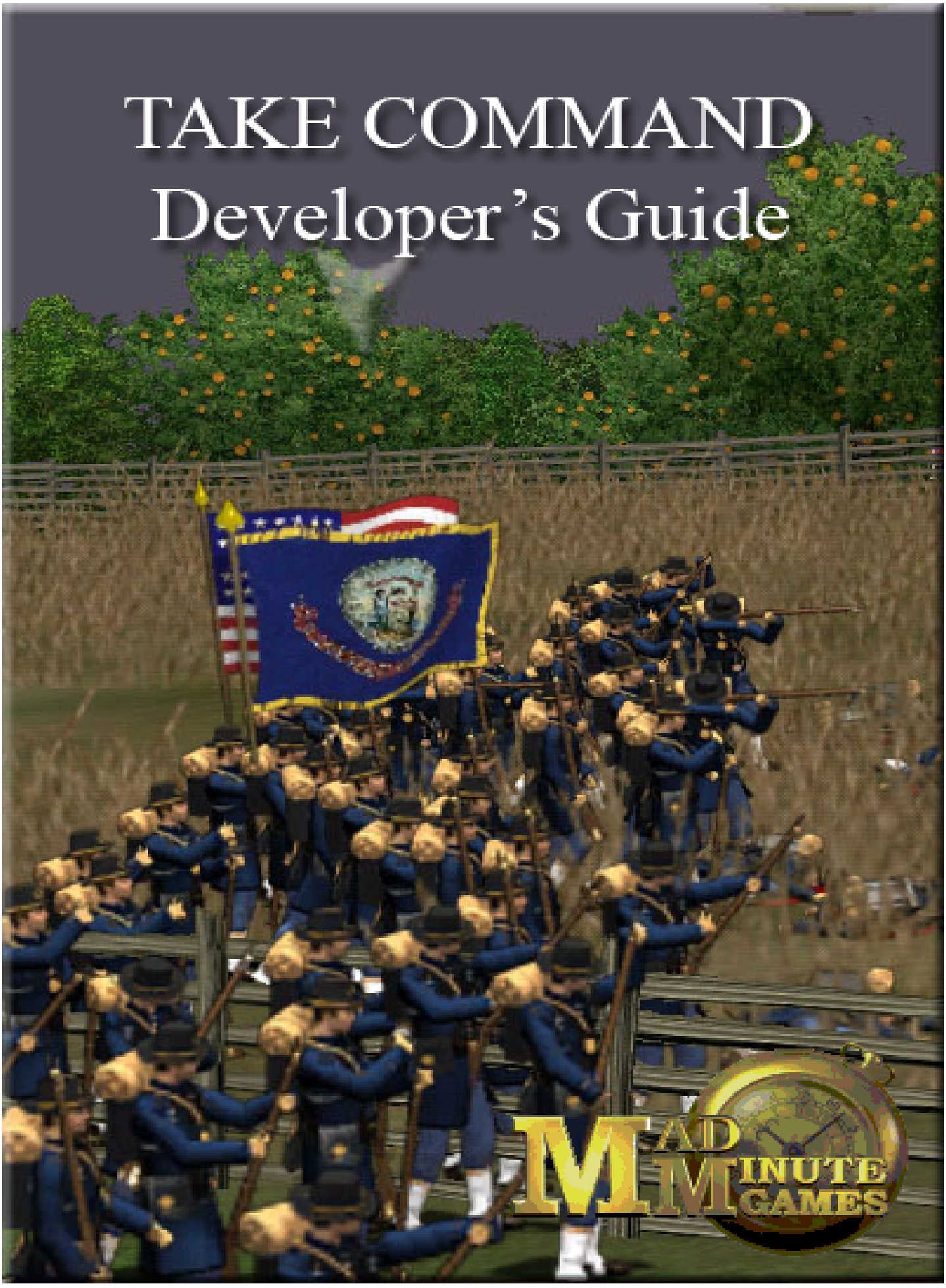
TAKE COMMAND
Developer's Guide



INSIDE FRONT COVER
PAGE INTENTIONALLY LEFT BLANK

Table of Contents

INTRODUCTION **5**

WHO AM I? **5**

MODDING TAKE COMMAND **7**

Conventions Used in this Guide 8
General Notes on File Editing 8
CSV Files: Columns, Rows, Cells, Headers, and Values 9

THE ORDER OF THINGS **10**

Folder Contents 10
Custom Scenario Folder Content 12
TC**.ini File 13
Level.ini File 14
The EndScenario Command and EndScreen Definitions 15
The Scenario Carryover Capability 16

DESIGNING AN OPEN PLAY ORDER OF BATTLE **18**

Types of Open Play OOBs 18
The Units.csv 18

DESIGNING A CUSTOM SCENARIO **29**

The Events.csv 39
The Objectives.csv 42
Intro.txt 44
Screen.txt 44
The Names.csv 45

MOD/ADD A UNIFORM **46**

The Sprites.csv 49
The Textures.csv 51
The Unitsprite.csv 51
The Unitcommon.csv 52

MOD/ADD A FLAG **54**

MOD/ADD A WEAPON **54**

The Weapons.csv 54
The Artyammo.csv 56
The Tables.csv: Artillery Section 56

MOD THE GAME INTERFACE **58**

The Toolbar.csv 58
The Tooltext.csv 60

MOD A MAP **61**

The Map Layout 61
The Map_Name.csv 61
Terrain Table Brush *62*
Terrain Table Sounds *62*

Terrain Table Objectives..... *63*
The Gamesounds.csv..... 63
MODDING THE TAKE COMMAND COMBAT MODEL..... 64
The Levels.csv..... 64
Table: Morale..... *67*
Table: Fatigue. *67*
Table: Fatigue Run. *68*
Table: Unit Morale Bonus. *68*
Table: Grades. *68*
Table: Retreat. *69*
Table: Fallback. *69*
Table: Elevation..... *69*
Table: Area Mod..... *70*
Table: Capture. *70*
Table: Artillery Accuracy. *70*
The Formations.csv..... 70
MISCELLANEOUS..... 74
The Mainscreens.csv..... 74
The Gamescreens.csv..... 75
The Mainsounds.csv..... 76
The Gamesounds.csv..... 76
The Effects.csv..... 77
USING THE WAREDIT UTILITY..... 79
USING THE WARPACK UTILITY..... 81
ON-LINE HELP..... 82
A NOTE FROM THE EDITOR..... 82
APPENDIX A: VARIABLES REFERENCE..... 83
APPENDIX B: EVENT COMMANDS REFERENCE..... 87
APPENDIX C: EVENTS.CSV EXAMPLES..... 90
INDEX..... 97

Introduction

The ***Take Command*** game series was created using MadMinute Games' ***War3D*** game engine. The ***War3D*** game engine was specifically designed to facilitate modding and scenario creation by the game owner. Included on the game CD-ROM is the ***Take Command Software Development Kit (SDK)*** folder. This folder contains all of our source csv (comma separated value) and txt (text) files that were used to build the scenario files for your particular ***Take Command*** game, and the ***WarEdit*** and ***WarPack Utilities***. With this SDK, you can modify the values in existing scenarios (troop strengths, morale, weapon ranges, etc). You can build your own historical or fictional Orders of Battle for Open Play. You can build historical and fictional scenarios from scratch. With additional third-party software, you can add or change all the artwork. You can modify the game interface or create a new one. In other words, you can do basically anything with the ***War3D*** game engine except create new game maps (though you can modify existing ones). A few variables and functions (very few) are still hard coded in the game. In future editions of the ***War3D*** game engine, more and more access to what "goes on under the hood" will be provided to the modder.

All of the technical information needed to mod the files used by the ***War3D*** game engine can be found in this guide. Many of the areas covered may be hard to understand, so if you need assistance after reading the information presented here, please visit ***The Modder's Corner*** at the MadMinute Games forum and post your questions there. You'll find many experienced modders on the forum who have already "seen the elephant".

Again, many of our design decisions for the ***War3D*** game engine were made to increase the flexibility of the player to do "his own thing". Though powerful—gaining an understanding of how the engine works can be challenging. However, the more you work with it, the easier it becomes to realize your game design goals. Start small and build on what you learn. This Developer's Guide is the first step towards designing your own orders of battle, scenarios, and mods.

Now—ask yourself the question…

Who am I?

Even though the intended audience of this guide is modders and scenario designers, there is something in it for everyone.

If you are a serious Take Command gamer but aren't interested in modding, check out the **Units.csv** section of this guide. You will find a few pieces of information there that might just enhance your game play.

If you are interested in expanding your game play options quickly and without a lot of fuss, then see the section on **Designing an Open Play Order of Battle**. This is the quickest way—for the least amount of effort—to expand the re-playability of any game in the ***Take Command*** series.

If you are interested in putting a little more time and effort into the craft of design—but want to keep the computer ju-ju down to a minimum—see the section on **Designing a Custom Scenario**.

If you are computer savvy, have an imagination, and want to push to the very limit your gaming and/or historical experiences with the Age of Linear Tactics, then **Take Command** of this Developer's Guide and start hitting the keyboard and history books! For you there is no easy button—the ***War3D*** *engine* is a set of power tools that will allow you to create—YOUR WAR, YOUR WAY! ("Har! Har! Har!…ahhhuuoo?" – Tim Allen, *Home Improvements*)

Modding Take Command

"Nothing is particularly hard if you divide it into small jobs."—Henry Ford

The SDK gives you the power to create an entirely new game. ALL of the artwork can be replaced—even the game screens and toolbars. Basically, any period in history where armies employed a regimental-based, linear combat system could be replicated using the ***War3D*** game engine. Since you can specify the size of the regiments used in the game, you could even create a powerful, single-man infantry unit (though it would still have a flag bearer). You could create a fantasy or sci-fi game—the possibilities are endless. Of course anything you create with the ***War3D*** engine must be given away for free. You cannot sell anything that you create using it, but that's what great mod communities are all about—modding and sharing more content for your favorite games.

There are some limitations of course. If we don't support the function that you need, you can't add it. For example, some people have been talking about a Napoleonic Mod. We can do almost anything with the engine, but one major item that we can't do is form a square. That's because within our existing formations, all units and sprites must face in the same direction. You could use our modding techniques to create a square formation, but once melee combat started the square would fall apart, and still—everyone would be facing in the same direction.

Now, open the SDK folder to see where all of the csv files, text files, and ***War3D Utilities*** are located. Our game directory structure was designed to support lots of experimenting with these files using the ***WarEdit*** utility—without worrying about messing up your main game files. So go ahead and give modding a shot. Read the following information and instructions and see what you can come up with. Start small to get a handle on things—and then build from there.

When you are finished building your Custom Scenarios, use the ***WarPack*** utility to package it all up. This utility will zip up all the files in your scenario folder into one compressed file—an .mmg file. This utility allows you to distribute your scenarios easily by email or from a webpage for others to play.

The ***WarEdit*** and ***WarPack*** utilities are found on your ***Take Command*** game CD-ROM in the SDK Folder.

So have fun—and show the ***Take Command*** gaming community what you can do. Good Luck!

Conventions Used in this Guide

The term **TC**.ini** is used throughout this guide to represent the main **.ini** file for any game in the ***Take Command*** game series. For example, the main **.ini** file in ***Take Command: 2nd Manassas*** is called the **TC2M.ini** file. As more games are published by MMG, the last two letters of each subsequent release will change…and keeping this guide up to date will be a little easier using this naming convention.

To help you quickly identify key terms within the document, all csv and text file names are in **BOLD** text; Commands, Variables, and Attributes are in ***BOLD ITALICS*** text.

If you see the less than, greater than symbols enclosing some text that needs to be typed, exclude the less than, greater than symbols. For example, if you see this:

<Paradox_Off>

…then type it in this way:

Paradox_Off

DESIGN TIPS: These boxes contain a description of a way to perform certain tasks. The methods they describe are optional.

NOTES: These boxes emphasize important aspects of the War3D engine. Pay close attention to what is found in these boxes.

General Notes on File Editing

What you need. The tools needed to edit the csv files in the ***Take Command*** game series are minimal. All of the files used to define a scenario are either plain text files or Comma Separated Value (csv) files. Both of these types of files can be edited with Microsoft's Notepad program. However, we highly suggest using a spreadsheet program (like Microsoft Excel) or the ***WarEdit*** utility to edit the csv files. These programs will import and export the ***Take Command*** csv files and provide a user-friendly environment in which to work on them.

DESIGN TIP: Commenting your .csv files. If you want to put a comment in one of our csv files, you must put these in the second column (or greater) so the game program doesn't read that row as data. Putting something in the first column of a row makes the program think that there is data in that row…and strange things could happen as a result.

If you want to mod or build new graphics, then you will need an image editing program that is capable of saving your artwork in either .tga file or .dds file formats. PaintShop, PhotoShop or GIMP are all logical choices; but if you are really serious about this aspect of modding—get PhotoShop and the .dds plug-in.

If you want to create new uniform, wagons, or weapon sprites from scratch, then you will need a 3D modeling and animation program such as 3DStudio Max or LightWave 3D. This is probably the most difficult task in modding the ***Take Command*** game series—but it is doable.

CSV Files: Columns, Rows, Cells, Headers, and Values

When editing .csv files it is important to understand the various terms related to them. Each ***Take Command*** .csv file has its own internal structure; but common to all is the idea of **headers, columns, rows, cells,** and **values.**

Headers. In most of our csv files (unless otherwise specified) the first **row** contains what are called, "the headers". These are short word descriptions of the data contained in any particular column. The game program does not read this row so you can change these headers as you see fit. If another name description of a data column makes more sense to you, then change it. Also if the first cell of a row is blank, then that row will be skipped (i.e., it will not be read by the game program). Unless otherwise specified in a csv file definition, you can add all the comments you want by leaving the first cell on the row blank.

Example of Terms. Let's use the **units.csv** file to see how these various terms all work together. The **units.csv** file is a list of all the Commanders and Units for the Union and the Confederacy that will be available in an Open Play OOB or Custom Scenario. It also lists the different attributes of each of these Commanders and Units. Each attribute will have its own **column**. The attribute names are the column **headers**. All the actual **values** of an attribute (header) for one Commander or Unit will be listed on one **row** in the **cells** under the appropriate column header.

Columns
(A thru G)

Rows
(1 thru 8)

	A	B	C	D	E	F	G
1	ARMY	CORPS	DIV	BRIG	REGT	ID NAME	CLASS
2	1	1	1	0	0	U_RKing	Div_BrigGen
3	1	1	1	1	0	U_JPHatch	Brig_BrigGen
4	1	1	1	1	1	U_22nd_New_York	Infantry_1
5	1	1	1	1	2	U_24th_New_York	Infantry_2
6	1	1	1	1	3	U_30th_New_York	Infantry_3
7	1	1	1	1	4	U_84th_New_York	Zouaves_5thNY
8	1	1	1	1	5	U_2nd_US_Sharpshooters	Infantry_5

To illustrate, in the **units.csv** file extract above we see seven columns and eight rows. In Row 1, we see seven column headers (***ARMY, CORPS, DIV, BRIG, REGT, ID NAME,*** and ***CLASS***). In Rows 2 thru 8, we see the actual values for the attributes of two Commanders and five Units. Row 2 lists a portion of the data relevant to Brigadier General Rufus King; reading the data cells from left to right, we see that he is in Army=1 (Union), Corps=1, Division=1. Because King is a division commander, the Brigade and Regiment columns=0. ***U_RKing*** is the scenario unique ***ID Name*** that will be used by the engine to perform various functions, and the ***CLASS Div_BrigGen*** identifies the **unitcommon.csv *CLASS*** that will be used by Brig-Gen King's sprite during game play.

NOTE: Corps=1 and Division=1 does not mean this is the 1st Division/I Corps; it means that is the first corps listed under the Union Army and the first division listed under that first corps. See the *Units.csv* section of this guide for more details.

The Order of Things

This section explains how the ***War3D*** program chooses files from the game directory structure. Let's take a look at it. If you loaded ***your Take Command game*** using the default loading options then you will find the main game directory on your C:\ hard drive. Using Explore, navigate to:

C:\Program Files\Paradox Interactive\Take Command - 2nd Manassas

Now left click on this folder and it should display the following folders:

Data Files
Graphics
- Effects
- Flags
- Misc
- Screens
- Terrain
- ToolBar
- Units

Maps
Movies
Open Play packages
Saved Games
Scenarios
screenshots
Sounds

Folder Contents

Data Files Folder. This folder is empty when you first load your game. Any .csv files that you modify and place in this folder will be used by the game engine for all Open Play and Custom Scenarios. If the Date Files folder is missing from your main game directory—then you need to add it.

Graphics Folder. The graphics folder contains seven sub-folders (labeled: **Effects, Flags, Misc, Screens, Terrain, ToolBar,** and **Units**). All the graphics files used in the used in the game are located in these folders.

Effects Folder. This folder contains the graphics files that are used to represent explosions and smoke effects during game play.

Flags Folder. This folder contains the hi-res and lo-res graphics files that represent the flags "carried" by Commanders and Units during game play.

Misc Folder. This folder contains the graphics files that represent game objective locations and the Commander and Unit wheeling indicator.

Screens Folder. This folder contains the graphics files for the main game screens.

Terrain Folder. This folder contains the graphics files that represent the various forms of vegetation and certain "cultural" entities (trees, crops, grass, graveyard stones, etc.).

ToolBar Folder. This folder contains all the graphics files that are used on the toolbar graphic. It also contains the Compass graphic used for navigation on the main game screen.

Units Folder. This folder contains all the hi-res and lo-res 2D graphics files that represent the Commanders and Units in the game.

Maps Folder. This folder contains all the map graphics files and map .csv files.

Movies Folder. This is where the movie files that shipped with the game are stored.

PLAY TIP. Tired of watching the start-up videos play? You can turn these video files "off", by either deleting them from this folder or re-naming them. For example, right click on the Paradox video file in the Movies folder. Select <**Rename**> from the pop-up window that appears and rename the file <**Paradox_OFF**>. To "turn it back on", repeat this process but rename the file <**Paradox**>. Voila! No video!

Open Play Folder. The folder contains the Open Play Orders of Battle that came with TAKE COMMAND. This is also the folder you put any OOBs that you or other modders have created for use in Open Play.

Packages Folder. Don't mess with anything in here. "These aren't the Droids you're looking for…Move along…"

Saved Games Folder. This folder is where all saved game files are stored.

Scenarios Folder. At the lowest level of the game directory structure is the **Scenarios** folder. This folder contains all the playable scenarios in the game (with the exception of Open Play scenarios). These scenarios can be either scenario .mmg files or "Custom" Scenario folders. There is no limit to the number of .mmg files or Custom Scenario folders that can be placed here.

IMPORTANT NOTE: Scenarios are loaded by the game program from either an .mmg file **OR** from a Custom Scenario folder--but never both. Any files found in an .mmg file or Custom Scenario Folder will be loaded. **These files will take precedence over any other files located within the main game program directory.**

NOTE: Even if an .mmg file and a Custom Scenario folder have the same exact name, they are considered two separate entities as far as the game program is concerned; they will **BOTH** show up in the Custom Scenarios game menu. The only .mmg files that will not show up are the ones displayed and accessed from the ***TAKE COMMAND Battle Screens***. These are hard coded and will not display in the Custom Scenarios game menu.

Screenshots Folder. This folder is where screenshots are stored. When you hit the F4 key during game play, an image is taken of the game screen and is saved in this folder.

Sounds Folder. This folder is where all sound files are stored.

Main Game Folders. At the next level up in the game directory structure are the main game folders.

IMPORTANT NOTE: If a required game file is **not** found in a scenario .mmg or Custom Scenario folder, then the game program will look for it in one of the main game folders. If the required game file is **not** found in any of the main game folders then the game program will look for it in the **TC**.mmg** file.

For example: Let's say the game program is loading a scenario folder and the **2ndM_29th.csv**. The **2ndM_29th.csv** is normally located in the Maps folder of the main game directory. First the game program looks to see if you have a Maps folder in your scenario folder. If it doesn't find one, then it looks in the main directory Maps folder. If it doesn't find one there, it looks next in the **TC**.mmg**.

Custom Scenario Folder Content

The following folders can exist within your Custom Scenario folders: **Data Files**, **Graphics**, **Maps**, **Sounds**, and **Text**. The files in these will override the files in both the main game folders and the **TC**.mmg** file: One minor exception to this rule is that the **screen.txt** file will add to and replace the strings in the **main screen.txt** file. So if there is a default string in the main file and you don't want to add it, if you do not place that string in the scenario **screen.txt** file, it will be retrieved from the main **screen.txt** file. All of the other files will completely replace their main directory equivalents.

This directory design allows you to experiment inside a scenario folder without ever having to worry about messing up your main game directory. Basically any file that you want replaced in your scenario can be placed in your scenario folder directory using the matching structure of the main directory—the file will be picked up when that scenario is selected for play. A basic Custom Scenario folder's directory (with basic required files shown) looks like this:

```
\Scenarios
    Custom Scenario Name
        Data Files
            events.csv
            objectives.csv
            units.csv
        Text
            intro.txt
            screen.txt
        level.ini
```

A fully developed Custom Scenario folder's directory (without files shown) could look like this:

```
\Scenarios
    Custom Scenario Name
        Data Files
        Graphics
            Effects
            Flags
            Terrain
            Toolbar
            Units
```

Maps
Sounds
Text

NOTE: While modding the ***War3D*** engine, you will probably encounter many crashes. This is because the modding abilities are so powerful—and can get quite complex with creative modders. We have added numerous checks in the ***War3D*** engine to help diagnose problems with user developed mods. However, we can't anticipate every potential mistake a modder might make. The crash is your friend; it tells you that you have done something incorrectly. So make small changes and check your work often until you get a feel for how everything ties together.

TC**.ini File

There is just one **TC**.ini** file for the entire game. This file might be displayed on your It can be found in the main game directory with the executable. This file is used to set many global variables in the game and is also used for storage by the dynamic variable system.

File entry	Explanation
[Initialization]	Section header for initial part of the file
Full Screen=0	***TAKE COMMAND*** actually will work in windowed mode, it will show up in the upper left corner of the screen and is not movable. This is a great way to work during development of scenarios. Set this to 1 to activate windowed mode; 0 to deactivate windowed mode.
DbgLvl=0	By turning on the debug level you turn on some of the information that we use to develop the game. The first number is the number of frames per second, followed by the number of AI calls in the last frames with the maximum reached in parentheses, next is the location of the camera and then how many sprites were drawn the last frame with the total that could have been drawn in parentheses. You will also see the coordinates of the currently selected unit. Set this to 1 to activate; 0 to deactivate.
LoseFocus=1	This determines if the game pauses when it loses focus, really only useful if you're running in windowed mode. Set this to 1 to activate; 0 to deactivate.
NoAI=0	Turns off all of the AI in the game. No until will fire. This is useful for scenario creation. Because once you have everyone where you want them to start and facing the right direction, you can then hit the **'L'** key to dump their locations into **unitlocs.csv** in the main folder. You can paste the columns from here right into your OB. Set this to 1 to activate AI; 0 to deactivate AI.
Alpha Omega=1	This capability allows you to display all units for both sides. This is a useful tool during scenario creation. Set this to 1 to activate; 0 to deactivate.
[Variables]	This section stores all of the dynamic variables used in the game.

DESIGN TIP: Once you have your TC**.ini set-up the way you like it, you can "turn off" an initialization command by typing a colon <;> in front of it

Level.ini File

The **level.ini** file must exist for every Open Play and Custom Scenario. It defines game options that can only be used at the scenario level. If you are designing a Custom Scenario, this file must be created in your Custom Scenario folder. Open Play level.ini files are created automatically when you generate an Open Play Scenario. An Open Play level.ini file can be found in the Open Play folder of the main game directory.

File entry	Explanation
[Init]	Section header for initial part of the file
map=2ndM_28th.lsl	Name of the map file in the **/maps** directory.
Weather=7	Sets the visibility and weather effects for the scenario: 0=very low visibility (300 yards), 8=as clear as it gets (1,000 yards). See the **levels.csv** (rows 1-25) for more details.
TimeOfDay=1	This sets the color of the sky in the scenario. 0=Day, 1=Dawn/Dusk, and 2=Night.
StrategicAI=0	If you set this to **1**, the scenario will run like Open Play with the AI commanders implementing a strategy to take over the objectives. This would be useful if you wanted to write a scenario that just defined the starting positions and objectives. It makes sense to use this capability with the Commander selection capability defined below (see Rank).
Carryover=1	The ***Carryover=1*** command enables a scenario to act as the source of **units.csv** data for another scenario. What "carryover" means is the end strengths of a scenario will be "carried over" and become the **beginning strengths** for another scenario that contains some or all of the same Commanders/Units. This carryover applies to both sides—Union and Confederate. For example: The ***TAKE COMMAND*** scenario **SM03 - 28 Aug - King (U_Div)** has the file entry ***Carryover=1*** in its **level.ini** file. This enables the SM03 scenario to act as the **source** of data for the carryover scenario's **units.csv**.
CarryOverFrom= <Scenario Name>	The ***CarryOverFrom=*** command enables a scenario to act as the **recipient** of **units.csv** data from another scenario. For example: In the **level.ini** file of the scenario **SM17 - 29 Aug - Hatch (U_Div)**, we see the file entry ***CarryOverFrom=SM03 - 28 Aug - King (U_Div)***. Together, the effects of the ***Carryover=*** and ***CarryOverFrom=*** commands together mean the player must complete the SM03 scenario before he can play the SM17 scenario. The strengths of Union and Confederate units that are in both scenarios will be carried over from SM03 to SM17. See *The Scenario Carryover Capability* section of this guide for more carryover commands.
StartTime=09:15:00	Specifies the time of day that the scenario starts. Use military time (24-hour clock). Example: 1 P.M. = 13:00:00
CommandHeight=40	Limit of camera height above the ground in yards. If you do not include this, the player will be able to move the camera very high off the ground.
CommandRadius=500	Limit the camera can move away from the player's character. This distance is in yards. If you do not include this, the player can move the camera anywhere on the game map.
CantKillMe=1	Prevents player's character from being killed in the scenario. If you use this option the player's character will never have an **evtdeath** event.
[Rank]	This section sets the player's character in the scenario. The numbers refer to the same numbering system used in the file **units.csv** file (columns A-E). If all of these values are set at **0**, then the player will be able to choose to play any Commander displayed in the Custom Scenario game screen.
Army=1	**ARMY** from column A of the **units.csv**
Corps=1	**CORPS** from column B of the **units.csv**
Division=2	**DIV** from column C of the **units.csv**
Brigade=2	**BRIG** from column D of the **units.csv**
Regiment=0	**REGT** from column E of the **units.csv**
This section shows an example of all the endscreen definitions for one Custom Scenario's level.ini file. See *The EndScenario Command and EndScreen Definitions* section of this guide for a detailed explanation pf endscreen	

definitions. Also, study the level-ini files in the SDK Scenarios folder for other examples.

[EndPlayerDie]	**[EndPlayerDie]** **screen=End_Union** **headline=End_U_Tragedy.dds** **article=EndKilled**
[EndMajWin]	**[EndMajWin]** **screen=End_Union** **grade=320** **headline=End_U_MajVictory.dds** **article=EndMajorSuccess** **CarryOver=1**
[EndWin]	**[EndWin]** **screen=End_Union** **grade=240** **headline=End_U_Victory.dds** **article=EndMinorSuccess** **CarryOver=1**
[EndTie]	**[EndTie]** **screen=End_Union** **grade=160** **headline=End_U_Victory.dds** **article=EndDraw**
[EndFail]	**[EndFail]** **screen=End_Union** **grade=80** **headline=End_U_Defeat.dds** **article=EndMinorFail**
[EndMajFail]	**[EndMajFail]** **screen=End_Union** **grade=0** **headline=End_U_MajDefeat.dds** **article=EndMajorFail**

The EndScenario Command and EndScreen Definitions

endscenario. This is a new command and it is now the correct way to end a Custom Scenario. You should call this command on the **events.csv** whenever a scenario is supposed to end. We no longer call ***loadscreen*** to end the scenario.

There are two ways to call ***endscenario***. One way is to use the command by itself; the other way is to follow it by a specific screen definition name (see below). If you call ***endscenario*** by itself, it will search the screen definitions in the scenario's **level.ini** file to find the appropriate end screen for the ***grade*** the player has achieved when the ***endscenario*** command was triggered. Since ***endscenario*** must always be called, you must also put the ***carryover=*** flag in the screen definition.

Screen Definitions. These are found in each scenario's **level.ini** file. There are no default settings —you must have them. They are in .ini format. Only one screen can be defined per heading. The ***screen=*** is the name of the specific screen defined in the **gamescreens.csv** file. Currently there are two screens defines—***End_Union*** and ***End_Confed***. These just show the newspapers for the different sides. The ***grade=*** is the grade that this screen will be used for. If the player achieves a grade that is greater than or equal to the ***grade=***, then the defined screen will be displayed.

The lowest grade defined in the file will be used for all lower grades. The ***carryover=1*** means that if this screen is displayed, the carryover .**mmc** file will be created. All other **=** are variables from the

screen in the **gamescreens.csv** file. Currently we have two, a headline for the headline graphic, and a newspaper article. This entry is the name of the ***textid*** from **screen.txt** file that will show up as the article. Using this method, any number of variables can be added to the screen and defined here. Only include the grade line if you want this screen used in the default grade screen searching process. Do not include it if you only want this screen to show when you call it specifically. Example:

```
[EndMajWin]
screen=End_Union
grade=300
headline=End_U_MajVictory.dds
article=EndObjs
CarryOver=1
```

The Scenario Carryover Capability

In this section, we will take a look at the underlying concepts for the scenario carryover capability. In the examples below, we will use two hypothetical scenario names—Scen1 and Scen2—to act as our examples. Scen1 is the first scenario to be played. Once Scen1 has been played to completion, then the values from Scen1 will be carried over to Scen2.

- Units are found by ***name*** and ***name2***. The **units.csv** file from Scen2 is loaded and then the **units.csv** file of Scen1 is loaded. The game program then searches Scen2 for the same units in Scen1 (by name). If a match is found, then the game program transfers the appropriate data. So not every unit in Scen1 has to be in Scen2, and Scen2 can include units that were not in Scen1. This comparison is case insensitive.

- Scen2 REQUIRES that Scen1 has been completed before it will run. In Scen1, the designer must call the command ***endscenario*** as an entry in the **events.csv** file using the criteria they feel the player must pass to successfully complete the scenario—and thus be allowed to continue to Scen2. This will create a file <**scen1.mmc**>. This file will be created in the **saved games\cont** folder. If this file is not there, then any scenarios that depend on Scen1 being complete will not be able to load.

- The difficulty level for Scen2 will be the same as that set at the beginning of Scen1.

- Guns captured in Scen1 will be completely gone from Scen2. In fact, anything captured in Scen1 is completely gone from Scen2. This includes captured infantry and cavalry Units—consider them on the way to prison camps.

- **Events.csv** (Scen1)
  - **endscenario** – This command must be called via the events.csv to create the pseudo saved game file that is required for carryover from Scen1.

- **Level.ini** (Scen1)
  - ***CarryOver=1***
    - This command says that if ***endscenario*** is called, then create the .**mmc** file.

- **Level.ini** (Scen2)
  - ***CarryOverFrom=***Scen1
    - This says that the Scen2 scenario requires the Scen1.mmc file to begin, and will use the info in that file for carryover.

- ***CarryOverMorale=1***
  - If the value of ***CarryOverMorale=1,*** then Scen2 will use the morale levels from the carryover file of Scen1. If the value of ***CarryOverMorale=0,*** then Scen2 will use the morale levels from Scen2 **units.csv**.
- ***CarryOverMoraleValue=-1***
  - This number will be added to the morale levels of all Units in Scen2. It will be a level, not points. So if you enter the value as **-1**, then all Units in Scen2 will lose one morale level. If you enter a value of **0**, then their morale levels stay the same. A value of **1** will cause all Units to gain one morale level. The program is designed to make sure that Units do not go below the minimum morale level or above the maximum morale level.
- ***CarryOverFatigue=1***
  - If the value of ***CarryOverFatigue=1***, then the fatigue levels for all Units will carry over from Scen1 to Scen2. If the value of ***CarryOverFatigue=0***, then the fatigue levels from Scen2 will be used.
- ***CarryOverFatigueValue=-1***
  - This number will be added to the fatigue levels of all Units in Scen2. It will be a level, not points. So if you enter the value as **-1**, then all Units in Scen2 will lose one fatigue level. If you enter the value as **0**, then their fatigue levels stay the same. A value of **1** will cause all Units to gain 1 fatigue level. The program is designed to make sure that Units do not go below the minimum fatigue level nor above the maximum fatigue level.
- ***CarryOverDead=1***
  - A value of 1 means that the program will restore all dead body sprites to the map. A value of 0 means that it will not.
- ***CarryOverCasPct=50***
  - This is the percent of casualties that will be restored upon carryover. Casualties are defined as the number of men alive that you started with minus the number of men alive that you ended with. Do not use decimals or percent signs here. The above value means 50 percent.

Designing an Open Play Order of Battle

The easiest mod to do with the quickest and highest personal payoff is to design an Open Play Order of Battle (OOB). The only file you need to build for this type of mod is the units.csv file. When complete, save this **units.csv** file as an **OB_<name>.csv** file; put it in your Open Play folder—and play! This section describes what you need to do to build your own OOBs.

Types of Open Play OOBs

There are at least three types off OOBs that you can build for Open Play. These are as follows:

Historical OOBs. These OOBs are a true reflection of whatever history recorded them as. When designing these, there is no consideration for "fair play" or "play balance" or of the extreme possibilities that can occur when the Open Play scenario routine randomly selects units. War is all hell. If you can't hack it —pack it.

Competitive OOBs. These OOBs are built specifically to insure a balanced, "fair fight", regardless of the Commander or Command Level selected in Open Play. These will pit the player against the AI in an "all things being equal" kind of scrap. Names of Commanders and Units are not relevant—pick whatever. An equality of organizations, strengths, and similar Commander and Unit attributes is the goal for a competitive OOB.

Hybrid OOBs. These OOBs are a cross between the Historical and Competitive types of Open Play OOBs. Start with an Historical OOB. Identify and eliminate any potential extreme combinations of Union and Confederate forces. Consider adjusting strengths or minor organizational changes to "balance" things out. The goal is to maintain the flavor of the Historical OOB while reducing the chances of an "impossible" situation being generated in Open Play.

The Units.csv

The **units.csv** file provides the Order of Battle (OOB) for ***Take Command***. This file lists all of the Commanders and Units that will be available for play in either a Custom Scenario or for Open Play. Each Custom Scenario has a **units.c**sv file that is used specifically for that scenario. This file is located in that Custom Scenario's .mmg file and/or scenario folder. There are also three **units.csv** files located in the Open Play folder of the main game directory that are used for Open Play.

NOTE: A units.csv file used as an Open Play OOB must be named and saved using the following convention: **OB_<name>**. For example: **OB_2ndM**. If the **OB_** prefix is not used, then the OOB file will not show up in the Open Play game screen OOB window. Though it has a different file name, an **OB_<name>**.csv has the same content as a **units.csv** file.

IMPORTANT NOTES:

- All headings for the columns are in the first row of the **units.csv**.
- You must list commanders and units in numerical sequence when building the army structure; you cannot skip numbers.
- Supply wagon units must be entered as the last row of a division.
- A side's courier unit must be entered as the last row of an army or the game will crash to the desktop (CTD). As an example, check out the **units.csv** files contained in the SDK to see how the last rows of both armies look.

Column	Explanation
A: Army	This value must equal 1 or 2. 1 is for the Union Army, and 2 is for Confederate Army. The first five columns designate who the unit is. They must be in order and no number(s) can be skipped. These columns are used to designate the command structure of the army. The structure MUST be correct and in order or the game will crash when the AI starts to issue commands. There can be only two armies. No gaps in #s...all values must be consecutive.
B: Corps	The Corps ID number.
C: Div	The Division ID number.
D: Brigade	The Brigade ID number.
E: Regiment	The Regiment ID number. If you want to have companies instead of regiments, then this would be the company ID number.
F: ID Name	This acts as a nickname for the unit and allows the scenario designer to name them specifically for use in a scenario. Whenever this unit is referenced in another csv file, they will use this ID. This is especially used in the **events.csv** file.
G: Class	ID name from Column A of the **unitcommon.csv** file. Each unit must be associated with one and only one class from the **unitcommon.csv** file. This will define attributes such as graphics and sounds.
H: Weapon	This column references the **Weapons.csv** file and is the weapon assigned to the unit.
I: Ammo	Specify the ammo levels of the specific unit type. This is the amount of ammo that **EACH SOLDIER** in the regiment will have. So multiply this number times the regiment's strength to see how much the entire regiment is carrying. For artillery, this amount is also per soldier, so for a 15 man gun crew, multiply this value by 15 to calculate the total ammo available for the gun.
J & K: Dir X/Dir Z	This is a normalized 2D vector that determines the facing of the unit when the scenario starts. This is pretty hard to compute, so we recommend that you use our automated process. For full details please see the help on **TC**.ini** and the ***NoAI*** flag. By hitting the "**L**" key while in game…the game will spit out a file into the main folder called **unitlocs.csv**…use this file to set positions for Custom Scenarios.
L & M: loc X/loc Z	This is the 2D coordinates of this unit's flag bearer on the map. The valid values are from 0.0 - 131072.0. Though we recommend not getting too close the edge of the map. As above, this is much easier to set using the **NoAI** process.
N: Flags	Specifies the flag graphic, from **sprites.csv**, of the specific unit.
O: Formation	Specifies the starting formation, from **formation.csv**, of the specific unit for the scenario.
P: Name	The name of this unit. Stored in the **$name** variable.

Q: Name2	The 2nd name for this unit. Stored in the **$name2** variable. Also known as ***2nd Line*** on the **units.csv** file.
R: Name3	The 3rd name for this unit. Stored in the **$name3** variable. Also known as ***3nd Line*** on the **units.csv** file.
S: Strength	This is the number of men in the unit. Note that we always create a flag bearer, so he does not count in this number. Also we create one sprite for every ten men. We show a max of 100 sprites per unit. So a unit will be full with 990 men, since one sprite is used for the flag bearer. If there are more men, they will be distributed evenly among all the sprites of the unit. For units with one man (ammo wagons and commanders), you can specify anywhere between 1-10 men and still only have your one sprite plus the flag bearer. The number of men does not matter for commanders and wagons. Artillery batteries have 15 men per gun.
UNIT TRAINING	The rest of the columns have to match exactly the ROWS from the **levels.csv** file. Starting after weather, each row of the **levels.csv** file is matched here in the **units.csv** file. This is important because you CAN add more rows to the **levels.csv** file and match them in the units.csv file, and it will work. You can add your own skills to the game. Refer to the **levels.csv** file section to see what the min and max values can be for each different attribute.
T-X: Commander Attributes	These columns only affect commanders. They are explained in detail in the **levels.csv** file section.
Y: Experience (Quality)	This column affects both commanders and fighting men. It's the amount of battle experience that they have. This is explained in more detail in the **levels.csv** file section.
Z-?: Regiment Attributes	The rest of the columns affect only the fighting men, be they infantry, cavalry, or artillery. They are explained in more depth in the **levels.csv** file. Note that the morale and fatigue columns affect many items in the game that may not be mentioned in the **levels.csv** file. This is because they are such a core part of the AI and fighting in ***TAKE COMMAND***.

Column F: ID NAME. Column F is the Commander or Unit ID NAME. In the extract above, we see that the data entry for King is ***U_RKing***. You can use whatever naming convention you like in these cells but there is one important rule to keep in mind: The names must be unique. In other words, no other Commander or Unit can share this name. So...one unit, one name--and **nobody** else can have this same name on your Open Play **OB_** or **units.csv** file.

Naming Conventions for ID NAMEs. Here are some examples of naming conventions you might use. If you don't like these, you can make up your own—just remember the rules:

Commanders	Units
RufusKing	**22ndNewYork**
AoV_RufusKing	**AoV_22ndNewYork**
IIICorps_RKing	**IIICorps_22ndNewYork**
III_1_AoV_CDR	**III_1_AOV_22NY**
RKing_U	**22ndNewYork_U**
King1	**22NY**

Column G: CLASS. These CLASS names come from the **unitcommon.csv** file. If you want to create a new class, it must be defined on the **unitcommon.csv** file before it can be used on the **units.csv** file.

Column H: WEAPON. These Weapon names come from the **weapons.csv** file. If you want to create a new weapon, it must be defined on the **weapons.csv** file before it can be used on the **units.csv** file.

Column I: AMMO. You can put any number you want in these cells but here are some rules of thumb.

Infantry. A good rule of thumb for assigning ammunition to the infantry is 40 rounds per man--that's what a soldier could carry in his ammo pouch. This was the standard issue while on the march because of the inevitable wastage due to exposure to rain, stream crossings, morning dew, humidity, etc. Plus soldiers didn't like to carry anything more than what they thought they should have to carry (many an "extra load" of ammo got tossed into the weeds if it lightened the load a bit).

If battle was imminent--and time and ammo was available to do so--it was not uncommon for soldiers to be issued 60-100 rounds each. However, this additional ammo was often carried in a soldier's pockets or backpack. It was very common to ground the backpack (take it off) before entering combat. This allowed soldiers more freedom of movement while loading and firing, increased their agility for moving quickly on the battlefield, and reduced the chances of fatigue. It also meant being away from your extra ammo.

Consult the historical records and personal accounts if you want more precision to the ammo count--you will often find references to how much ammo was actually carried in any particular battle. Lastly, rifle-musket rounds came in packages of 10. Use this number as a guide for increases/decreases of ammo carried by the infantry. Barring any information you might find to the contrary, **40 rounds per man** may be as accurate a figure as you can get your hands on.

Cavalry. The rule of thumb for cavalry is 40-100 rounds per man. As a portion of this ammo was carried in the saddlebags—which will be with the horse if the trooper dismounts—40-60 rounds per cavalryman is about right.

Artillery. Here are some rules of thumb for artillery ammo—these are maximum ammo loads.

Artillery Piece	# of Rounds
6 lb Model 1841	13
6 lb Wiard	13
6 lb Model 1857	13
10 lb Parrott	8
12 lb Napoleon	10
20 lb Parrott	8
30 lb Parrott	6
3 in Ordnance	8
12 lb James	13
14 lb James	8
24 lb James	8
12 lb Whitworth	13
12 lb Armstrong	13
12 lb Blakely	13
12 lb Clay	13
12 lb Dahlgren (Howitzer)	16

Wagons. It is a rare occasion that a wagon will run out of ammo, but if you want to have the right amount represented in the game, here are some general rules.

Infantry and Cavalry. A good rule of thumb for small arms ammunition is 100 rounds per man is carried in the wagon that supports a particular organization (normally a division). To calculate this, total the number of infantry and cavalry **sprites** in the organization and multiply this number by 100

(or by whatever number you have decided your basic load is going to be). Enter this number in the ammo cell for that organization's wagon.

Artillery. A good rule of thumb for artillery ammunition is the # of rounds from the table above per gun in the wagon for the organization it is supporting. Add up all the ammunition being carried by the artillery batteries of an organization and add that amount to and organization's wagon.

Columns J, K, L, & M: Dir/Loc. Where am I at? Which way am I pointed? As we are designing an Open Play OOB, we don't have to worry about these questions--positioning of Commanders and Units will be determined by the game program when you start an Open Play scenario.

What to do? Enter 1, 0, 1, 1 under ***dir x***, ***dir z***, ***loc x***, and ***loc z*** on your **units.csv** file for ALL Commanders and Units in your Open Play OOB. For example:

dir x	dir z	loc x	loc z
1	0	1	1

Column N: Flags.

Column O: Formation.

Column P, Q, & R: Names.

Columns S thru Y: Commander Attributes. The range of legal values listed below must be used in defining Commander Attributes. Failure to do so could result in a crash to desk top (CTD).

Initiative. A Commander's initiative rating affects the rallying times of "broken" Units. AFFECTED BY: Completing orders on time. There seven legal values for the Initiative rating. They are 0 through 6. Here is what they mean:

Initiative Ratings	Initiative Value	Rally Points	Commanders run or walk to "Support" units (summed up - 0 or greater they run)
Poor	0	10	-3
Mediocre	1	20	-2
Average	2	30	-1
Competent	3	40	0
Good	4	50	1
Very Good	5	60	2
Excellent	6	70	3

Leadership. The base Command Radii are measured in yards and are HARDCODED in the game. The actual Command Radius assigned to a commander is based on the unit level he commands. The hardcoded radii are as follows:

Level of Command	Command Radius (Hardcoded)
Brigade	10 yards
Division	25 yards
Corps	50 yards
Army	65 yards

A Commander's leadership rating influences the size/area of his command radius. AFFECTED BY: Combined battle grades of all subordinate Units. There seven legal values for the Leadership rating. They are 0 through 6. Here is what they mean:

Leadership Ratings	Leadership Value	Command Radius Modifier	Commander Bonus	Commander "Support" Distance	Commanders run or walk to "Support" units (summed up- 0 or greater they run)
Poor	0	-50	0	0	-3
Mediocre	1	0	10	-5	-2
Average	2	50	20	-7	-1
Competent	3	100	30	-10	0
Good	4	150	40	10	1
Very Good	5	200	50	7	2
Excellent	6	250	60	5	3

Loyalty. A Commander's loyalty rating reduces the number of deserters from player's routed Units. AFFECTED BY: Total kill ratio of subordinate Units. There seven legal values for the Loyalty rating. They are 0 through 6. Here is what they mean:

Loyalty Ratings	Loyalty Value	Command Radius Modifier	Rally Time (in seconds)	Commanders run or walk to "Support" units (summed up - 0 or greater they run)	Regts max morale modifier (leaders "loyalty")
Poor	0	-50	105	-3	0
Mediocre	1	0	90	-2	0
Average	2	50	75	-1	20
Competent	3	100	60	0	40
Good	4	150	45	1	60
Very Good	5	200	30	2	80
Excellent	6	250	15	3	100

Ability. A Commander's ability rating determines the amount of morale bonus given to a Commander's subordinate units. AFFECTED BY: Completing orders/mission goals. There seven legal values for the Ability rating. They are 0 through 6. Here is what they mean:

Ability Ratings	Ability Value	Commander Bonus	Commanders run or walk to "Support" units (summed up - 0 or greater they run)
Poor	0	10	-3
Mediocre	1	20	-2
Average	2	30	-1
Competent	3	40	0
Good	4	50	1
Very Good	5	60	2
Excellent	6	70	3

Style. A Commander's style rating determines...random behavior tendencies. There five legal values for the Style rating. They are 0 through 4. Here is what they mean:

Style Rating	Style Value	Personality	Commanders run or walk to "Support" units (summed up - 0 or greater they run)	Max ammo per/man from ammo wagon (in rounds)
Cautious	0	-100	0	50
Defensive	1	-50	0	55
Balanced	2	0	1	55
Bold	3	150	1	60
Daring	4	200	2	60

Quality. A Commander's quality rating influences the size/area of his command radius. There eight legal values for the Quality rating. They are 0 through 7. Here is what they mean:

Quality Rating	Quality Value	Command Radius Modifier	Commanders run or walk to "Support" units (summed up - 0 or greater they run)	Player's Commander runs or walks to "Support" units (summed up - 0 or greater they run)
Green	0	-100	4	0
Trained	1	-50	3	0
Fair	2	0	2	1
Regular	3	50	1	1
Good	4	100	0	3
Veteran	5	150	-1	3
Elite	6	200	-2	7
Crack	7	250	-3	7

Columns Y thru AG: Unit Attributes. The range of legal values listed below must be used in defining Unit Attributes. Failure to do so could result in a crash to desk top (CTD).

Quality. Also known as Experience.

Quality Rating	Quality Value	Firing (1,000 pts)	Misfires (1,000 pts)	Melee (1,000 pts)	Wheeling Locked (distance in yards)	Units run or walk to ammo wagon (summed up - 0 or greater they run)	Ammo taken from dead	Unit Morale Bonus Radius (yards)	Firing Time (Rate of Fire)	Cover Stand Time
Green	0	0	0	0	140	-5	0	100	20	20
Trained	1	5	5	5	125	-4	0	100	10	10
Fair	2	10	10	10	110	-3	0	100	5	5
Regular	3	15	15	15	100	-2	0	120	0	0
Good	4	20	20	20	90	-1	0	120	-5	-5
Veteran	5	25	25	25	80	0	0	120	-10	-10
Elite	6	30	30	30	70	1	0	150	-15	-15
Crack	7	35	35	35	60	2	0	150	-20	-20

Fatigue.

Fatigue Rating	Fatigue Value	Firing (1,000 pts)	Misfires (1,000 pts)	Melee (1,000 pts)	Units run or walk to ammo wagon (summed up - 0 or greater they run)
Exhausted	0	-25	-10	0	-9
Weary	1	-20	0	5	-7
Tired	2	-10	0	10	-5
Winded	3	0	0	15	-3
Okay	4	0	0	20	-1
Fresh	5	0	0	35	0
Rested	6	0	0	50	3

Morale.

Morale Rating	Morale Value	Melee (1,000 pts)	Units run or walk to ammo wagon (summed up - 0 or greater they run)
Routed	0	0	-6
Broken	1	0	-5
Panicked	2	0	-4
Wavering	3	0	-3
Shaken	4	0	-2
Uneasy	5	10	-1
Willing	6	30	0
Good	7	40	1
Confident	8	50	2
High	9	70	3

Accuracy. Number is added to the base value (# in red row in "Weapons" file)….random # is selected & if same or higher…then a hit is scored.

Accuracy Rating	Accuracy Value	Firing (1,000 pts)
Untrained	0	0
Familiar	1	10
Qualified	2	15
Competent	3	20
Skilled	4	25
Expert	5	30
Specialist	6	35

Loading. Number is added to the base value in the misfire column ("Weapons" file)….random # is selected & if same or higher…then the shot is MADE.

Loading Rating	Loading Value	Misfires (1,000 pts)
Untrained	0	0
Familiar	1	10
Qualified	2	15
Competent	3	20
Skilled	4	25
Expert	5	30
Specialist	6	35

Formation (Drill). Increases speed of wheeling and formation changes.

Drill Rating	Drill Value	Limber Time	Cover Stand Time
Untrained	0	0	0
Familiar	1	0	-3
Qualified	2	-4	-4
Competent	3	-5	-5
Skilled	4	-10	-10
Expert	5	-15	-15
Specialist	6	-20	-20

Bayonet Drill. Increases hit % during melee.

Bayonet Drill Rating	Bayonet Drill Value	Melee (1,000 pts)
Untrained	0	0
Familiar	1	25
Qualified	2	50
Competent	3	75
Skilled	4	100
Expert	5	125
Specialist	6	150

First Aid (Medical). Returns higher % of wounded troops to active duty after the battle. This attribute is NOT IMPLEMENTED IN THE TAKE COMMAND GAME SYSTEM YET.

First Aid Rating	First Aid Value	First Aid Modifier
Ignorant	0	-2
Familiar	1	-1
Informed	2	0
Educated	3	1
Knowledgeable	4	2

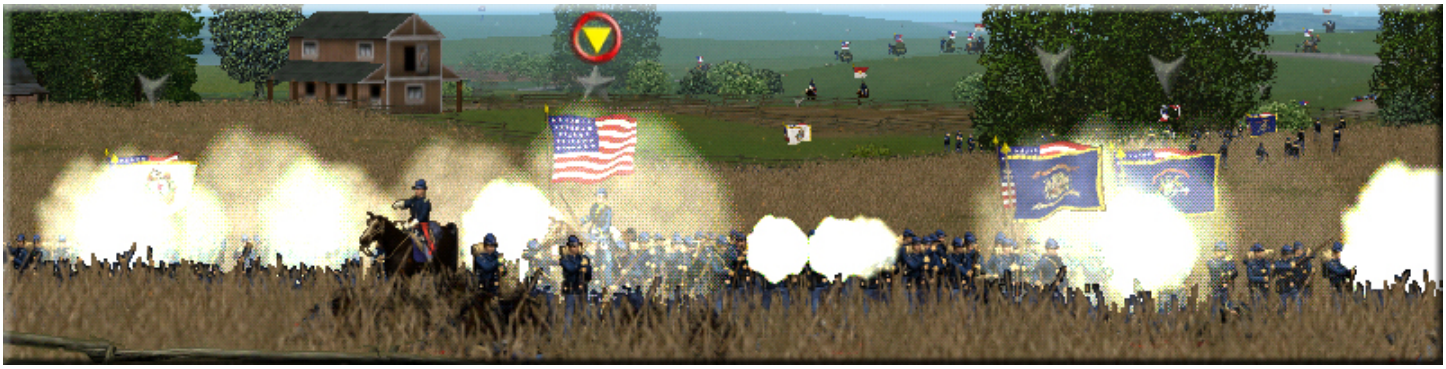
Breastworks. Speeds a unit's ability to make entrenchments/breastworks/defenses. This attribute is NOT IMPLEMENTED IN THE TAKE COMMAND GAME SYSTEM YET.

Breastworks Rating	Breastworks Value	Breastworks Modifier
Untrained	**0**	**-**
Familiar	**1**	**-**
Qualified	**2**	**-**
Competent	**3**	**-**
Skilled	**4**	**-**
Expert	**5**	**-**
Specialist	**6**	**-**

Unit Attribute Values by Year. The following table is a quick reference guide for assigning attribute values to units. This is intended for OOB and scenario designers who don't have the time or resources to conduct extensive historical research and analysis to make the subjective judgments required for the assignment of attribute values to units in their OOBs.

Attributes	Values	Raw	Green	Trained	Fair	Average	Good	Veteran	Elite	Crack
Quality	0-8	0	1	2	3	4	5	6	7	8
Fatigue	0-6	Rested	Rested	Rested	Rested	Rested	Rested	Rested	Rested	Rested
Morale	0-10	Good	Good	Good	Good	Good	Good	Good	Good	Good
Accuracy	0-6	0	0	1	2	3	4	5	6	6
Loading	0-6	0	1	2	2	3	4	5	6	6
Formation Drill	0-6	0	1	2	3	3	4	5	6	6
Bayonet Drill	0-6	0	1	1	2	3	4	4	5	6
First Aid	0-4	1	1	1	1	1	1	1	1	2
Breastworks	0-6	0	0	1	2	3	3	4	4	6
		1	4	8	12	16	20	24	28	32
		1	4	8	12	16	20	24	28	32
		1861								
		1862								
		1863								
		1864								
		1865								

The best way to use this chart is to view it as a set of value possibilities that gets larger over time. For example, in 1861, the options available for unit assignments are limited to the Raw, Green, and Trained attribute values. In 1862, the set of available options expands to Raw, Green, Trained, Fair, and Average. This reflects the fact that new units were still being recruited and older units were gaining experience.



Designing a Custom Scenario

TAKE COMMAND was designed from the ground up to be modified by users to create new scenarios using the historical and fictional battlefield maps that are included with the games. This section will show how to create a new scenario from scratch.

DESIGN TIP: You can grab the **units.csv**, **objectives.csv**, and **level.ini** file from the Open Play folder (after you have created an open play game) and use them to create a scenario which you can mod any way that you want.

To demonstrate the process of building a new scenario, we will create a hypothetical fictional Custom Scenario.

Planning. One of the most important aspects of scenario design is planning. If you are creating a new historical or historical variant scenario, a good knowledge of the actual course of the battle is needed. Even with a fictional scenario, some knowledge of the units involved is required. In addition, a good study of the map that your battle will take place is needed. You do not need to have your units starting at opposite sides of the map unless you really enjoy watching units march. A design tip is to print out the Mini Map graphic file. The Mini-Map files are located on your computer at:

C:\Program Files\Paradox Interactive\Take Command - 2nd Manassas\maps\map name_MM.dds

For this exercise we will set up a meeting engagement on the Cedar Ridge map. It will help to print out the map: **Cedar_Ridge_MM.dds**. Note that you may need to get a program to change the file format from .dds to a more traditional format like .jpg that programs such as Microsoft Word can handle.

The basic concept for this battle is to have a collision between a Union division and a Confederate division while both are marching to a distant destination—neither knows the other is in the area. We will further assume that neither unit has cavalry pickets scouting the road ahead. This lack of scouting happened more often than you might suspect. The Union forces will be marching on the road just north of Deer Hollow with the goal of continuing east towards the Montgomery farm. The Confederate forces will be coming out on the road just north of Cedar Ridge and heading north towards Rocky Junction. If we design things correctly, this will lead to a collision of forces in the area of Massey's Store.

Scenario Files. Before we begin, we need to understand the basic structure of the scenario folder. To take a look at the basic structure look at the directory on your CD-ROM **SDK\Scenarios\OH3 - Difficult Run - Starke (C-Div).** In the root of the scenario file structure is one required file, **level.ini**,

which holds some basic information about the scenario and will be discussed in more detail below. In addition to that file, there are two other required directories, **Data Files** and **Text**, and optional directories. During the loading of a scenario, the game engine checks the scenario files and uses any game file found there in preference to the equivalent file in the main game. This gives the scenario designer wide latitude in altering the game, a potential that has barely been touched. Any of the five main directories can exist in the scenarios folder and can override the main files of the game (maps, text, data files, sounds, and graphics). So you can experiment by creating a scenario and copying the files that you want to modify into your scenario folder without ever messing up the game.

Remember, file choices by the game engine are defined as follows:

1) Scenario loaded from .mmg or Custom Scenario Folder, and files in here will override those files in the main game directory.
2) Next the main directories, the top level folders in the same folder as the .exe, files in here will be used next.
3) Next the main **TC**.mmg** file. If the file does not exist in the previous two choices, the file will then be loaded from here.

The **data files** directory must contain three files, **events.csv, objectives.csv**, and **units.csv**. Other files can be added as needed. The **text** directory must contain a file **screen.txt** and a file **intro.txt**. The first contains the text of various messages used in the scenario and **intro.txt** has the text used in the bottom section of the 'Choose a Scenario' window on the 'Custom Scenarios' selection game screen. Details of the contents of the ***.txt** and ***.csv** files are covered below.

New Scenario Example: For our new scenario, we are going to start with the CM03 division level scenario. Go into the /SDK/Scenarios directory on the CD and copy the directory: **OH3 - Difficult Run - Starke (C-Div)** into the **C:\Program Files\Paradox Interactive\Take Command - 2nd Manassas\scenarios** directory. Next rename the directory to: **M1 Meeting**. In this scenario, we are going set up a speculative meeting engagement on one of the non-historical open play maps.

First steps: MMG has built in some very useful tools for the scenario designer which we will take advantage of. Using your text editor open up the game .ini file: **C:\Program Files\Paradox Interactive\Take Command - 2nd Manassas\TC**.ini.**

Add the following text at the beginning of the file:

[Initialization]
DbgLvl=2
NoAI=1
Alpha Omega=1

Save the file and exit the editor. These commands do three very useful things for the scenario designer. The first turns on the display that shows the location of the selected unit or officer in the game **X,Z coordinate system**. This system is use by numerous files to specify the location of units on the battlefield. The numbering starts at the lower left with coordinates 1,1 and ends at the top right at coordinates 131071, 131071. For scale, the engine uses 32 units per yard. This makes the map 4096 yards by 4096 yards or 2.33 miles square. The second command totally disables all AI functions so that units will not fight at all. This allows units to be marched all over the map without starting odd battles. The third allows you to see and take command of any unit in either army.

Next shift to the scenario directory (**C:\Program Files\Paradox Interactive\Take Command - 2nd Manassas\scenarios\M1 Meeting** and open up the file **level.ini** with your text editor. Edit the second line to read: **Map=Cedar_Ridge.lsl**. This changes the site of the battle from the original

Chantilly map to the Cedar Ridge map. Change the start time to: **9:00:00** so that we have all day to fight it out.

DESIGN TIP: Using the ';' character at the beginning of a line in any *.ini file tells the program that this line is a comment; the program will ignore all content of that line after the semi-colon. If you use this technique, you can minimize the adding and deleting of commands in the main and level .ini files.

Let's give this design tip a try. Put a semi-colon at the beginning of these lines:

CarryOverFrom=SM29 - 30 Aug - Starke (C_Div)
CarryOverCasPct=5

So that they look like this:

;CarryOverFrom=SM29 - 30 Aug - Starke (C_Div)
;CarryOverCasPct=5

These changes disable the carryover commands that prevent you from starting this scenario.

Next change to the **Data files** directory and open **events.csv**. While setting **NoAI=1** in the **TC**.ini** file prevents conflict, it does not stop unit movement that is dictated in the events.csv file. Delete row 4 and rows 7 through 125. Change all of the remaining entries in the time column to **9:00:01** then save the file and exit.

Moving US Units. Now, having done that—start the game. Go into Custom Scenarios game screen and select the new M1 Meeting scenario. Select General Pope as your character and load the game. After the game loads, nothing will happen until you specifically order it. You want to put the units in march order along the road past Blacksmith back towards Reed's Tavern and then down to Benton Mills (see the map below). Find Colonel Marsena Patrick and select him. March him and his entire brigade so that they are on the road passing Blacksmith. The head of the column should be intersection with the road from Rocky Junction to Massey Store. When the head of the column is in position, hit the Halt Brigade button to stop everyone in road march position. Use the right arrow key to identify the next unit in line. Now march them into position. Repeat this process until all units are in position as shown on the map below. Move Generals Pope, Heintzelman, and Hooker to just north of the road near Patrick's brigade. When you are all done, it should look like this:

Last and most importantly **hit the 'L' key**. This writes a file named **unitlocs.csv** intp the game's root directory (**C:\Program Files\Paradox Interactive\Take Command - 2nd Manassas\).** This file contains the XZ location of each commander and unit on the map, as well as the direction that they are facing at the time the **'L'** key was hit.

Exit the game. Now open up **unitlocs.csv** in the main game directory, and also open up **units.csv** in the M1 Meeting scenario directory. In **unitlocs.csv** select all of the first four columns (A-D) and select **Copy**. Then open the **units.csv** and paste the information into columns J-M of the **units.csv**. Save the **units.csv** and exit from both files. The Union units are now in the desired starting position. Restart the game and select General Pope; make sure that all commanders and units are in the correct positions.

DESIGN TIP: Make a "fast move" **unitcommon csv** to get troops into their scenario start-up positions quickly. Here's what to do. Make a copy of the **unitcommon csv** and place it in the Data Files folder of the Main Game Directory or in your custom scenario folder. Open this file and change all the RUN speed entries (column F) to 300. Save and close. With this "fast move" **unitcommon.csv**, your Commanders and units will now move into position at 300 yards per second! Don't forget to remove this file when you get ready to play.

Moving CSA Units. The process outlined above for the Union units is quite similar for CSA units. Start the game, this time selecting General Jackson. When the game opens, hit the down arrow key two times to change to Colonel Grigsby. Now road march his unit into position on the road south of the Massey Store (shown below). March the rest of Starke's Division into position along the road just north of Cedar Ridge. Move Generals Jackson and Starke into the field just to the right of Grigsby.

NOTE: The **'t'** key was used to remove the trees from the display so that the location of all units could be seen clearly in the screenshot.

Again hit the **"L"** key to write all of the unit locations to **unitlocs.csv**. Exit the game and copy all unit positions into **units.csv** as you did above. Save all files, and reopen the game to check that all units are in position. On the in-game Mini Map, the positions of both sides should look like this:

M1 Meeting – Starting Positions for Both Armies

Finding X,Z locations. Next we need to find map locations for objectives and for destinations for movement. On the display the lower line of white text on a black background shows the selected units current location. The first number is the **X coordinate** and the last number is the **Z coordinate**. For this picture:

The **X coordinate** is 67841 and the **Z coordinate** is 109581. The decimal fractions can be ignored for scenario design.

Restart the game and select General Pope as your character. Move him to the to the road from Rocky Junction to Massey Store, about half way between the intersection with the Blacksmith Road and the intersection with Deer Hollow Road. Write down the X and Z coordinates. This will be the marching destination for the Union forces. Next, move General Pope up to the intersection between Blacksmith and Fisher's Farm and record the X and Z coordinates to serve as a goal for the initial CS marching orders. Exit the game. Finally, move him south and record the X and Z coordinates in the middle of the field above where the text 'Massey Store' is located on the mini-map and just even with the north end of the smaller patch of trees. This will serve as the victory location for this scenario.

Creating Objectives. Now open the **objectives.csv** file in the **Data files** directory. For this scenario we will have one objective. It will be the Massey Store location recorded just above. In the **Objective Name** column in row 16, enter **Massey Store** and in the **Objective ID** column enter **Massey_Store**. Enter **100** in the AI column since we want this objective to be winnable by either side. Next in the **loc x** and **loc z** columns enter the values that you recorded earlier. Scroll right and change the **start time** to **9:15:00**, leave the **end time** at **23:00:00** which means that it will stay up for the duration of the game or until won. Next change the radius value (column H) to **150** which is the distance in yards that must be clear of opposing forces for the objective to be won. Finally, set the interval for **0:20:00 (20 minutes)**, which is the time the objective must be occupied for a win. Since we want to have winning the **Massey Store** objective to end the scenario, the type must be set to **Major**. Adjust the other values as shown in the table below. Here is how the revised **events.csv** file should look, because it is too wide for the page, it is shown in two parts.

Left side:

Massey Store	**Massey_Store**	**Major**	**hold**	**100**	**74113**	**66475**	**150**	**200**

Right side:

50	**0**	**0**	**0**	**1**	**9:15:00**	**23:00:00**	**0:20**	**Obj_Major**	**Obj_UMajor**	**Obj_RMajor**

Save the file and exit. Restart the game and scenario to be sure that changes in this file have not caused problems.

Managing Events. This is the core of the scenario. In this file, you dictate the specific movement of units, courier messages, and leader stances with the goal of setting up the battle the way you would like it to start. How it comes out is the intersection of the player's decisions and the AI decisions. There is significant complexity in these files and they amount to a miniature programming language. In spite of that, it can be learned and some interesting and challenging scenarios can be devised. In this case, we will be making one nearly from scratch since the existing one is for an entirely different battle.

For this part open up both **units.csv** and **events.csv** as we will need info from the units.csv file to feed into the commands in **events.csv**. Remember that the setup for this scenario is an inadvertent meeting between two divisions on the march. So we will set up the initial commands for each unit to be on the march towards their intended destinations. Let's start with the CS forces since the initial march command will be the only part of their action in the **events.csv** files since the player is responsible for the rest of their actions.

The remaining lines from the starting scenario are left in so that the senior commanders are available to send messages and so that the chain of command is intact. The command ***tcommon*** is shorthand for 'take command' and removes that character from the control of the AI. This results in the division commanders having command. The player's character is **C_WEStarke2** and the player's character is always ***tcommon*.**

The simple command to move Starke's division would be to add one row to the **events.csv** file:

9:00:02 AM	**C_WEStarke2**	**Amoveto**	**85060**	**83550**

However, because the previous row is to ***tcommon*** C_WEStarke2, this will not execute correctly. The workaround is to issue the commands to the brigade and battery commanders. That looks like this for a single brigade:

9:00:02 AM	**C_AJGrigsby**	**Amoveto**	**85060**	**83550**
9:00:02 AM	**C_AJGrigsby**	**Aform**	**Brig_Column**	
9:00:02 AM	**C_AJGrigsby**	**Auseroad**		

The ***Aform*** command gives the unit a formation to assume when they reach their destination. The ***Auseroad*** command forces the brigade CO and all subordinate regiments to march on the closest available road.

Edit the **events.csv** to give the same commands to the remaining infantry brigades, the artillery batteries, and the cavalry unit. Note that the ending formations will have to be changed, for artillery use ***Artillery_Line*** and for the cavalry use ***CavBrig_Line***. Save the events.csv file and load the scenario. Select General Starke as your player and let the scenario run. The CS units should all road march towards their destination. If all is correct, we will move onto the Union forces.

Since the Union forces will be fully under AI command, we can issue a single order to the division and it should work correctly. Copy the last 3 rows for the CS unit and paste them below. Change the time to 9:00:03 and the destination to the X,Z value you wrote down earlier. The ending formation can be ***Div_Battle_Line***. Change the unit name to **U_JHooker**, save the file and see what happens. If all was done correctly, the Union forces all started along their road march as well. We have one more row to add to the timed section of the scenario. This is the row to end the fight at a specific time. Copy a time down one row and edit it to read 10:00:00, leave column B empty and put ***endscenario*** in column C. This will end the scenario at 10:00 A.M. unless some other ending criteria are met before then.

Testing. If the edits were made correctly, the scenario will load. It is not done yet but it is time for the first test of the new battle. Edit the **level.ini** file in the root scenario directory so that the rank section looks like this:

[Rank]
Army=2
Corps=1
Division=1
Brigade=0
Regiment=0

This will put you back in charge of Starke's Division only and let the AI handle all of the other units. In addition, edit the **TC**.ini** file and comment out the lines:

; DbgLvl=2
; NoAI=1
; Alpha Omega=1

Here's how to turn off the location display and to re-activate the AI. Save the files and load the game and let's see a new battle in action. It's probably a good idea to take notes of things that need to be changed during the course of this first test.

Play through the game and pause to take notes on what is working well and what might need to be changed.

In my battle, the Union forces were slow to react and to deploy. To help them out, let's add some explicit commands to the **events.csv**. If you restart the game and follow General Hooker, he sights the first CS units about 9:00:40 or so. In war, sighting an enemy unit would be of high importance. Hooker would want to learn as soon as possible how big the opposing force is. Let's have the

general gallop out into the field where he can get a better view of what forces are approaching. The commands will be:

9:00:45 AM	**U_JHooker**	**moveto**	**75686**	**74010**
9:00:45 AM	**U_JHooker**	**run**		

NOTE: The ***moveto*** command moves only the leader, whereas the ***Amoveto*** command moves both the leader and all units under his command.

Having seen that the CS forces are substantial, General Hooker would decide to deploy his division for battle. To keep things simple, we will have the division deploy based on General Hooker's current location. The commands for this are:

9:01:33 AM	**U_JHooker**	**Aform:Div_Battle_line**
9:01:33 AM	**U_JHooker**	**Auseroad**
9:10:00 AM	**U_JHooker**	**orders:probe**

The last command gives him an explicit stance of ***Probe***, which is attacking, but not an all-out attack.

Messages. One issue that must be fixed is that the messages from the courier are not relevant and not arriving at the correct times. We will need to edit two files, **events.csv** and **screen.txt.** Open both files and let's see what is there.

In the **events.csv** file from OH3, here is a row that sends a courier message:

15:00:02 PM	**C_TJJackson**	**courier:C_WEStarke2:loadscreen:Courier:C_Starke1**

Here is the corresponding message:

$C_Starke1 <FONT=Georgia14,L,20,20,20>No.1 - General Jackson

Near Ox Hill
3:00 p.m. September 1, 1862.

<FONT=Georgia14,L,20,20,20>General Starke,

Resume the advance with your division down the Little River Turnpike;
clear any blocking forces so the Corps may advance.

<FONT=Georgia14,L,20,20,20>
Maj Gen T. J. Jackson
General Commanding
Left Wing, Army of Northern Virginia

//
//
//

The line in **events.csv** is a timed event and is used to send Starke the initial orders. In the original scenario, no enemy units are in sight at the start of the scenario and Starke must march down the road a ways to sight Union forces. Now, the first paragraph needs to be rewritten to fit the new

scenario and the second needs an adjustment. Here is text that can be used for the new version, or you can write your own.

Near Cedar Ridge
9:00 a.m. August 10, 1862.

<FONT=Georgia14,L,20,20,20>General Starke,

Advance your division to Rocky Junction and then move east to the next village. Await further orders there. Scouts have reported Union forces in the area, be ready for a fight at any time.

<FONT=Georgia14,R,20,20,20>
Maj Gen T. J. Jackson
General Commanding
Left Wing, Army of Northern Virginia

Make the needed edits to **screen.txt** and save the revised file. Now we need to add the commands to deliver and log the message into the **events.csv** file. Insert a row in the **events.csv** file at the end of the current list of commands to CS forces. The new command should be:

9:00:02 AM	**C_TJJackson**	**courier:C_WEStarke2:loadscreen:Courier:C_Starke1**

This sends the courier from ***C_TJJackson*** to ***CWEStarke2*** and delivers message ***C_Starke1***. Next we need to log the message so it is available during game play on the message log screen. The command for that is:

9:00:30 AM		**logmsg:Courier:C_Starke1**

Always allow time for the courier to reach the recipient before logging the message. Save all of the open files and run the scenario again to see how the new version plays. Take notes on anything else that might need changing.

Additional items. One small item is that some of the units in the Union forces are not in march order to start. Open **units.csv** and edit column O for Marsena Patrick's 4 regiments. Change them all to read ***Road_Column*** and change the row for Patrick himself to read ***Brig_RdCol***. Repeat this for the other two Union brigades and then save the file.

The file **intro.txt** in the text subdirectory contains the text that shows up in the lower left portion of the Custom Scenario screen when that particular scenario is selected. Edit this file to reflect the new scenario and save it with the same name.

Finally open the level.ini file and edit the victory levels. The victory point levels are usually set after the playtesters have played the scenario a number of times. However, if you set the levels as follows, this will be in the ballpark for this type of battle. To set the victory point levels, open the file **level.ini** and scroll down. In the section **[EndMajWin]** edit the line to read grade**=300**. This means that if you get 300 or more points, you are awarded a major victory. In the same way set the values in each section as follows:

```
[EndWin]      150
[EndTie]       50
[EndFail]    -150
[EndMajFail] -300
```

At this point, the scenario should be both playable and interesting. I should give you fair warning that a number of iterations of testing and error finding were omitted as they did not contribute to teaching you about making scenarios. However, if you choose to build new scenarios or modify existing ones, you should expect many rounds of debugging before you are finally happy with the scenario.

Last Thoughts. This has hopefully given you a basic introduction into the rather complex world of creating or modifying scenarios for the ***TAKE COMMAND*** game series. Combine the ideas you have learned here with your imagination and you will be amazed at the results.

The Events.csv

The **events.csv** file is the script of events and triggers for everything that a scenario designer wants to occur in a custom scenario. With this file, you can make any unit do just about anything you want. However, there are limits. For example, if you give a brigade an order to march right through the enemy lines, they will never make it. They will stop and fight as soon as they are in range of the enemy. Because of the numerous variables involved, even if you scripted every single movement, a scenario will not play out exactly the same twice in a row. The commanders and units have been programmed to protect themselves, and based on their "stance" they will react to enemy forces they can see. So realize this as you design your scenarios. Realize that the "Chaos that is War" has been built into the game by design.

NOTE: The events.csv file is a required custom scenario file. However, it is NOT required to be filled in. A scenario designer can set the ***strategicAI*** flag in the **level.ini** file. This will allow the AI to take control of the battle.

Types of Event Commands. There are two major groupings of commands in the **events.csv** file. These groupings are the T**ime Triggered Event Commands** and the **Action Triggered Event Commands**. The top of the **events.csv** file contains the time triggered events and the bottom contains the Action triggered events.

Time Triggered Event Commands. These event commands will execute at a specific time listed in a scenario's **events.csv** file. These event commands are synchronized to execute at a stated time based on the scenario start time. The scenario start time is designated in the scenario's **level.ini** file. These event commands will occur at the exact time stated on the **events.csv** file no matter what else is going on in the game.

Time Triggered Events	Explanation
A: Time	This is the time of the day that the event should occur. These are of the format **HH:MM:SS (Hour:Minutes:Seconds)**. This is NOT the time since the scenario started. This IS the time of the day. The start time of the scenario is in the **level.ini** file. Use military time for afternoon or evening events, e.g. 15:00:00, NOT 3:00:00 PM.
B: ID Name	This is the ID Name of the unit from the **units.csv** file. Each unit that you want to reference in this file must have an ID. This is the unit that will be affected by this event.
C: Command	This is the command that will be run for this event. Commands that take multiple parameters have their parameters separated by "**:**" (colons), except those commands that take coordinates as described below. Please see the commands reference for a complete list of commands.
D & E: X & Z Coord	On our map there are two coordinates that matter, x and z. Since y is up and we don't allow airplanes, y does not matter. All commands that require map coordinates will get the x and z from these columns. For example: ***moveto,***

	forcemove, and ***safeplace***.
F: Time Var	This column specifies to run this command a certain amount of seconds in the future. So if you want something to happen two minutes after an event is triggered, you can set this column to 120. Then, two minutes after the event is triggered, the command on this row will run. Note that this affects only the row that it is on and not any others that may be in an event triggered action grouping.

Time Triggered Event example:

Time	ID Name	Command	X Coord	Z Coord	time var
17:30:00 PM	**U_NPBanks**	**tcommon**			

What this means is at 5:30 P.M., the game engine will take command of Major-General Nathaniel P. Banks away from the AI. In other words, he will not be able to issue orders to his subordinate units.

NOTE: Game performance will lag if you have too many time triggered events happening at the same time; so we suggest spacing them out time-wise. Try to schedule your events so that no more than 10 events are triggered every second at the beginning of a scenario. Later when fighting may be going on, this should be reduced to an event every 5 seconds or so.

It really only makes sense to use timed triggered events at the beginning of a scenario to get things rolling—but that's up to you. You may want to use timed triggered events later in a scenario; the arrival of reinforcements for instance. Experience has shown that scenarios work best when timed triggered events are used early in the fight and when action triggered events are used later in the battle.

Action Triggered Event Commands. These events will trigger when some specific action happens during game play. The following table shows the **events.csv** layout for the action triggered events. It is much like the layout of the time triggered events with the exception of the first two columns on the trigger row.

Triggered Events	Explanation
A: Trigger Type	This is the event trigger type as defined in the table above. This can also be an ***evtcont*** to link this row with the previous event trigger type.
B: ID Name	This is the ID Name of the unit from the ***units.csv*** file or the objective from ***objectives.csv***. Each unit that you want to reference in this file must have an ID Name. This is the ID Name of the commander or unit that will be affected by this event.
C: Command	This is the event command that will be run for this event. Event Commands that take multiple parameters have their parameters separated by: colons, except those commands that take loc x, loc z coordinates as described below. Please see the ***Commands Reference*** section for a complete list of commands.
D & E: X & Z Coord	On our map there are two coordinates that matter, x and z. Since y is up and we don't allow airplanes, y does not matter. All commands that require map coordinates will get the x and z from these columns. For example: ***moveto***, ***forcemove***, and ***safeplace***.
F: Time Var	This column specifies to run this command a certain amount of seconds in the future. So if you want someone to happen two minutes after an event is triggered, you can set this column to 120. Then two minutes after the event is triggered, the command on this row will run.

Action Triggered Event example:

Time	ID Name	Command	X Coord	Z Coord	time var
evtdeath	C_JAEarly	endscenario:EndPlayerDie			

What this means is regardless of what time it is during the game play of this scenario, if Brigadier-General Jubal Early gets killed—it's Game Over! Go to endscreen…

DESIGN TIP: There are many game commands in ***TAKE COMMAND***. They can be used to support toolbar buttons, menus, and in the events.csv file for any Custom Scenario. All commands are in lowercase. If you preface a command with the capital letter **A**, then this command will affect all units subordinate to the Commander receiving the command. So if you have a brigade Commander selected and you click a button that sends the 'run' command. Then only he runs. But if you have that button send the 'Arun' command, then the brigade Commander and all regiments under him will run.

As stated above, an action in the scenario during game play triggers these events. They may never happen, but with some experience you'll find that the triggered events are the most powerful. In the table below we describe all of the Action Triggered Events.

Action Triggered Events	Explanation
evtarrived	This event is triggered when the unit defined in column: B has arrived at the location define in columns D & E. This means that the unit in column B is stopped and within 100 yards of the location. Also that all of his subordinates are also stopped. This is used to determine that a formation is halted and set up.
evtlarrived	This is when you want to trigger an event on just the leader. This triggers when the leader only has arrived at the location defined in columns D & E. The subordinate units can be anywhere on the map. The most common use is to trigger an event when the leader of a unit arrives at a specific road location without waiting for the rest of the unit to arrive.
evtcont	This is not a trigger, but is used to add other event commands after an event trigger. Since one row on the events.csv can only be used to trigger one command, this is the command to use to add more commands to one event trigger.
evtcourier	This event is triggered when the unit defined in column: B has received a courier message. Since the AI sends couriers all around the battlefield, use this command carefully as the first ***evtcourier*** command is executed when the first courier arrives, and the second ***evtcourier*** command with the second courier. Proper sequencing must be well thought out.
evtdeath	This event is triggered when the unit defined in column: B has died. This is how we end the game when the player dies. It can also be used to add some variety to a scenario, because units may or may not actually die.
evtfailcheck	This event is triggered when the unit defined in column: B has reached the fail grade. The fail grades are command specific. These are the events that we use in open play to end the battle. Each level of command has its own fail grade. By adding this event to a unit, that unit will now do fail checks. If their grade falls below the fail grade for their command level, then this event will be triggered.
evtfighting	This event is triggered when the unit defined in column: B has first engaged the enemy.
evtgiveup	This event is triggered when the unit defined in column: B has given up. This means that they have lost all of their subordinate units.
evtgrade	This event is triggered when the unit defined in column: B has reached a certain grade. Note that this is a greater than or equal to >= comparison and will not work

	to see if a unit has reached a low grade, only a high grade.
evtintrouble	This event is triggered when the unit defined in column: B is in trouble. This state can be seen by the displaying of the in trouble icon over that unit. It usually means that all of their men need rallying and that this unit is really no longer a fighting force.
evtobjarmy1	This is just like ***evtobjdone***, except that it is only triggered when army 1 has completed the objective.
evtobjarmy2	This is just like ***evtobjdone***, except that it is only triggered when army 2 has completed the objective.
evtobjdone	This event is triggered when the objective defined in column: B has been completed. If this is a hold objective, then this will trigger the first time it is completed. Note that this requires an objective ID, not a unit ID Name.
ranevt	Define a random event. Example: **ranevt:Union:3** In this example, <**ranevt**> is the command that tells the engine that you are going to declare a random event. <**Union:**> is the name of this random event. This random event name can be anything you want it to be. The entry <**3**> tells the game engine that you want it to generate a random number between 1 and 3. This random number (once generated) tells the game engine which <**evtran**> to execute. This example assumes you have only three alternative course of action (**evtran**). You can have any number of alternatives (**evtran**)—just change the random number generation requirement to match the number of alternatives in your design.
evtran	Execute a random event. Example: This is a continuation of our <**ranevt**> example above. We need to define at least one <**evtran**> but we can define no more than three (because we stated in the example above that the engine needed to generate a number between 1 and 3). An evtran statement has the command <**evtran**> plus the random event name <**Union**> plus the random number generated <**1**, **2**, or **3**>. So for example, the final <**evtran**> statements would look like this: **evtranUnion1** **evtranUnion2** **evtranUnion3**
evtseetarg	This event is triggered when the unit defined in column: B has first seen the enemy. This is used for commanders and it means that one of the regiments under their chain of command can see the enemy.

The Objectives.csv

The objectives.csv file defines all of the objectives (waypoints, victory sites, victory points, etc.) for a scenario. There are different types of objectives and different ways to win them.

NOTE: All of the points and bonuses are awarded each time a hold objective is won—even if it has been won 100 times previously.

Column	Explanation
A: Objective Name	This is the name that is displayed to the player. This is currently used on the strategic map, the objective countdown timer, and the commander's pop-up display.
B: Objective ID	This is the ID name of the objective that is used in other files to reference this objective. Mainly this is used in **events.csv**.
C: Major/Minor	This is just the word **major** or the word **minor**. The only difference in these two values is the display on the strategic map and how the AI evaluates the objective. It does not affect the player at all. Originally we had some more ideas for this and there may be more in the future, but for now, it doesn't do much.
D: Type	This column really determines how an objective acts. There are two valid values here: **hold** or **waypoint**. These are very different beasts. A **hold objective** will always reset itself. It will never end unless an event terminates it. Every time that it is won, points and bonuses are awarded and the timer will reset. A **waypoint** will not. A **waypoint** can only be won once, and then it will disappear.
E: AI	This determines who cares about this objective. An objective can be set up to be valid only for 1 army, or only for the player, or for everyone. This will affect the AI, as the AI will not care about a player only objective. 0: player only 1: army 1 (Union) 2: army 2 (Confederate) 100: everyone
F: loc x	The X coordinates of the objective.
G: loc y	The Y coordinates of the objective.
H: radius (yds)	This is the radius of the invisible circle surrounding the objective. The only units that are counted in determining who is holding this objective are those within the circle or within this radius.
I: Number of men	The minimum number of fighting men that must be within the radius to hold the objective. The only way to win an objective is to have more than this number of men within the radius as well as a commander. They must remain within the radius for the duration of the objective. If there are enemy also within the radius, you will lose the objective unless you have twice as many men as the enemy. If the enemy has twice as many men, then they will be holding the objective. If neither side meets these criteria then the objective is considered not held.
J: Points	The number of points added to the grade of the commander that is holding the objective when the time limit runs out.
K: Fatigue Bonus	This is the number of fatigue points added to all of the fighting men that are subordinate to the commander that won the objective. It is much better to hold an objective with a high ranking commander than a low ranking commander, as many more units will receive the bonus.
L: Morale Bonus	This is the number of morale points added to all of the fighting men that are subordinate to the commander that won the objective. It is much better to hold an objective with a high ranking commander than a low ranking commander, as many more units will receive the bonus.
M: Ammo Bonus	This is the amount of ammo added to all of the fighting men that are subordinate to the commander that won the objective. It is much better to hold an objective with a high ranking commander than a low ranking commander, as many more units will receive the bonus.
N: Occupied Modifier	This value is multiplied by the points and all of the bonuses if the objective site was previously occupied. This means that you can make it possible to give them more points if they had to fight for the objective.
O: Start Time	This is the time of day that the objective will become active and appear on the map. If you want to only have the objective become active when triggered by an event, then set this to some time that will never be reached in the scenario, such as 23:00:00.
P: End Time	This is the time of day that the objective will become inactive and disappear from the map. It does not matter if it has been won or not, if the end time is reached and the objective is still active, it will disappear. This is awesome for creating scenarios where units have to reach a location by a certain time.
Q: Interval	This is how long an objective has to be consecutively held to win the objective. If the commander leaves the radius or the number of men drops too low, or if the enemy comes in with too many men, this timer will be reset. When all the criteria are met, this will start counting down. You can see this

	timer by selecting the commander that is holding the objective.
R: Sprite	This is the sprite name from **sprites.csv** that will be shown if no one is holding the objective.
S-Z: Army 1-?	This is the sprite name from **sprites.csv** that will be shown if this army is in control of the objective.

Intro.txt

The **intro.txt** file is test that is displayed in the window in the lower left in the Custom Scenarios screen. The first line must start with $Intro and include the font, justification and color information.

$Intro <FONT=Georgia16,L,8,12,24>Uneven Stevens, Steven's Division

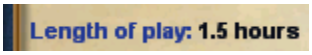
The text between <> are the formatting commands, and the rest is the part actually displayed on screen. Here is the interpretation:

Entry	Interpretation
FONT=Georgia16	This specifies the font that the message text is displayed with. Fonts and font equivalents are listed in the file Fonts.csv in the \SDK\Data Files directory.
L	Left justify text, can also have C for center or R for right justification
8,12,24	Text color in RGB coding.

The <FONT=Georgia16,L,8,12,24> information does not need to be repeated unless you wish to change the font, size or color of the text. For example, this line:

<FONT=Georgia14,L,38,58,150>Length of play:<FONT=Georgia14,L,8,12,24> 1.5 hours

Displays the text “Length of play:” in blue color and “1.5 hours” in black text, and looks like this on screen:



Screen.txt

The **screen.txt** file holds the text of any screens and messages that appear during game play. The formatting is the same as in **intro.txt**. The first line of each message must have the message header string that is called in **events.csv**. Here is the beginning of the first message of the OH1 scenario:

$UN_Stevens1 <FONT=Georgia14,L,20,20,20>No.1 - General Reno

Near Ox Hill
5:00 p.m. September 1, 1862.

The $UN_Stevens1 is the message header, as above the text between the <> is the font and formatting specification, and the rest is the text that is displayed on screen. If you have double forward slash (//) alone on a line, this is displayed as a blank line.

The **screen.txt** file also holds customized text that is displayed as the headlines on the End of Battle newspaper screen. The formatting is the same as above. As an example here is the first headline and text for the OH1 scenario:

$EndMajFail <FONT=Georgia16,C,0,0,0>Slaughter in the Storm!

<FONT=Georgia14,L,0,0,0>Brigadier General Isaac I. Stevens gave an embarrassing display of command ineptitude, even by the low standards of the Union Army. He sent his small division headlong on the attack into the entire Corps of the vaunted Rebel General Stonewall Jackson. This attack was pressed forward in a driving rainstorm, rendering ammunition useless. The remains of his division should be placed in the command of a general with some sense!
//
//
//

This is the text that is displayed if the player is scored with a major defeat in the scenario.

The Names.csv

The **Names.csv** file is a list of names that we use to generate random commander names to replace commanders killed during game play.

Column	Explanation
A: Rank	Randomly assigns a rank to a fallen commander.
B: First Name	The list of random first names. If you put your own first name in here, it will eventually show up during game play.
C: Last Name	The list of random last names. If you put your own last name in here, it will eventually show up during game play.

Mod/Add a Uniform

Overview. There are two parts to the process, editing the graphics files themselves and modifying the text files that specify which unit uses which graphic file for their uniform.

Memory Usage. Uniform graphics are held in memory – the standard set of eight uniform files for infantry takes up about 3.4 MB of memory. The new Hi-res Graphics sprite sets take up about 8.2 MB of memory. The memory usage can be changed by the various selections on the Options screen.

However, there is no requirement to use graphics in the ***TAKE COMMAND*** game series format. Uniform files can be less detailed and have fewer frames of animation, with much less memory usage. One early decision is where to set your tradeoff between number of uniforms and level of detail on each uniform. How critical this is to you depends entirely on how much RAM you have on your system. If you have 4 GB of RAM, go wild.

Graphics files. Each uniform requires a set of eight matched graphics files for the required game actions: stand, shoot, melee, march, charge, double quick, prone, and death. All unit graphics in ***TAKE COMMAND*** are in *.dds format. The standard infantry graphics show 16 angles in each row of each figure with between 10 and 40 rows per file. In theory, each of the 2240 frames that go into one uniform set could be edited by pixel. However, this is a somewhat time consuming method. An alternative is to use programs like ***3DStudio Max or LightWave***. In addition, graphics files from other games can be imported provided that the image angles go across and the frames go down. The correct values for these files need to be entered in the **sprites.csv** file but they should work. Whatever the solution adopted, the full set of eight sprite files must be either imported or created.

CSV files. After you have made the set of eight graphics files you need to get the new uniform in the game. This can be done on a per scenario basis by editing the set of four .csv files that specify which graphics files go with which action and which unit uses which uniform type(s). This is done fairly easily by using the ***WarEdit Utility*** or a spreadsheet program like Microsoft Excel or to keep everything lined up properly. The four files are: **sprites.csv**, **unitsprites.csv**, **units.csv**, and **unitcommon.csv**. Let's look at these in more detail.

Sprites.csv is the file that deals with the details of using the *.dds file as the source of the animation in the game. Each graphic file is assigned a name used by other files in the game. Other parameters dealing with the nuts and bolts of animation are in this file as well. See the ***Mod/Add a Uniform: The Sprites.csv*** section of this guide for details.

unitcommon.csv is the file where the specific uniform graphics files are assigned to a unit class. Up to six uniforms can be assigned to a unit class. These uniforms will appear at random on the sprites in that unit. There is no limit on how many uniform classes may be declared. The name in each uniform column must match a name in the 'type' column of **unitsprite.csv**. See the ***Mod/Add a Uniform: The Unitsprite.csv*** section of this guide for details.

units.csv is the primary OOB file. The column of interest for uniform modding is column G, 'Class'. The entry here must match the 'Class' entry in **unitcommon.csv** and determines which uniform(s) are worn by which specific unit in the game. See the ***Designing an Open Play Order of Battle: The Units.csv*** section of this guide for details.

Unitsprite.csv is the file where the eight names that are associated with the eight *.dds files in sprites.csv, are linked to a uniform 'type' used in the **unitcommon.csv** file. The name in each column must match the name in the 'Name' column of **sprites.csv**. ***See the Mod/Add a Uniform: The Unitcommons.csv*** section of this guide for details.

Example of a simple uniform mod.

To demonstrate the process we will walk through the process of doing a simple uniform mod. For this example, we will insert a set of uniform files for Confederate soldiers with bandages, representing walking wounded returned to the ranks.

NOTE: We have not actually created this set of uniform graphics files, so please do not ask for them. Perhaps someone in the mod community will actually create this set of files.

The first step is to do the actual graphics editing. There are two strategies, pixel editing which is cheap but very tedious or 3D editing which requires one or more comparatively expensive program(s), but is much faster.

For the pixel editing example, we will add an arm bandage to the standard **C_Regular** file set. If you were doing this by pixel editing, these are the steps. First, in the graphics directory copy the file **C_REGULAR_Stand.dds** and name the copy **C_Regular_B_Stand.dds**. Repeat the process with these files:

C_REGULAR_March.dds
C_REGULAR_Shoot.dds
C_REGULAR_Melee.dds
C_REGULAR_DblQuik.dds
C_REGULAR_Charge.dds
C_REGULAR_Prone.dds
C_REGULAR_Death.dds.

When you are done, you will have eight **C_Regular_B_*.dds** files to edit.

Using pixel editing requires a program that can read and save *.dds files. There are a number that can do the job that can be purchased either commercially or downloaded as shareware or freeware. Regardless of the program, open the **C_Regular_B_Stand.dds** file in the editor. In the first image, edit the pixels to add the white bandage on the right arm. Repeat 2239 more times to convert the rest of the images in this and the other six **C_Regular_B_*.dds** files.

A somewhat less tedious method is to load the files into a 3D rendering program, do the editing there and then write out the edited files with the appropriate parameters. One program that will do this is a program called 3D Studio Max. The details of using these 3D programs are beyond the scope of this guide. Consult the 'Modders Corner' on the MMG discussion boards for pointers to books or websites describing this process in more detail. After the 3D models are created, the 2D graphics files need to be created or rendered from the 3D models. This must be done for all eight file types.

After you have finished up creating or editing all eight of the graphics files, you need to do some editing of *.CSV files to get the game to recognize them. Let's assume that you will be opening the *.csv files in a spreadsheet program; thus we will refer to row & column designations accordingly.

First open up the file **sprites.csv**, and move down to row 117. Insert eight (8) blank rows below. Next type in the name for each of the eight types and the matching name for the * dds file. Next,

enter the appropriate numbers for each of the animation columns. If you are using the same number of frames and rows as the base game, you can simply copy the data from the existing equivalent file. See the section below on the **sprites.csv** for details of the animation values. For the wounded Confederate example, the end results should look like this. Using this order will make things easier for other files.

C_REGULAR_B_March	**C_REGULAR_B_March.dds**	**64**	**64**	**16**	**15**	**3**	**-1**	**0**	**0**
C_REGULAR_B_Stand	**C_REGULAR_B_Stand.dds**	**64**	**64**	**16**	**10**	**250**	**-1**	**0**	**0**
C_REGULAR_B_Shoot	**C_REGULAR_B_Shoot.dds**	**64**	**64**	**16**	**40**	**10**	**35**	**0**	**0**
C_REGULAR_B_DblQuik	**C_REGULAR_B_DblQuik.dds**	**64**	**64**	**16**	**15**	**4**	**-1**	**0**	**0**
C_REGULAR_B_Charge	**C_REGULAR_B_Charge.dds**	**64**	**64**	**16**	**15**	**3.2**	**-1**	**0**	**0**
C_REGULAR_B_Melee	**C_REGULAR_B_Melee.dds**	**64**	**64**	**16**	**30**	**10**	**20**	**0**	**0**
C_REGULAR_B_Death	**C_REGULAR_B_Death.dds**	**64**	**64**	**16**	**15**	**4.5**	**-1**	**0**	**0**
C_REGULAR_B_Prone	**C_REGULAR_B_Prone.dds**	**64**	**64**	**16**	**15**	**4.5**	**-1**	**0**	**0**

The values in column B must be the exact filenames of the graphics files in the **\Graphics\Units** directory. Save the file in *.csv format.

Next open up the file **unitsprite.csv**, move down to the bottom of the data and enter a type name in column A, for this example, use **C_Regular_B**. Next enter the names from column A of **sprites.csv** into columns B-H. A useful technique is to copy the names in column A and then <**Paste Special/Transpose**> with the active cell in column B. The end result should look like this:

C_Regular_B C_REGULAR_B_March C_REGULAR_B_Stand C_REGULAR_B_Shoot C_REGULAR_B_DblQuik C_REGULAR_B_Charge C_REGULAR_B_Melee C_ REGULAR_B_Prone C_REGULAR_B_Death

The values in columns B-H must match the values in column A of **sprites.csv**. Save this file in *.csv format and continue with the next file.

The third file to edit is **unitcommon.csv** in this file. Scroll down and insert a row below row 75. Copy all of the information from the row above and paste it into the blank row. Next change the entry in column A, ‘Class’ to **C_Regular_w_Bandage**. Change the value in column F, ‘Uniform 1’ to read **C_Regular_B**. Leave all of the other columns alone unless you have created new sound files to match the uniform. In this case, units using the **C_Regular_w_Bandage** uniform set will have one in every four sprites with the bandage uniform set.

The name in column F must exactly match the text from column A of **unitsprite.csv**. If you have additional uniforms for this unit, they can be entered in columns G-K. Matching entries must be in the **unisprite.csv**. A maximum of six (6) uniforms is allowed per regiment; these will used to randomly populate formations with individual sprites. Save this file in *.csv format and continue with the final file.

The last file to edit is **units.csv** from a scenario or an Open Play OP_*.csv file. Go down to the row for the unit you want to have the bandaged uniform set and over to column G ‘Class’ and enter **C_Regular_w_Bandage** which is the exact text from column A of **unitcommon.csv**. Save in *.csv format.

If you have done things correctly, the new uniform type will show up in on the game in open play or in any custom scenario where you have specified this uniform in the **units.csv** file.

The Sprites.csv

The **sprites.csv** file defines all of the in game sprites. If it's not listed here, it's not in the game. All of the sprites here are kept in memory during the entire game (not on the main menus). There is a filtering method to not load sprites that are not referenced, but once it's loaded, it stays.

NOTE: This is important to remember because the number of sprites that we have is directly related to why we demand a 64MB video card. These things take up a lot of memory that has to be switched to the video ram for display. That's why if you boost your system ram very high and your video card very high, you sometimes can get the game to run on less than a 1Ghz processor. I've had it running on a P3 600Mhz, with 1Gig system ram and a 128MB video card. I had to run in low mode, but it ran. The number of sprites that we use has been a constant battle between Adam and me. He was looking to make the best looking game and I was trying to make it run well. We compromised.

NOTE: Whenever you make a 3D game you must understand the payoff of using sprites vs. 3D models. 3D models are cooler and use less memory because you only need a small texture to map on the model. You don't have to create a separate texture for each frame of animation. But the problem is that 3D models use a lot of polygons. This is what affects video performance. Each sprite uses 2 polygons, where as a 3D model could use 10-20 plus and it wouldn't look that good. By using sprites we are able to get a lot more men on the map. We have over 20,000 sprites on the map at any given time. We could never do this with 3D models. Everyone would have to buy a $500 video card. So the payoff is that we can show a lot of guys, but we have these gigantic animation files and require lots of video ram and system ram. The houses, stone bridge, fences, and walls are actually 3D models.

The top of the **sprites.csv** file starts with a small list of sprites that must exist. That means that the game itself will be looking for the sprite name and if it's not there it may crash. You can change the graphic and anything else about the sprite, but the sprite name must exist.

Column	Explanation
A: Name	This is the identifier by which this sprite will be called in all of the other csv files and sometimes directly from the program. It cannot have spaces. I suggest using some type of naming convention because as the sprite list grows, you can lose yourself in the detail.
B: File	This is the name of the graphic file on disk that this sprites name refers to. Now the graphic directories are split up into sub directories just for organization purposes. All sprites referenced in **objectives.csv** are in the **misc** folder of the main game directory. Sprites used in the **unitsprite.csv** are in the **units** directory. Sprites in **units.csv**, which are only the flags, are in the **flags** directory. Sprites in **effects.csv** are in the **effects** directory. Sprites in **map_name.csv** are in the **terrain** directory. Sprites in **toolbar.csv** are in the **toolbar** directory. The graphics for TAKE COMMAND are all in .dds format which uses significantly less memory that the TGA graphics used in CWBR. This memory saving has allowed us to bring in the Hi-Res files. Files should be saved with no MIP maps. For figures for infantry, artillery, cavalry or flags, use DXT5 and for terrain objects, use DXT1. Graphics can also be 24bit TGA files with a separate alpha channel or PCX files with solid black as the alpha. We believe these will work in all cases. The only problem with PCX files is that they have a hard alpha and don't look as nice, but the files are smaller.
C: Width	This is the width of one frame of the sprite in pixels. It is not the entire size of the file unless the file only contains one graphic. Since video cards will always change the size of the sprites to the next power of 2, there is no point in making a sprite 60x60, since the game will automatically move it up to 64x64 which is the next power of 2. It may reduce the size of the file, but there will be no performance

	boost.
D: Height	This is the height of one frame of the sprite in pixels. It is not the size of the file unless there is only one frame in the file.
E: Scale	This tells the game engine what size you want the sprite to be in the 3D world. You can shrink or grow the sprites depending on your specific needs. For example if you put 0.5 here, the sprite will be 50% of its original size in the world. If you put 2.0 here, then it will be 200%. You can leave this empty if you want it the same size as shown in your particular .dds graphics file.
F: Hi Width	This is the width for one Hi Res sprite frame in pixels.
G: Hi Height	This is the height for one Hi Res sprite frame in pixels.
H: Hi Scale	This is the scale factor for Hi Res sprites
J: Angles (across)	This is the number of cells across that the file has. Currently we use 16 frames for our units. This is because it looks much nicer in the game. Previous sprite based games only had 8 angles, which showed a lot of popping as their direction changed. By using 16 angles the popping is greatly reduced but the file size and memory image is greatly increased. You can define as many angles you want for your sprites. That is completely up to you. It is suggested that you keep them to a power of 2, otherwise it would look strange. The toolbar buttons also use this as they are referenced by specific cell in the toolbar.csv file. **Note**: When creating a new unit graphics file, you must follow the Take Command facing conventions in sequencing your animation frames. The first frame must be facing away from you (looking at its back). The frames then continue rotating to the right across the row. They must rotate to the right up until the last frame, which would then rotate to look like the first frame. This is easily seen by opening up any of our unit graphics files.
K: Frames (down)	This is the number of cells down each column. The frames of animation go down a column. With each cell down being the next frame and the last cell continuing back to the first cell at the top of the column. Any unit or terrain sprite can have angles and frames. They will automatically be handled by the program. Currently our trees and terrain sprites only have one cell in their files. But it would be easy to add animated trees with multiple angles to give the forests a must more realistic look.
L: Animation Speed	This is the speed of the animation in ticks. As you can see each sprite can have a different animation speed, which is the amount of time it takes to switch from one frame cell to the next. Ticks are defined as 60 ticks per second. So if you wanted the animation to take place once every second, you would put in 60. This is a float, meaning that it can have decimal places in the number.
M: Action	This is the action or firing frame. It is only used for unit sprites that show the men firing their weapons. We have synchronized the program to always fire when they reach this frame. There is more on this in column M and N, as it's a little hard to understand how it all works. Remember that frames are 0 based. Meaning the top frame is frame 0, not 1. So if you have 20 frames, you have frames numbered 0-19, there is no frame 20.
N: Above Terrain	This is the number of game units above the terrain that the sprite will be drawn. This is not in yards or any other standard unit of measure. For our scale, we have 32 game units per yard. So if you want to convert it, just do the math. This is used for the VP and orders icons that float in the sky.
O: Sharp	This is tells the engine how to load in the graphic. The value can be 0 or 1. Basically this determines if the graphic is loaded in exactly as defined (sharp=1) or if the engine can smooth out the graphic (sharp=0) and blur it a little. We use sharp for the ***TAKE COMMAND*** toolbars.
P: Shadow	Our forests are dynamic, meaning that our game fills them in when the level is loaded. See the map_name.csv file write up for more detail. When the terrain graphic is placed we can draw a shadow for it right on the terrain. This really adds to the realistic look of the world. Without a shadow it looks kind of dumb. The shadows for the units are drawn in the cells of animation. So if you create a terrain sprite, you'll want to fill this in. The name ***BigRock*** is just a model from our map that is used to draw the sprite on the ground. We don't have any others, so if you want a shadow, just fill this in with ***BigRock*** like all of the other terrain sprites.
Q: Begin Loop	Since it would not make sense to have the weapon firing rate dependant on the number of frames of animation, we have defined a firing time in the **weapons.csv** file. Certain unit skills can modify this time, so it's variable. In order to pull this off, we need to know what to do while we're waiting to fire the weapon. That's where this loop comes in. This loop will tell the engine which frames to continuously go through until it's time to fire the weapon. We use this for our infantry. They will continue to push in the ramrod until it's time to fire. This must be defined for any firing animation, even if it's only one frame. Also, these frames cannot depend on circling back to the first frame. For example: Our men fire on frame 35, but loop between frames 13 and 19 while waiting to fire. The engine calculates how long it will take to reach frame 35, then will loop between frames 13 and 19 until it's time. You could

	not have them loop between frame 26 and frame 5 because it would require going back to frame 0 and the game won't handle that situation. For best results have the firing frame towards the end of the frames, because it looks better while falling back and retreating. Remember that these frames are 0 based, not 1 based. The begin frame is the first frame of the loop.
R: End Loop	This is the last frame of the firing loop. If it is not time to fire and this frame is reached, the next frame will be the one mentioned in the Begin Loop.

The Textures.csv

All dds sprites must also have an entry in the **textures.csv** file. This allows a single texture to be broken up into multiple files. All textures must have dimensions that are a power of 2. So in order to use the least memory possible, it is good to break up textures into multiple files.

Column	Explanation
A: Name	The .dds name from column B of the **sprites.csv** file.
B: File	The actual file name
C: Width	The width of the .dds file.
D: Height	The height of the .dds file.

In the following example you can see that the texture **C_Haversack_Melee.dds**, as referenced in the sprites.csv file, is actually broken up into 6 different files. All of these files are square and are powers of 2 in dimension. Note the "cont" in column A to show that the following lines are all part of the first line.

Example:
C_Haversack_Melee.dds C_Haversack_Melee1.dds 1024 1024
cont C_Haversack_Melee2.dds 512 512
cont C_Haversack_Melee3.dds 512 512
cont C_Haversack_Melee4.dds 512 512
cont C_Haversack_Melee5.dds 256 256
cont C_Haversack_Melee6.dds 256 256

The Unitsprite.csv

The **unitsprite.csv** file was created to define one set of sprites that define one type of unit. Each unit in the game requires eight sprites types: **march, stand, shoot, run, charge, melee, prone, and dead**. Since not all units use all of the animations, you can just reuse a sprite type in these instances.

Column	Explanation
A: Type	This is the name of this sprite set. This name will be used in other files to reference this sprite type.
B: Low Res	This is the name of the alternate sprite from the unit**sprites.csv** file. It is used when the Minimum Uniforms option is selected on the Options Game screen.
C: March	This is the name of the sprite from the **sprites.csv** file that will be used while this sprite type is marching.

D: Stand	This is the name of the sprite from the **sprites.csv** file that will be used while this sprite type is standing still.
E: Shoot	This is the name of the sprite from the **sprites.csv** file that will be used while this sprite type is firing their weapons.
F: Run	This is the name of the sprite from the **sprites.csv** file that will be used while this sprite type is running; this is also used for retreating.
G: Charge	This is the name of the sprite from the **sprites.csv** file that will be used while this sprite type is charging the enemy.
H: Melee	This is the name of the sprite from the **sprites.csv** file that will be used while this sprite type is engaged in melee combat.
I: Take Cover	This is the name of the sprite from the **sprites.csv** file that will be used while this sprite type is taking cover (prone).
J: Dead	This is the name of the sprite from the **sprites.csv** file that will be used while this sprite type is dead. This can have multiple animation frames if a dying animation is desired for this sprite set. The last frame of animation will be used as the final dead body.
K: Dead2	This is the name of a secondary sprite from the **sprites.csv** file that will be used while this sprite type is dead. Currently this is only used for the men dying around a cannon. The first dead sprite is the actual cannon and the 2nd dead sprite is the men of the crew.

The Unitcommon.csv

The **unitcommon.csv** was created in order to better organize often reused properties of units from the **units.csv** file. Basically we define a unit class. In the units.csv file we assign a class to every unit and then that unit picks up all of the properties from the unitcommon class. It's an organizational tool and prevents mistakes.

Column	Explanation
A: Class	This is the name of the class. It is used in the units.csv file. Each unit in the game has an assigned class.
B: Type	This determines what type of unit this is referring to. The following values are valid: 0 = Infantry 1 = Cavalry 2 = Artillery 3 = Ordnance 4 = Courier Note that we only have one type of division, corps, and army commander. They must be set to infantry. You cannot create an artillery division commander. You can create an artillery, cavalry, or infantry brigade commander.
C: Alt Class	This is used for limbered artillery and dismounted cavalry, as well as when guns are captured. The internals of the program know to switch the class to the alt class when certain functions are called or certain events occur. This is just another class name. When the class switch is called for either by the AI or the player's interaction with the toolbar, the referenced unit switches to using the alt class.
D: Capt Class	This is the class that is used when an artillery unit is captured. This is used for artillery units only.
E: March Speed	This is the number of yards per second that this unit can march. This can be a float value.
F: Run Speed	This is the number of yards per second that this unit can run. This can be a float value.
G-L: Uniform 1-6	This are the sprite set names from **unitsprite.csv**. You can have up to six different uniforms in the same unit. The program will randomly assign these. You must fill these cells in sequentially. You cannot have a name in just cells 1 and 3. You must fill in cell 2 as well.
M: march sound	This is a looping sound from **gamesounds.csv**. It is played while this unit is marching.
N: stand sound	This is a looping sound from **gamesounds.csv**. It is played while this unit is standing still.

O: shoot sound	This is a looping sound from **gamesounds.csv**. It is played while this unit is firing their weapons.
P: run sound	This is a looping sound from **gamesounds.csv**. It is played while this unit is running.
Q: charge sound	This is a looping sound from **gamesounds.csv**. It is played while this unit is charging the enemy.
R: melee sound	This is a looping sound from **gamesounds.csv**. It is played while this unit is engaged in melee combat.
S: take cover sound	This is a looping sound from **gamesounds.csv**. It is played while this unit is taking cover (prone).
T: flag bearer	This is the sprite set for the flag bearer from **unitsprite.csv**. This 1 sprite in a formation can have its own sprite set. Currently we don't really use this feature because we couldn't get the flag staff to line up with the flag. You can put whatever you want here; you can make this a regimental commander sprite if you wish.
U: toolbar	This is the toolbar name from **toolbar.csv**. This is the toolbar that is shown when this unit is under your control.
V: friendly toolbar	This is the toolbar name from **toolbar.csv**. This is the toolbar that is shown when this unit is in your army but not under your control.
W: enemy toolbar	This is the toolbar name from **toolbar.csv**. This is the toolbar that is shown when you click on a flag from the enemy army.
X: fighting formation	This is the formation name from **formation.csv** that is used by the AI as the default formation to use when fighting.
Y: march formation	This is the formation name from **formation.csv** that is used by the AI as the default formation to use when marching.
Z: Melee Hit	This is the chance out of 1000 of successfully getting a kill when fighting in melee combat. This value will be modified by the **levels.csv** file based on the skills and experience of the unit. So this value should be the base for green unskilled troops. We keep this value pretty low, since each sprite represents multiple men. If this value is too high, melee would be over instantly.

Mod/Add a Flag

The flag graphic has to be included in the file **sprites.csv** which needs to be in the *data files* directory. Then the flag can be assigned to a specific unit in **units.csv** or an Open Play **OB_*.csv** . It will show up in open play or in any custom scenarios you build that include that **units.csv** file.

Mod/Add a Weapon

The ***War3D*** game engine offers significant power for the modder to change the behavior of the infantry and artillery weapons. Please note that changes described here will not apply to the battle scenarios shipped with the game that appear on the 'Battles' screen. The behavior for those is hard coded in the *.mmg files. This means that you cannot mod the game and have the changes affect the scores that will appear on your 'Service Record'. If you want to try and make "Guns of the South", it will have to be as a custom scenario.

To understand how to mod a weapon, it is essential to understand the structure of three files: **Weapons.csv**, **artyammo.csv**, and the artillery section of **tables.csv**. Read through these sections and then the discussion of how to mod weapons in ***TAKE COMMAND*** will resume after the section on **tables.csv.**

The Weapons.csv

The **weapons.csv** file describes the firing abilities of every ranged weapon in the game. Weapons such as artillery have extra parameters that are described below. There is no limit to the amount of weapons that you can create.

The **weapons.csv** file is a little obscure, so please read this carefully to understand how it all works.

Infantry weapons: There are seven base ranges for infantry weapons that are used by the game. They are described in columns C through I. Each weapon will set its own particular yardage for each range. The first row of the csv is just header text that is not read by the program. The second row is the most important row because it sets the percent chance of a hit for each of the seven ranges. This chance of a hit is set in row 2, columns C - I. In each of these boxes there is a number that describes the chance of a hit out of 1000. This is the same for every weapon in the file. This is the way that the weapons differentiate themselves is by setting the yardage for each range.

For example: The first range is Min Range. We have set this to have a 400 out of 1000 chance of a hit, or 40%. This is our highest percent chance because this is the closest that you will get. Our rifles are different; many of them have this range set at 30 yards. That means for these rifles to have the same 40% chance of a hit, they would have to be at least within 30 yards of the target. Although the 40% chance of a hit does not change, the range to get that 40% chance can does change for every ranged weapon.

These ranges also affect how the AI works. The AI will not tell their men to move within 50 yards of the enemy, that's not how it thinks. The AI will tell their men to move to Medium range (for example). So depending on the rifles that their men have, their distance to the enemy will vary.

NOTE: The base chance of a hit, the base chance of a misfire, and the firing times are also modified by a number of factors. For the chance of a hit, these are: weather, unit quality, fatigue, and accuracy. The weather is set by the scenario designer; the initial values for the other three parameters are set in **units.csv.** The misfire value can also be affected by: weather, unit quality, fatigue, and loading rating. Firing time is affected only by unit quality. The tables that show how the modifying parameters change the base values are found in **levels.csv**.

Column	Explanation
A: Name	The name of this ranged weapon. This name is used in other csv files to reference this weapon.
B: Name2	This is an extra line of text to describe the weapon. This field is used for display only, and is not referenced anywhere except by the variable that displays it.
C: Min Range	The number of yards away from the enemy for this weapon to be considered at minimum range. The minimum range value will be used for 0 - Min Range.
D: Optimal Range	The number of yards away from the enemy for this weapon to be considered at optimal range. The optimal range value will be used for Min Range - Optimal Range.
E: Medium	The number of yards away from the enemy for this weapon to be considered at medium range. The medium range value will be used for Optimal - Medium Range.
F: Typical	The number of yards away from the enemy for this weapon to be considered at typical range. The typical range value will be used for Medium - Typical Range.
G: Long	The number of yards away from the enemy for this weapon to be considered at Long range. The Long range value will be used for Typical - Long Range.
H: Longest	The number of yards away from the enemy for this weapon to be considered at Longest range. The Longest range value will be used for Long - Longest Range.
I: Max Range	The number of yards away from the enemy for this weapon to be considered at maximum range. The maximum range value will be used for Longest - Max Range. This also specifies the maximum range in which this type of weapon can be fired. The men will not fire if the enemy target is farther away than this range.
J: Misfires	This column is probably labeled incorrectly. It's really the chance you have of loading your rifle correctly. It's out of 1000 like the chance of a hit. This is also modified by the unit's levels in **levels.csv**. So if this number is 200, then you have a 20% chance of even getting the shot off.
K: Firing Time	This is the number of seconds that it takes between firing your weapon and getting it reloaded. Obviously the lower time here the better. This, as well as misfires and shooting, should be set for the lowest level of men, for the green soldiers. Because unit experience and skills will bring this number down.
L-O: Arty Ammo Type Quantities	*Columns from L through T are for artillery only*. These next four columns need to be in the same exact order as the ammo types listed in columns P-S. They are matched 1-1 by the program when the level is loaded. These four numbers need to add up to exactly 100. Basically when an artillery unit is loaded, they are given a total number of rounds from the units.csv file. Since arty has 4 types of ammo, we use these numbers as percentages to divide up the ammo into the different types. It also effectively allows certain artillery types to not receive certain ammo types. So you can use 0 as a valid value if that piece of artillery would never shoot that type of ammo. If these numbers add up to more or less than 100, the results are undetermined, but may result in a rapid visit to your desktop.
P-S: Arty Ammo Types	These entries show which ammo type from the file **artyammo.csv** is used for this artillery type. The value in each cell must be an exact match for an entry in column A of **artyammo.csv**. In some cases the ammo type differs whether the battery is Union or Confederate. This is because the CS had great difficulty in manufacturing reliable fuses for shrapnel and shell.
T: Accuracy Table Number	The value in column T indicates which accuracy table found in **Tables.csv** should be used for this artillery weapon type. A value of '1' in column T shows that the table 'Arty1' in **Tables.csv** should be used in the accuracy calculations for this type of artillery.

The Artyammo.csv

The **artyammo.csv** file has one row for each artillery ammo type. We use this file to further modify the effects of the different artillery ammo types. The number of ammo types is not hardcoded, you could add new ammo types to this file.

Column	Explanation
A: Name	This is the name of this artillery ammo type. The name here is used in **Weapons.csv** to determine the ammo performance characteristics.
B: Particle Effect	This is the name of the graphics effect from column A of the **effects.csv** file. This effect will be played at the point of impact, or in the air as dictated.
C: Sprite Graphic	This is the name of the explosion graphic from the **sprites.csv** file that will be played at the point of impact. This graphic will slowly be faded out.
D: Smoke	This determines whether or not to play the smoke effect at the point of impact.
E: Display on Ground	This determines whether or not to display a missed artillery explosion on the ground, if not on the ground it will be played in the air.
F: Min Range	This is the minimum range in yards that this ammo type would be used.
G: Max Range	This is the maximum range in yards that this ammo type can be used. This also represents the maximum theoretical range for the gun type.
H: Probability of kill with a hit	If the accuracy calculations show that the ammo type passed through the target area, this value shows the probability that casualties resulted. The values range from 0 to 1, with 1 representing a 100% chance of casualties. For solid shot, this value is always 1, the values for shell and shrapnel represent the probability that the fuse detonates the round at the right time to cause casualties.
I: Max Kill	This is the maximum number of men that would be killed by 1 successful hit by this ammo type
J: Miss Morale decrease	This value is the amount that morale is decreased by a miss where no casualties result. The minimum value is 1 and the maximum value (for big Naval weapons) is 3.
K: Canister angle	This is the dispersion angle of the shot from a canister round after leaving the barrel of the gun. The values are in degrees and should be between 0.5 and 0.9 for all single canister rounds. Double canister is not implemented as a separate ammo type in ***TAKE COMMAND***, but may be added in a future game. Double canister is mimicked by simply doubling the casualty value when the target is closer than 75 yards. Although there are **artyammo.csv** entries for double canister, they are not used and that it's hardcoded for 75 yards.
L: Canister shot #	This is the actual number of shot in the canister round used by this weapon type. It is used in the calculations to determine the number of hits by a canister round.
M: Sound File	This is the name of the sound from the **gamesounds.csv** file that is played when this ammo type impacts.

The Tables.csv: Artillery Section

This part of the tables are used to describe the accuracy of the various types of artillery used in ***TAKE COMMAND***. This is new for ***TAKE COMMAND*** and is intended to bring a greater historical accuracy to the differing performances of artillery depending on a variety of factors. The new artillery engine initially calculates the group diameter at the actual range to target. This is based on the values in the Arty# table section in **tables.csv.** These values were derived from actual scores in cannon target matches conducted by the North-South Skirmish Association (http://www.n-ssa.org/). There are currently five tables which cover rifled guns (e.g. 3 inch Ordinance Rifle), long smoothbores (e.g. 12 lb Napoleon, short smoothbores (e.g. 12 lb howitzer), the Wiard 6 and 12 lb guns, and the

Whitworth gun. The accuracy data for the Wiard and Whitworth were estimated as no actual performance data has been yet located. The table data are arranged in this way:

Column in tables.csv	Explanation
A	Unit Quality Rating: These exact terms must be used as they match values used elsewhere in the game.
B	Observed or estimated group size for 10 rounds fired at 100 yards. The spread in values matches that seen from top to bottom in actual cannon target matches.
C	Combat Stress Factor Adjustment. Set at 2 but can be changed. This is a global multiplier to represent the factor that performance in combat is always degraded from that on the target range.
D	Adjusted Group Diameter: This is simply the value in column B multiplied by the value in column C. This is the only value actually used by the artillery engine to calculate the results of that round being fired.

Now that you have some familiarity with the tables, we will discuss how to mod the game to add additional weapons. We will start with infantry weapons and then deal with the more complex case of artillery weapons.

The mechanics of adding a new infantry weapon are actually relatively simple. If you are adding a weapon for a scenario, the copy **Weapons.csv** from the SDK folder into the **data files** folder of your custom scenario. If you want the change to appear in open play, then the **Weapons.csv** file should be copied into the **data files** directory of the main game. Let us assume that you have discovered that your favorite unit, the 6th Arkansas Volunteer Infantry (CS), was armed with the Model 1819 Hall breech loading musket. In the interest of historical accuracy, you may want this to be in the OOB for a future game.

In the **data files** directory, open the file **Weapons.csv** in your favorite spreadsheet program. The infantry weapons are sorted generally in decreasing caliber. Insert a row at row 26, between the entries for 1854 Austrian Lorenz and Sharps Rifle. Column A holds the weapon name, that will be used in **units.csv**; enter *US M1810 Hall Flintlock* here. For column B, copy up the value of *.52 cal Breechloader* from the row below. In columns C-K, the values that reflect the actual performance of this weapon in the hands of the units that carried it. Please note that this is not the bench-rest accuracy of the weapon, but the actual performance in the hands of the infantry that carried it. Since rifle practice was quite rare in most units, the actual performance was much worse than the weapons were usually capable of. The values in each cell are the range at which the 'Base % Chance of HIT' matches that shown in row 2. The data here are simply a best estimate as quantitative data are nonexistent. My choice was to copy the values from the Dimick Rifle (row 14) for columns C-I as these are intermediate between the smoothbores and the mid-century rifled muskets. For the misfire value (column J) the 175 value for the US M1816 Flintlock was used, as was the firing time of 40 (column K). This is all that is needed to make the weapon available in the game.

The next step is to assign the weapon to a specific unit. This is done in the **units.csv** file. Open up the **units.csv** file for the scenario where you wish the weapon to appear. Move to column H (Weapon) and scroll down to the specific regiment that you want to have this weapon, in our example, the 6th Arkansas. Copy the value from column A of **weapons.csv** and paste in column H of **units.csv** and save the file. The 6th AR will now appear in the game carrying the Hall flintlock.

Modding artillery is more complex, due to the need to account for the gun accuracy, crew competence, type of round, and the very different performance of fuses between CS and US suppliers.

Mod the Game Interface

One of the strengths of the TAKE COMMAND system is that the entire user interface can be changed by the determined modder. The game allows new buttons to be added or existing ones rearranged—or deleted. The graphics used for the buttons or for the toolbar itself can be changed as well.

The Toolbar.csv

The **toolbar.csv** file defines all of the toolbars in the game. This file describes all graphical components of toolbars; the text for the toolbars is defined in the **tooltext.csv** file. These files use the same exact names to refer to the different toolbars. The **unitcommon.csv** file assigns the toolbars to the different commanders and units in the game. There is no limit to the number of toolbars that can be defined and used in the game.

There are three types of tool bars. The first type is used for commanders and units that will be under the player's command (i.e. a subordinate). The second type is used for commanders and units belonging to the same army (side) as the player–but not part of the player's command (i.e. peer and superior commanders and units). The third type is the toolbar that is shown for the enemy. Depending on your design, you could possibly combine the friendly and enemy toolbars, but that choice is up to you, we do not.

Toolbars are drawn on the game screen in descending order during play. This means that you must list the background of a tool bar first on the **toolbar.csv**. If you place the buttons and then place the background, the buttons will be hidden because they are behind the tool bar background. The tool bar graphics are drawn in the same exact order as they are presented in the file.

Toolbars support inheritance. This means that you can create a base class toolbar with common functions, and have other toolbars **INHERIT** attributes from it. This really eases the process when making global changes and keeps the graphics/toolbar folder files smaller and easier to manage.

Each button supports 4 states: Normal - the normal button with no changes, Highlighted - shown when the highlight condition is met, gray - we do not currently use this state, Active - shown when the mouse cursor is over the button.

Column	Explanation
A: Toolbar	This is the name of the toolbar. This same exact name is used in **tooltext.csv** and **unitcommon.csv**. Since there can be a lot of toolbars, we suggest using a naming convention that will be easily understood, because you can become lost.
B: Sprite	This is either the name of the sprite from the sprites.csv file or the word ***INHERIT***. If you use the word ***INHERIT*** here, the next column must contain the name of the toolbar from which you want to ***INERHIT*** from. Meaning that the inherited toolbar will be completely included for this toolbar.
C: loc x (across)	If you are using ***INHERIT***, this column will contain the name of the toolbar from which you wish to ***INHERIT***. Otherwise it contains the 2D x coordinate of the upper left corner of where you want to draw the sprite. Remember that these coordinates are 2D with the screen being a 1024x768 area.
D: loc y (down)	This is the 2D y coordinate of the upper left corner of where you want to draw the sprite.
E: Button Sound	This contains the name of a sound from the **gamesounds.csv** file. This is only used when this row references a button with an associated function. The default sound is **click**, unless a different one is specified here. This sound is played when the user presses and releases the button.
F: Normal (across)	This is the 0 based index of the graphic from sprite referenced, just start at 0 and count across until you reach the column of the sprite that you want. We group the button sprites together in one file for

	ease of use. This specifies the index normal state of the graphic. This state will be shown if the graphic does not qualify for any of the other states explained below. Leave this as 0 if there is only 1 graphic in the sprite.
G: Normal (down)	This is the 0 based index of the graphic from sprite referenced, just start at 0 and count down until you reach the row of the sprite that you want. We group the button sprites together in one file for ease of use. This specifies the index normal state of the graphic. This state will be shown if the graphic does not qualify for any of the other states explained below. Leave this as 0 if there is only 1 graphic in the sprite.
H: Highlighted (across)	Same as Normal (across), but specifies the highlighted state.
I: Highlighted (down)	Same as Normal (down), but specifies the highlighted state.
J: Gray (across)	Same as Normal (across), but specifies the grayed out state.
K: Gray (down)	Same as Normal (down), but specifies the grayed out state.
L: Active (across)	Same as Normal (across), but specifies the active state. The active state is used when the cursor is over the button.
M: Active (down)	Same as Normal (down), but specifies the active state. The active state is used when the cursor is over the button.
N: mod left	If you want to have a smaller active area for particular buttons, you can use the mod parameters. We currently only use this for the wheeling buttons. Basically if you put a number in here it is added to the current side of the rectangle in order to obtain a hit area. For example: in our wheel right button, we put a value of 19 in here. This means that 19 is added to the left side of the rectangle of this button. So this effectively creates two buttons out of one, but specifying a hit area on only the right side of the button.
O: mod top	This works the same as mod left, but adds the value to the top of the rectangle; we do not currently use this option in any of our buttons.
P: mod right	This works the same as the mod left, but adds the value to the right side of the triangle. For example: in our wheel left button, we put a value of -19 in here. This means that -19 is added to the right side of the rectangle of this button. So this effectively creates two buttons out of one, but specifying a hit area on only the left side of the button.
Q: mod bottom	This works the same as mod left, but adds the value to the bottom of the rectangle; we do not currently use this option in any of our buttons.
R: func	This is the function for the button. If this field is blank, then the program assumes that this is not a button. Please see the **command reference** for a list of valid function names.
S: text	This is the ***tooltip*** text. This will appear whenever the user has the cursor over this graphic. You can have a ***tooltip*** for any graphic; it does not have to be a button.
T: Special	Currently the special field is only used for progress bars. We show them in many of our screens. On our toolbars, we use them for the morale, strength, and fatigue backgrounds. A sample progress bar looks like this: **progbar:#moralemin-#moralecur-#moralemax** This defines a progress bar. The sprite for a progress bar is assumed to be the proper size for when the progress bar is fully extended. The program will shrink the progress bar in order to show values that are smaller. Our progress bars currently only work left to right, we do not support up and down progress bars. Notice the three values used. These are variables that can be found in the **variable reference** document. All of the variables are preceded by the **#** sign. This means that the variable will be read and used as a number. The first variable is the value for the left side of the progress bar, the last variable is the value for the right side of the progress bar. The middle variable is the current value. The program will automatically calculate the length of the progress bar based on these three values. Don't forget to put the - between the values and make sure that there are no spaces. The code that reads these is not very robust.
U: Show Condition	The show condition column is what it says; it will determine whether or not to draw this graphic. Please see the **variable reference** document on how to create condition functions. This can be used for any graphic; it does not have to be a button.
V: Highlight Condition	This works the same as the **Show Condition** column, except that this determines whether or not to show the button in the highlighted state.

The Tooltext.csv

The **tooltext.csv** file defines all of the text that is written on the toolbars.

NOTE: Rather than writing exactly the same definitions for multiple toolbars, this file and the toolbar.csv file support inheritance. In addition to specifying specific information for a toolbar, you can say that a toolbar inherits from another, meaning that it will get all of the same information than another toolbar has.

Column	Explanation
A: Toolbar	This is the name of the toolbar that this text is displayed on. The toolbar name is used in toolbar.csv and **unitcommon.csv**. These names much match exactly to get the toolbars correctly displayed. If the name **all** is here, the text will be applied to all toolbars.
B: Font	The font names are from the **graphics/fonts** folder. These files are created by engine specific tools, so fonts cannot be created by modders. The font names come from the **fonts.csv** file where modders can add their own fonts.
C: Place(C/L/R)	This describes the placement of the text. **C** means that the text will be centered on the coordinates. **L** means that the text will be written to the left of the coordinates, and **R** means that that text will be written to the right of the coordinates. If this is an inheritance row, then the name of the toolbar from which you want to inherit should be placed here.
D: Scale	This capability is no longer supported.
E: Color R	The color of the font is describes in RGB format. This is the **R** or red value of the color.
F: Color G	The color of the font is describes in RGB format. This is the **G** or green value of the color.
G: Color B	The color of the font is describes in RGB format. This is the **B** or blue value of the color.
H: X Coord	This is the X coordinate is which to draw the text based on the placement value. These values are based on our default resolution of 1024x768. These are only 2D values.
I: Y Coord	This is the Y coordinate is which to draw the text based on the placement value. These values are based on our default resolution of 1024x768. These are only 2D values.
J: Variable Names	This is the text that will be written in this position. You can put a game variable here or just write text. All variables are preceded by a **$** (for string) or **#** (for number). If neither of these symbols is found, the game will assume that it's just straight text and print the exact text at the coordinates. You can find a list of all variable in the variables reference section. Note that even though the variable may be displaying a number, use the **$** symbol because to display something it must be a string. The **#** symbol is only used in calculations and would not make sense in this file.

Mod a Map

You can't build a new map in ***Take Command*** but you can modify certain aspects of existing ones. This section explains how to do it.

The Map Layout

The maps in ***Take Command*** are actually defined by a set of four map files. As an example, the 2nd Manassas 2ndM_28th map set consists of the following files:

2ndM_28th.lsl	The **.lsl** map file is used by our 3D engine. This file cannot be edited at this time.
2ndM_28th.csv	The **.csv** map file contains all of the information about the terrain. It defines the forests, movement speeds, defensive bonuses, etc. This file is covered in more detail in the **map_name.csv** section.
2ndM_28th.tga	The **.tga** map file is the grayscale of the map. This file is used to define line of sight areas. All grayscale values in this map are defined in the csv map file described above.
2ndM_28th_MM.dds	The **MM.dds** file (or mini-map file) is used for the in-game strategic mini-map. This is a great file to print off and use to devise your plans for any particular scenario. It contains information that is useful for gaining a better understanding the area of operations.

NOTE: If you open the tga map file with an image editing program, you will notice that North is not up. These tga map files are rotated by the game engine when loaded.

NOTE: The **.lsl** file cannot be modded. The **.csv** file can be edited in the same way that the other .csv files are edited, using a spreadsheet program. The **.tga** and **.dds** files must be edited using a higher end graphics editing program such as PhotoShop.

The Map_Name.csv

Each map has its own csv file that defines the map. The .**csv** files for the maps are in the **\SDK\maps** directory. Just like all other csv files, this can be overridden. You just create a maps directory under you scenario folder and place the **map_name.csv** file there. You replace 'map_name' with the name of the map.

The map csv file is based on the grayscale map (**.tga**) that comes with every map. The grayscale map is just a bit map with a value between 0-255 for each pixel. The maps are 2048x2048 pixels. The map designer creates the grayscale and then takes all of the values that they used and puts them in the map csv file to define them.

Since you can easily modify the grayscale and the csv files, you can redefine the maps that shipped with the game. You can move forests, although the terrain image will not move, you can redefine defensive bonuses and movement speeds across different terrain types. You can rename terrain. This does not give the freedom of a map editor, but it's a start.

The **map_name.csv** file is broken into three parts. The first part is the map definition section. This basically defines all of the colors used in the grayscale. The second part is for defining the ambient sounds. To do an ambient sound, we just place a terrain sprite at a specific location on the map and attach a sound file to it. Since this file is broken into two pieces, there can't be any blank rows in either table. When the first blank row is found, the engine assumes that the next table has started. The third part is the list of possible objective sites. This is used by the game engine to select objectives for an Open Play Scenario.

Terrain Table Brush

Column	Explanation
A: Terrain	This is the name that will show up in the toolbar when a unit is on this grayscale value. It will be filled in the $terrain variable.
B: Grayscale	This is the value between 0-255 that this row pertains to.
C: Move Rate	This value is added to the formation movement rate modifier and then multiplied by the base movement rate of the unit to provide their current in game movement rate.
D: Density	This defines how to fill in this grayscale value with terrain sprites. All of our terrain features are filled in dynamically during level load. The lower this number is, the denser the foliage will be filled in. The higher this number is, the lower the foliage density will be. Think of this as a random number. Every time a grayscale of this color is found, the computer will pick a number between 1 and this number on how many pixels to skip before it places another terrain sprite.
E: Visibility	This is the distance in yards that you can see through this type of terrain. For example, if woods have a visibility value of 40, then a unit can see 40 yards into the woods. Any units more than 40 yards away would not be visible.
F: Def Bonus	This is the defensive bonus that this terrain gives to units on this terrain type. If the flag bearer is on this terrain, then the entire unit is defined as being on that terrain. This is because it would be crazy to calculate separately for every man. It would just use up too much CPU time. The defensive bonus is a random number between 1 and 100. If that random number is more than this def bonus number, a hit is made. If it is equal to or less than, then the target is missed. So the greater the number here, the higher the defensive bonus. A value of 0 is no bonus.
G: Fatigue	Every so many ticks (defined in **tables.csv**), a fatigue check is made. When that check is made, if the unit is marching, they will lose this many fatigue points. The point system is defined in **levels.csv**. If the unit is running, the fatigue penalty is taken from **tables.csv**. These are negative numbers and are added to the fatigue value. If you want a unit to get rested while marching over certain terrain, you could put a positive number in here.
H: Can't Halt	This defines those terrain types that units are not allowed to stop on. A value of 1 means that units cannot stop on this terrain. We use this for rivers and streams. A value of 0 means that stopping is allowed.
I-N:Sprite1-6	If the density value is filled in, then these are the types of sprites that the level loader will use to fill in the terrain. You can have up to six different types to add some variety. These names come from the **sprites.csv** file.
O: Draw Distance	This value is in yards and defines how close the camera has to be to this location to draw this sprite. If you leave this blank the engine will handle this for you. I don't believe we use this in any of our csv files, but the code is still there if you wanted to use it.
P: Make Transparent	This is a flag to tell the engine to make this terrain type transparent as you get within a specified distance. This is currently used only for trees in forest areas.

Terrain Table Sounds

Column	Explanation
A: loc X	This is the x coordinate of the sprite location.
B: loc Y	This is the y coordinate of the sprite location
C: dir X	This is the x value of the direction unit vector.
D: dir Y	This is the y value of the direction unit vector.
E: Sprite	This is the sprite name from the sprites.csv file.

F: Sound File	This is the terrain sound to play at this sprite location. This is how we do the ambient terrain sounds. This sound will be constantly played at this location. This sound is from the **gamesounds.csv** file.
G: Terrain	This column is used for reference only and is not read in by the game.

Terrain Table Objectives

This table is a list of objectives and their locations. This list is used to determine VP Sites (objectives) in Open Play scenarios. When a map is selected for Open Play, the game engine will randomly determine the objective sites to be used in that scenario. Try to place these objectives at crossroads and in open fields, but never in the woods.

Column	Explanation
A: ID Names	This is the name that will be displayed for this objective
B: X	This is the x coordinate of the objective site.
C: Z	This is the z coordinate of the objective site.

The Gamesounds.csv

All sounds used in the game must be in **wav** format and be located in the **\Sounds** directory. All sounds that are used in the game must be predefined in the **gamesounds.csv** file and then referenced in the other files by the name. The **gamesounds.csv** file defines those sounds that are used in game. The **mainsounds.csv** file defines those sounds that are used in the pre-game menus.

NOTE: There must be one sound with the name of **click**. This is the default sound for button clicks. This sound can be over ridden in the appropriate file, but this default must exist in case there is no specific sound defined for a particular button.

Column	Explanation
A: Name	The name of the sound. This name is used in all of the other csv files that need access to the sound.
B: File	This is the name of the wav file that must exist in the **Sounds** folder. This will attach the name to this file.
C: Min Dist	This is the minimum distance in yards away from the sound that the max volume of the sound will be heard. The sound is played at the highest volume at the minimum distance all the way to the actual location.
D: Max Dist	This is the maximum distance away from the location of the sound that you will hear the sound. The sound is lowered in volume from the min to max dist. 0 - Min Dist yards : highest volume Min Dist - Max Dist yards: sound is gradually lowered to 0. Greater than Max Dist yards : no sound is heard
E: Loop	This is a 0 or a 1 defining whether the sound loops or not. Looping sounds are used for all ambient sounds and most state sounds (Marching, Standing, Shooting). Non-looping sounds are used for the button clicks and explosions.
F: Vol	This is the volume to play the sound. The volume is defined as a number between -10000 and 0. With 0 being the highest possible volume of the sound and -10000 being absolutely silent. No sounds are amplified; a value of 0 just plays at the normal volume of the wav file.

Modding the Take Command Combat Model

A model is an abstraction of a thing—and the model is static. A combat model is an abstract representation of combat—in a static form. In the case of ***Take Command***, this model happens to be Civil War combat from the perspective of a commander. The central focus of this model is on the command and control aspects of Civil War combat at the division level and higher. As such, the granularity of things gets "less focused" as you move from this center. All of it by definition is an abstraction, and there is more abstraction as you move away from the central focus.

The various .csv files discussed previously in this guide—and those that follow—form the essence of the Take Command combat model. Though there are sub-models of the activities of regiments and brigades in Take Command, they are optimized to support the central focus—the trials, tribulations, and challenges of a Commander at division level command and above.

A simulation is the exercising of a model over time—that's where the War3D engine comes in to play. Working hand in hand with the Take Command combat model, it is through the ***War3D*** engine that we are able to see the effects of the interactions between processes and variables within the combat model over time. (You can now go and impress your friends with this newly found information…or just tell them, "Hey—it's a really cool game! And you can change stuff in it if you want…know what I mean?")

This section explains the remaining parts of the ***Take Command*** Combat Model that don't fit so nicely into the previous sections.

The Levels.csv

I'm going to approach this file a little differently because it is very different from other files. The main idea behind this file was to allow the modder to add their own types of attributes to the fighting men without having to change the code. We accomplished this to a lesser degree. You can add new rows

representing new attributes, and you can then add new columns to the **units.csv** file to specify values for those new rows. So there really is no limit on how many attributes you can add to the fighting men. The limit is how they affect the game.

The following table defines the columns across the top of the file. These are the ways that you can affect the game in this file. Each column has a specific purpose and must exist. You cannot add to or subtract from the columns. When figuring out a column, you must find every attribute that affects that column, then find the specific unit's level in those attributes, then add up the values of that column to get your total.

The morale attribute (row 99) is the last hard coded row attribute. After that the order of attributes and number of attributes does not matter. To see what an attribute affects, just look across the rows and see which columns have values. You cannot add attributes to commanders; they are stuck with the first attributes listed (Initiative - Experience). The fighting men start with experience and continue to the end of the list. Note that commanders and troops share the experience. If you look at our **units.csv** files you will see this reflected in which columns are filled in to the right.

Column	Explanation
Attribute Definition	
A: Attribute Name	This is the name that is used to describe this attribute. Each attribute starts with a name and then the levels of the attribute.
B: Points	These are the points that are associated with each attribute level. Although they are really not used in the game at this point, it is still very important for them to be there. Certain aspects of the game expect them there.
C: Label	This is the text name of each level of the attribute.
Gameplay Columns - each of the following columns is a hard coded part of the game play	
D: Firing (1000 pts)	The values in this column are added to a total accuracy value for the fighting unit. The closer the value is to 1000, the better the accuracy is for the unit. The accuracy is the chance to hit a target with a properly loaded weapon.
E: Reload (1000 pts) - incorrectly called misfires in most files	The values in this column are added to a total reload skill value for the fighting unit. The closer the value is to 1000, the better chance that unit has of properly loading their weapon. A low value here would result in misfires or simply the inability to fire the weapon.
F: Melee (1000 pts)	The values in this column are added to a total melee value for the fighting unit. The closer the value is to 1000, the better chance this unit has to make a kill while engaged in melee combat.
G: Commander Radius Percentage	This is the percentage that is multiplied by the base commander radius for the command level. All of the values are totaled, then the result is used as a percentage to multiple against the base: 10 yds - Brig 25 yds – Div 50 yds - Corps 65 yds – Army So the higher this total, the larger the command radius would be for this commander. Therefore his morale bonus would affect more men.
H: Commander Morale Bonus	The values in this column are added to a total morale bonus value for the commander. This total is the number of points (not levels) of morale that will be added to fighting units that are within this commander's radius.
I: Rally Time	The values in this column are added to a total value that represents a number of seconds which is how often a commander gives out his rally points. The lower this total, the more effective the commander in rallying troops.
J: Rally Points	The values in this column are added to a total morale points value. Every time the rally time comes up this total is added to the morale of all troops within this commander's

	morale radius. The troops must be standing still and out of range of any enemy for this bonus to be added. There is also a cap as to how far any commander can rally troops.
K: Wheeling Locked	The values in this column are added to a total distance in yards. If the enemy is within this distance to the troops, than the troops are less likely to wheel. Less experienced troops would have a larger distance because they would not be able to wheel while the enemy is close.
L: Commander Stance Modifiers	The total of these values represent how commanders would interpret orders. We tend to have out low numerical orders being more cautious while our higher numbers are more aggressive. So by modifying these values you can increase/decrease the aggressiveness of certain personality types.
M: Commander Support Distance	This is how close to the lines the commander will actually get. Certain commander's may get closer and have a higher chance of getting killed, but also have a higher chance of constantly giving out their morale bonus. Other commander's my want to stay back. If this total is negative, it is added onto the commander's radius, if it is positive it represents the exact number of yards behind the men that the commander will stand.
N: Units Run or walk for Ammo	The values in this column are added up and if the total is greater than 0, they run, otherwise they walk.
O: Commander's Run or Walk	The values in this column are added up and if the total is greater than 0, they run, otherwise they walk. This applies when a commander is trying to rally their men.
P: Ammo Taken From Dead	This total represents the amount of bullets that can be taken from the dead within their own units.
Q: Max Ammo Per Man From Ammo Wagon	This is the maximum amount of ammo that the ammo wagon will give to one man within a fighting unit. Lower this value to conserve the ammo.
R: Unit Morale Bonus Radius	The values in this column are added to a total radius in yards. This represents the radius in which this unit will give out its own morale bonus.
S: Choose VP Bonuses	This should only be set in one attribute as this cannot total correctly. This represents the AI preference when choosing VP sites to conquer: 1 = morale 2 = fatigue 4 = ammo This value works like a bit field in that you can just add the numbers to use multiple types. So if you want certain commanders to go after morale and ammo, you could put in a 5.
T: Max Morale Modifier	The cap on morale is determined by different means. It has to do with your commander and the number of men that you have left compared to what you started with. This total represents a little bonus that will up the cap. The cap being the highest morale that any commander can rally you to. So if you like your commander, he can up your morale a little higher.
U: Firing Time	The values in this column are totaled and then added to the base firing time of the weapon that the fighting unit is using.
V: First Aid	In some small way this affects the OOB game screens that show your total casualties. This will determine how many dead vs. wounded. This is not really effective in this game, but in future games you'll be able to get some of those wounded back on the battle field.
W: Limber Time	The values in this column are totaled and then added to the base limber time of the weapon that the fighting unit is using.
X: Cover Stand Time	The values in this column modify how long it takes a regiment to get back to their feet from the take cover (prone) position. As you can see, this timing is very important when being approached by a valid enemy target.

The Tables.csv

The **Tables.csv** is where we threw everything that didn't have another home. We really tried not to hardcode any values. We didn't do too bad a job, but there are a lot of hard coded values in the game. A lot of times when we had something new to code, we created a table in the **tables.csv**.

This prevented us from hard coding the values and also provided a place to test the game with different sets of numbers.

The very first thing to notice of any table is the table name. That is on the first row of the table in the first column. I will list the tables in order here, but the program loads in the tables by the table name, not by the order.

NOTE: You can place comments between tables by not filling in the first column. This goes for most csv files. As long as the first column remains blank, the game will skip that row.

Table: Morale.

The morale table defines how much morale is lost when one man is killed. This morale loss only happens at certain percentages. Those percentages are listed on the left side of the table. So morale is not lost EVERY time a man is killed. Morale is only lost when a man is killed and by that death the total percentage of men dead (versus the starting strength) falls to be equal to one of the percentages listed in the left column.

Where	Explanation
Left most column	This is the percentage dead and is figured by taking the total dead and dividing by the total starting strength of the regiment. This column should be in ascending order. In the default table the first value is 2. This means that when the unit has lost two percent of it men, it will receive the first morale penalty from this table.
Top rows	There are two top rows. The first row is the experience value. The second row is the experience label. These rows are not read by the game. What is important to remember is that if you add new experience levels to the game by modifying the **levels.csv** file, you must add an equal number of columns here to reflect your change.
Value (min-max)	By referencing the percent dead, then the experience level of the regiment, you will find a value of type (min-max). This is the min and max morale penalty that will be taken. The game will pick a random number between and including these values.

Table: Fatigue.

The fatigue table defines how, when, and how often a unit loses or gains fatigue. Fatigue figures into numerous calculations throughout the game, and it all starts here. It is important to note that the base fatigue of Marching does not get the fatigue penalty from this table, but from the **map_name.csv** file where it takes into account the terrain that you are marching on. It is interesting to note that only marching fatigue is affected by terrain, if the unit is not marching, their fatigue is not directly affected by terrain. How are they affected? Using running as an for example, Units are slowed down and therefore their already high penalties for fatigue will come into play more often.

Where	Explanation
Left most column	These are the unit states; you should never modify these because they cannot be changed by the modder. They are hard coded in the game and their order is as important as their quantity.
Points	These are the number of points that will be added to the fatigue value of the regiment. You can put positive or negative numbers here, depending on how you want this activity to affect fatigue.
Seconds	Every x seconds, the fatigue will be modified.

Table: Fatigue Run.

The fatigue run table defines the movement penalty associated with running or charging at certain fatigue levels. It is important to note that if you add new fatigue levels to the **levels.csv** file, you must add those fatigue levels to this table.

Where	Explanation
Left most column	These are the fatigue levels, in the exact order and with the exact name as found in the **levels.csv** file.
Effect	This is the movement penalty that will be applied to the movement speed of the unit while trying to run or charge at the left fatigue level.

Table: Unit Morale Bonus.

The Unit Morale Bonus table defines how many morale bonus points one unit gives another by being near that unit. Units (regiments) feel much more secure by being around other units. No one wants to be left alone. The radius, or the proximity, that one unit has to be in relation to another unit is defined in the **levels.csv** file. If the units are close enough, or within the radius, then this table defines how many bonus points they receive. The morale bonus is based on experience and the number of men.

Where	Explanation
Left most column	This defines the number of men in the regiment giving the bonus. Any amount up to and including the number on the left of the row, will use this row for the bonus.
Top Rows	There are two top rows. The first row is the experience value. The second row is the experience label. These rows are not read by the game. What is important to remember is that if you add new experience levels to the game by modifying the **levels.csv** file, you must add an equal number of columns here to reflect your change.
Values	These are the number of points that this unit gives as a morale bonus. Look up the number of men on the left and the experience across the top to find the correct bonus.

Table: Grades.

The grades table defines all of the grades that can be won by a unit. The only grades that this does not include are those won by taking objectives. When a unit causes another unit to reach the state defined in the left column, they are awarded the points to their grade and the morale bonus. The unit that is caused to be in that state will lose the points in the grade column. Do not change the left most column; the order is defined in the game.

Where	Explanation
Left most column	This defines the different states that a unit can reach. One or more units will cause another unit to reach this state. Do not change the order or names of this column, as the order is defined in the game and is expected to be as presented here.
Grade	These are the number of points that will be awarded to the victorious units and taken away from the defeated unit for reaching the state defined on the left. Note that if more than one unit causes the defeated unit to reach that state, the points will be divided among all the units that have a kill in the defeated unit.
Morale	This is the morale bonus that will be received by the victorious units. Each unit causing the state will receive the full morale bonus.

Table: Retreat.

The retreat table defines the percent chance that a unit commander will get scared and order a retreat on their own. If the unit commander orders the retreat on their own, their commander will lose points for not watching his men closely enough. If the brigade commander calls a retreat before the regimental commander does, they will not lose points.

Where	Explanation
Left most column	These are all of the morale levels from the **levels.csv** file. If you modify the levels.csv file and add or take away morale levels, then you must also add or take them away from this column.
Top Rows	There are two top rows. The first row shows some comments; the second row is the experience label. These rows are not read by the game. What is important to remember is that if you add new experience levels to the game by modifying the **levels.csv** file, you must add an equal number of columns here to reflect your change.
Value	These values are the percent chance out of 100 that the regimental commander will call a retreat when that regiment FIRST reaches that morale level. There will be one check that occurs whenever a unit changes morale levels.

Table: Fallback.

The fallback table defines the percent chance that a regimental commander will get scared and order a fallback on their own. If the regiment commander orders the fallback on their own, their commander will lose points for not watching his men closely enough. If the brigade commander calls a fallback before the regimental commander does, they will not lose points.

Where	Explanation
Left most column	These are all of the morale levels from the **levels.csv** file. If you modify the **levels.csv** file and add or take away morale levels, then you must also add or take them away from this column.
Top Rows	There are two top rows. The first row shows some comments, the second row is the experience label. These rows are not read by the game. What is important to remember is that if you add new experience levels to the game by modifying the **levels.csv** file, you must add an equal number of columns here to reflect your change.
Value	These values are the percent chance out of 100 that the regimental commander will call a fall back when that regiment FIRST reaches that morale level. There will be one check that occurs whenever a unit changes morale levels.

Table: Elevation.

The elevation table defines accuracy bonuses for units that are at a higher elevation and accuracy penalties for units that are at a lower elevation. These values are added to whatever other accuracy bonuses or penalties they may have.

IMPORTANT NOTE: Right below the table name (Elevation) is the number of (rows-columns). If you change either of these values, you must change this value to reflect the correct amount; otherwise the game will not read in all of the data.

Where	Explanation
Left most column	This is the elevation difference in yards. This column is ascending so that the smallest difference is at the top.
Top Rows	Across the top is the distance the targets are away from each other in yards.
Values (bonus-penalty)	The values show the bonus and penalty for each elevation difference and distance combination. This value is added to or subtracted from whatever the current accuracy value is. The higher unit will get the bonus while the lower unit will get the penalty.

Table: Area Mod.

This table is used by the AI to determine where they want to attack. It is used by the Corps and Army Commanders to evaluate the field of battle during the initial strategy planning. It is broken into different sections for each type of evaluation and each type of commander style.

Where	Explanation
Left most column	These are the commander styles. If you add new styles to the **levels.csv** file, then you must also add them here. In the base table 0 is the most cautious personality and 4 is the most daring.
Top Rows	The top row lists the different types of situations that the AI will evaluate. Do not change this row at all, as it is expected in this format by the game.
Values (min-max)	These values represent the range of getting a random number. This random result will determine how many subordinate units this commander will commit to the particular rows type of attack.

Table: Capture.

This table is no longer used.

Table: Artillery Accuracy.

There are five of these tables. These tables define the base accuracy of the cannon/crew combination. The data supporting these tables comes from actual cannon target matches run by the North-South Skirmish Association.

Where	Explanation
Left most column	These are the gun crew quality rating. These cannot be changed.
Column B: group size	This is the size of a group of 10 rounds at 100 yards for this specific cannon type and crew quality.
Column C: Combat Stress Adjustment Factor	Nobody shoots as accurately in combat as they do on the target range. This factor is multiplied by the group size to give the Adjusted Group size.
Column D: Adjusted Group Diameter	This is column B times column C and gives the final group size. **NOTE:** This column is the only one actually used, the others are to show how this number was derived.

The Formations.csv

The **formations.csv** is one of the most complex csv files in the game. It has many rules that must be followed exactly.

The most important thing to remember when modifying or designing a formation is that they are real. When marching in formation, soldiers line up on other soldiers in that formation. A soldier aligns on the soldier to his right or the soldier to his direct front. This continues all through the formation until you get to the soldier carrying the colors. In the large formations he's lining up on someone too. "Dress Right"…

If you keep this in mind, creating formations can be fairly simple. Think of the formation that you want, then place the flag bearer, then the next soldier and keep going until you've filled in all of the slots, with each soldier lining up on another soldier lower in number than he is.

TAKE COMMAND formations can have a maximum of 100 sprites—the equivalent of 1,000 soldiers. The first sprite is in slot 1. This is the flag bearer and every formation must have one flag bearer. You then place the remaining 2-100 sprites in position. You do not have to define all 100 sprite slot positions if the formation you are designing will never contain that many sprites. For example, our commander formations only have two sprites, the flag bearer and the commander so there would be no reason to define all 100 slots.

The **formation.csv** file is broken into two sections for each formation definition. The first section is the first row of a formation definition. It defines the formation properties. The second section is a set of rows that define the formation locations. There can be no blank rows within a formation definition. The facing of a formation definition on the **formations.csv** is always towards the top of your monitor.

All units have formations that they can employ. These formations are defined using the **formations.csv**.

Column	Explanation
A: Form #	This is the name of a particular formation. This name will be used in other csv files to reference this formation. This name is used on the following csvs: Toolbar (func) **units.csv**, column O (Formation) **unitcommon.csv**, column W (Fighting Formation) **unitcommon.csv**, column X (March Formation) This is not a number, but the name of the formation. It is suggested that you some sort of naming convention because as you create more and more formations, they can get very confusing.
B: Rows	This is the number of rows or lines down that will make up the formation. You can fill this in after you have completely defined the formation by just counting the number of rows.
C: Cols	This is the number of columns or rows across that make up the formation. You can fill this in after you have completely defined the formation by just counting the number of columns.
D: Row Distance	This is the number of yards between each row of men. So as the men stand there will be this many yards in the front and back of each man.
E: Col Distance	This is the number of yards between each column of men. So as the men stand there will be this many yards to the left and right of each man.
F: Subformation	This is the name, from column A of this file, of the sub-formation to use. This is used for commander formations. For example our brigade commanders only have two people in their formation, the flag bearer and the commander. Once a formation has been filled in with all of the men for that unit, the game will then look to see if that unit has any subordinates. If it does, then it will place the subordinates in the remaining formation locations, and call the sub-formation for each of those subordinates.
G: Keep Formation	This flag determines if this formation is used for marching or if the unit should switch to their marching formation for marching. We use this flag to maintain skirmish formation.
H: Can Wheel	This is just a 0 or a 1 determining whether or not wheeling is allowed in this formation.
I: Can Fight	This is just a 0 or a 1 determining whether or not fighting is allowed while in this formation.
J: Move Rate	This is one of the two parameters that affect the movement rate. This number is added to the movement rate modifier of the terrain, from the **map_name.csv** file, and then this result is multiplied by the current movement speed of the unit and then added to the movement speed of the unit. So make this a percentage decimal. For a 20% increase in speed while in this formation, use 0.2, to decrease the speed by half while in this formation, use -0.5.
K: AboutFace	This one is a little confusing. The formations can about face and the AI does use this functionality to turn around quickly. Also if you click directly behind a unit, they will usually about face to the location. If this value is 0, then the unit will not be able to about face from this formation. If you want to support about face from this formation, this number should be the position in your formation where you want the flag bearer to end up after the about face is completed. So after you

	have your formation designed, take a look at where all of the numbered slots are, then think of the formation as turned around, where would the flag bearer be? Now we don't have our columns allow about face, but if we did, then this value would have to be 100. Meaning for a good about face to occur, then the unit must occupy the same exact space that they did before the about face was called. So you would want to move the flag bearer all the way down to the last slot and have everyone else line up behind him.
L: Artillery Formation	This is the sub formation for artillery. This works exactly like the other sub formation, but applies to artillery. This is really only used for the division formations. Divisions are the meeting place for each type of unit (cavalry, artillery, and infantry). The cavalry and infantry share most formations, but the artillery does not. So if a division commander calls an entire division formation, the artillery will use this formation instead of the other sub formation defined earlier.
M: Closest Enemy	This column determines the number of yards that men in this formation will allow an enemy to approach this formation. If an enemy gets closer than this value, then the unit will run a short distance away.

After you have defined the formation above or at least filled in as many parameters as you can, it's time to design the actual formation. This is a list of numbers from 1 to 100 placed on the grid exactly how you would want them to look in the formation. This is a good reason to use a spreadsheet program to edit these csv files.

1. Place your flag bearer. It's easier to work on a blank spreadsheet and place your flag bearer somewhere where there is a lot of room all around. So type a 1 in a box and you have your first formation consisting of 1 flag bearer.

2. Place your number 2 guy. Remember that he must be able to line up to the left, right, front, or back of the 1. He cannot be diagonal from the 1. You may leave blanks to the left, right, and front, but not in the back. So if 2 is behind 1, he must be directly behind 1. But if he is to the left, right, or front, he may skip some blocks and does not have to be directly next to 1.

3. Continuing placing men until you have placed all you want, and remember—no more than 100.

4. Also remember that the numbers you place are as if there were that many sprites in the formation. So you start with just one side of the formation and then move to the other, when there are less than max men in the formation, it will look unbalanced. Always balance the formation as you add new numbers.

5. We have designed many formations in this file, they all work. So use our examples to help you on your way. Also if something that I have written is too cryptic, our example should hopefully clear it up.

6. When the game loads in the formation, it looks in a specific order to determine who to line up on for each slot. First it looks to see if it's behind a lower number, then it looks to see if it's to the right of a lower number, then to the left, and finally ahead. It does not look to find the lowest number that it's around; it looks in the above order. If it cannot find a lower number, then the formation fails.

Special Situation	Explanation
(row dist - col dist - sprite)	Before every number in your formation design, except number 1 (flag bearer) since it would not make sense, you can specify specific properties that affect that formation slot alone. These additional properties will override anything else and will only affect the formation slot number that they precede. For example, the following is the entry for slot 2 in the formation. **(10-0-0)2** It overrides the row distance and the column distance for this slot. We use this often in the larger formations. In the division formations we want the commander close to the flag bearer, but we want

	large gaps between the bigger brigade formations. So we set the default distance for the formation to a large number, then we override the commander slot with a smaller distance, so he's closer to the flag bearer. It does not make sense to fill in both the row and column distance, since each slot only lines up on one other slot, so you must read your design and determine which slot this slot is lining up on (using step 6 above) and then determine if it's the row or column distance that you must modify.
row dist	The row distance, ahead or behind, in yards. A value of 0 uses the default.
col dist	The column distance, to the left or right, in yards. A value of 0 uses the default.
sprite	Generally, you should set this number to 0. However, it is functional. Currently, the ***War3D*** game engine uses this number as the index to the list of uniform sprites found in the **unitcommon.csv**. In that file you can define up to six different sprites that will be used randomly to populate a formation. This sprite number, if set to 1-6 will use this specific sprite set (as enumerated on the **unitcommon.csv**) for this particular formation slot. Since these sprites are used randomly throughout the formation, other slots may also use this sprite. If you want to make the use of a sprite exclusive to a particular formation slot, then all the formation slots should be defined with a specific uniform sprite number.

Miscellaneous

The following csv files deal with the graphics user interfaces, the game sound files, and other visual effects contained in the Take Command game series.

The Mainscreens.csv

The **mainscreens.csv** file is hard to describe. It is best understood by just opening it up and taking a look at what is in it. Your familiarity with playing the game will provide some clues that will help you to understand much of what you see here.

Column	Explanation
A: Type	This lists the type of functionality required/desired for a particular gamescreen. The type "NEW" starts a new screen definition. Example: NEW Main MainScreen.dds 0 0 0 1024 768 1024 1024 $lastscreen
B: Screen	This is the name of a defined mainscreen.
C: Targa	This column contains either the name and file type for a graphics file or specific text entries used on a defined mainscreen.
D: FONT-L/R/C-R-G-B	name-justified-R-G-B
E: Run Game	This column entry tells the game engine what to do while the called mainscreen is active. 0 = Game paused; 1= Game runs & drawn; 2= Game paused & drawn
F: X Coord	This is the distance in pixels from the bottom of the screen that the top left hand corner of a graphic or text entry will be displayed.
G: Y Coord	This is the distance in pixels from the left side of the screen that the left edge of a graphic or text entry will be displayed.
H: Width	This is the width dimension of a graphic in number of pixels.
I: Height	This is the height dimension of a graphic in number of pixels.
J: X Source	On our .dds screens, we keep all graphics that show up on the screen in the same .dds file. The x & y source columns are their location in that file. For every line but the new line, this is the location of the upper left hand corner of this graphic in the mainscreen.dds file.
K: Y Source	On our .dds screens, we keep all graphics that show up on the screen in the same .dds file. The x & y source columns are their location in that file. For every line but the new line, this is the location of the upper left hand corner of this graphic in the mainscreen.dds file.
L: Draw Condition	The conditions that must exist in order for a graphic or text to be displayed/written to the screen in question.
M: Exec Condition	The execute condition is a conditional statement that allows the function of the button to occur. If the condition fails it will not execute the function of the button.
N: Function	This is the game function called when the defined button is selected by the player.
O: Depends1	The game function called depends on this additional condition.
P: Depends2	The game function called depends on this additional condition.

The Gamescreens.csv

The **gamecreens.csv** file is hard to describe and is best understood by just opening it up and taking a look at what is inside. Your familiarity with playing the game will provide some clues that will help you to understand much of what you see here.

Column	Explanation
A: Type	This lists the type of functionality required/desired for a particular gamescreen. The type "NEW" starts a new screen definition. Example: NEW Main MainScreen.dds 0 0 0 1024 768 1024 1024 $lastscreen
B: Screen	This is the name of a defined gamescreen.
C: Targa	This column contains either the name and file type for a graphics file or specific text entries used on a defined gamescreen.
D: FONT-L/R/C-R-G-B	name-justified-R-G-B
E: Run Game	0 = Game paused; 1= Game runs & drawn; 2= Game paused & drawn
F: X Coord	This is the distance in pixels from the bottom of the screen that the top left hand corner of a graphic or text entry will be displayed.
G: Y Coord	This is the distance in pixels from the left side of the screen that the left edge of a graphic or text entry will be displayed.
H: Width	This is the width dimension of a graphic in number of pixels.
I: Height	This is the height dimension of a graphic in number of pixels.
J: X Source	On our .dds screens, we keep all graphics that show up on the screen in the same .dds file. The x & y source columns are their location in that file. For every line but the new line, this is the location of the upper left hand corner of this graphic in the mainscreen.dds file.
K: Y Source	On our .dds screens, we keep all graphics that show up on the screen in the same .dds file. The x & y source columns are their location in that file. For every line but the new line, this is the location of the upper left hand corner of this graphic in the mainscreen.dds file.
L: Draw Condition	The conditions that must exist in order for a graphic or text to be displayed/written to the screen in question.
M: Exec Condition	The execute condition is a conditional statement that allows the function of the button to occur. If the condition fails it will not execute the function of the button.
N: Function	This is the game function called when the defined button is selected by the player.
O: Depends1	The game function called depends on this additional condition.
P: Depends2	The game function called depends on this additional condition.

NOTES: MAINSCREENS & GAMESCREENS

X,Y coord: The game screen is drawn to 0,0. On width and height, the screen is 1024x768 pixels X&Y source, for the "NEW" line, this defines the dimensions of the dds file

BUTTON 373 687 289 36 317 821 exit

This button will be drawn at 373,687 based on the location of this screen (not the entire game window, though they are the same in this example), the width and height of this button is 289x36 pixels. It is in the file <**mainscreen.dds**>. It will be drawn like this on the game screen. We don't support shrinking or stretching here.

The Mainsounds.csv

All sounds used in the game must be in wav format. All sounds that are used in the main menus must be predefined in the **mainsounds.csv** file and then referenced in the other files by the name. The **gamesounds.csv** file defines those sounds that are used in game; the **mainsounds.csv** file defines those sounds that are used in the pre-game menus.

Note: There must be one sound with the name of **click**. This is the default sound for button clicks. This sound can be over ridden in the appropriate file, but this default must exist in case there is no specific sound defined for a particular button.

Column	Explanation
A: Name	The name of the sound. This name is used in all of the other csv files that need access to the sound.
B: File	This is the name of the wav file that must exist in the **Sounds** folder. This will attach the name to this file.
C: Min Dist	This is the minimum distance in yards away from the sound that the max volume of the sound will be heard. The sound is played at the highest volume at the minimum distance all the way to the actual location.
D: Max Dist	This is the maximum distance away from the location of the sound that you will hear the sound. The sound is lowered in volume from the min to max dist. 0 - Min Dist yards : highest volume Min Dist - Max Dist yards: sound is gradually lowered to 0. Greater than Max Dist yards : no sound is heard
E: Loop	This is a 0 or a 1 defining whether the sound loops or not. Looping sounds are used for all ambient sounds and most state sounds (Marching, Standing, Shooting). Non-looping sounds are used for the button clicks and explosions.
F: Vol	This is the volume to play the sound. The volume is defined as a number between -10000 and 0. With 0 being the highest possible volume of the sound and -10000 being absolutely silent. No sounds are amplified; a value of 0 just plays at the normal volume of the wav file.

The Gamesounds.csv

All sounds used in the game must be in wav format. All sounds that are used in the game must be predefined in the **gamesounds.csv** file and then referenced in the other files by the name. The **gamesounds.csv** file defines those sounds that are used in game; the **mainsounds.csv** file defines those sounds that are used in the pre-game menus.

Note: There must be one sound with the name of **click**. This is the default sound for button clicks. This sound can be over ridden in the appropriate file, but this default must exist in case there is no specific sound defined for a particular button.

Column	Explanation

A: Name	The name of the sound. This name is used in all of the other csv files that need access to the sound.
B: File	This is the name of the wav file that must exist in the **Sounds** folder. This will attach the name to this file.
C: Min Dist	This is the minimum distance in yards away from the sound that the max volume of the sound will be heard. The sound is played at the highest volume at the minimum distance all the way to the actual location.
D: Max Dist	This is the maximum distance away from the location of the sound that you will hear the sound. The sound is lowered in volume from the min to max dist. 0 - Min Dist yards: highest volume Min Dist - Max Dist yards: sound is gradually lowered to 0. Greater than Max Dist yards : no sound is heard
E: Loop	This is a 0 or a 1 defining whether the sounds loops or not. Looping sounds are used for all ambient sounds and most state sounds (Marching, Standing, Shooting). Non-looping sounds are used for the button clicks and explosions.
F: Vol	This is the volume to play the sound. The volume is defined as a number between -10000 and 0. With 0 being the highest possible volume of the sound and -10000 being absolutely silent. No sounds are amplified; a value of 0 just plays at the normal volume of the wav file.

The Effects.csv

The **effects.csv** file defines all of the explosions and smoke effects visible during gameplay. These effects are basically a pcx file played a variety of ways. The pcx file uses solid black for the alpha channel. All of the pcx files referenced in this file must be found in the **graphics/effects** folder.

NOTE: There must be one effect that is named **smoke**. This name is hardcoded for use at the end of all barrels of all weapons when that weapon is fired.

Column	Explanation
A: Name	This is the name of this effect. This name is referenced in other csv files that need access to an effect.
B: Particle File	This is the name of the pcx file for this effect. This file uses solid black as the alpha channel.
C: Alpha Start	When an effect is played in the game, it slowly disappears until completely invisible. This has nothing to do with the alpha channel of solid black in the pcx file for parts of the file that will never be seen. This value is between 255-0, with 255 meaning that the effect starts as completely solid and 0 meaning completely invisible. We start most of our effects at 225, meaning that they are semi transparent when first appearing. If someone wanted to make the smoke last longer, they could just set this value higher.
D: Alpha Step	This is how must alpha the effect will lose on every frame of the game. We set this value to -1, meaning that the alpha will be decreased by 1 each frame until it is completely invisible.
E: Scale Start	This is the scale of the effect. If a value of 0 is here, then the effect will be played at the exact size that it exists in the file. This value is a percentage meaning that if you put in 0.5, then the effect will start at half size.
F: Scale Step	This is how much the scale will change on each frame. This value will be added to the current scale on each frame to get the final scale of the effect.

	If you want the effect to increase in size over time, then use a positive number. If you want it to get smaller, then use a negative number.
G: Gravity	This sets the height of the effect over time. If you use a positive number here, the effect will appear to rise in each frame. A negative number will make it appear to sink towards the terrain.
H: Should the Particles Move	This determines whether or not the particle will move. Use a value of 1 to have the effect move, a value of 0 will keep the effect in its original position.
I: Emitter Frames	An emitter is a random bunch of very small pixels that will explode out from the area of the effect. These are randomly generated by the engine for this amount of frames. We only use one frame for a small amount of emitters, to get more just increase this number.
J: Should the emitter move	This determines whether or not the location of the emitter should move. The emitter being the source from where the small pixels are generated.
K: Num Particles	This is really the number of small pixels that the emitter should generate each frame in which they are being generated.
L: Random Particle Velocity	This is the speed that the small pixels should move. I don't know how to compare it to speed in the game, so just experiment until you get it looking like you want it to.

DESIGN TIP: From the **effects.csv** - Setting the length of an effect such as Smoke

"C" Alpha Start - (255 through 1) (int)
"D" Alpha Step - (-1 through -255) (int)

For each "frame", the Alpha step is added to Alpha to get the new Alpha until the Alpha reaches 0 and the effect disappears.

"E" Scale Start - no limits - starting scale (float)
"F" Scale Step - no limits - added to scale (float)

For each "frame", the Scale step is added to Scale to get the new Scale until the particle disappears (still based on Alpha).

All of these routines are grabbed right from the engine and the problem is obvious--you cannot set the time of the frames. The longest an effect can be on screen is 255 frames, so depending on how fast your computer runs determines how long you will see the smoke or any other effect. You can shorten the time of the effect, but you cannot make it longer.

There is a routine where if you set the alpha step to 0, you could set the number of frames for the effect, But of course this is tied to frame time as well and would not be the same across all computers due to varying CPU speeds. This routine is not supported in the ***War3D*** engine so it is not available to modders.

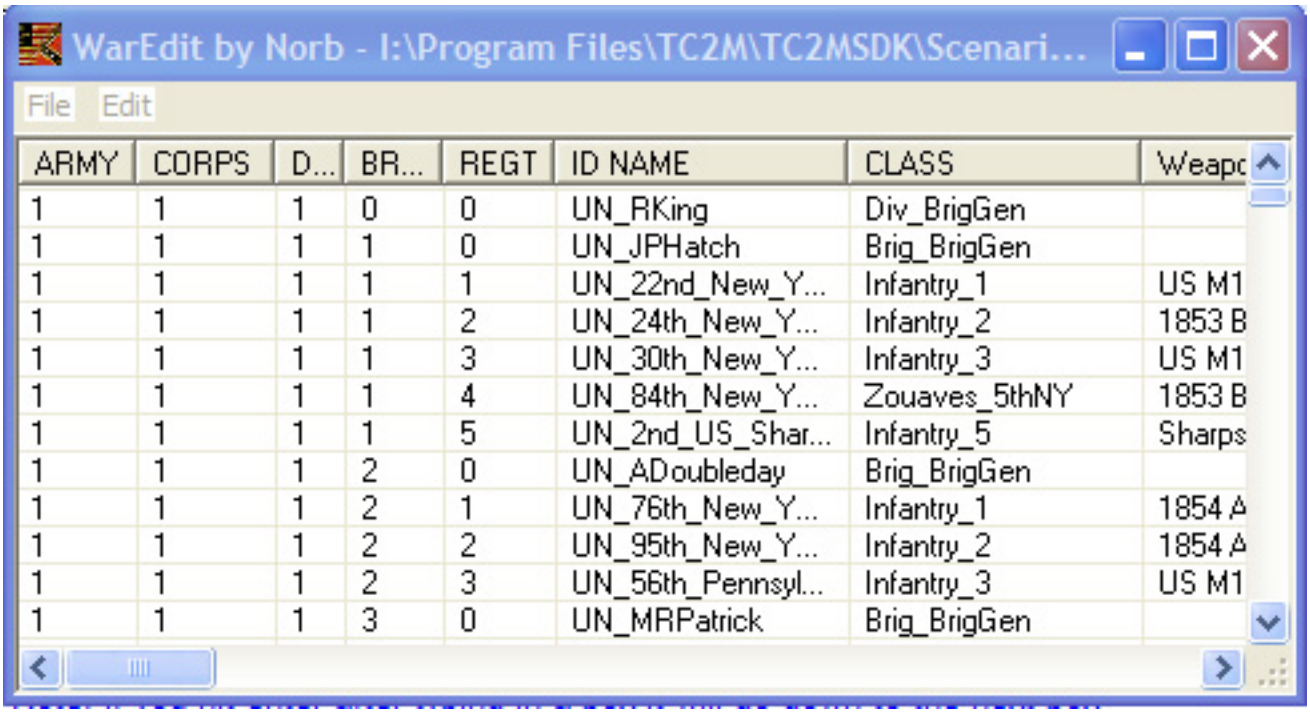
Using the WarEdit Utility

The ***WarEdit Utility*** was designed to allow the modder and scenario designer to easily edit the .csv files that support the ***War3D*** game engine. This section contains an overview of how to use it.

Installation.

Unzip the ***WarEdit Utility*** to main game directory. It will add **WarEdit.exe** file.

Double click on the WarEdit.exe file. It will open and display the last .csv file you were working on (the first time that you use it, it will be blank). You can several instances of the ***WarEdit Utility*** open on your desktop at the same time. This makes it easy to have all the information available from the different, interrelated—csv files.



ARMY	CORPS	D...	BR...	REGT	ID NAME	CLASS	Weapc
1	1	1	0	0	UN_RKing	Div_BrigGen	
1	1	1	1	0	UN_JPHatch	Brig_BrigGen	
1	1	1	1	1	UN_22nd_New_Y...	Infantry_1	US M1
1	1	1	1	2	UN_24th_New_Y...	Infantry_2	1853 B
1	1	1	1	3	UN_30th_New_Y...	Infantry_3	US M1
1	1	1	1	4	UN_84th_New_Y...	Zouaves_5thNY	1853 B
1	1	1	1	5	UN_2nd_US_Shar...	Infantry_5	Sharps
1	1	1	2	0	UN_ADoubleday	Brig_BrigGen	
1	1	1	2	1	UN_76th_New_Y...	Infantry_1	1854 A
1	1	1	2	2	UN_95th_New_Y...	Infantry_2	1854 A
1	1	1	2	3	UN_56th_Pennsyl...	Infantry_3	US M1
1	1	1	3	0	UN_MRPatrick	Brig_BrigGen	

The WarEdit Utility screen with a units.csv file open

WarEdit Utility Functions.

File/Open. Selecting this function will allow you to navigate through your computer directory to locate .csv files.

File/Save. Selecting this function will allow you to save the .csv file you are working on with the same name.

File/Save As. Selecting this function will allow you to save the .csv file you are working on with a new name.

File/Exit. Selecting this function will close the ***WarEdit Utility***. If you have changed the open file in any way, a pop-up window will appear asking you if you wish to save.

File/About. Selecting this function will display the version of the ***WarEdit Utility*** that you are using.

Edit/Insert Blank Row. Selecting this inserts a blank row above the selected row in the ***WarEdit Utility*** window. Use the <Insert> key on the keyboard to perform the same function.

Edit/Delete Row. This function allows you to delete the entire selected row from your .csv file. Use the <Delete> key on the keyboard to perform the same function.

Edit/Undo Delete Row. This function reinserts the last deleted row. Use the <**Ctrl+Z**> keys on the keyboard to perform the same function. The <Undo Delete Row> is function is unlimited. It will go back as many as you want.

Edit/Copy Row. This function allows you to copy the selected row to the clipboard. This function reinserts the last deleted row. Use the <**Ctrl+C**> keys on the keyboard to perform the same function.

Edit/Paste Row. This function automatically adds a row above the selected row and pastes the copied data into this new row. Use the <**Ctrl+V**> keys on the keyboard to perform the same function. You can keep adding the same cut data using the <Paste Row> function. You cannot undo the <Paste Row> function.

NOTE: You can keep adding the same data from the same copied row using the <Paste Row> function. You cannot undo the <Paste Row> function (use the Delete Row function to do this).

Edit/Cut Row. This function cuts the row from the .csv file and moves the rows that follow up one row. Use the <**Ctrl+X**> keys on the keyboard to perform the same function.

Edit/Import Unitlocs.csv. This function imports dir x, dir z, loc x, loc z data from the unitlocs.csv file into the currently open units.csv file.

Other Keyboard functions.

<**Home**> Key. This will move the view within the ***WarEdit Utility*** window to the top of the current file.

<**End**> Key. This will move the view within the ***WarEdit Utility*** window to the bottom of the current file.

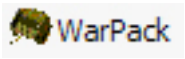
<**Top of Page**> Key. This will move the view within the ***WarEdit Utility*** window up one page in the current file.

<**Bottom of Page**> Key. This will move the view within the ***WarEdit Utility*** window down one page in the current file.

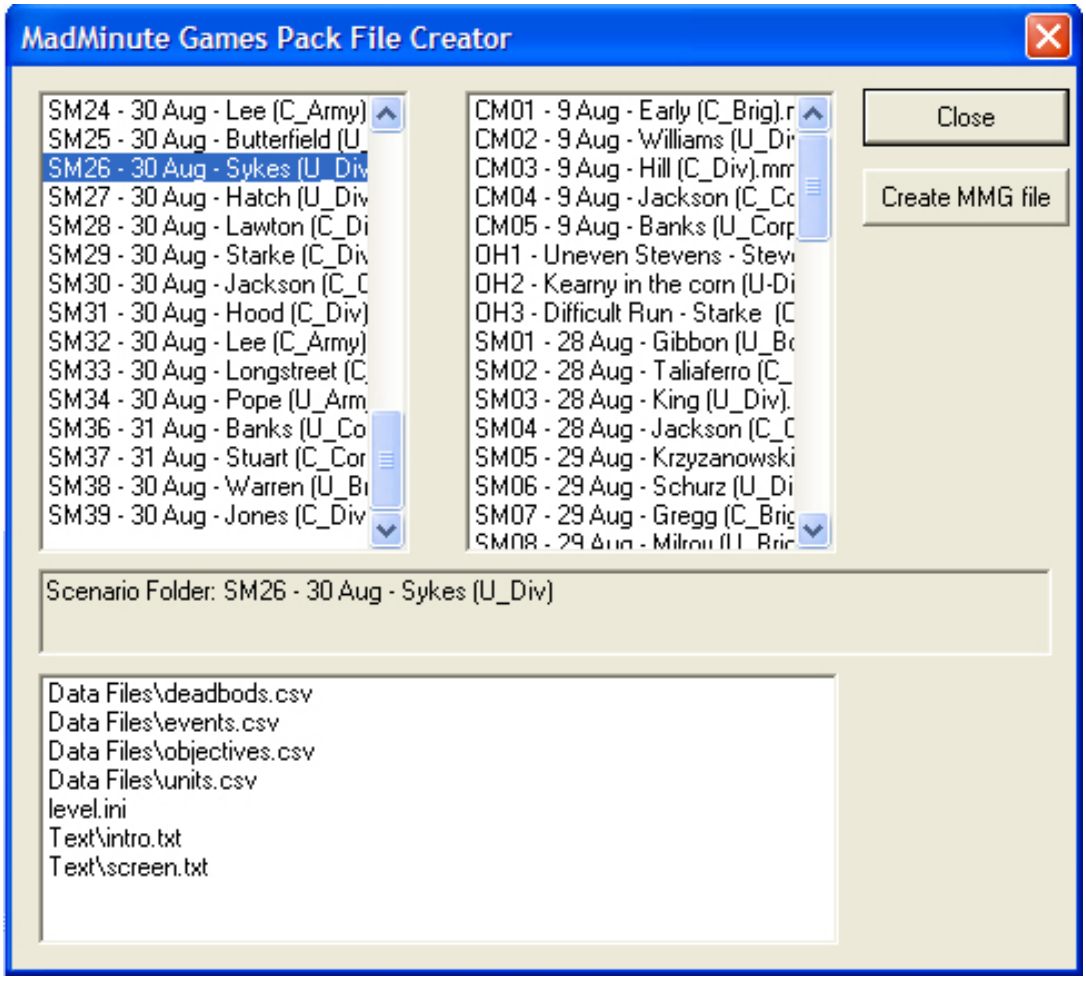
<**Esc**> Key. If you are typing in a cell and hit the <**Esc**> key, then the cell contents will revert back to the information that was in the cell before you typed.

<**Enter**> Key. If you hit the <**Enter**> key after typing in a cell, the focus of the ***WarEdit Utility*** will change to the next cell down in the column.

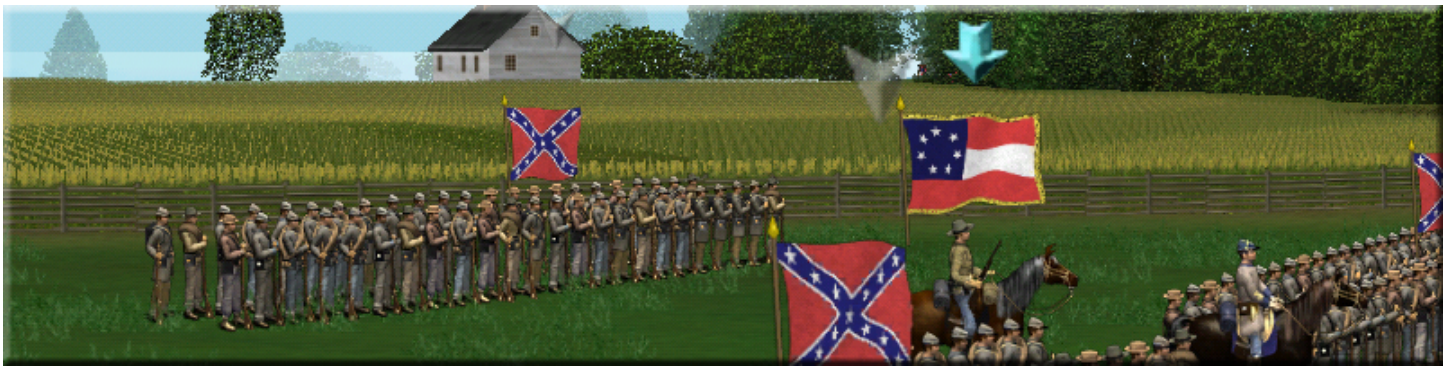
Using the WarPack Utility



Creating an .mmg file for distribution. If you wish to distribute a regular Custom Scenario or a Custom Scenario containing modded .csv files it is relatively simple. From your CD, copy the file **SDK\WarPack.exe** into the main game directory. Run this program by double-left clicking on the **WarPack** icon. If you have done this correctly, then the MadMinute Games Pack File Creator window will open. It looks like this:



The **WarPack Utility** looks in the **scenarios** directory and will show in the top left box all of the subdirectories present there (i.e. all Custom Scenario folders). The box at the top right of the window shows all the existing .mmg files that are in the scenarios directory. In the example above, all of the scenario folders from the SDK have been placed in the **scenarios** directory and scenario **SM26 -30 Aug – Sykes (U_Div)** has been selected. The box at the bottom of the window displays all the folders and files contained in the Sykes Custom Scenario. To create a new .mmg file of this scenario, click on the <**Create MMG file**> button. This file will automatically be created in the **scenarios** directory. This compressed file (about 40-150 KBs for an average Custom Scenario) can be easily shared among other players as an email attachment. Select the <**Close**> button to exit the WarPack Utility.



On-Line Help

Visit our website **http://www.MadMinuteGames.com** for additional information about TAKE COMMAND, including, additional modding information, downloads, game reviews, updates, patches and user created scenarios by players from all over the world.

Also, be sure to drop by our helpful forums for answers to any questions you may have and to learn hints, tips, and strategies on how to beat those darn Rebs/Yanks. You can visit the MMG Forums at:

http://www.madminutegames.com/MadMinuteBB/index.php

A Note from The Editor

The ***Take Command Developer's Guide*** is the result of the efforts of several folks. A large portion of the text was written by Norb Timpko and Jim Weaver with supporting efforts from Adam Bryant and Brett Schulte. As compiler of it all, I tried to transform it into "fine literature"—with the possibility of a movie option staring Bruce Willis as "Norb" and Keanu Reeves as "Adam". Towards that end—I have failed. As such, you will find several "voices" in this document. I didn't edit these out. They are reminders—to me anyway—that the game and this document are the result of the efforts of many people who love this lil' engine. If something is found to be in error within the guide the fault is mine—I was the editor.

So now you know enough to be dangerous. As an independent modder and volunteer scenario designer for the Take Command game engine for over a year, I'll offer one last thought. I must point out that the responsibility for debugging, testing, and supporting a mod or player-created Custom Scenario rests squarely with the builder—not MMG. Remember—MMG is only two guys working on this game series in their spare time. The Take Command Modding Community needs to "kick in" and help other modders through the "Norb Screens of Death"—yes your ever present friend—the CTD. Every minute you save Adam and Norb from having to answer a question puts us all another MadMinute closer to getting to Shiloh.

So hang tough, keep up the fire—and git moddin'.

v/r

Wrangler

"I haven't failed. I've just found 10,000 ways that won't work."—Thomas Edison

Appendix A: Variables Reference

There are a ton of variables that are filled in based on the currently selected unit. The entire variable list is below. These variables are used on the toolbars and the in game screens.

Variables are used in conditions for many screens and toolbars. A condition is a function that is either true or false. For example each toolbar graphic has a show condition column. In this column you can place a condition which will be evaluated when drawing the toolbar. If this statement evaluates to true, then the graphic is drawn, if it evaluates to false, then the graphic is not drawn. The following is the list of valid condition operators with examples. Note that all conditions use the **#** symbol before the variable name. This means that we need to use the variable as a number. We do not support the use of strings in our conditions.

#visible == 1 This is a basic equality operator. If the variable on the left is equal to the value on the right, then this condition will be true, otherwise it will be false.

#leaderbonus != 0 This is a basic inequality operator. If the variable on the left is not equal to the value on the right, then this condition will be true. Otherwise it will be false.

#moralelvl < 5 This is a basic less than operator. If the variable on the left is less than the value on the right, then this condition will be true. Otherwise it will be false. This can also be combined with an equal **=** symbol for less than or equal.

#ammoman > 57 This is a basic greater than operator. If the variable on the left is greater than the value on the right, then this condition will be true. Otherwise it will be false. This can also be combined with an equal **=** symbol for greater than or equal.

1 <= #ordernum <= 2 This is how we combine symbols for a range. It's the same as doing 2 of the above operators in one line. In this case **#ordernum** must be greater than or equal to 1 AND **#ordernum** must be less than or equal to 2 for this statement to be true.

Variable	Explanation/Where Used
terrain	Used as a string to display the type of terrain a Commander or unit is in. [gamescreens.csv]
name	Used as a string to display Commander and Unit names. [units.csv/gamescreens.csv]
name2	Used as a string to display Commander and Unit names. [units.csv/gamescreens.csv]
name3	Used as a string to display Commander and Unit names. [units.csv/gamescreens.csv]
men	Used as a string to display the number of men in a Command or Unit. [units.csv/gamescreens.csv]
quality	Used as a string to display the quality of a Commander or Unit. [units.csv/gamescreens.csv]
morale	Used as a string to display the morale of a Unit. [units.csv/gamescreens.csv]
fatigue	Used as a string to display the fatigue of a Unit. [units.csv/gamescreens.csv]

weather	Used as a string to display the weather above the game compass. Used as a string to display the morale of a Unit. [levels.csv/gamescreens.csv]
timer	
grade	Used as a string to display the quality of a Commander or Unit. [units.csv/gamescreens.csv]
disposition	Used as a string to display the disposition (moving, stationary) of a Commander or Unit. [gamescreens.csv]
orders	Used as a string to display the current orders (stance) of a Commander. [gamescreens.csv]
moralecur	current morale number [toolbar.csv]
moralemin	minimum morale number [toolbar.csv]
moralemax	maximum morale number [toolbar.csv]
fatiguecur	current fatigue number [toolbar.csv]
fatiguemin	minimum fatigue number [toolbar.csv]
fatiguemax	maximum fatigue number [toolbar.csv]
mencur	current number of men [toolbar.csv]
menstart	starting number of men [toolbar.csv]
mencas	total casualties [toolbar.csv]
attached	#attached == 0 means the Commander or Unit is attached. #attached == 1 means the Commander or Unit is detached from his/its Superior Commander. [toolbar.csv]
rallying	#rallying == 1 means the Unit is rallying. [toolbar.csv]
leaderbonus	#leaderbonus != 0 means the Unit is not receiving the Leader Bonus [levels.csv/toolbars.csv]
unitbonus	#unitbonus != 0 means the Unit is not receiving the adjacent Unit Bonus. #unitbonus != 1 means the Unit is receiving the adjacent Unit Bonus and will display an icon to that effect on the selected unit's toolbar [levels.csv/toolbars.csv]
visible	#visible == 0 means the Units cannot be seen by any enemy units. #visible == 1 means the Unit is Visible to Enemy units and will display an icon to that effect on the selected unit's toolbar [levels.csv/toolbars.csv]
defbonus	#defbonus == 0 means the Unit is not receiving the Defensive Terrain Bonus. #defbonus == 1 means the Unit is receiving the Defensive Terrain Bonus and will display an icon to that effect on the selected unit's toolbar [terrain.csv/toolbars.csv]
ammo	current ammo level [toolbar.csv]
ammostr	ammo total amount string [toolbar.csv]
ammostart	starting ammo level [toolbar.csv]
ammoman	ammo per man [toolbar.csv]
ammospr	ammo per sprite [toolbar.csv]
status	Used as a string to display the status of a Unit. [units.csv/gamescreens.csv]
moralelvl	#moralelvl is a number representing a unit's current morale level. [toolbar.csv]
fatiguelvl	#fatiguelvl is a number representing a unit's current fatigue level. [toolbar.csv]
targets	number of valid targets
blocked	#blocked != 0 means that no enemy targets are blocked by friendly units. #blocked != 1 means that A Target is Blocked and will display an icon to that effect on the selected unit's toolbar [toolbars.csv]
casstart	string that shows cas/start, 100/1000
weapon	Used as a string to display the type of weapon a Unit has. [weapons.csv/gamescreens.csv]
weapon2	Used as a string to display a second type of weapon a Unit has. [weapons.csv/gamescreens.csv]
weaponmax	Used as a number to display a weapon's maximum range. [weapons.csv/gamescreens.csv]
weaponopt	optimal distance for weapon
mendead	Used as a number to display how many men have been killed in a friendly Unit. [gamescreens.csv]
enemydead	Used as a number to display how many enemy men have been killed by a friendly Unit. [gamescreens.csv]
obj1	Objective name.
obj2	Objective name.
obj3	Objective name.

objdone1	Flag showing whether or not obj1 is completed.
objdone2	Flag showing whether or not obj2 is completed.
objdone3	Flag showing whether or not obj3 is completed.
flankfire	#flankfire==0 means the Unit is not receiving flanking fire. #flankfire==1 means the Unit is Getting Flanked and will display an icon to that effect on the selected unit's toolbar [toolbars.csv]
elevbonus	#elevbonus==0 means the Unit is not receiving the High Ground Bonus. #elevbonus==1 means the Unit is receiving the High Ground Bonus and will display an icon to that effect on the selected unit's toolbar. [tables.csv/toolbars.csv]
resting	#resting==0 means the Unit is not resting. #resting==1 means the Unit is Resting and will display an icon to that effect on the selected unit's toolbar. [toolbars.csv]
skill%dcur	current level of the skill (replace %d with number)
skill%dmax	maximum level of the skill (replace %d with number)
expcur	Current Experience (Quality) Level. Game engine variable.
expmax	Maximum Experience (Quality) Level. Game engine variable.
leadcur	Current Leadership Level. Game engine variable.
leadmax	Maximum Leadership Level. Game engine variable.
abilcur	Current Ability Level. Game engine variable.
abilmax	Maximum Ability Level. Game engine variable.
loycur	Current Loyalty Level. Game engine variable.
loymax	Maximum Loyalty Level. Game engine variable.
initcur	Current Initiative Level. Game engine variable.
initmax	Maximum Initiative Level. Game engine variable.
takecomm	#takecomm == 1 means Take Command from AI. #takecomm == 0 means Relinquish Command to the AI. [toolbar.csv]
ordernum	The index of the order currently assigned. [toolbar.csv]
formnum	This is the formation as derived from Column A of the formations.csv. The first formation named = 0, the second formation named = 1, etc. [formations.csv/toolbar.csv]
regstate	This is the movement state of a regiment. #regstate == 1 means stop, #regstate == 3 means run. #regstate == 4 means charge. #regstate == 6 means takecover. #regstate == 7 means fallback. #regstate == 8 means retreat. #regstate == 9 means advance. [toolbar.csv]
useroads	#useroads == 0 means the Commander or Unit has not been directed to use roads. #useroads == 1 means the Commander or Unit has been directed to use roads. [toolbar.csv]
artyammo	#artyammo == 0 means use canister. #artyammo == 1 means use solid shot. #artyammo == 2 means use shell. #artyammo == 3 means use shrapnel. [toolbar.csv]
artyfire	#artyfire == 0 means AI chooses target. #artyfire == 1 means Target: Troops. #artyfire == 2 means Target: Artillery. #artyfire == 3 means Conserve Ammo or Battery Hold Fire. [toolbar.csv]
artylimber	#artylimber == 0 means unlimber the gun. #artylimber == 2 means limber the gun. [toolbar.csv]
limcur	Time left to limber or unlimber. [toolbar.csv]
limmax	Total time to limber or unlimber. [toolbar.csv]
artyammo1	Used as a number to display the number of rounds of canister. [gamescreens.csv]
artyammo2	Used as a number to display the number of rounds of solid shot [gamescreens.csv]
artyammo3	Used as a number to display the number of rounds of shell. [gamescreens.csv]
artyammo4	Used as a number to show the number of rounds of shrapnel. [gamescreens.csv]
mounted	#mounted == 0 means the cavalry Unit is mounted. #mounted == 1 means a cavalry unit is dismounted. [toolbar.csv]
objtimer	Used as a string to define position and font for the objective timer. [tooltext.csv]
officer	Used as a string to display the commanding officers name on a pop-up window. [gamescreens.csv]
artyminrange%d	Used as a string to display the minimum range of a type of round of artillery ammunition. [gamescreens.csv]
artymaxrange%d	Used as a string to display the maximum range of a type of round of artillery ammunition. [gamescreens.csv]
rank	Rank level. [gamescreen.csv]

sellead1	Selected units 1st leader. [gamescreen.csv]
sellead2	Selected units 2nd leader. [gamescreen.csv]
sellead3	Selected units 3rd leader. [gamescreen.csv]
sellead4	Selected units 4th leader. [gamescreen.csv]
unittype	Used as a string to display a Unit's branch of service. [gamescreens.csv]
caskilled	Used as a number to display the number of men killed in a friendly Unit. [gamescreens.csv]
caswound	Used as a number to display the number of men wounded in a friendly Unit. [gamescreens.csv]
casdesert	Used as a number to display the number of men deserted in a friendly Unit. [gamescreens.csv]
mencaparty	Used as a number to display the number of men or guns captured. [gamescreens.csv]
reloadsec	Used as a number to display the number of seconds required by a Unit to reload its weapons. [gamescreens.csv/weapons.csv/levels.csv]

Appendix B: Event Commands Reference

There are many game commands in ***TAKE COMMAND***. They can be used to support toolbar buttons, menus, and in the events.csv file for any Custom Scenario. All commands are in lowercase. If you preface a command with the capital letter <**A>**, then this command will affect all units subordinate to the Commander receiving the command. So if you have a brigade Commander selected and you click a button that sends the 'run' command. Then only he runs. But if you have that button send the <**Arun**> command, then the brigade Commander and all regiments under him will run.

NOTE: All game commands won't necessarily work on every csv file. Study the various csv files in the SDK that use these commands to get a feel for what will work and where. When all else fails, don't be afraid to experiment.

Command	Explanation / Where Used
aboutface	Commands the selected Commander or Unit to turn 180 degrees from current facing. [events.csv/toolbar.csv]
advance	Commands the selected Commander's subordinates or the selected Unit to advance 50 yards towards an enemy Unit that is within open fire range. [events.csv/toolbar.csv]
Ahalt	Commands everyone in the named Commander's organization to stop right where they are, rather than in the currently designated formation of their Commander.
applyoptions	Game engine command. Applies options selected on the Options Game Screen [mainscreens.csv]
artyfireai	Commands the AI for the selected artillery battery to pick its own targets. [events.csv/toolbar.csv]
artyfirearty	Commands the AI for the selected artillery battery to fire only at enemy artillery Units. [events.csv/toolbar.csv]
artyfirehold	Commands the AI for the selected artillery battery to hold its fire. An artillery battery under the ***artyfirehold*** command will still fire canister in self defense if approached within canister range by an enemy unit. [events.csv/toolbar.csv]
artyfiretroops	Commands the AI for the selected artillery battery to fire only at enemy infantry and cavalry Units. [events.csv/toolbar.csv]
attach	Commands the selected Unit to attach itself to its Superior Commander. [events.csv/toolbar.csv]
begrandom	Game engine command to load an Open Play scenario. [mainscreens.csv]
charge	Commands the selected unit to charge a spotted enemy Unit. [events.csv/toolbar.csv]
close	Game engine command to close a screen. [gamescreens.csv]
closelevel	Game engine command to close a level. [gamescreens.csv]
courier	Commands the named Commander to send a message to the player's Commander. [events.csv]
delevt	Commands that an event listed on the **events.csv** be deleted from the in-game event execution list. [events.csv]
delsave	Game engine command to delete a saved game. [mainscreens.csv]
detach	Commands the selected Unit to detach itself from its Superior Commander. [events.csv/toolbar.csv]
doneloc	Game engine command for Open Play courier message to Superior Commander. "I have arrived at the location ordered." [gamescreens.csv]
donequad	Game engine command for Open Play courier message to Superior Commander. "I have arrived at the quadrant ordered." [gamescreens.csv]
donevp	Game engine command for Open Play courier message to Superior Commander. "I have captured the objective ordered." [gamescreens.csv]
endscenario	This command must be called via the **events.csv** to create the **.mmc** saved game file required for carryover. [events.csv]
exit	Game engine command to exit a main or game screen. [mainscreens.csv/gamescreens.csv]
fallback	Commands the selected/named Unit to fall back at walk speed 50 yards to the rear of its current

	facing. The Unit will continue to fire at enemy forces is range while executing this command. [events.csv/toolbar.csv]
fightform	Assigns the fighting formation for a unit highground - arty move to best ground host - unused - remove join - unused - remove loadlevel - loads scenario loadspec - loads scenario plyrdest - unused - please remove setleaders - assigns leader vars for current unit
fire	Commands the selected/named Unit to open fire. [events.csv]
forcemove	Commands the selected/named Commander/Unit to move to the selected/designated location regardless of the presence of enemy Units. [events.csv]
form	Commands the selected/named Commander/Unit to assume a formation. [events.csv/toolbar.csv]
gatherhigh	Game engine command. [mainscreens.csv]
gathercarry	Game engine command. [mainscreens.csv]
goto	Game engine command. [events.csv]
help	Game engine command for Open Play courier message to Superior Commander. "I need help!" [gamescreens.csv].
hideunit	Commands the named Commander/Unit to "disappear" from the game map. The Commander/Unit is still there—it just doesn't show up in play. [events.csv]
highground	Game engine command.
highscore	Game engine command. List high score. [mainscreens.csv]
host	Game engine command.
join	Game engine command.
killoff	Commands the named Commander to be killed. [events.csv]
limber	Commands the selected artillery battery or gun to limber. [events.csv/toolbar.csv]
loadbullrun	Game engine command. Load a battle<#>num. [mainscreens.csv]
loadgame	Game engine command. Load a saved game. [mainscreens.csv]
loadlevel	Game engine command.
loadscen	Game engine command. Load a scenario. [mainscreens.csv]
loadscreen	Commands that a game screen appear during game play. [events.csv/mainscreens.csv/gamescreens.csv]
loadspec	Game engine command.
logmsg	Commands that a courier message be logged for later review via the Message Log game screen. [events.csv]
movedir	Commands the named Commander/Unit to move in a specified direction (dir x, dir z). [events.csv]
moveto	Commands the named Commander/Unit to move to a specified location (loc x, loc z). [events.csv]
mycourier	Game engine command for Open Play courier message to Superior Commander. Used as the preceding command for help, donequad, doneloc, and donevp [gamescreens.csv].
objactivate	Commands that an objective listed on the **objectives.csv** be activated. [events.csv]
orders	Commands that a stance order be issued to the named Commander. [events.csv/toolbar.csv]
playmp3	Game engine command. Play MP3 file at endscenario. [gamescreens.csv]
plyrdest	Game engine command.
quickload	Game engine command. Used for an in-game scenario load. [gamescreens.csv]
rally	Commands that the named Unit's morale level be returned to the level it started the scenario with. [events.csv]
ranrandom	Game engine command. Generate a random Open Play scenario. [mainscreens.csv]
resupply	Commands the selected/named Unit to move to the Superior Commander's supply wagon to upload ammo. [events.csv/toolbar.csv]
retreat	Commands the selected/named Unit to retreat at walk speed 100 yards to the rear of its current facing. [events.csv/toolbar.csv]
route	Commands the selected/named Unit to rout at run speed 500 yards to the rear of its current facing (generally away from the enemy). [events.csv/toolbar.csv]
run	Commands the selected/named Commander/Unit to double-quick. [events.csv/toolbar.csv]
safeplace	Designates a specific location on the game map for subordinate Units of the named Commander to retreat to—if forced to retreat. [events.csv]
savegame	Game engine command. Save the game in play. [gamescreens.csv]
setammo	Command sets the ammo type for a gun (Solid, Shell, Canister, Shrapnel)

	[events.csv/toolbar.csv]
setleaders	Assigns variables for current Commander [gamescreens.csv]
setvars	Assigns variables for current unit [gamescreens.csv]
showunit	Commands the named Commander/Unit to appear on the game map. [events.csv]
stop	Commands the selected/named Commander/Unit to halt. [events.csv/toolbar.csv]
switchcmn	Commands the selected/named cavalry unit to mount or dismount. [events.csv/toolbar.csv]
takecover	Commands the unit to go prone. [events.csv/toolbar.csv]
takecommand	Game engine command [toolbar.csv]
tcommoff	Release command of the selected/named Commander/Units back to the AI. [events.csv/toolbar.csv]
tcommon	Takes command of the selected/named Commander/Units away from the AI. [events.csv/toolbar.csv]
unlimber	Commands the selected artillery battery or gun to unlimber. [events.csv/toolbar.csv]
useroad	Commands the selected/named Commander/Unit to use road movement to move to his/its destination. [events.csv/toolbar.csv]
wheelleft	Commands the selected/named Commander/Unit to wheel left 10 degrees. [events.csv/toolbar.csv]
wheelright	Commands the selected/named Commander/Unit to wheel right 10 degrees. [events.csv/toolbar.csv]

Appendix C: Events.csv Examples

Command	Explanation	Time	ID Name	Command	CoordX	CoordZ	vartime
aboutface	Commands the selected Commander or Unit to turn 180 degrees from current facing.	9:15:08	U_14th_IA	aboutface			
advance	Commands the named unit to perform the advance function (the opposite of the fallback function) (also ***Aadvance***).	9:15:08	U_14th_IA	advance			
Ahalt	Commands everyone in the named Commander's organization to stop right where they are, rather than in the currently designated formation of their Commander.	9:15:08	U_WTShaw	Ahalt			
artyfireai	Commands the AI for the named battery to select targets for the named battery.	9:15:08	Pelham	artyfireai			
artyfirearty	Commands the AI for the named battery to only fire at artillery type targets.	9:15:08	Pelham	artyfirearty			
artyfirehold	Commands the AI for the named battery to hold fire.	9:15:08	Pelham	artyfirehold			
artyfiretroops	Commands the AI for the named battery to only fire at infantry/cavalry type targets.	9:15:08	Pelham	artyfiretroops			
attach	Commands the named leader/unit to attach to its owning leader (as designated on the units.csv).	9:15:08	U_14th_IA	attach			
charge	Commands the named unit to charge (also ***Acharge***)	9:15:08	U_14th_IA	charge			
courier	Commands that a courier be sent from one named Commander/Unit to another named Commander/Unit.	9:15:08	Hunter	courier:Burnside:loadscreen:Courier:Burnside			

delevt	Delete an event. This is useful for canceling an event if it no longer is required or makes sense. Example: an ***evtdeath*** message lets the player know a leader has been killed. Any other messages from this dead leader should "die" with him.	9:15:08		delevt:evtarrived:C_RSEwell			
detach	Commands the named Commander/Unit to detach from its parent Commander/Unit (the opposite of ***attach***).	9:16:15	Bee	detach			
enddir	The direction you want a formation to face upon arrival at a location (also ***Aenddir***)						
evtarrived	This event is triggered when the unit defined in column: B has arrived at the location define in columns D & E. This means that the unit in column B is stopped and within 100 yards of the location. Also that all of his subordinates are also stopped. This is used to determine that a formation is halted and set up.	evtarrived	U_WTShaw	Aform:Brig_Line			
EvtArrived	Same as above except at the regiment level.	Evtarrived	U_14th_IA	form:ManuverColumn			
evtcont	Predicate for continuing a series of ***evt*** events.	evtcont	U_14th_IA	takecover			
evtcourier	This event is triggered when the commander defined in column: B has received a courier message. Since the AI throws couriers all around, use this carefully. The first event occurs when the first courier arrives, and the second event is triggered with the arrival of the second courier.	evtcourier	U_WTShaw	Aform:Brig_Line			
evtdeath	This event is triggered when the commander defined in column: B has died. This is how we	evtdeath	U_WTShaw	endscenario:EndPlayerDie			

	end the game when the player dies. It can also be used to add some variety to a scenario, because units may or may not actually die.						
evtfailcheck	This event is triggered when the commander defined in column: B has reached the fail grade listed on the scenario level.ini file (?). The fail grades are command specific. These are the events that we use in open play to end the battle. Each level of command has its own fail grade. By adding this event to a unit, that unit will now do fail checks. If their grade falls below the fail grade for their command level, then this event will be triggered.	evtfailcheck	U_WTShaw				
evtfighting	This event is triggered when the Commander defined in column: B has first engaged the enemy.	evtfighting	U_WTShaw	Aadvance			
EvtFighting	Regiment level-This event is triggered when the Unit defined in column: B has first engaged the enemy.	EvtFighting	U_14th_IA	advance			
evtgiveup	This event is triggered when the Commander defined in column: B has given up. This means that they have lost all of their subordinate units.	evtgiveup	U_WTShaw	Arout			
EvtGiveUp	regiment level	EvtGiveUp	U_14th_IA	rout			
evtgrade	This event is triggered when the Commander defined in column: B has reached a certain grade. Note that this is a greater than or equal to >= comparison and will not work to see if a unit has reached a low grade, only a high grade. This event is the opposite of ***evtfailcheck***.	evtgrade	U_WTShaw		20		
EvtIArrived	Regiment level.		U_14th_IA	form:ManuverColumn			

evtintrouble	This event is triggered when the unit defined in column: B is in trouble. This state can be seen by the displaying of the in trouble icon over that unit. It usually means that all of their men need rallying and that this unit is really no longer a fighting force.	evtintrouble	U_WTShaw	Aretreat			
EvtInTrouble	Regiment level	EvtInTrouble	U_14th_IA	retreat			
evtobjarmy1	This is just like ***evtobjdone***, except that it is only triggered when army 1 has completed the objective.	evtobjarmy1	objective1	courier:Burnside:loadscreen:Courier:Burnside			
evtobjarmy2	This is just like ***evtobjdone***, except that it is only triggered when army 2 has completed the objective.	evtobjarmy2	objective1	courier:C_WBTaliaferro2:loadscreen:Courier:C_WBTaliaferro26			
evtobjdone	This event is triggered when the objective defined in column: B has been completed. If this is a hold objective, then this will trigger the first time it is completed. Note that this requires an objective ID from the **objectives.csv**, not a unit ID Name.	evtobjdone	objective1	courier:C_WBTaliaferro2:loadscreen:Courier:C_WBTaliaferro26			
evtran	Execute a random event.	evtranUnionStrategy6		delevt:evtarrived:C_RSEwell			
evtseetarg	This event is triggered when the commander defined in column: B has first seen the enemy. This is used for officers and it means that one of the regiments under their chain of command can see the enemy.	evtseetarg	U_WTShaw	orders:Attack			
EvtSeeTarg	Regiment level. This event is triggered when the Unit defined in column: B has first seen the enemy.	EvtSeeTarg	U_14th_IA	courier:Prentiss:loadscreen:Courier:Prentiss2			
fallback	Commands the named Unit to perform the fallback function (the opposite of the advance function).	9:15:08	14th_IA	fallback			
fire	Commands the selected/named Unit to open fire.	9:15:08	14th_IA	fire			

forcemove	Command forces movement regardless of enemy.	evtcont	Porter	forcemove	##	##	
form	Commands a unit/leader to assume a formation (also ***Aform***).	evtcont	Porter	Aform:Brig_Line			
goto	Go to screen message.	evtdeath	Beauregard	goto			
guard:unit_id	Cavalry command only. It directs the named unit to guard a named unit.	9:15:01	CoA_1stUS CAV	guard:U_14th_IA			
hideunit	Commands the named leader/unit to not be shown on the 3D map. This command is useful for hiding reinforcements until they are scheduled to arrive on the battlefield. (also ***Ahideunit***)	9:15:01	Heintzelman	Ahideunit			
killoff	Remove a Commander /Unit from the game.	9:15:01	Heintzelman	killoff			
limber	Commands the named artillery battery to limber.	9:15:01	Pelham	limber			
loadscreen	Commands the Courier Screen to be displayed	9:15:08	Hunter	courier:Burnside:loadscreen:Courier:Burnside			
logmsg	Add message text to the message log.	evtcourier	Burnside	logmsg:Courier:Burnside3			
movedir	Commands the named Commander/Unit to move in a specific dir x, dir z	9:15:08	U_14th_IA	movedir	1	0	
moveto	Commands the named leader/unit to move to a specific loc x, loc z	9:15:08	Imboden	moveto	##	##	
objactivate	Activates an objective (as named on the **objectives.csv**)	evtcont		objactivate:Objective2			
orders	Commands the named Commandert/Unit to assume a specified stance (**Attack, Probe, Defend, Hold**).	evtcont	Jackson	orders:Attack			
playmp3	Play an mp3 file. MP3 sound file can be in the Sound folder or in the scenario folder. (see ***stopmp3***)	evtcont		playmp3:charge.mp3			
raid	Cavalry command only. It directs the named unit to raid.	9:15:08	CoA_1stUS CAV	raid			
rally	Rally a unit or command.	9:15:08	U_14th_IA	rally			
ranevt	Declare a random event.	9:15:08		ranevt:UnionStrategy:6			
resupply	Resupplies ammo to	9:15:08	U_14th_IA	resupply			

	selected/named Unit.						
retreat	Instructs selected/named Commander/Unit to retreat.	9:15:08	U_14th_IA	retreat			
route	rout[e] - Instructs unit to rout.	9:15:08	U_14th_IA	route			
run	Instructs leader/unit to run (also ***Arun***)	evtcont	Terry	Arun			
safeplace	A predesignated loc x, loc z for units to retreat to; this command does not work with routed units.	9:35:00	U_WKrzyzanowski	safeplace	##	##	
scout	Cavalry command only. It directs the named unit to scout.	9:15:08	CoA_1stUSCAV	scout			
screen	Cavalry command only. It directs the named unit to screen.	9:15:08	CoA_1stUSCAV	screen			
showunit	Commands the named leader/unit to appear on the 3d map (the opposite of ***hideunit***) (also ***Ashowunit***)	evtcont	Porter	Ashowunit			
stop	Commands the named leader/unit to stop movement (also ***Astop***)	9:15:08	U_14th_IA	stop			
stopmp3	Stop playing an mp3 file			stopmp3:charge.mp3			
switchcmn	Commands the named cavalry unit to mount or dismount	9:15:12	CoA_1stUSCAV	switchcmn			
takecover	Infantry command only. Directs the selected /named unit to assume the prone position.	9:15:08	U_14th_IA	takecover			
tcommoff	Turns off the Take Command Function (AI is activated).	evtcont	Heintzelman	tcommoff			
tcommon	Turns on the Take Command Function (AI is deactivated).	9:15:02	Tyler	tcommon			
useroad	Commands the named leader/unit to use the road while moving to a new location (also ***Auseroad***).	9:15:12	C_WTaliaferro	Auseroad			
wheelleft	Commands the named leader/unit to wheel to the left 10 degrees.	9:15:08	U_14th_IA	wheelleft	1	0	
wheelright	Commands the named leader/unit to wheel to the right 10 degrees	9:15:08	U_14th_IA	wheelright	1	0	
followone	Cancels ("turns off") the cavalry screen, guard, scout, and raid commands.	9:15:08	CoA_1stUSCAV	followone			

getaway	A 100 yard retreat (normal is 300), with no moral or grade penalties.	9:15:08	U_14th_IA	getaway			
weather=	Command changes the weather state as defined in rows 1 thru 25 of the levels.csv file.	18:15:00		weather=9			

Index

Ability 24
Accuracy..... 26
Alpha Omega=1.....13, 30, 36
Army= 14
Artyammo.csv..... 56
Bayonet Drill..... 27
Breastworks..... 28
Brigade=..... 14
CantKillMe=..... 14
Carryover=..... 14
CarryOverFrom=..... 14
cells 9
columns..... 9
CommandHeight=..... 14
CommandRadius=..... 14
Corps=..... 14
Custom Scenario Folder Content.. 12
Data Files..... 10
DbgLvl=0..... 13
Division=..... 14
Effects Folder..... 10
Effects.csv..... 77
EndFail..... 15
EndMajFail..... 15
EndMajWin..... 15
EndPlayerDie..... 15
endscenario..... 15
EndScreen Definitions..... 15
EndTie..... 15
EndWin..... 15
Event Commands Reference..... 87
Fatigue..... 25
File Editing..... 8
First Aid (Medical)..... 27
Flags Folder..... 10
Folder Contents..... 10
Formation (Drill)..... 27
Formations.csv..... 70
Full Screen=0..... 13
Gamescreens.csv..... 75
Gamesounds.csv..... 63, 76
Graphics Folder..... 10
headers..... 9
Initiative..... 22
Leadership..... 23
Level.ini File..... 14
Levels.csv..... 64
Loading..... 26
LoseFocus=1..... 13
Loyalty..... 23
Main Game Folders..... 12
Mainscreens.csv..... 74
Mainsounds.csv..... 76
Map Layout..... 61
Map_Name.csv..... 61
Misc Folder..... 10
Morale..... 26
NoAI=0..... 13
On-Line Help..... 82
Open Play Folder..... 11
Open Play OOBs..... 18
Packages Folder..... 11
Quality..... 24
Regiment=..... 14
rows 9
Saved Games Folder..... 11
Scenario Carryover Capability..... 16
Scenarios Folder..... 11
Screens Folder..... 10
Screenshots Folder..... 11
Sounds Folder..... 11
Sprites.csv..... 49
StartTime=..... 14
StrategicAI=..... 14
Style 24
Table

Artillery Accuracy..... 70
Table\
Elevation..... 69
Fallback..... 69
Fatigue..... 67
Fatigue Run..... 68
Grades..... 68
Morale..... 67
Retreat..... 69
Unit Morale Bonus..... 68
Tables.csv..... 66
Tables.csv\.....
Artillery Section..... 56
TC**.ini File..... 13
Terrain Folder..... 11
Textures.csv..... 51
TimeOfDay=..... 14
ToolBar Folder..... 11
Toolbar.csv..... 58
Tooltext.csv..... 60
Unit Attribute Values by Year..... 28
Unitcommon.csv..... 52
Units.csv..... 18
Unitsprite.csv..... 51
values 9
Variables Reference..... 83
WarEdit Utility..... 79
WarPack Utility..... 81
Weapons.csv..... 54
Weather=..... 14
[Rank] 14