, Turrets) they are currently occupying. If they are not inside a container, nothing should happen. Still, it is good to be aware that this might have unexpected results, as the action has not actually been tested. This action requires no other parameters.

AGENT_Heal

This action restores myAmount of health to the specified or context agent(s), up to their maximum (original) health.

NUMBER myAmount

This is the amount of health regained by myAgents.

AGENT_Hurt

This is the inverse of AGENT_Heal. The action will remove myDamage amount of health from all agents specified.

NUMBER myDamage

This is how much damage will be detracted from myAgents current health. Detracting more health than they currently have will kill them. All agents lose this same amount of health.

DamageType myDamageType

Specifies what type of damage is used to harm the agents. This decides the visual effect displayed on the agent when it takes damage. Available damage types are: ALIEN, ANTI_PERSONNEL, ANTI_TANK, CHEMICAL, ENERGY, FRAGMENTATION, MACHINE_GUN and MISSILES.

AGENT_MoveTo

This command issues a move command to all scripted agent(s) listed. The order requires a target Area to exist. The move command will be targeted at the centre of the area specified. It is not recommended to give such an order to more than a few units at once as they will circle around the center position of the area, all trying to fit in the same spot. Instead, give one move order to each agent involved and the result will look much better.

XSphereInstance myPosition*

To give the order, there must be a target point for the agents to move to. Areas are used for this. You can give a move order to any area, as long as it is reachable and the center is not blocked. You could order agents to move to a Landing Zone's area, for example.

AGENT_Remove

Removes all agents listed from the game without triggering a kill. TRIGGER_AgentsWereKilled will not start because of this command and no score changes will be generated for the removed agent(s). This is useful for removing agents when they are no longer needed as objectives, for example. To save performance, it is still recommended that the level designer uses SCRIPT_KillTriggers to deactivate any triggers connected to the removed agents.

AGENT_SetFireBehaviour

This action sets the fire behaviour for a list of agents. This command has an unreliable effect on human-controlled agents – their agents' behaviour will change, but the change will not be visible in the user's GUI.

FireBehavior myBehaviour

This value specifies the agents' new behaviour to FREE_FIRE, HOLD_FIRE or RETURN_FIRE mode. In FREE_FIRE mode, the agent will fire on any enemy in range. In RETURN_FIRE mode, the agent will fire at any enemy who has damaged the agent. In HOLD_FIRE mode, the agent will only fire upon enemy units it has been directly ordered to attack.

AGENT_SetMoveBehaviour

This action sets the move behaviour for the listed agents. The command was never used in the single player campaign, but it should work. It might suffer from the same UI problems as AGENT_SetFireBehavior when used on HUMAN-owned agents.

MoveBehaviour myBehaviour

Possible values are FOLLOW_TARGET, where the agents will pursue any enemy encountered, and HOLD_POSITION, where the agents will follow their existing move orders (or lack thereof) and never move to own to attack the enemy by their own volition.

AGENT_SecondaryMode, AGENT_SetEnabled, AGENT_SetIndestructable, AGENT_AttackAgents, and AGENT_AttackArea

These commands were removed or not implemented because they would have taken a long time to implement correctly.

To make agents indestructible, spawn an INVULNERABILITY agent from the agent list, give it to the relevant player / team and remove it when you want to return the player to mortality. Please note that although the effective radius of an INVINCIBLE barrel is quite large, it is not enough to cover the entire map from only one corner of it. As long as the barrel is placed near the centre of a map, or at least near the action, it should cover all relevant agents.

To make agents attack the enemy, move them to an area with hostile agents nearby and set the agents to FireBehavior FREE_FIRE.

Campaign flags

Despite the name, campaign flags only persisted throughout a mission, including savegames. They do not carry over between missions due to a conflict with how savegames were implemented. Also see the triggers CampaignFlagSet and CampaignFlagNotSet, as well as the Conditions dealing with campaign flags.

Common attribute

TEXT myFlag

Both Int (integer number) and Str (text string) campaign flags have names in plain text. The first time a Set-command is used for a specific campaign flag name, the campaign flag is created. This name is then used to change, check and remove the campaign flag's contents. Any flag that has been created, but not removed is "set".

There are also a few hard coded campaign flags which can be used to check game settings, victory conditions and winning team. In most cases, 1 means yes and 0 means no.

MISSION_OPTION_ALLOW_DROPIN	Is drop-in enabled?
MISSION_OPTION_IGNORE_SCRIPTED_DEFEAT	Can script commands cause victory?
MISSION_OPTION_WIN_ON_ALL_VL	Do players win by taking all VLs?
MISSION_OPTION_LOSE_ON_NO_LZ	Do players lose without an LZ?
MISSION_OPTION_LOSE_ON_NO_UNITS	Do players lose without units?
MISSION_OPTION_LOSE_ON_NO_PRESENCE	Do players lose without presence?
TEAM_n_HAVE_UNITS	Does TEAM_n still have units?
TEAM_n_HAVE_PRESENCE	Does TEAM_n still have presence?
MISSION_OPTION_WIN_TEAM	Which team number has won? If no team has won yet, this flag is not set. If the game is a tie, it is 0 (zero).

The meaning of "presence" is when a team still has any ability to fight back and return to the map to make a difference. A team does not lose "presence" until it has no units, owns no zones on the map, or cannot afford to land any units. Here is one example used in many multiplayer maps:

```
Script START
{
  SCRIPT_EvaluationBranch SCRIPT_EvaluationBranch__0
  {
    myTrueScript myInstances.myScripts.MAP_IS_SYNC
    myFalseScript myInstances.myScripts.MAP_IS_DROPIN
    myConditions
    {
      CONDITION_IntCampaignFlagEquals myScriptclassEntry
      {
        myFlag MISSION_OPTION_ALLOW_DROPIN
        myValue 0
      }
    }
  }
}
Script MAP_IS_SYNC
{
  SCRIPT_RandomAddZone SCRIPT_RandomAddZone__2
  {
    myZones
    {
      TEAM_AddZone* LZ1_cap myInstances.myScripts.zones_random.LZ1_cap
      TEAM_AddZone* LZ2_cap myInstances.myScripts.zones_random.LZ2_cap
    }
  }
}
Script MAP_IS_DROPIN
{
  SCRIPT_RandomAddZone SCRIPT_RandomAddZone__3
  {
    myZones
    {
      TEAM_AddZone* LZ1 myInstances.myScripts.zones_random.LZ1
      TEAM_AddZone* LZ2 myInstances.myScripts.zones_random.LZ2
    }
  }
}
```

In the START script, a SCRIPT_EvaluationBranch is used to determine whether the campaign flag MISSION_OPTION_ALLOW_DROPIN is set to 0. If it is, that means this is a "synch start" game. If the flag is NOT 0, it must be 1, and this is therefore a drop-in mission. In drop-in missions it is important for Landing Zones to always be available so that a player who connects later on in the game will always have a landing zone.

The two scripts MAP_IS_SYNC and MAP_IS_DROPIN will then activate two different sets of Landing Zones. In the sync start case, the zones LZ1_cap and LZ2_cap, which are both capturable, will be used. In the drop-in case, LZ1 and LZ2 will be used, which are not capturable. This behaviour is controlled by the value myIsCapturableFlag on the TEAM_AddZone command.

See also individual headings for SCRIPT_EvaluationBranch, SCRIPT_RandomAddZone, TEAM_AddZone and the chapter on Conditions.

DATA_SetIntCampaignFlag

This action sets an integer campaign flag to a numeric value. If the flag does not exist, it will be created and set to the value specified. Please note that setting an integer campaign flag to 0 (zero) will mean that it is set, but contains 0 (zero). This can be confusing.

NUMBER myValue

This is the value that the new Integer campaign flag will be set to.

DATA_SetStrCampaignFlag

This action sets a string campaign flag to the specified text content. If the campaign flag name does not exist, it will be created, and then set to myValue.

TEXT myValue

This is the text string that the campaign flag will be set to. It can contain spaces, but quote marks may be a bad idea.

DATA_ChangeIntCampaignFlag

This action modifies the specified integer campaign flag with a positive or negative number. For example, using 1 as the amount will increase the value by one. Using -2 as the amount will decrease the value by two.

NUMBER myAmount

This is the integer value that will be used to modify the integer campaign flag.

DATA_RemoveCampaignFlag

This action removes a campaign flag regardless of integer or string type. The campaign flag will no longer be set. This command only takes the myFlag parameter.

Objective commands

Objectives are the orders given to human players to let them know how the map should be played, but there are also several objectives which control specific AI behaviour.

When given to a human player the short description will appear on the left side of the in-game screen. The long description is used for more extensive information that will only appear when the player views the objective screen.

The AI believes it will win when it completes all of its objectives. It is not necessary to use objectives for the AI. As long as a map is constructed in a logical manner, using capturable zones, the AI will understand how to play it. Also see the TEAM commands AddMinimapMarker, FlashPositionInMinimap and RemoveMinimapMarker, which add and remove markers on the mini map. See the TEAM command ApRewardOrPenalty for a way to give AP to teams. The following parameters are shared by all OBJ_AddObjective commands:

Common attributes

Player myPlayer

This is the player who should receive the objective. There is no harm in giving AI objectives to human players, as long as there are sensical short and long descriptions available, but be careful what objectives you give to AI players, or players who could be AI if the map is played in skirmish mode.

LOCTEXT myShortDescription

This is the short description, the "name" of the objective. Keep it short and simple to understand. The AI pays no attention to this.

LOCTEXT myLongDescription

This is the longer description available when the objective is clicked. The AI pays no attention to this, but human players would like to know what to do on the map.

DECIMAL myImportance

Normally 1.0, this is the relative importance of the objective. For humans, this determines the order in the objective list. For the AI, this is a multiplier for the relative importance of this objective. 1.5 is a very large value. Using myImportance, the AI can be told that a specific objective is more (or less) important than it seems.

For some objective types, the AI requires extra information. These are some of the AI-specific parameters in Zone, SearchAndDestroy and DefendArea objectives:

DECIMAL myAIImportance

This is an absolute Importance value. Where myImportance only multiplies on a value calculated by the AI, this value can be used to tell the AI "exactly" how important the SearchAndDestroy ("SAD") objective should be. 15 000 is a good starting value, but should be tweaked if the AI does not seem to value the SAD-objective enough while playing.

DECIMAL myCautionFactor

This is an objective-specific multiplier of the AI-specific general value EngageSuccessProbability, described below in the description for the command PLAYER_AiChangeEngageSuccessProbability. Basically, a low Caution factor means the AI will try to complete the objective even with very small or weak forces. A high Caution factor can mean that the AI will never "dare" to try completing the objective.

NUMBER myDifficulty

This is the unit strength required to complete the objective. This value is based on a calculated "worth" of each type of agent. A lower value means a smaller, weaker force is sent to the objective, while a greater value means that a more powerful force will be sent. For comparison, a single Liberator Terradyne has a calculated value of 74.

OBJ_AddObjective_HumanObjective

This objective is only used for Human players. It displays objective text for the player, but does not relate this to any area or agent on the map. This command has no parameters but the four common ones mentioned above. The AI will ignore objectives of this type.

OBJ_AddObjective_DefendArea

This objective tells an AI to guard an area. The AI only uses the centre of the specified area, so the size of this area does not matter, but the centre point **must** be accessible. It cannot be blocked by a structure or a slope in the terrain.

Specifying an area of this type among neutral turrets will motivate the AI into using the turrets, if there is no other objective nearby. This is useful for making the AI defend a base with no zones, or to guard choke points, entrances and exits to isolated areas. This objective has a myCaution value.

OBJ_AddObjective_DefendUnit

Do not use. This is a very special purpose objective only used on one mission in the GC2 campaign. It makes the AI defend an immobile agent. The defended agent must be added in scripts prior to this objective. This command may cause crashes on other maps. Do not use it.

OBJ_AddObjective_SearchAndDestroy

Special objective which makes the AI patrol a list of areas. When the squadron assigned to this objective finds an enemy they will attack at all costs. It is best for the areas in the patrol route to be connected. This Objective command requires a specific Difficulty and AIImportance.

AreaRefList myAreas

This parameter lists all the areas that the “SAD” objective covers.

OBJ_AddObjective_Zone

This is a standard AI objective that tells the AI to capture and defend the specified VL or LZ. The objective command is normally unnecessary, because the AI will always prioritize capturable VLs and LZs by default. One reason to create a Zone objective for the AI is the ability to set Caution and Importance values. This command has no effect on non-capturable zones, as the AI reasons that it does not need to capture of defend something which cannot be captured from or lost to an enemy. This objective uses an individual Caution factor.

TEAM_AddZone myZone*

This parameter is a reference to the Zone connected to this objective. The TEAM_AddZone command must be created before the Objective. The zone must have been added in scripts prior to this objective.

OBJ_AddObjective_Annihilate

Not used. This command was intended to give the AI an objective to dominate the entire map, killing all opposition in the most aggressive mode possible. Hopefully it no longer works.

OBJ_ChangeObjectiveImportance

This command changes the importance value of an existing AI objective. Changing this during a mission can be a good way to divert the AI’s attention and make it seem to respond to a situation which the AI might otherwise ignore. The AI is a clever tactician, but it has no flair for the dramatic.

DECIMAL myNewImportance

This decimal value is the new importance multiplier for the listed objectives.

ObjectiveRefList myObjectives

This is a list of references to the objectives which should have their importance changed. The objectives must have been created and not removed before this command is used.

OBJ_ObjectiveCompleted

This marks an existing objective as completed. The command is recommended for use only on human players’ objectives. The objective will be marked as complete in the objective screen, but will remain visible in the list. Please note that even a Failed objective can be changed to Completed, so be careful with this command.

ObjectiveRefList myObjectives

This is a list of references to the objectives to be completed. The objectives must have been created but not removed.

OBJ_ObjectiveFailed

This command marks one existing objective as failed. The command is recommended for use only on human players. The objective will be marked as failed in the player’s objective screen, but will remain visible in the list.

ObjectiveRefList myObjectives

This is a list of references to the objectives which have failed. The objectives must have been created, but not removed. Please note that even a Completed objective can be Failed, so be careful with this command.

OBJ_RemoveObjective

This removes an objective for a Human or AI player. This is the only correct way to remove an objective for an AI player. The objective will no longer be on the human player's list of objectives, and the AI will no longer attempt to complete it.

ObjectiveRefList myObjectives

This is a list of references to the objectives to be removed. The objectives must have been created before they can be removed. It does not matter if the objective is failed, completed or still active, though it is less confusing for human players if the objective becomes either failed or completed before removing it.

Thing commands

A thing is based on a prop instance but has the ability to change state. A thing has no collision so Agents can move through Things. All commands except THING_AddThing itself use references to a THING_AddThing command:

THING_AddThing myThing*

This is the affected thing, which should move or change animation state or turn to look at another object.

To create a Thing, select one you want it from the Instances list. Click in the world to place it. Then make it usable and visible through scripting with the command THING_AddThing. Remove it with THING_RemoveThing. Note that the THING_AddThing command uses a reference to the thing instance, while THING_RemoveThing uses a reference to the THING_AddThing command used.

THING_AddThing

This command adds a thing that has been placed in the 3D view. All script references to the thing are then made to this command.

XThingInstance myThingInstance*

This is a reference to the thing instance. Place it into the world before connecting an AddThing script command to it, and then rename the command appropriately.

THING_LookAtAgent

This command was never used and might not work. It was probably intended to rotate the thing so that it faces towards a specified agent.

PLAYER_AddAgent myAgentToLookAt*

This is a reference to the agent the thing should rotate towards.

ActionAffector myAffector

This value determines if the specified or the context agent should be used.

THING_LookAtCamera

This command was never used in Ground Control II and might not work. It was probably intended to rotate a thing so that it faces towards the camera of the specified team. What the effect would be in an on-line game is difficult to say, as each team member has a camera of their own.

Team myCameraToLookAtTeam

The team whose camera the Thing should turn to look towards.

THING_LookAtThing

Not used, possibly not implemented. The command was intended to rotate a Thing so that it faces toward another Thing.

THING_AddThing myThingToLookAt*

This is a reference to the thing myThing should turn towards.

THING_MoveTo

Not used. This command moves a THING_AddThing instance instantly to the centre of an area. Because of the instant transportation, this is not a good way to make a Thing move as part of a cinematic cutscene.

XSphereInstance myPosition*

This reference is the area the Thing should be transported to. The Thing will end up in the absolute centre of the area.

THING_RemoveThing

Removes a thing added previously by the THING_AddThing command. The thing will no longer be visible in the 3D world. Note that it will still be visible in XEd, because XEd does not parse script commands, and displays all instances whether they are in use or not. This command does not use any values other than myThing.

THING_SetAnimationState

This command sets the animation state of an added thing. The default animation state is STND, for stand, an idle animation. Frequently used four-letter states are: STND, WALK, FIRE, normally for “idle”, “active” and “death”.

TEXT my4LetterAnimationState

This is the animation state that the Thing should change into. It is encoded into four letter combinations. STND is always the default, the state every newly spawned thing is created in.

Thing name	**Animations**
Tutorial Beacon	WALK activates light & smoke.
Vir ClanShip	WALK activates damage effects.
Vir ClanShip TakeOff	WALK launches the ship.
WaterExplosion, Explosion	WALK triggers the explosion effect.
BG Block House 05 Explode, BG Block House 07 Explodable, explohouse1, OrbitalGun, Shield Generator BG, Power Generator, FL commandcenter	In STND the thing is undamaged. In WALK, the thing is destroyed. If the thing is changed back to STND after WALK, it will snap back into an undamaged look.
RD Main	WALK activates the dish. STN2 deactivates it.
Des Xenofact 02	WALK makes the Xenofact glow. FIRE explodes it.
Des Vlaana mountainbase Entrance 01	STND closes doors. WALK opens doors. FIRE explodes the structure.
TRADER_SHIP	STND lands the ship, during 17s. WALK launches it.
Imp ComShuttles	WALK launches the craft.
Imp ComShuttles LAND TAKEOFF	FIRE lands the craft and WALK launches it.
LaunchPad Arm BG VISUALMODEL	WALK activates the launch pad arm.
Escape Pod BG	WALK activates a shuttle, FIRE launches the shuttle and BRNY explodes it.

THING_SetPlayerColor

This command was never used in Ground Control II. It can be used to change the colour of a spawned thing to that of a certain player, much like the landing zone and victory location markers change colours based on who owns it. Therefore, the VL_thing and LZ_thing objects will probably work with this command, but most things might not.

Player myPlayer

This is the player whose colour should be used on the thing.

Triggers

Triggers check for a specific situation and will activate a target script when the situation arises. When they have done so, they have been spent and are automatically destroyed. To activate the same script when the same situation arises again, it will be necessary to create a new trigger, or to somehow restart the one which was spent.

If a trigger monitors specific agents it is necessary for the agent to exist when the trigger is created. Similarly, triggers which monitor players require the player to be connected to the game when the trigger is created. Because of this, it is important to start Triggers after players have connected and agents have been created. Also note that all triggers cost processing power. It is not much, but do try to keep the number of active triggers to a minimum and kill any triggers you no longer need.

If an Agent-trigger contains several agents in a list and some of these agents die or are removed, the trigger can still be activated by the remaining agents. This can give rise to peculiar situations. For example: A map has three agents: AGENT_1, AGENT_2 and AGENT_3. The designer creates a TRIGGER_AgentsEnterArea on an area named GOLD_AREA, listing all three agents, with myFireWhenFlag set to AllAgents and myTriggeredScript will display the message *"All agents have found the gold!"*.

During the course of playing this map, AGENT_1 and 2 enter the GOLD_AREA, but AGENT_3 is held up on the way there. While AGENT_3 is delayed, AGENT_1 and 2 have time to leave the GOLD_AREA. When AGENT_3 dies several minutes later, without ever reaching GOLD_AREA, the message *"All agents have found the gold!"* will appear, because now all (living) agents in the list have entered the area even though none of them is within the area at this time. To the player who just destroyed AGENT_3, this will seem very odd. Be aware of this possibility.

Trigger context

Most triggers update the current CONTEXT, but what is it updated to? The principle is that the agent, or player, or team who caused the Trigger to become active will become the new CONTEXT. Some triggers are confusing, however. For example, what context does AgentsWereKilled create? The killed agent is obviously dead and not a very useful context. Therefore it is logical to assume that it is the killer who becomes stored as the new context agent. But if that is the case, then what about AgentsUnderAttack? Is it the attacker or the attacked who becomes stored as context in that case? It is all very confusing.

See the chapter on the CONTEXT system for more detail.

Common attribute

All triggers share this attribute, and TRIGGER_ConditionBranch can have several:

Script myTriggeredScript*

This is the script which will be activated when the trigger fires.

TRIGGER_AllPlayersReady

This trigger waits for all players to be connected to a game before it runs a script. This command is very useful for delaying the start of a multiplayer or cooperative mission until all players have connected to the game. This trigger requires no attributes other than myTriggeredScript. See also the chapter on Starting.

TRIGGER_Timer

The simplest of all triggers. Create the trigger, set the timer and when the time runs out, the target script will be started. Easy to use, and extremely difficult to do without.

DECIMAL myTimeInSeconds

This is the time, in seconds, after which the trigger will fire and activate its myTriggeredScript. The value is decimal, and will therefore accept fractions of seconds.

Due to imperfect accuracy in the conversion between numbers and text strings, please do not be alarmed if 10.0000 suddenly comes out as 10.0001 or 9.9998.

TRIGGER_TriggerIsTriggered

This trigger activates a script when a certain number of listed triggers have fired. This trigger is special in that it does not require its monitored triggers to exist when it is started. TRIGGER_TriggerIsTriggered will wait for the a trigger of the specified name to fire. Only SCRIPT_KillTriggers can prevent this, either by killing the monitored triggers or by killing the TRIGGER_TriggerIsTriggered itself.

TriggerRefList myTriggers

This is a list of references to triggers to be monitored. The triggers do NOT need to have been created, or even be created. The listed triggers will use full names, such as myInstances.myScripts.START.TRIGGER_Timer__0.

NUMBER myNeededNumberOfTriggeredTriggers

A mission designer can choose to fire the trigger only when this amount of monitored triggers have fired. If this number of triggers never fire, neither does the TRIGGER_TriggerIsTriggered.

```
TRIGGER_TriggerIsTriggered TRIGGER_TriggerIsTriggered__TWO_TIMERS
{
  myTriggeredScript myInstances.myScripts.TWO_TIMERS_SCRIPT
  myNeededNumberOfTriggeredTriggers 2
  myTriggers
  {
    TRIGGER_Timer* Timer_1 myInstances.myScripts.TRIGGER.Timer_1
    TRIGGER_Timer* Timer_2 myInstances.myScripts.TRIGGER.Timer_2
    TRIGGER_Timer* Timer_3 myInstances.myScripts.TRIGGER.Timer_3
  }
}
```

The above simplistic example will activate the script TWO_TIMERS_SCRIPT once two of the listed timers in the TRIGGER script have run out. For example, if the timers are 1, 2 and 3 minutes long, the script will be started once 2 minutes have passed.

TRIGGER_CampaignFlagNotSet

Triggers a script if the specified campaign flag has not been set, i.e. does not yet exist. Even if the campaign flag is set to zero or <null>, it is considered to be "set".

TEXT myFlag

Enter the name of the campaign flag in this value.

TRIGGER_CampaignFlagSet

Triggers a script if the specified campaign flag has been set, i.e. that it exists. Even if the campaign flag is set to zero, it is considered to be “set”.

TEXT myFlag

Enter the name of the campaign flag in this value.

TRIGGER_ConditionBranch

Sets up one or more Options. Only one of these options will execute its script when all of its Conditions have been met.

Options myOptions

This is the list of options. Each option will have a target script, and each has a set of conditions which need to be fulfilled to start that script. Compare this to a multiple-choice questionnaire. Several answers could be right, but only the first correct one will actually be chosen. Each Option consists of the following parameters:

Script myTriggeredScript*

This is the script which will start if this Option is the one to become true first.

Conditions myConditions

These are the conditions for this Option to start its Script. Each option can have several Conditions which all have to be true for the option itself to be considered true as well.

```
TRIGGER_ConditionBranch TRIGGER_ConditionBranch_LZ3_or_LZ4
{
  myOptions
  {
    Option op1
    {
      myTriggeredScript myInstances.myScripts.LZ3
      myConditions
      {
        CONDITION_IntCampaignFlagEquals con1
        {
          myFlag FLAG_LZ3
          myValue 1
        }
      }
    }
    Option op2
    {
      myTriggeredScript myInstances.myScripts.LZ4
      myConditions
      {
        CONDITION_IntCampaignFlagEquals con1
        {
          myFlag FLAG_LZ4
          myValue 1
        }
      }
    }
  }
}
```

In this example, the ConditionBranch has two options, op1 and op2, with only one condition each. Option Op1 checks to see if the CampaignFlag FLAG_LZ3 equals 1, and starts the script LZ3 if it does, while the option Op2 instead checks the flag FLAG_LZ4 and starts the

script LZ4 if that equals 1. If either option had more than one condition, all conditions under that option must become true before the option starts its script. Also see definition of SCRIPT_ConditionBranch and the separate chapter on Conditions.

TRIGGER_GameOver

This trigger will fire when a team has won the game. A special campaign flag called MISSION_OPTION_WIN_TEAM will be set either to 0, for a tied game, or 1 to 8 for the team which won the game. This campaign flag can then be checked with a TRIGGER_ConditionBranch to produce different scripted endings depending on which team won the match.

TRIGGER_MessageBoxClosed

This trigger activates a script when the specified message box has been closed. The messagebox command must have been started when this trigger is created, or the trigger will never fire.

MessageBoxRefList myMessageBox

This list should contain only the message box to be monitored. The value is a list so that it can contain all three types of message box: TEAM_ShowAutomaticMessagebox, TEAM_ShowTimedMessageBox or TEAM_ShowButtonMessagebox, but the list should only hold one item at a time.

Agent triggers

The following triggers all deal with Agents. For practically all of these triggers, the one agent which causes the trigger to fire is saved as the new CONTEXT.AGENT, even when several agents are involved. Aside from myTriggeredScript, these triggers all have a few more attributes in common.

Common attributes

AgentRefList myAgents

This is a list of references to agents monitored by the trigger. The agents must have been created before the trigger. This list works just like the list of the same name on AGENT commands. If myAffector is SPECIFIC, all listed agents will be affected by the trigger, in conjunction with the myFireWhenFlag value, which determines if all agents are affected, or only the first one to perform the required action.

AgentsFlag myFireWhenFlag: FirstAgent, AllAgents

This value can be either FirstAgent or AllAgents, which will determine if the trigger fires as soon as one of the listed agents enter the area, or if it should wait for all agents. If there is only one agent in the list this value has no function. If any of the agents in the list are dead or do not exist, AllAgents means only all agents which are still alive, which can have some strange side-effects.

TriggerAffector myAffector: SPECIFIC, CONTEXT or ANY

This is different from ActionAffectors used by non-triggers, which only be SPECIFIC or CONTEXT. When set to SPECIFIC, the myAgents list is used. When set to CONTEXT, the trigger will wait for the agent currently saved into CONTEXT.AGENT instead of any agents listed in myAgents, which can be left empty. When set to ANY, the trigger can be activated by any unit what so ever, no listing or context required.

TRIGGER_AgentsEnterArea

This trigger activates a script when a listed number of agents have entered the defied area. The trigger can be set to wait for all the listed agents or to fire when the first listed agent enters. Please mote that the larger an area is, the more processing power it will require for a trigger to monitor it.

XSphereInstance myArea*

This is the area monitored by the trigger. As soon as the required agents enter this area, the trigger fires and activates myTriggeredScript.

TRIGGER_AgentsLeaveArea

Activates a script when the specified agents have left the area. The agents may have to be inside the designated area when the trigger is set up in order for it to trigger when they leave.

XSphereInstance myArea*

When the agents exit this area, the trigger will activate its script.

TRIGGER_AgentsEnterBuilding

This trigger activates a script when a list of agents has entered the specified building or other Prop. Obviously the listed myAgents need to be capable of entering props, and the prop needs to be capable of harbouring agents, or the trigger is useless. This is something you as a designer need to be aware of. See also AGENT_EnterBuilding.

XPropInstance myBuilding*

This is the building or other Prop which the agents must enter to activate the trigger.

TRIGGER_AgentsEnterContainer

Triggers a script when a list of agents has entered another agent. Typically, infantry can enter turrets and APC vehicles, while vehicles can enter the NSA Transport Copter. Also refer to AGENT_EnterContainer.

PLAYER_AddAgent myContainer*

This is the agent with a container parasite which myAgents must enter for the trigger to fire. The agent must be created before the trigger.

TRIGGER_AgentsExitBuilding

This trigger fires when the specified agents exit the myBuilding prop. The trigger must be created after the agents, but it does not matter if the agents have entered the building already. This trigger will calmly wait until they Exit before activating the target script.

XPropInstance myBuilding*

This is the building or other Prop which the agents must exit to activate the trigger.

TRIGGER_AgentsExitContainer

This trigger fires when the specified agents exit the myContainer agent. The trigger must be created after the agents, but it does not matter if the agents have entered the Container already. This trigger will calmly wait until the agents exit this container before activating the target script.

PLAYER_AddAgent myContainer*

This is the agent with a container parasite which myAgents must exit for the trigger to fire. This agent must be created before the trigger.

TRIGGER_AgentsHaveKilled

Triggers a script when the listed agents have killed any agent. When myAffector is set to ANY, obviously this trigger will fire as soon as any agent has been killed by any other agent. The agent which causes the kill is stored into Context. This trigger requires no additional attributes.

TRIGGER_AgentsUnderAttack

Triggers a script when the listed agents have been attacked and taken damage. Attacks that do no damage does not activate this trigger; nor should friendly fire, but the detection code might not be able to distinguish between friendly and hostile splash damage. The agents responsible for the attack will become the new CONTEXT.AGENT. This trigger takes no additional attributes.

TRIGGER_AgentsWereKilled

Triggers a target script when the specified agents have died. This trigger was used very often throughout the Ground Control II campaign. The agent that died will be the new

CONTEXT.AGENT which will not be very useful, but CONTEXT.PLAYER and CONTEXT.TEAM can be. This trigger requires no additional attributes.

Agent Type triggers

These triggers all deal with agent types. The names of all agent types can be found in the Instances window in XEd, listed under the Agents branch. F1_Assault_APC is one agent type, F2_Covinus is another.

The greatest benefit over Agent triggers is that there is no requirement for the expected agent to exist at the time the Agent Type trigger is created, as there is no specific references to agents, only agent types. Agent type triggers cannot use CONTEXT for detection, but will still update context by saving data on the specific agent which activated the trigger. Because of this, they behave similarly to Agent triggers, but lack myAffector and myFireWhenFlag attributes, and replace myAgents with myAgentType.

Common attribute

TEXT myAgentType

The myAgentType is written in plain text and cannot use Context. The name of the agent type must be exactly the same as in the agent instance list in XEd. There is no selection box in XEd for this value, so it is up to the designer to type the exact agent type name, for example F1_Engineer_Vehicle, F2_Missile_Walker or F3_UnmelderEgg.

TRIGGER_AgentTypeEnterArea

Activates a script when an agent type has entered an area. Tip: You can detect melding by checking an area for F3_MelderEgg, F3_MelderEggAir and F3_MelderEggInfantry.

XSphereInstance myArea*

This area will be checked several times per second for an agent of myAgentType.

TRIGGER_AgentTypeEnterBuilding

Starts a script when one agent of the specified agent type has entered the building. Remember that there is no point in waiting for a tank to enter a hotel, since it will never happen. Not even an aircraft flying over the building will activate the trigger, because it expects agents to actually take up a position slot inside the building.

XPropInstance myBuilding*

Select the building you wish to monitor for the specific agent type. Unlike the myBuilding attribute for the AgentEnterBuilding trigger, XEd will display all props in this list; not just actually enterable buildings.

TRIGGER_AgentTypeExitBuilding

This trigger fires when agents of the correct type exit myBuilding. It does not matter if agents have already entered the building. This trigger will wait until agents exit before activating the target script.

XPropInstance myBuilding*

This is the building or other Prop which agents must exit to activate the trigger. Please note that in contrast to AgentExitBuilding, XEd will list all buildings here, instead of just enterable ones. Choose carefully.

TRIGGER_AgentTypeEnterContainer

This trigger activates a script when an agent type has entered the specific container, normally an APC or a gun emplacement. Please note that there is no check to see if myContainer is really a container – all PLAYER_AddAgent commands are listed.

PLAYER_AddAgent myContainer*

This is the container agent to be monitored for entering agents of the correct type. This agent must exist when the trigger is created.

TRIGGER_AgentTypeExitContainer

This trigger fires when agents of the specified type exit the specified gun emplacement, APC or other agent which can carry agents. There is no point waiting for tanks to enter an APC, unless you have given them a parasite enabling them to, so make sure you pick a sensible agent type.

PLAYER_AddAgent myContainer*

This is the agent with a container parasite which myAgents must exit for the trigger to fire. This agent must be created before the trigger.

TRIGGER_AgentTypeHaveKilled

Triggers a script when an agent type has killed another agent. The name of the agent type must be exactly the same as in the agent instance list in XEd. The agent which successfully kills another agent is saved as the current CONTEXT.AGENT. This trigger requires no additional attributes.

TRIGGER_AgentTypeLeaveArea

This trigger activates myTriggeredScript when an agent of the correct type has left the specified area. This can be used to detect if Virons have finished melding, because their F3_MelderEgg will “leave” the area into thin air.

XSphereInstance myArea*

This is the area which will be monitored for an agent of the matching type. The smaller the area, the more efficient the detection code will be.

TRIGGER_AgentTypeUnderAttack

Triggers a script when an agent of a specific type has been attacked and damaged. The agent which first takes damage will become the new CONTEXT.AGENT. This simple trigger requires no additional attributes.

TRIGGER_AgentTypeWasKilled

This trigger will start a script when any agent of the specific type has been killed. This trigger requires no other attributes. This can be a quick way of knowing when an NSA drop ship (F1_Dropship) has been destroyed, for example. This simple trigger needs only myAgentType and myTriggeredScript.

PLAYER triggers

Common attributes

Obviously, all PLAYER triggers will have a value in common for which player it acts on, as well as a myTriggeredScript value.

Player myPlayer

This value can be anything from PLAYER_PLAYER1 to PLAYER_PLAYER8 as well as CONTEXT.PLAYER and PLAYER_ANY. Setting this value to PLAYER_NOPLAYER will not work, as that means the trigger will “wait for no player”. It does not work that way.

TRIGGER_PlayerAction

Special trigger used in the tutorials and for skipping cutscenes. It triggers on many different kinds of input from the player.

PlayerAction myAction

This is the only command which uses a “PlayerAction” type attribute. This enumerated type can have any one of the following values:

AGENT_SELECTED	Player selects one agent.
AGENT_SELECTED_MULTIPLE	Player to selects more than one agent.

REPAIR_BUTTON_PRESSED	Player presses the Repair button.
SECONDARY_MODE	Player activates a secondary mode.
DROPSHIP_CONFIG_PRESSED	Player presses the drop ship configuration button.
DROPSHIP_DEPLOY	Player presses the Deploy button.
DROPSHIP_UPGRADE	Player views the drop ship upgrade screen.
DROPSHIP_UPGRADE_STARTED	Player has activated a drop ship upgrade.
DROPSHIP_SELECT	Player has selected his drop ship.
DROPSHIP_SELECT_LZ	Player has selected an LZ for the drop ship.
MEGAMAP_PRESSED	Player has pressed the objectives button.
MEGAMAP_CLOSED	Player has closed the objectives screen.
SUPPORT_WEAPON_USED	Player has used a support weapon.
SKIP_CUTSCENE	Player attempts to skip a cutscene.
NO_ACTION	Player has done nothing. Never used.

Most of these commands were only ever used in the tutorial missions, but the SKIP_CUTSCENE action is used throughout the Ground Control II campaign:

Script example

```
TRIGGER_PlayerAction CINE3_CutSceneSkipper
{
  myPlayer PLAYER_PLAYER1
  myAction SKIP_CUTSCENE
  myTriggeredScript myInstances.myScripts.NULL
}
```

As seen in this example, the trigger CINE3_CutSceneSkipper waits for PLAYER_PLAYER1 to press any combination of keys which are meant to skip cutscenes. It is script code responding to this simple trigger which ends all cutscenes, removing actor agents, silencing message boxes, et cetera. There is no programming involved, cinematic cut scenes must be created and removed piece by piece through scripting.

The reason this trigger points to the script NULL is that a TRIGGER_TriggerIsTriggered monitors CINE3_CutSceneSkipper as well as triggers that indicate that the cinematic cut scene has run its course. Then the IsTriggered activates CINE3_EXIT, which contains all the necessary clean-up scripts. See the chapter on Cutscenes for more information.

TRIGGER_PlayerConnected

Triggers a script as soon as a player has connected. This uncomplicated trigger only requires a player. In all retail multiplayer missions, one TRIGGER_PlayerConnected will wait for PLAYER_ANY to connect, and then his camera was moved to his LZ and he received his objectives.

Script example

```
Script Player_connect_ANY
{
  TRIGGER_PlayerConnected TRIGGER_PlayerConnected_ANY_1
  {
    myTriggeredScript myInstances.myScripts.Player_connected_ANY
    myPlayer PLAYER_ANY
  }
}

Script Player_connected_ANY
{
  SCRIPT_ActivateScript Reactivate_Player_connect_ANY
  {
    myScript myInstances.myScripts.Player_connect_ANY
  }
  PLAYER_MoveCameraToLz PLAYER_MoveCameraToLz__17
  {
    myPlayer CONTEXT.PLAYER
    myMoveTimeInSeconds 0.000000
  }
  OBJ_AddObjective_HumanObjective OBJ_DeathMatch
  {
    myPlayer CONTEXT.PLAYER
    myShortDescription "Deathmatch"
    myLongDescription "Kill all enemy units"
    myImportance 0.0
  }
}
```

In this example, TRIGGER_PlayerConnected_ANY_1 waits for PLAYER_ANY, and then that player is saved as the CONTEXT.PLAYER. The script Player_connected_ANY will first restart the Player_connect_ANY and the spent trigger. Then the player's camera will be moved to his LZ, and the player receives a simple objective. The next player to activate the now-resurrected TRIGGER_PlayerConnected_ANY_1 will receive the exact same treatment, as well as cause the trigger to be resurrected once more and wait for the third player who connects.

TRIGGER_PlayerEnterArea

This trigger is activated when agents belonging to the specified player enters myArea. Remember to make the area only as large as strictly necessary. It is often better for performance and accuracy to use several smaller areas and tie them together with a TRIGGER_TriggerIsTriggered instead of using one very large area.

XSphereInstance myArea*

Point this value to the area you wish to monitor. Prepare by creating this area before setting up the trigger.

TRIGGER_PlayerLeaveArea

Activates a script when all agents belonging to the player have left the specified area. The agents may have to be inside the area when the trigger is set up in order for it to become active once they leave.

XSphereInstance myArea*

When the player has left this area, the trigger will fire.

TRIGGER_AnyOtherPlayerEnterArea

This trigger is the direct opposite of TRIGGER_PlayerEnterArea. Instead of waiting for the specified player to enter myArea, the trigger will be activated when any player other than myPlayer enters the area. This can be useful to let one player know if another is intruding, instead of setting up one PlayerEnterArea trigger for every player in the game.

XSphereInstance myArea*

This is the area to be monitored by the trigger.

TRIGGER_PlayerEnterBuilding

Starts a script when one agent belonging to the specified player has entered the building. The trigger expects agents to take up a position slot inside the building.

XPropInstance myBuilding*

Select the building you wish to monitor. Only enterable buildings will be listed.

TRIGGER_PlayerExitBuilding

Starts a script when one agent belonging to myPlayer exists the specified building.

XPropInstance myBuilding*

Select the building you wish to monitor. Only enterable buildings will be listed.

TRIGGER_PlayerEnterContainer

This trigger activates a script when an agent belonging to myPlayer has entered the specific container agent. Using this, you can perform script actions on a player who captured a radar tower, gun emplacement or a capturable vehicle. Why not blow him up in a good old fashioned mafia car bomb using the AGENT_Die or AGENT_Hurt commands and CONTEXT.AGENT?

PLAYER_AddAgent myContainer*

This is the container agent to be monitored for agents belonging to the specific player. There is no check to see if myContainer really is a container – all PLAYER_AddAgent commands are listed. This agent must exist when the trigger is created.

TRIGGER_PlayerExitContainer

This trigger fires when agents belonging to the player exit from the specified container agent.

PLAYER_AddAgent myContainer*

This is the agent with a container parasite which the player must exit to activate the trigger. This agent must be created before the trigger.

TRIGGER_PlayerHaveKilled

This trigger starts a script when the player has killed an agent. It is possible that this trigger will fire even if the player kills one of his own agents. When myAffector is set to PLAYER_ANY, this trigger will fire as soon as any player has killed anything. This trigger requires no additional attributes.

TRIGGER_PlayerUnderAttack

Triggers a script when any unit belonging to the specific player has been attacked and damaged. Again, it is possible that this trigger will be activated even if the player damages his own agents. Those tricksy humanses will do anything to break a good script. This trigger requires no additional attributes.

TRIGGER_ZoneTakenByPlayer

This trigger starts its target script when a player captures myZone, a TEAM_AddZone. This means that the player must have held the zone until it has passed from Neutral ownership to Team ownership, but that this must have been done by this exact player. It may seem confusing that this trigger can detect which exact player who took over a zone, when zones can only be owned by teams, but this trigger certifiably works, as it is an essential part of many Ground Control II missions. The trigger makes no distinction between different types of zone, as long as they were created by a TEAM_Addzone command.

TEAM_AddZone myZone*

This is the zone which should be monitored. The zone must have been created before the trigger is set up. Please note that this trigger can only monitor a single zone, in contrast to ZonesTakenByTeam which uses a ZoneList instead of a TEAM_AddZone reference.

TEAM triggers

These triggers have a few benefits over player triggers. For one thing, it is often more relevant to keep track of teams than of players. Zones are controlled by teams, not players, and because players can freely choose their teams it is never certain what team a specific player will be on. Also in the case of Cooperative missions it can be very useful to trigger by TEAM_1 instead of using three or more separate player-related triggers expecting either one of the players on the cooperating team.

Common attributes

All TEAM triggers deal with specific teams, and so they all have a value distinguishing which team is expected.

Team myTeam

This is the team which the trigger expects. It can be set to TEAM_1 to TEAM_8, but also to TEAM_ANY for any team at all as well as CONTEXT.TEAM. TEAM_NOTEAM is also available as an option, and covers neutral gun emplacements and other agents without a player in control.

TRIGGER_TeamEnterArea

This trigger is very similar to TRIGGER_PlayerEnterArea. The only difference is that this trigger monitors an area for agents belonging to any player in the team.

XSphereInstance myArea*

Point this value to the area you wish to monitor. Prepare by creating the area before setting up this trigger.

TRIGGER_AnyOtherTeamEnterArea

Activates a script when all agents belonging to all players on the team have left the area. The team may have to have agents inside the area when the trigger is set up in order for it to become active once they leave, but it's possible that the trigger works even if the team hasn't passed through yet.

XSphereInstance myArea*

When all agents belonging to this team have left this area, the trigger will fire.

TRIGGER_NumberOfZonesTakenByTeam

This trigger activates the targets script when the specified team has ownership of at least the specified number of TEAM_AddZones. This trigger is very practical in that it contains no

references to the specific zones, and thus has no requirement for being created after the zones have been created.

NUMBER myNumberOfZonesNeeded

This value is a simple integer number. Remember that the trigger will fire when the team has **at least** this number of zones. For example, a value of four will activate the trigger instantly if the team already has five zones.

```
TRIGGER_NumberOfZonesTakenByTeam TRIGGER_Player_Takes_All_Zones
{
  myTriggeredScript myInstances.myScripts.EV_13
  myTeam TEAM_1
  myNumberOfZonesNeeded 3
}
```

The above example is from C2M13, the first mission in the second campaign and the thirteenth mission total. It is a Viron mission with three Victory Locations. Because the NumberOfZonesTakenByTeam trigger does not require zones to be created before the trigger, it is perfectly suited to this level where zones appear over the course of the mission.

As soon as TEAM_1, meaning the human team of up to 3 cooperative players, are holding three zones simultaneously, the EV_13 (event 13) script is started, and the success sequence starts playing from there.

TRIGGER_ZonesTakenByTeam

The ZonesTakenByTeam trigger is unique in that a single trigger of this type can monitor several zones at the same time. It waits for the specified team to take over one or all of the TEAM_AddZones listed in its ZoneList myZones. Much like agent triggers, it also features an attribute – myAllZonesFlag – which controls whether all are required zones to have been taken or if it is enough if one of the listed zones falls under the control of the specified team.

The greatest benefit from this construction is that the singular trigger will not be activated if the team takes one of the zones, loses it, then takes another zone. This trigger keeps track of all that, without the need to create additional trigger to check if any another team takes the zones back.

If the listed zones already belong to myTeam, the trigger will fire instantly.

ZoneList myZones

This is a list of references to TEAM_AddZone commands. If this list contains only one zone, myAllZonesFlag has no effect as that one zone is both the first zone and all listed zones. Evidence from NSA mission 10 suggests that in contrast to standard, the zones listed here do NOT have to be created before the trigger.

Flag myAllZonesFlag

TRUE or FALSE. When this flag is TRUE, the trigger will wait until all myZones are under control by myTeam simultaneously before starting myTriggeredScript. To script this kind of control system in any other way would be very, very complicated if at all possible. When this flag is false, it is enough for either one of the zones listed in myZones to be controlled by myTeam in order for the trigger to activate the target script.

Script example

```
Script TRIGGERS
{
  TRIGGER_ZonesTakenByTeam Player_Has_All_Zones
  {
    myTriggeredScript myInstances.myScripts.NULL
    myTeam TEAM_1
    myAllZonesFlag TRUE
    myZones
    {
      TEAM_AddZone* LZ1 myInstances.myScripts.ZONES.LZ1
      TEAM_AddZone* LZ2 myInstances.myScripts.Second_Wave.LZ2
      TEAM_AddZone* LZ3 myInstances.myScripts.Second_Wave.LZ3
      TEAM_AddZone* LZ4 myInstances.myScripts.Second_Wave.LZ4
    }
  }
  TRIGGER_TriggerIsTriggered TRIGGER__time_is_up
  {
    myTriggeredScript myInstances.myScripts.EV_11_time_is_up
    myNeededNumberOfTriggeredTriggers 1
    myTriggers
    {
      TRIGGER_ZonesTakenByTeam* Player_Has_All_Zones
myInstances.myScripts.TRIGGERS.Player_Has_All_Zones
      TRIGGER_Timer* Timer_EV_11
myInstances.myScripts.EV_10.Timer_EV_11
    }
  }
}
```

This is more or less an exact example from C1M10, NSA mission 10. Within the TRIGGERS script we find a TRIGGER_ZonesTakenByTeam trigger listing all the possible Landing Zones on the map. The trigger waits for TEAM_1, the human-and-allies team to take over all four zones. When they do, it will only start the script called NULL which has no commands in it, but there is also the TRIGGER_TriggerIsTriggered to watch out for. It needs only one out of its two monitored triggers to fire and then it will activate script EV_11, event eleven. One of these monitored triggers is our ZonesTakenByTeam trigger mentioned above, while the other is a 1200 second Timer that is not created until in script EV_10, event ten.

What does all this mean? Well, if it works, this means that a skilled player or team of cooperative players could manage a quick win on C1M10 by taking over all four landing zones and control them all at once. Did you know that?

Conditions

Conditions are not commands in the ordinary sense. They cannot be selected directly in the Script window, but they are available as options for TRIGGER_ConditionBranch and

SCRIPT_EvaluationBranch, where they help decide if an Option is either True or False. Several Conditions can be used in conjunction when several are listed under a single Option, in which case all conditions need to be or become true in order for the Option to be true.

As you can see, a non-existent campaign flag will return the value null (0) which is the same value as if the campaign flag existed, but contained the value zero (0). There is no condition which can directly determine if a campaign flag is set (exists) or not, but the condition IsTriggered can be used in conjunction with a CampaignFlagNotSet trigger to make this distinction.

Note that when a Condition is added to a list of conditions, it will be named myScriptclassEntry by default. Be aware that this is only a default name, and you can freely change it to distinguish the different conditions you use.

CONDITION_IntCampaignFlagEquals

This condition checks if the integer campaign flag in myFlag has a value which equals myValue.

TEXT myFlag

This is the name of the integer campaign flag that should be checked. If the flag does not exist, or the name is incorrect, its value will be null (0).

NUMBER myValue

If the campaign flag number is equal to myValue, the condition will be true.

CONDITION_StrCampaignFlagEquals

This condition checks if the string campaign flag in myFlag has text content which equals the text string in myValue.

TEXT myFlag

This is the name of the string campaign flag that should be checked. If the flag does not exist, or the name is incorrect, the value will be null (0).

TEXT myValue

This is the text string compared to the contents of the string campaign flag.

CONDITION_IntCampaignFlagNotEquals

TEXT myFlag

This is the name of the integer (number) campaign flag that should be checked. If the flag does not exist, or the name is incorrect, the value will be null (0).

NUMBER myValue

If the campaign flag number is NOT equal to myValue, the condition will be true.

CONDITION_StrCampaignFlagNotEquals

TEXT myFlag

This is the name of the string (text) campaign flag that should be checked. If the flag does not exist, or the name is wrong, its value will be null (0).

TEXT myValue

If the campaign flag text string is NOT the same as myValue, the condition will be true.

CONDITION_IntCampaignFlagGreaterThan

TEXT myFlag

This is the name of the string (text) campaign flag that should be checked. If the flag does not exist, or the name is wrong, its value will be null (0).

NUMBER myValue

If the campaign flag number is greater than myValue, the condition will be true.

CONDITION_IntCampaignFlagLessThan

TEXT myFlag

This is the name of the integer (number) campaign flag that should be checked. If the flag does not exist, or the name is wrong, its value will be null (0).

NUMBER myValue

If the campaign flag number is less than myValue, the condition will be true.

CONDITION_Timer

When this condition is set up, the timer will start to count. Once the time has passed, the condition will become true.

DECIMAL myTimeInSeconds

The time it will take before the conditions becomes true.

CONDITION_IsAreaEmpty

This condition checks to see if a specified area is empty of Agents.

XSphereInstance myArea*

This is a reference to the area that should be checked. Place the area in the world like any other area, and then link to it here.

CONDITION_IsTriggered

This condition works just like TRIGGER_TriggerIsTriggered, checking the listed triggers. Using this command, any trigger at all can be used as a Condition. If the trigger has already been activated when the condition is set up, the condition will be considered true instantly.

TriggerRefList myTriggers

The condition becomes true when the listed triggers have been activated.