Force 21 Editor Help Overview

Introduction to the FED

Welcome to the Force 21 Editor. This tool has been released to provide gamers the opportunity to try out their own design ideas for Force 21 environments.

The editor will allow you to create basic multiplayer maps quickly. If you are interested in creating campaign missions or more detailed multiplayer maps, the FED provides a complete set of development tools for implementing your ideas.

The FED should be run at resolutions of 1024x768 or higher.

The FED is unsupported software. However, this help file and the map files shipped with the game should provide enough information to make the editor accessible to most gamers. Check the web frequently for the latest Force 21 information.

www.force21.com
www.redstorm.com

Welcome to armored warfare in the 21st century…

1. Run the New Environment Wizard

The FED includes a wizard to help in the creation of new maps. The wizard is available from the File Menu. The first page of the wizard prompts for the selection of an elevation layer. Experienced users may want to create a custom elevation layer, but it is easiest to select a pre-existing elevation layer. This is the bottommost layer in the FED's layering scheme and is required before proceeding with environment building. Once the elevation layer is in place, the mini-map will display an overview of the terrain. The green boxes framing the mini-map and the main map view mark the boundary of inaccessible terrain in the game. Areas outside this box may not be traversed by the players.

The second page of the wizard presents a choice of terrain tilesets. The tileset chosen will determine the terrain types available when painting the terrain layer of the map. A map must be fully textured to be used by the game. Make sure "Cover terrain layer with base texture" is checked to get up and running quickly. The third page of the wizard asks for cloud and water texture selections to be made in a similar fashion.

The last page of the wizard explains the basics of default object placement. New users should allow the wizard to create the default objects and layers. Once the wizard is finished, the default objects can be repositioned as desired. This is particularly important if the default placement results in objects on cliff faces or over water regions. The four objects represented by rectangular gray icons are platoons, which mark the initial vehicle unit for each player. If a map does not need to support four players, objects can be removed starting with objects owned by player 4. Keep in mind that there must be at least one platoon with one vehicle in it, or the map will not run in the game. Use the Object Forces toggle on the View Menu to display player ownership.

The wizard does not setup the side and force properties necessary for the game to be playable. This should be done manually in the Environment Properties.

Next: Apply terrain textures

2. Apply terrain textures

The elevation layer defines the basic topological features of the land. These features should be refined by applying textures to the map. Available base terrain textures are listed in the Terrain folder. Some tiles must be applied on top of other tiles in order to transition smoothly. Below the Terrain folder are several other folders named after terrain tiles. Each tile within a folder must be applied on top of the tile the folder is named after.

The FED texturing system was designed to give as accurate a representation of the game world as is possible in 2D. Painting one tile in the FED will generally result in one tile in the game engine. Exceptions occur in steep regions where texture coordinates are significantly influenced by elevation. On level ground, terrain textures are 32x32 meters in the game and 32x32 pixels in the FED at 1:1 zoom.

The fastest way to paint textures into the environment is using the Autotexture feature. Use the Autotexture configuration dialog, accessed via the toolbox or Tool Menu to set the parameters for an Autotexture operation. Choose a relative or absolute paint operation. Then choose whether to paint based on elevation or slope data. The boundary conditions are set with the two slider bars. Elevation bounds are specified in grayscale values. Slope bounds are specified in degrees. The "no overwrite" check box can be used to fill only the areas of the map that currently have no terrain type specified. For absolute operations, terrains are painted globally wherever the boundary conditions are met. For relative operations, the autotexture algorithm begins at a user-specified starting point and spreads to neighboring tiles.

Paintbrushes are also available to help with more detailed terrain painting. Terrain layer information can be reviewed from the Terrain Information dialog, accessed via the Report Menu. Once terrain layer modifications are complete, use the Create Terrain Bitmap command from the Layer Menu to generate the two bitmaps neededfor the game itself. The two bitmaps generated (the minimap and the strategic map) must be in the Scenarios directory for the mission to be playable.

Next: Place additional objects

3. Place additional objects

In addition to the required objects placed by the New Environment Wizard, there are a wide variety of objects available for placement in a map. The placeable objects are located within the Structures folder. Clicking on the folders in the Directory View, the middle window, reveals lists of objects in the Palette View, the rightmost window, each of which can be selected. When an object is selected it will appear as the active object in the toolbox. Note that many objects, particularly platoons, should not be placed in impassible terrain. Use the barrier toggles in the View Menu to display impassible regions.

A typical Force 21 map may contain hundreds of objects. To make managing these objects easier, objects may be organized into multiple layers. The best time to create these layers is before the objects are placed. Using the Layer Management dialog, accessed from the toolbox or Layer Menu, object layers can be added, renamed, and deleted. One approach is to place only objects of a certain type in each layer. For example, all forces could be placed in one layer, all structures in another, and so on. All maps maintain at least one object layer. There is exactly one "active" object layer at any time. This is the layer into which objects will be placed. The active layer can be selected via the toolbox or the Layer Management dialog. Information on all placed objects can be reviewed from the Object Information dialog, accessed via the Report Menu.

Once an object is placed, its properties may be edited with the Object Editor. The most important property associated with all objects is the object owner. You can set the force for one or more selected objects without opening the Object Editor. Just use Set Force from the map popup menu to specify owner 0-4. Force IDs (and object IDs) may be quickly reviewed by toggling Alt+I and Alt+O, or using the View Menu. Force IDs can also be specified automatically as objects are placed by setting a default force. This is done via the Palette Menu. The Default Force feature never affects objects already placed on the map. Object orientations can be set in similar fashion using Set Orientation from the map popup menu or Default Orientation from the Palette Menu. Choosing "random" orientation generates arbitrary rotation values for each object.

Next: Try out the new map!

4. Try out the new map!

Save the completed map as a *.gee file in the \Scenarios folder. Whether this is designed to be played as a single player scrimmage map or a multiplayer map, it will be played from the Force21 Multiplayer shell, so be sure that the flags are set correctly in the Environment Properties..

To use the new map in a multiplayer game, run Force 21 as usual. The new *.gee file will appear with the rest of the multiplayer maps in the \Scenarios folder. Important Note: All computers in the multiplayer game must have copies of five files to play a custom map.
1) The *.gee map file in the \Scenarios folder.
2) The two *.bmp bitmap files for the in-game map (minimap and strategic) located in the \Scenarios folder.
3) The *.tgz elevation file in the \Terrain folder (only if using a custom elevation layer)
4) The *.idx game summary file in the \Scenarios folder

Back to the Overview

File Menu Commands

The File menu offers the following commands:

New	Creates a new, empty Force 21 map.
Wizard…	Walks the user through the first steps of map design.
Open…	Opens an existing map.
Close	Closes the current map.
Save	Saves the current map using the same file name.
Save As…	Saves the current map to a specified file name.
Exit	Exits the Force 21 Editor.

Edit Menu Commands

The Edit menu offers the following commands:

Undo Autotexture	Reverses previous autotexture operation.
Mass Edit…	Edits the properties of many objects at once.
Cut	Moves selected objects to the clipboard.
Copy	Copies selected objects to the clipboard.
Paste	Copies objects from the clipboard into the map.
Delete	Deletes the selected objects.
Properties…	Edits the properties of the selected object.
Script…	Edits the script attached to the selected object.
Set Force	Assigns the ownership of the selected objects.
Set Orientation	Assigns the orientation of the selected objects.

View Menu Commands

The View menu offers the following commands:

Land-based Barriers	Toggles display of land-based barriers.
Helicopter Barriers	Toggles display of helicopter barriers.
Hide Textures	Hides terrain layer textures.
Blend Textures	Displays both terrain and elevation data.
Show Textures	Shows terrain layer textures.
Object IDs	Toggles display of object IDs over icons.
Object Owners	Toggles display of ownership over icons.
Zoom 1:1	Sets map display to 1:1 scale.
Zoom 1:2	Sets map display to 1:2 scale.
Zoom 1:4	Sets map display to 1:4 scale.
Zoom 1:8	Sets map display to 1:8 scale.
Zoom 1:16	Sets map display to 1:16 scale.
Mini-map	Shows or hides the mini-map.
Toolbox	Shows or hides the toolbox.
Status Bar	Shows or hides the status bar.

Environment Menu Commands

The Environment menu offers the following commands:

Select Clouds…	Selects the cloud texture for the map.
Select Water…	Selects the water texture for the map.
Properties…	Edits the properties of the environment.
Script…	Edits the environment script.

Layer Menu Commands

The Layer menu offers the following commands:

Select Elevation File…	Selects an existing elevation layer.
Import Elevation Layer…	Imports a bitmap as the elevation layer.
Edit Elevation Layer	Edits barriers and water level.
Select Terrain Tileset…	Selects a set of textures for the terrain.
Reload Terrain Layer	Reloads terrain textures from disk.
Create Terrain Bitmaps…	Generates bitmaps for the in-game maps
New Object Layer…	Creates a new layer of objects.
Delete Object Layer	Deletes the active object layer.
Manage Object Layers…	Manages all object layers.

Palette Menu Commands

The Palette menu offers the following commands:

Rename Variant	Renames the selected variant object.
Delete Variant	Deletes the selected variant object.
Object Properties…	Edits properties of the selected palette object.
Object Script…	Edits the script of the selected palette object.
Set Base Terrain	Sets the selected terrain as the base terrain.
Default Owner	Assigns new objects to the specified owner.
Default Orientation	Assigns new objects the specified orientation.

Tools Menu Commands

The Tools menu offers the following commands:

Paint Terrain	Paints the terrain layer with current texture.
Place Objects	Places objects of the selected type.
Autotexture…	Automates painting and mass placement.
Select Objects	Allows selection of placed objects.
Scroll Map	Enables mouse scrolling and measuring.
Small Brush	Paints textures with a small brush.
Medium Brush	Paints textures with a medium brush.
Large Brush	Paints textures with a large brush.
Random Fill	Paints textures with a random fill pattern.

Report Menu Commands

The Report menu offers the following commands:

Terrain Information…	Displays and manages all terrains.
Object Information…	Displays and manages all objects.
Script Information…	Displays and manages all scripts.

Window Menu Commands

The Window menu offers the following commands:

Cascade	Arranges windows in an overlapped fashion.
Tile	Arranges windows in non-overlapped tiles.
Arrange Icons	Arranges icons of closed windows.
Split	Splits the active window into panes.
Window 1, 2, ...	Goes to the specified window.

Help Menu Commands

The Help menu offers the following commands:

Help Topics	Displays this help information.
About FED…	Displays the version number of this application.

Compound Objects

Compound objects have an X, Y position like all placeable objects, but they may also include additional subpoints. When a compound object is dropped on the map, a line extends from the new object icon to the current mouse position. Each subsequent click will place a subpoint, represented by a small blue cross icon, and a line connecting the new point to the previous point. A right-click will end compound object placement, as will completing placement of all subpoints allowed for the object. When placement is complete, the subpoint icons will be hidden, but the connecting lines will remain. To edit a placed compound object, double-click the object icon in selection mode. This will display the subpoint icons, which may then be dragged to new positions or deleted with the delete key. If there are fewer than five subpoints in the object, additional subpoints may be added by shift-dragging a subpoint. Some compound objects, namely rects and radii, are limited to one subpoint. Compound objects may be Cut, Copied, and Pasted like any other objects. They will always be pasted so that all subpoints are at valid map locations.

Cut, Copy, and Paste

These standard edit operations affect the current selection of map objects. If multiple objects are cut or copied, their relative positions will be maintained for use in paste operations. If a paste would place objects off the map, the positions will be adjusted to values at the edge of the map. When pasting from the Edit Menu, the new objects will be centered in the map window. Objects are always pasted into the currently active object layer, as noted in the toolbox. Objects may be copied and pasted between different environments.

Mass Edit

To change properties of multiple objects simultaneously, use the mass edit tool. This tool will permit editing of templates, variants, and placed objects. First select a set of objects to consider with the radio button filters at the top of the dialog. Then choose a property to edit from the left listbox. Select one or more objects from the right listbox to modify. Placed instances are identified by class name and object ID, whereas templates and variants will have no object ID. The editbox at the bottom of the dialog can then be used to enter a new setting for the property. If all selected objects have the same value for the property, that value will appear above the editbox. Mass editing is particularly useful if a designer decides to modify a template after many instances of the template have already been placed in the world. If the designer wants the instances to have the new template property settings, the mass edit tool simplifies the update process. Mass editing can also be used to attach a script to multiple objects at once.

Editing Modes

The FED has five basic modes. The terrain painting mode allows access to the texture painting tools. Object placement mode allows the designer to place new objects into the active object layer. Selection mode permits the modification of existing objects and the editing of compound objects. Autotexture mode can be used to automate the placement of terrain. The scroll mode allows the designer to scroll freely around the map by clicking and dragging. This mode doubles as a ruler tool. The displacement from the last mouse-down position is displayed in the coordinates area of the toolbox.

The user may toggle back to the previous mode with the spacebar. Many actions in the FED will automatically trigger a switch to the appropriate mode. For example, selecting an object from the template list in the rightmost pane will switch the FED to object placement or terrain painting mode. Right clicking the map will prompt a switch to selection mode.

Autotexture

Selecting the autotexture tool will present a configuration dialog. Using this feature, the designer may automatically apply the selected terrain type globally or locally, based on relative or absolute elevations or slopes. For example, the designer could request that all map elevations between grayscale values 40 and 60 (= 80-120 meters elevation) be set to the selected terrain type. This is a global operation that would be performed immediately. To paint all terrain +/- 10 grayscale values around a specific map location, use the dialog to set this configuration and then return to the map to select the location or locations at which to apply the autotexture algorithm. Note that autotexture can also be set to avoid overwriting existing placed terrain tiles.

Object Editor

The Directory View, the middle window, and the Palette View, the rightmost window, are used to display all objects recognized by the game engine. Most of these objects can be placed. The Internal folder contains objects that are not usually placed, but which can still be modified to affect gameplay for the current map. All of these objects listed on the right are called template objects. There is also a special Environment object, accessed from the Environment Menu, that controls characteristics of the terrain engine and the map as a whole.

Each game object has a set of properties that describe its appearance and behavior in the game. In the FED, they can be modified on a per map basis and a per placed object basis. To change the property values for all instances of an object type in the map, edit the object template found in the Palette View. The values set here will be assigned to all objects placed on the map after the change and to all objects of this type that are spawned after the game begins. Note that objects already placed on the map will not be affected by changes to the template object, but that vehicles assigned to platoons will inherit the new characteristics if the vehicle template is changed, even if the change is made after the platoon is set to include the vehicle. All new maps have all properties initialized to Force 21 default settings.

Object variants may also be created in the Palette View. Variants are simply a convenient way to track multiple versions of an object type. If you know you will be placing several versions of an item, bring up the Object Editor for the object template in question. Then select the variant button, name the new version of the object, and alter the properties as desired. The variant will appear in the Palette View list below the object template and may be placed on the map in the usual manner. Variants are the only objects in the Palette View that can be renamed and deleted. Variants are simply a convenience for the designer and are ignored by the game engine.

In Force 21, Variants can only be used for objects which can be directly placed on the map (buildings, trees, etc). Vehicles, which are included as part of Units **cannot** have Variants.

Environment Properties

TimeOfDay
Determines how many seconds into the day the game begins - noon is 0 seconds.

DayLength
The length of a game day in seconds.

WRNoon - WBNoon
RGB color values which tint the directional light from the sun.

WRMidnight - WBMidnight
RGB color values which tint the ambient light in the environment.

SRNoon - SBNoon
RGB color values to tint the sky and fog.

CloudFile
Cloud texture filename, can be set via the New Environment Wizard.

CloudsVelX, CloudsVelY
Velocity of the cloud texture.

Water1File
Water texture filename, can be set via the new environment wizard.

WaterVelX, WaterVelY
Velocity of the water texture.

CloudsRatio
Controls the tiling of the cloud texture.

WaterRatio
Controls the tiling of the water texture.

Nearfog
Sets distance (in meters) from the camera that the fog starts.

Farfog
Sets distance (in meters) from the camera that the fog ends.

SoundLoop
Main sound loop for the game

SoundLoopVolume
Volume for main sound loop, 1.0-100.0

SoundSpot1
First random environmental sound

SoundSpotVolume1
Volume of this sound, 1.0-100.0

SoundSpot2
Second random environmental sound

SoundSpotVolume2
Volume of this sound, 1.0-100.0

SoundSpotInterval
Average time in seconds between sound spots played

SoundSpotDeviation
Variability between intervals

IsWinter
0 = fall vehicles, leafy camo
1 = winter vehicles, snow camo
2 = fall vehicles, desert camo

MiniMapFilename
The name of the minimap bitmap file. If blank, this will default to *filename*Hires.bmp when loaded into the game.

StrategicMapFilename
The name of the strategic map bitmap file. If blank, this will default to *filename*Strat.bmp when loaded into the game.

NumForces
The number of non-neutral forces in the game. Each player or AI force has its own number, with Force 0 being the neutral objects (trees, bridges, some buildings, etc).

Force1Faction - Force4Faction
The faction for the specified force. This force must be either 1 (for the US/RFS forces) or 2 (for the Chinese forces).

Force1Name - Force4Name
The name for the specified force. This is a simple string of up to 31 characters. It will be checked against the objectives string table, any matches will be expanded out to the localized version. For user defined missions, where there is no match, the string will be used as typed.

Force1Points - Force4Points
For multiplayer game play, players may choose their vehicles based on point costs. Each force has an assigned point total from which to pick, instead of using the default units assigned in the FED. Values of 0 or less mean that the force may not select units even if that option is allowed for the game as a whole. If there is no MultiplayerInsertionZone defined for the force, players will also be unable to select their choice of units. AI controlled players will never use points to make selections, and will always use the default selections.

NumSides
The number of non-neutral sides in the game. Each side will be assigned a color (Side 1: Red, Side 2: Blue, Side 3: Green, Side 4: Gold), with each force within that side given a unique shade. Forces on the same side cannot directly attack each other, and should share victory conditions. Sides cannot be changed during a mission.

Side1Forces - Side4Forces
A comma separeted list of the forces which make up a side. So to have Forces 1 and 3 be part of side 1, the Side1Forces field should be set to 1,3

MissionName
The name of the mission. This will only be used for multiplayer games, or for single player games played as scrimmages from the Multiplayer game. Like the force names, it is checked against the objectives string table for internationalization, and like the force names, if there is no match found the string will be used as entered.

IsMPMission
Non-zero if this mission should be available for selection from the Multiplayer game setup. This field is slightly

deceiving, because it is possible to create single player games (complete with AI opponents) and play them from the Multiplayer option.

MPGameType
A descriptive string (still checked against the objectives string table) to give a brief overview of what type of game the mission will be. This is completely freeform, and isn't used by the game engine in any way, except to display the string to the user.

MPForce1CanBeAI - MPForce4CanBeAI
Non-zero if the specified force can be controlled by an AI. The AI will use the default units for that force, but if there is no scripting for that force, all the units will merely remain in place and fight tactically if enemies come near. They will take no other action.

MPForce1MustBeAI- MPForce4MustBeAI
Non-zero if the specified force must be controlled by an AI. This force is now unselectable by human players.

Basic Object Properties

ObjectID
Each object is assigned a unique, positive ID. Do not change this value.

PlayerID
The army that controls the object, 0-4. 0 is reserved for neutrals.

Script
The ID of the script attached to the object, else -1. Be sure the referenced script exists.

x, y, z
The initial position of the object. The Z value is generally overridden.

Duration
Number of seconds object exists, -1 to ignore duration.

Visible Object Properties

Heading
Initial orientation of the object in degrees, 0 is East.

Scale
Scale factor for object model.

DistFromGround
Distance above the ground, if applicable

RestsOnGround
Set to 1 if object height is adjusted to the ground.

OrientOnGround
Set to 1 if object is oriented to the ground.

ResourceName
The resource name for the models for the object

Physical Object Properties

Mass
The mass of the object in kilograms.

CollideType
The collision type of the object. Valid values are:

0: No collisions (i.e. bridges, rubble)
12: Immobile (i.e. structures)
60: Mobile (i.e. vehicles)

Front_Armor
Right_Armor
Left_Armor
Rear_Armor
Top_Armor
Bottom_Armor
The armor value based on location. Armor blocks its value in damage outright, and does not ablate.

DamagePoints
How many points of damage the object can take before being destroyed

Signature
A floating point value indicating how difficult the object is to detect. The values center around 1.000, with lower values being more difficult to detect. Negative values are illegal, and should not be used.

DamageLevel
The starting damage state of the object. Valid values are:

Undamaged: 0
Damaged: 1
Destroyed: 2

Unit Object Properties

Vehicle1 - Vehicle5
The vehicle types which should be created to fill out the five units of the platoon. Unused slots are left blank. Ground units and helicopters should never be mixed in a single platoon.

Formation
The formation in which the platoon starts. The valid formations are:

Line: 1
Column: 2
Wedge: 3
Reverse Wedge: 4

ROE
The rules of engagement for the platoon. Rules of engagement values are as follows:

Fire At Will: 1
Return Fire: 2
Hold Fire: 3

Vehicle Object Properties

Weapon1 - Weapon4
The weapons on the vehicle, by name. If a weapon is unused, it is left blank.

MaxSpeed
Maximum vehicle speed in meters/second

MaxSlope
Maximum slope the vehicle can climb as a gradient.

MaxAcceleration
Maximum acceleration in meters/second^2

MaxTurn
Maximum turn rate in radians/second

Detection
The detection modifier for the vehicle. This defaults to 1.0, and a higher detection rating makes it easier to spot other vehicles. This should never be negative.

RoleFlags
The capabilities and type of the vehicle. Bridgelayers are a special case, and are not determined entirely by this field. Role flags are as follows:

1: Fighting Line Unit
2: Fire Support
4: Reconnaissance
8: Mine Layer
16: Electronic Warfare/Signal Intelligence
32: ADA
64: Artillery

PointCost
The cost to purchase the unit in multiplayer planning. If the point cost is zero or negative, the unit cannot be purchased.

Faction
The faction is used to determine the ability of a player to purchase units in multiplayer games. There is no requirement in designing missions that the factions adhered to otherwise. Valid values are:

Neutral: 0
USA/RFS: 1
China: 2

Miscellaneous Properties

Subpt1x - Subpt5y (Locations)
Subpoints that complete the spatial description of a compound object. It is best to set these with the FED compound object interface.

Elevation Maps

Force 21 uses many different elevation maps (*.tgz files), any of which can be selected for use in a new game environment created with the FED. However, it is possible to create entirely new elevation maps, allowing the design of completely new worlds.

There are four steps to creating a new elevation map.
1. Create the grayscale bitmap
2. Import the bitmap and set a water level
3. Place barriers
4. Save as *.tgz with shadow data

1. Create the grayscale bitmap

The FED uses 8 bit grayscale bitmaps to specify the basic terrain of an environment. The bitmap values are interpreted as elevations. These elevation points are spaced 16 meters apart in the game world. Thus, a 384x384 elevation bitmap would specify terrain for a world 6144 meters on a side. Each increment in the 256 color gradient is interpreted as a 2 meter height difference. Thus the maximum elevation change over an environment is 512 meters.

These bitmaps cannot be edited in the FED. Microsoft Paint (included with Windows) is the recommended application for elevation bitmap creation. It is important to smooth areas of high contrast in the bitmap so as to avoid overly rough terrain. The game engine requires elevation bitmaps to be square, in multiples of 64. Valid map sizes range from 192x192 to 1024x1024.

Next: Import the bitmap and set a water level

2. Import the bitmap and set a water level

Once an elevation bitmap is prepared, the file must be imported into the FED using the Import Elevation command from the Layer Menu. The editor will then compute geometry information for the elevation map and display the imported bitmap. This process may take several minutes depending on the size of the map and the speed of the computer.

When the elevation layer import process completes, the FED will be in elevation layer editing mode. The elevation layer tool window will be open with controls for manipulating the layer. From this dialog, the water level of the map can be set. As the water level changes, the new value will be reflected by flooding or receding waters in the map view.

Next: Place barriers

3. Place barriers

The next step required to prepare a new elevation layer for use involves the placement of barriers. These barriers are placed and edited in much the same way as compound objects and define the passability of the terrain. Left click to place subpoints, then right click to close the barrier. When vehicles move across the map, they will be forced to avoid regions marked by the elevation layer barriers. This helps ensure that units do not attempt to scale mountains or, worse, walk off a steep cliff edge. When in elevation layer edit mode, areas of the map with steep slope (greater than 45 degrees) will be painted red. These are areas that should generally be surrounded by barriers, as units are likely to have trouble traversing these reasons because of the slope.

There are actually two types of barriers, represented by different colors. Land-based barriers appear in yellow and mark areas not to be entered by units bound to the ground. In addition to blocking red zones, these barriers should mark water regions as impassible. Helicopter barriers appear in blue and mark areas not to be entered by helicopters. In addition, forests automatically generate air and land barriers (although high flying helicopters can fly over them) and don't need to have these added separetely.

There are several general rules to follow when placing and editing barriers. One barrier of each type should surround the entire play area, marking the outside limits of unit travel. This is the only barrier that is permitted to contain other barriers. Rather than placing this barrier at the very edges of the elevation layer, consider hugging the green square marking the limits of player camera movement. Units that leave the green square will be difficult to control in the game.

No barrier should ever cross itself or another barrier of the same type. The more complex the barrier system of a map, the more difficult the pathfinding computations, which can affect overall performance. See the elevation layers used by the campaign missions for examples of appropriate barrier use. Each time a barrier placement or edit operation is completed there will be a delay as pathfinding data is recomputed. If this delay becomes especially long, the map may contain an overly complex system of barriers.

Next: Save as *.tgz with shadow data

4. Save as *.tgz with shadow data

New elevation layers must be saved to *.tgz format by selecting the Save button in the elevation layer editing tool window. The new file should be saved to the \Terrain directory. Before the FED saves the layer, it will ask permission to compute shadow data. These additional computations will take several minutes, but are necessary for the elevation map to load properly in the game. When the save process completes, the new elevation file can be used by any number of maps created with the FED. Remember, however, that any computer that attempts to run GEE files using the new elevation map must have the *.tgz file in its \Terrain directory. Use the Close button in the elevation layer dialog to end layer editing and return to normal FED operation.

Back to Elevation Maps

Scripting

Once map objects are in place, custom gameplay functionality can be added with the FED scripting system. This is the primary means of implementing the opponent strategic AI and guiding overall game flow, and is the mechanism by which victory conditions are determined.

The script editing interface may **not** work at resolutions of less than 1024x768.

The FED Scripting Language

Once map objects are in place, custom gameplay functionality can be added with the FED scripting system. This is the primary means of implementing the opponent strategic AI and guiding overall game flow, and is the mechanism by which victory conditions are determined. Attaching a script to an object involves writing "handlers" that respond to event triggers. A handler is a block of text that describes actions that an object is to perform in response to a particular event. Each object may be scripted to respond to one or more game events, such as receiving damage or colliding with another object.

Scripts may be attached to templates, variants, placed objects, and the general environment object (accessed via the Environment Menu). Terrain types are the only exception - no scripts are attached to terrains. The Script Editor dialog is used to create and edit scripts attached to objects. Use the popup menus to attach a script to a selected object. Type the script text into the dialog's main edit box. Pressing the Debug button will send the text through the FED's script debugger, which will:

- Parse the script for syntax errors
- Look for the use of uninitialized local variables
- Perform type checking
- Verify function arguments
- Check for valid trigger and action symbols
- Check for redeclared variables
- Monitor internal limitations in the scripting system

The Script Editor compiles the text script into a series of virtual stack machine instructions. The script can be viewed in its compiled version by pressing View Compiled. Both text and compiled versions of the script are stored in the GEE file. The game engine will utilize the compiled form and the FED will reload the text form for future editing. If the Script Editor is exited with the OK button and no text appears in the script edit box, any previous script or script reference will be deleted. All scripts in the environment can be reviewed from the Script Information dialog, accessed via the Report Menu.

The scripting language recognized by the FED is an almost-strict subset of C syntax. Local and global variables may be declared. The only datatypes are int, float, and string, however arrays of these types are also possible. The assignment operator is the only valid string operator. The following operators can be used on int and floats.

Arithmetic: +, -, *, /
Assignment: =, +=, -=, *=
Boolean: &&, ||, ! (zero = FALSE, non-zero = TRUE)
Comparison: ==, <, >, <=, >=, !=
Increment/Decrement: ++, -- (prefix and postfix)

Built-in functions (also known as Actions) may be called. Some of these functions perform actions in the game, while others are simply helpers for the scripts (fetching information, string operations, etc.).

These functions may be called with int, float, and string arguments. They may return values of the same types. Some functions specify that they take or return bool values, these are actually declared ints which are used as booleans, where zero is false, and non-zero is true.

The language supports if-else conditionals and while loops. Global variables and user functions may be defined in a global script, which can be accessed from the Script Editor. User functions are essentially implemented as macros - they are expanded inline, not implemented with a call stack.

C and C++ style comments (// and /**/) are supported. User-defined types and pointers are not supported.

The best way to become familiarized with FED scripting is to study scripts from the Force 21 campaign. Scripting is primarily used for the following tasks.

- Monitoring victory and defeat conditions
- Displaying objectives and messages
- Enemy strategic AI and mission events

Triggers

TriggerInitialization(void)

This trigger is sent to the environment script and to all game objects (vehicles, special regions declared in the editor, platoons, etc) when the scripting is started. This occurs after vehicles have been created, so any global variables which need to be initialized before individual vehicle **TriggerSpawned** handlers (for example, determining a count for a type of unit) should be set in **TriggerBriefing**.

TriggerTick (int UnitID, int ForceID)

This trigger is sent every three seconds to each object in the game and to the environment. The environment ignores the arguments. Each object will have the UnitID for it's unit passed in (if applicable) and the force it is assigned to passed in (if applicable).

TriggerBriefing(void)

This trigger is sent just before the briefing starts to the environment and to all placed objects (including the units). It will not be sent to vehicles, because the vehicles do not exist in the game at the time this trigger is sent. The primary purpose of this is to allow the mission objectives to be initialized for appropriate display. This is the only scripting trigger which is ever processed on a client in a multiplayer game

TriggerTimer (int TimerID)

This trigger is only sent to the environment, and is sent when the specified timer expires. Timers are created with the SetTimer function.

TriggerSpawned (int UnitID, int ForceID, int ObjID)

This message is sent to each object as it is created. It includes the containing Unit ID, the Force the object is assigned to, and the ID of the object itself.

TriggerDamaged (int UnitID, int ForceID, int ObjID)

This message is sent to an object and to its unit (if applicable) when the object is damaged. The UnitID, ForceID, and ObjID represent the object in question. Some objects (such as structures) will not have a meaningful UnitID.

TriggerDestroyed (int UnitID, int ForceID, int ObjID)

This message is sent to an object if it is destroyed. The UnitID, ForceID, and ObjID represent the object in question. Some objects (such as structures) will not have a meaningful UnitID. Units will receive a TriggerDestroyed after their last unit has been destroyed, but not if the last unit is moved to another platoon.

TriggerTaskComplete (int UnitID, int ForceID)

This message is sent to a unit when it completes an assigned task, or to a CAS object when the sortie is complete. The UnitID, and ForceID represent the object in question. Note that this is sent to a unit, and not to any of the vehicles inside the unit.

TriggerAttacked (int UnitID, int ForceID, int ObjID)

This message is sent to a vehicle (and to its unit, if applicable) when it is being attacked. The UnitID, ForceID, and ObjID represent the object in question.

TriggerEnemySeen (int seeingUnitId, int seenUnitId)

This message is sent to a unit when it detects an enemy unit.

TriggerTimeExpired(void)

This trigger is sent when a multiplayer game time limit is reached. Immediately after this timer is triggered, any force which has not already won or lost will be determined to have lost the game, and the game will end. This is primarily used for "hold until time limit" missions, to determine whether or not the defender has held.

Actions and Functions for Scripting

The Force 21 script engine provides access to a number of accessor and modifier functions that allow the script access to the game core.

These functions are broken down into the following categories: Game Rules, Object Control, Orders, Information, and Utility Functions

Game Rules
int AddObjective (string objective, int ForceID)
void CompleteObjective (int objectiveID)
void ForceHasWon (int ForceID)
void ForceHasLost (int ForceID)
int GetMPTimeLimit(void)
void KillTimer (int TimerID)
void ObjectiveIncomplete (int objectiveID)
void SetCommander (int UnitID, int commanderIndex)
int SetTimer (int duration)

Object Control
void HideObject (int ObjID, bool Setting)
void Kill (int ObjID)
bool ObjectExists(int ObjID)
int Spawn (string type, int ForceID, int LocationObjID)

Orders
void MoveFollow (int UnitID, int followUnitID)
void MoveLocation (int UnitID, int locationID, float heading)
void MovePatrol (int UnitID, int pathID)
void MoveRoam (int UnitID, int locationID, float distance)
void OrderAttack (int UnitID, int targetID)
void OrderCancel (int UnitID)
void OrderCamo (int UnitID)
void OrderFortify (int UnitID)
void OrderPlaceMines (int UnitID, int locationID)
void OrderRequestArty (int UnitID, int locationID)
void OrderRequestCAS (int CASID, int locationID)
void OrderSetFormation (int UnitID, int formation)
void OrderSetHeight (int UnitID, bool high/low)
void OrderSetJamming (int UnitID)
void OrderSetRadar (int UnitID)
void OrderSetROE (int UnitID, int roe)
void SetToAvoid (int UnitID, float distance)
void SetToEngage (int UnitID)
void SetToFlee (int UnitID, float distance)
void SetToPursue (int UnitID, float distance)

Information
float DistanceToObject (int ObjID1, ObjID2)
int GetForce (int ObjID)
int GetNumPlatoons (int ForceID)
int GetNumVehicles (int ForceID)
int GetNumVehiclesInUnit (int UnitID)
string GetType (int ObjID)
int GetUnitByForce(int ForceID, int UnitIndex)

float GetXPos (int ObjID)
float GetYPos (int ObjID)
bool IsAreaOccupiedBy (int faction, float range, int location ID)
bool ObjectInKeyRadius (int ObjID, keyID)
bool ObjectInKeyRect (int ObjID, keyID)
bool ObjectInKeyRegion (int ObjID, keyID)

Utility Functions
string Append (string front, string back)
int FloatToInt (float f)
string FloatToString (float toConvert)
float IntToFloat (int i)
string IntToString (int toConvert)
void Message (int ForceID, string text)
int RandInt (int low, int high)
float RandFloat (float low, float high)

void ForceHasWon (int ForceID)

Set the victory state for the specified force (which should never be the neutral force) to victory.

In a single player game, if the player force (which is always force 1) has won or lost, the game is over.

In a multiplayer game (which includes "scrimmage" matches designed to be played by one player against the AI), the game does not end until all forces have won or lost. As a matter of design, it is recommended that for multiplayer games, victory conditions for all forces be set at one time.

void ForceHasLost (int ForceID)

Set the victory state for the specified force (which should never be the neutral force) to failure.

In a single player game, if the player force (which is always force 1) has won or lost, the game is over.

In a multiplayer game (which includes "scrimmage" matches designed to be played by one player against the AI), the game does not end until all forces have won or lost. As a matter of design, it is recommended that for multiplayer games, victory conditions for all forces be set at one time.

int SetTimer (int duration)

Set a timer of the specified duration, and return the unique timer ID which represents that timer.
Duration is specified in milliseconds.

void KillTimer (int TimerID)

Cancel the specified timer.

int AddObjective (string objective, int ForceID)

Create a new objective and assign it to the force. A unique objective ID is returned, which is used by the script engine to report on the status of an objective. Although more than three objectives can be created for a given force, the game will only display three.

The objective string is first checked against the objectives string table for a match. If one is found, the internationalized entry from the string table is used, otherwise, the objective is used as entered.

void CompleteObjective (int objectiveID)

Mark a specific objective as having been met. This will also give an audio cue on to the player controlling that force.

void ObjectiveIncomplete (int objectiveID)

Mark a specific objective as having not been met. This will not give an audio cue.

int GetMPTimeLimit(void)

Return the time limit (in seconds) for a multiplayer game. If the value returned is 0, then there is no time limit set.

void SetCommander (int UnitID, int CommanderIndex)

Assign a commander to a specific unit. The index is the index into the commander list for that faction. This does not check for duplicate use of commanders, and will not work if the unit is controlled by a human player.

For single player games (including scrimmages from the multiplayer menu), this can be set at any point. For multiplayer games, this must be set in TriggerInitialization or later. It will not work if set in TriggerBriefing.

int Spawn (string type, int ForceID, int LcationObjID)

Spawn a new object. The type is the name of the object class, the ForceID is the force it should be assigned to, and the LcationObjID is the ID of an object where it should be located. This can be a hidden object (a KeyPoint, for example) or another object already in the game. This should never be used to create a vehicle.

The return value is the ObjID of the newly created object.

bool ObjectExists(int ObjID)

Determine if the object specified still exists.

void Kill (int ObjID)

Kill the object specified.

void HideObject (int ObjID, bool Setting)

Hide (true) or display (false) the specified object. If the object is a Unit, then all the vehicles that make up that Unit will also be affected. This is the only object modifier which can be called in TriggerBriefing and should be used there to hide objects which will be brought into play later in the mission.

Hiding or revealing a Unit will also hide or reveal all vehicles which make up that unit.

Hidden objects do not move or take damage, or otherwise interact with the game world.

Player controlled units should **never** be hidden.

void OrderAttack (int UnitID, int targetID)

Designates a target for the unit, which will ignore the order if out of range. It will not move the unit to the target or perform any action other than target designation.

This is disabled automatically in multiplayer games if the unit is controlled by a player.

void OrderCancel (int UnitID)

Cancel the orders for the specified unit.

This is disabled automatically in multiplayer games if the unit is controlled by a player.

void OrderPlaceMines (int UnitID, int locationID)

Place a minefield at the location of the specified object (often a KeyPoint object).

This is not valid if sent to a unit which does not have a minelayer, and is disabled automatically in multiplayer games if the unit is controlled by a player.

void OrderRequestArty (int UnitID, int locationID)

Request an artillery barrage from a specific artillery platoon, to be targetting at the location of a specified object (this can be a Keypoint, structure, sighted vehicle, whatever).

This is disabled automatically in multiplayer games if the unit is controlled by a player.

void OrderRequestCAS (int CASID, int locationID)

Request a CAS sortie from a specific airbase aimed at the location of a specified object (this can be a Keypoint, structure, sighted vehicle, whatever).

This is disabled automatically in multiplayer games if the CAS is controlled by a player.

void OrderSetFormation (int UnitID, int formation)

Set the formation of a given unit. The valid formations are:

Line: 1
Column: 2
Wedge: 3
Reverse Wedge: 4

This is disabled automatically in multiplayer games if the unit is controlled by a player. These values may also be set in the object properties.

void OrderSetHeight (int UnitID, bool high/low)

Set the height of a given platoon of helicopters. If the boolean argument is true, they should fly high, if false, they should drop low.

This is disabled automatically in multiplayer games if the unit is controlled by a player.

void OrderSetJamming (int UnitID)

Set any Radar/Jamming vehicles in the platoon into jamming mode.

This is disabled automatically in multiplayer games if the unit is controlled by a player.

void OrderSetRadar (int UnitID)

Set any Radar/Jamming vehicles in the platoon into Radar mode.

This is disabled automatically in multiplayer games if the unit is controlled by a player.

void OrderSetROE (int UnitID, int roe)

Set the rules of engagement for the specified platoon. The rules of engagement are:

Fire At Will: 1
Return Fire: 2
Hold Fire: 3

This is disabled automatically in multiplayer games if the unit is controlled by a player. These values may also be set in the object properties.

void SetToAvoid (int UnitID, float distance)

Set the specified platoon to avoid enemies within a specified distance in meters.

This is disabled automatically in multiplayer games if the unit is controlled by a player.

void SetToEngage (int UnitID)

Set the specified platoon to engage enemies. This is the default behavior.

This is disabled automatically in multiplayer games if the unit is controlled by a player.

void SetToFlee (int UnitID, float distance)

Set the specified platoon to flee from any enemies within a specified distance in meters.

This is disabled automatically in multiplayer games if the unit is controlled by a player.

void SetToPursue (int UnitID, float distance)

Set the specified platoon to pursue any enemies within a specified distance in meters.

This is disabled automatically in multiplayer games if the unit is controlled by a player.

void MoveLocation (int UnitID, int locationID, float heading)

Order the specified platoon to move to the location of the object specified. If this object is a KeyPath, then the platoon will follow that path.

The final heading value is in degrees, with 0 degrees being defined as due East. If it is negative, then the platoon will keep the heading value it had when it arrived at its destination.

This is disabled automatically in multiplayer games if the unit is controlled by a player.

void MovePatrol (int UnitID, int pathID)

Order the specified platoon to patrol along the KeyPath object specified.

This is disabled automatically in multiplayer games if the unit is controlled by a player.

void MoveRoam (int UnitID, int locationID, float distance)

Order the specified platoon to roam in an area within a specified distance in meters from the location object (often a KeyPoint).

This is disabled automatically in multiplayer games if the unit is controlled by a player.

void MoveFollow (int UnitID, int followUnitID)

Order the specified platoon to follow the target unit.

This is disabled automatically in multiplayer games if the unit is controlled by a player.

void OrderCamo (int UnitID)

Order any vehicles in the platoon which can camoflauge themselves to do so. There is no corresponding remove camoflauge order, but ordering a platoon to move will remove the camoflauge.

This is disabled automatically in multiplayer games if the unit is controlled by a player.

void OrderFortify (int UnitID)

Order any vehicles in the platoon which can fortify to do so. There is no corresponding unfortify order, but ordering a platoon to move will remove the fortification.

This is disabled automatically in multiplayer games if the unit is controlled by a player.

float DistanceToObject (int ObjID1, ObjID2)

Return the distance between two objects in meters.

bool ObjectInKeyRadius (int ObjID, keyID)

Return whether the specified object is within the KeyRadius specified.

bool ObjectInKeyRect (int ObjID, keyID)

Return whether the specified object is within the KeyRect specified.

bool ObjectInKeyRegion (int ObjID, keyID)

Return whether the specified object is within the KeyRegion specified.

string GetType (int ObjID)

Return the type of the specified object as a string.

int GetForce (int ObjID)

Return the force which the object is part of. Force ID 0 is used for neutral objects.

float GetXPos (int ObjID)

Return the X coordinate of the object.

float GetYPos (int ObjID)

Return the Y coordinate of the object.

int GetNumPlatoons (int ForceID)

Return the number of platoons in the specified force which have live vehicles attached to them. This can be used to determine whether or not a force has been eliminated (if it has no live platoons, it has been).

int GetNumVehicles (int ForceID)

Return the total number of live vehicles in the specified force. This can be used to determine if a force has been eliminated, but GetNumPlatoons is more efficient for that task.

bool IsAreaOccupiedBy (int faction, float range, int locationID)

Return whether or not there are any live vehicles of the specified faction within the given range of the location object. This is done by faction, and not by side or force, which can be an important consideration for multiplayer scripting.

int GetNumVehiclesInUnit (int UnitID)

Return the number of live vehicles in the specified unit.

int GetUnitByForce(int ForceID, int UnitIndex)

Each force has 16 platoons. This function returns the UnitID of the nth unit of the given force. It is primarily useful in iterating through all the platoons of a given force to check for victory conditions.

void Message (int ForceID, string text)

Send a text message to the specified force. If the ForceID is set to -2, then the message will be sent to all forces.

The text is checked against the objectives string table for internationalization. If it matches, the internationalized form is used, otherwise the text string is passed along verbatim

float IntToFloat (int i)

Return the floating point form of an int.

int FloatToInt (float f)

Return the integer form of a float.

string Append (string front, string back)

Create and return a new string of *front+back*.

string IntToString (int toConvert)

Return an integer as a string.

string FloatToString (float toConvert)

Return a floating point value as a string.

int RandInt (int low, int high)

Return a random integer between low and high, inclusive.

float RandFloat (float low, float high)

Return a floating point value between low and high, inclusive.

Types of Objects and IDs

Faction	*The game faction an object or force is part of. Valid values are:* *Neutral: 0 (Neutral is not valid in all cases)* *USA / RFS: 1* *China: 2*
KeyRadius	*An invisible object which defines a centerpoint and a radius used in scripting, primarily to check victory conditions or for AI. It has no meaningful force assignment or orientation.*
KeyRect	*An invisible object which defines a rectangle used in scripting, primarily to check victory conditions or for AI. It has no meaningful force assignment or orientation*
KeyRegion	*An invisible object which defines an arbitrary polygonal region used in scripting, primarily to check victory conditions or for AI. It has no meaningful force assignment or orientation.*
KeyPath	*An invisible object which defines a path of locations used in scripting, primarily for AI purposes. It has no meaningful force assignment or orientation*
KeyPoint	*An invisible object which defines a location used in scripting, primarily for AI purposes. It has no meaningful force assignment or orientation.*
MultiplayerPlacementRadius	*An object visible in multiplayer planning, this radius defines a region in which the force at belongs to may place units during multiplayer planning. It must have a non-neutral force setting, but has no meaningful orientation.*
Object ID	*Each game object (unit, vehicle, structure, Key object, etc) has a unique ID assigned to it.*
Objective ID	*Each time an objective is created in the script engine, even if the same text is assigned to another objective, it is assigned a new unique integer identifier. This is unique within the set of objectives, but may overlap other IDs*
Platoon	*Synonymous with Unit in Force 21*
Structure	*A structure is a destroyable building or object of some kind placed onto the map. They may be assigned to a given force, or specified as neutral. The tactical AI will not fire on even enemy buildings automatically.* *Each structure has a unique object ID.*
TextPoint	*This somewhat misleadingly named object is partially visible, but doesn't use any text. If it is placed on the map, the sector it is in will be marked by highlit grid lines on the strategic map. It is used primarily to highlight mission objectives in missions.*
Timer ID	*Each time a timer is created in the script engine it is given a new*

	unique integer identifier. This is unique within the set of timers, but may overlap other IDs
Unit	*A Force 21 unit is a platoon of up to 5 vehicles. Vehicles may be assigned to and from units by the player during game play. Each Unit has a unique ID.* *Helicopters and ground units should never be in the same platoon.*
Vehicle	*Each individual combat object (tank, truck, helicopter, armored vehicle) is a Vehicle. These should never be placed on the map directly, and should only exist as parts of Units.* *Each instance of a Vehicle has a unique Object ID. Scripts may be assigned to the Vehicle templates, if this is done, then those scripts are inherited by all instances of those vehicles.*

Tips

- Use the Autotexture feature to texture your map quickly. Several "absolute" operations at different elevation ranges will creates bands of textures at the different altitudes.

- If you want to replace bitmap artwork in the game, you must either delete the *.rsb file that corresponds with the *.bmp being replaced (a new *.rsb file will be generated the next time the game runs), or use the Red Storm RSB plugins, available at the Red Storm web page.