HOW TO BUILD A NEW WORLD FOR EAW

Terminology

To avoid confusion, I suggest using the following conventional names:

Terrain A set of tiles (square, flat graphic surfaces) which EAW uses to depict the ground cover beneath your plane. Note that if all you want to do is change the appearance of the ground below, all you need is a new terrain set, and this was how the first new EAW environments were created. *Terrain* can also mean the appearance (in the game) of the ground below, as in "forest terrain" or "mountain terrain".

Layout A specific arrangement of tiles taken from the chosen set (the terrain), over which you will fly. The arrangement of the tiles is what creates the appearance of coastlines, rivers, cities, forests and so on.

Topography Denotes the rise and fall of the virtual ground below your plane, which creates the appearance of hills and valleys.

Map (Or "Briefing map") A simple "picture" of the EAW world, viewed during the game while receiving a briefing, or in flight. Again, if all you want to do is change the place names, this is almost the only thing you need to amend.

Working map A graphical representation, based on a bitmap, of the items in your EAW world. I recommend making and maintaining one of these as you go along: it will save you immense amounts of time.

Target A location within the EAW world where ground objects are to be found. A target is, in effect, a named place, such as London, Berlin, Biggin Hill, Brandenburg, and so on.

Objects Also referred to as "3D objects", these are the individual items you see on the ground, such as buildings, vehicles, ships, hangars and so on. There is also such a thing as an air base object, which forms the runways and taxiways at an airfield. Objects are divided into two types:

Default objects, which are automatically positioned by EAW. Each tile in the tile set has its own default objects: thus you will see houses on a city tile, trees on a forest tile, and so on. Bridges are default objects, and so are the vehicles which move on the roads and railways.

Target objects, which are positioned at each target location. Each target has its own unique set of objects, each in its own specific position around the centre of the target area.

World Denotes a complete new EAW environment: tile layout, topography, target set, maps, 3D objects and so on.

The necessary steps

The full list of steps that you will need to go through in order to build a complete new EAW world is set out below. Note that depending on what you want to achieve, it is not always necessary to do all of these things, or to do them in this exact order. But I can say from experience that a bit of time spent at the beginning of your project, deciding which steps you need to take and in what order to do them, will pay off handsomely later!

1. Terrain (tile set)
2. Tile layout
3. Default objects
4. Topography
5. Front line
6. Targets
7. Target names
8. Air bases
9. 3D objects
10. Object layouts
11. Railways
12. Interdiction files
13. Briefing map and other screens

1. Terrain (tile set)

A *tile* is a square graphics file, essentially a modified .pcx file. It measures 256 x 256 pixels, and uses its own palette of 256 colours. In effect, it is a small square picture of a piece of landscape, such as grassland, city blocks, forest and so on.

There are 64 tiles in a full set, but EAW only uses 59 of them. The other 5 appear to be useless, in that even if you put them into a tile layout, they will not display properly in the game. Appendix 1 lists the 64 tiles, showing which ones are not used, and giving the file name for each one.

The most important thing to understand about the tiles is that each one has a fixed file name which you cannot change. Whatever the tile actually looks like when you fly over it in the game, it always keeps the same file name.

In changing a tile, you cannot change its file name. You can, however, change the graphics which will appear when that tile is used. For example, in DAW, the desert sands were actually represented by the Grass tile (*BNGrass.ter*). To do this:

1. We made a new graphic, depicting sand dunes instead of grass, and gave it the name of the Grass tile (*BNGrass.ter*).

2. This file was then put into the main DAW folder. This means that the program will use it in preference to the default (Grass) tile with the same file name.
3. As far as the game engine is concerned, those are still Grass tiles (*BNGrass.ter*) down there, but by substituting a new graphic, we are able to change the way the game looks to the player. Instead of grass, the player sees sand.

There are two things which this process will not change. The first is the appearance of default objects on the altered tile: in DAW, for example, the "sand" tile would still have trees growing out of it, as the default Grass tile does. You can, however, edit these default objects: see below.

The other thing that will not change is the basic nature of the tile: specifically, whether it represents land or water. This affects the appearance of crashes and bomb-bursts; and more importantly, it determines whether or not you can land your plane on that tile. Even if there appears to be an airfield present, you cannot safely land on it if it is on a tile that the game thinks is made of water!

On the other hand, in DAW we were able to reverse this process, and make aircraft carriers by:

1. taking a "land" tile (specifically, Road-1, filename *BNRoad1.ter*)
2. changing its graphic to look like the sea (in fact we just copied the sea tile, *BNWater.ter*, and renamed it *BNRoad1.ter*), and
3. putting a specially modified air base on it to represent the flight deck.

It is a good idea to begin your project by thinking about the tile set, because the available tiles will restrict the number of different kinds of ground cover that you can depict in your world. Remember that for each pair of terrain types, you may also need some transitional tiles. For example, suppose you decide that you need a tile to represent swamp terrain. At some points, swamp will give way to something else, perhaps to forest terrain. This means you will need some tiles which are part swamp and part forest. Usually you will find you need three of these transitional tiles, arranged something like this:

Swamp	Swamp	Swamp
Swamp	Swamp	Forest
Swamp	Forest	Forest
Forest	Forest	Forest

That is, in the case of the first tile shown above, the tile will mostly depict swamp, but there will be a bit of forest in one corner. The second tile will be half swamp and half

This is one of the biggest tasks in making a new world. The arrangement of the tiles is what creates the coastlines, cities, roads and rivers that you fly over.

The tile layout is contained in the hex file *EAW.tm*. Full details of the structure of this file, and the way it relates to the visual appearance of the EAW world, can be found in my separate set of notes called *Editing the tiles in EAW*; but for the moment, note that its main features are these:

1 The EAW world consists of 204,800 tiles, arranged in a block 640 wide and 320 deep. (So far nobody has succeeded in altering these dimensions.)
2 *EAW.tm* represents the tile layout by using a single byte for each tile. The byte indicates both the type of tile, and its orientation (each tile can be turned to "face" north, south, east or west). Appendix 1 lists the byte values used in *EAW.tm*.

There are two tile editors in use by EAW world makers. One is Woolfman's tile editor, about which I am afraid know nothing; the other is MapEdit, by The Photon Effect. This is the one I use. It was not developed for EAW, but it does a good job.

As alluded to above, I have written a set of notes on the use of MapEdit, called *Editing the tiles in EAW*. This explains how to use the program, what limitations it suffers from, and how to get around them. It also gives a lot of information about using bitmaps to create your basic layout, and how to convert files between different formats (bitmaps, MapEdit files, and EAW layouts). I recommend you get these notes from my web site.

The best way to get accustomed to tile layouts and the use of MapEdit is to take the default version of *EAW.tm*, convert it to a MapEdit file, and examine it in MapEdit. You will soon spot bits of the European landscape that you recognize. Try altering some tiles in MapEdit, converting the file back to *EAW.tm* format, and flying over it to see how it looks.

Try also converting *EAW.tm* to a bitmap and examining it in your favourite graphics program.

The best way to create a new layout is in fact to start with a bitmap of your chosen area, and do as much as possible of the layout work on that before moving on to MapEdit. For example, by filling an area with a random pattern of suitable pixels, you can create an area of (say) farmland where the tiles will be a mixture of types and orientations.

For this to work, you will need to use a palette which reflects the correct values for each tile (otherwise the bitmap will not convert successfully to an EAW tile layout) You can get a suitable palette for PaintShop Pro 7 from my web site.

There are several possible sources for these basic bitmaps. For example, the basic map for DAW was made up of screenshots from a digital atlas (specifically, Attica's *Interactive World Atlas*), which I got free from the cover CD of a computing magazine. The maps for GPAW, on the other hand, are based on a map of European Russia and Eastern Europe which I found on the West Point Military Atlas website at:

http://www.dean.usma.edu/history/dhistorymaps/WWIIPages/WWIIEurope/WWIIEToC.htm

In planning your layout, you need to be aware that there is a No Fly Zone all round the edge of the EAW world. This zone is 15 tiles deep. In other words, the first 15 tiles in each row cannot be flown into, and neither can the last 15, or the first and last 15 tiles in each column. If you put a target or an airbase inside the No Fly Zone, it will be unreachable. (The reason for the No Fly Zone is simple: it stops you flying to a position from which you can see over the edge of the world!)

Ultimately, though, however much bitmap work you do, you will have to use MapEdit to make the coastlines, roads and rivers run smoothly.

Don't throw away your bitmap. When the tile layout is finished (or at least, when you think it's finished!), convert the final version of *EAW.tm* into a bitmap and use that as a map of your tile layout. If at all possible, work on it using a graphics program that supports layers, such as PaintShop Pro 7 or Adobe Photoshop. You can then add a series of layers on top of it, each one representing a different type of item. One layer will show where your targets are positioned, another might show the railway lines, and so on. You can switch the layers on and off to de-clutter the display.

This layered file is your *working map*. I use this system to keep track of the "geography" of my EAW worlds, and it is an invaluable tool. (In fact you probably won't get far without it.)

3. **Default objects**

The default objects (or "DOs") are the houses, farms, trees and other things which pop up from the ground as you fly over it, but which are not tied to any specific target.

Each individual tile in the default set has a specific set of DOs associated with it. A forest tile will mostly have trees, a city tile will have buildings, and so on. (A sea tile, of course, has no DOs.) Whenever a particular tile appears in your layout, you will see the appropriate DOs when you fly over it.

Remember that because you cannot change the file names of the tiles, a tile will always have the same DOs associated with it, no matter what that tile actually looks like in the game. However, it is possible to change the associated DOs, with a bit of work.

One way to change the DOs is to change the 3D objects which the game uses to depict them. For example, you can download a set of objects which represent palm trees. These simply replace the existing European tree objects, having the same file names. The game associates the same tree DOs with the tiles as it did before, but to the player they now look like palm trees.

You can even eliminate objects entirely by using this method. Or to be more accurate, you can make them invisible! It is possible to get a file for an "object" which consists of

no visual appearance whatsoever. Any existing object which you replace with this file will be invisible. As far as the game engine is concerned, however, it's still there, and you can still crash into it!

Depending on the nature of your new world, however, you will often get better results by editing the DOs associated with some or all of the tiles you are using.

The DOs associated with each tile are specified in the hex file *EAW_TTD.dat*. The structure of this file is as follows:

- The file contains 303,260 bytes. It consists of 59 records, one for each type of tile in use. Each record consist of 5,140 bytes.
- The tiles are dealt with in their numerical order, as listed in Appendix 1.
- Each record begins with a header section of 20 bytes. All are zeroes except the 16th byte, which gives the number of DOs to be placed on that tile.
- The rest of the record contains 64 fields, each of 80 bytes. Each field deals with one DO. There can therefore be up to 64 DOs on a tile.
- In each field, the first byte gives the number of the 3D model used for the DO. Appendix 2 lists all the 3D models used in the game.
- The next 8 bytes give the (X,Y) position of the object on the tile.
- The rest of the field contains zeroes.

A complete analysis of *EAW_TTD.dat* is included in my EAW spreadsheet.

EAW_TTD.dat can be edited with a hex editor. Things you can do include:

1 Changing the number of DOs on a tile. For example, if the number is set to zero, there will be no DOs on that tile. In DAW, we did this for most of the desert tiles. There was then no need to edit the DOs themselves, because the game engine just ignores them. However, see the warning below on vehicles.
2 Editing the identity or position of an individual object. In DAW, we edited out the 3D forest objects (which nobody seems to like) and replaced them by single trees.
3 A combination of the two. In DAW, we did this to the coastal tiles. First the number of DOs on each tile was reduced, generally to about 10 or 12. Then the first 10 or 12 DOs in the record were edited to get rid of most of the buildings, replacing them with trees. (In the game, the trees were palm trees, of course, because of the 3D objects we were using for them.)

Finally, note that bridges, motor vehicles and trains are also DOs. However, the vehicles and trains do not appear to be controlled in any obvious way by *EAW_TTD.dat*. (The bridges are.)

Motor vehicles will appear on the roads running through two of the road tiles (Road-4 and Road-6), but not on every such tile: there seems to be a minimum allowable distance between road convoys. CAUTION: if you change the number of DOs on one of these tiles, it sometimes seems to prevent the vehicles from appearing at all. The best thing is to leave the number of objects at 64 (hex value 40), and just substitute 00 for the object identity numbers of those DOs you don't want.

Trains appear on railway lines (of course!), which are discussed in more detail later on.

4. Topography

Remember that by "topography" we mean the rise and fall of the landscape – or to put it another way, the hills and valleys of your EAW world.

The topography is controlled by the hex file *EAW16.hm*, which has the following structure:

1 Each tile in the tile layout is represented by two bytes. This means that each tile has associated with it a value from 0 to 65535. This value gives the height above sea level of the ground at the south-east corner of the tile.

(The height in what units, we aren't quite sure. There is a suggestion that 1 unit is equivalent to about 2.67 feet, but this is a bit uncertain.)

In any event, one of the limitations of EAW is that the AI pilots are not equipped with very sophisticated terrain avoidance. Put another way, if you make the hills too steep or too high, the idiots will crash into them. (This is why the mountains in DAW were not as rugged as many people would have liked.) Personally I don't now use mountains higher than about 80 units, which seems enough to prevent the problem.

To make a topology file, I use a graphical method, which is described more fully in my notes *Editing the tiles in EAW*. Briefly, I start by converting the tile layout file *EAW.tm* into a 24-bit colour bitmap.

I then fill in that bitmap with a set of colours whose digital representations translate into a suitable set of values for *EAW16.hm*. For example, I might colour the "lowlands" areas in a mixture of shades of very dark green, having the following digital representations:

(0,4,0) (0,8,0) (0,12,0) (0,16,0)

Note that the first value (corresponding to the colour red) is always zero.

The reason for doing it this way is that the small variations in the values of the second and/or third numbers produce small variations in the ground height. Thus the ground, instead of being flat, becomes gently undulating.

When the colouring process is finished, the bitmap is converted directly into an *EAW16.hm* file by stripping out the unwanted zeroes. This gives excellent results in quite a short time.

5. Front lines

The EAW world is divided into Axis and Allied territory by a front line, which can be set to be visible or not visible on the briefing maps. All the targets on one side of the line belong to the Allies; all the targets on the other side belong to the Axis. The front line also affects things like your chances of surviving a bail-out.

The front lines are controlled by the hex file *frntline.dat*. Its structure is fully described in a set of notes by Dominique Legrand, and Dom has also written a DOS program for building your own front lines. There is little for me to add to what Dom has written, except for a few practical observations:

Working map: I recommend using a separate layer of your working map to plan the front line. Draw in the front line as a series of straight lines, and then you can read the (X,Y) values of each turning-point in the line from the map (assuming your map has Y=0 at the top), and input them to Dom's Front Line program.

Dom's program: To get the right result, you must add 1 to the Y co-ordinate taken from your working map, before inputting it as a line number into Dom's program. (This does not, however, apply to the X co-ordinate!) Other tips:

1 Start with the lower left-hand point of the front line and work, in order, towards the upper right.
2 You must have the last point above and to the right of the first one. One or other of these conditions alone will not do.
3 Avoid using Line values less than 2, which seem to produce errors. Just end your front line a little below the top of the map.

London: The position of the London target seems to be critical to the working of the front line. Whichever side of the line London is on, that is the Allied side. I say "seems to be", because that is what I found in DAW (where London appears as the convoy VIGOROUS, somewhere to the east of Malta); but the builders of PT2 seem to have managed without using London at all, so there must be more to it that this. Personally I play safe and leave London in its default position, disguising it with a new name and target objects.

Single missions: Some of the air bases are coded (in a way we don't yet understand) according to which side may use them as its home base in a single mission. This mainly affects the Allied side. Most of the air bases in mainland Europe cannot be used as Allied home bases, even when they are on the Allied side of the front line. Appendix 3 lists those European bases which *can* be used by the Allies. Interestingly, this limitation appears not to apply to campaigns.

Moving front lines: If you have played a 1944 campaign, you will have noticed that the

front line gradually moves eastward from mission to mission. This process is still not very well understood, but I am trying to incorporate it in the Operation DOWNFALL campaign.

6. **Targets**

Constructing the target set is likely to be the longest and most laborious part of building your EAW world. There can be over 300 targets in a set, and some targets can have as many as 30 or 40 individual objects.

There is a lot to be said for deciding where you want to put all (as in every single one) of your targets before you start editing any of them. Otherwise confusion and re-working are sure to follow!

There are five inter-related files in a target set, as follows:

1 *Targets.dat*
2 *Tardata.dat*
3 *Tarnames.str*
4 *Griddata.dat*
5 *Airfield.dat*

Once again, Dominique Legrand wrote a set of notes about these files, and some DOS programs to go with them. They are indispensable. You need to be aware, however, that Dom made a small mistake in describing the file structures, because he did not identify the file headers. This was corrected by Moggy, on whose web site you can find some more advanced (and equally essential) notes.

For the sake of completeness, a brief description of each file structure is included at the relevant point in these notes, starting with:

Targets.dat This hex file of 9,668 bytes specifies the location (and other properties) of each of the targets within the EAW world. It consists of a 4-byte file header (which must not be changed), followed by 302 records, each of 32 bytes. Each record deals with one target.

The structure of a record is not yet completely understood, but the main parts are as follows (bytes are numbered from 0 to 31):

1 Bytes 0-7 are the X and Y co-ordinates of the target in the EAW world. See below for an explanation of how this works.

2 Bytes 16-17 give the number of the record in the file *targets.str* which holds the target's name. See section 7, below.

3 Bytes 18-19 set the numbers of records applicable to this target in the file *tardata.dat*, and therefore the number of actual objects that will

appear at that target.

4 Bytes 24-25 give the Airbase code ("A-code") if the target is an air base, or are zero if the target is not an airbase. See section 8, below, for more about airbase codes.

A complete analysis of the default version of *targets.dat* is included in my EAW spreadsheet.

The Target Co-ordinate system

EAW uses an (X,Y) co-ordinate system to identify the exact location within the EAW world of the centre of each target area. Each co-ordinate is represented by 4 bytes, with the smallest (least significant) byte first and the largest (most significant) byte last. In each record within *targets.dat*, the X co-ordinate is given by the first 4 bytes (bytes 0-3) and the Y co-ordinate by the next 4 (bytes 4-7).

Fortunately for us, the co-ordinate system ties up in a (relatively) easy way to the grid of tiles. Remember that the EAW world is made up of a rectangular arrangement of tiles, 640 tiles wide (east-west) and 320 tiles high (north-south).

If we first of all think of the EAW world as consisting of 640 columns of tiles, then bytes 2 and 3 in the *target.dat* record specify the column in which the target will appear.

1 Column 0 (the westernmost column) has the X co-ordinate 00 00 81 FD
2 Column 639 (the easternmost) has the X co-ordinate 00 00 80 07
3 Each increment of byte 2 denotes another column to the east. For example, the value 00 00 82 FD represents column 1; 00 00 83 FD represents column 2, and so on.
4 Each increment of byte 3 denotes 64 columns. For example, the value 00 00 81 FE represents column 64 (= 0 + 64).
5 After FE in byte 3 comes 00.
6 Bytes 0 and 1 represent fractions of a column width. You can usually ignore them.

Y co-ordinates correspond to rows of tiles in a similar way. The northernmost row (row 0) has the Y co-ordinate 00 00 81 E4, and the southernmost (row 319) has the co-ordinate 00 00 80 E9.

Confused? Don't worry. You don't really need to remember (or even understand) all this, because Dom has written a small program called *VERTCON.EXE* which converts tile column and row numbers (X and Y values) into 4-byte co-ordinates for insertion into *targets.dat*. I explain how to use it a bit later on.

The T-code

Each of the 302 targets available in the game has its own unique identity code, referred to (following Dom's convention) as the "T-code". The T-codes are not explicitly listed in *targets.dat*, but each target's T-code is the hexadecimal version of that target's record number in *targets.dat*. For example, the 79th record in *targets.dat* refers to the air base at

Beaumont-le-Roger. The T-code for Beaumont is 4F, which is the hex notation for 79.

My EAW spreadsheet shows the T-codes for all 302 targets. Note that the target with T-code zero (which is Brest) does not seem to work properly even in the default version of EAW, and is perhaps best avoided.

Griddata.dat This hex file of 6,400 bytes serves as a lookup table, which the game engine uses to decide which targets are currently within your field of vision. If the entries in *griddata.dat* are not correct, the target will appear on the briefing map, but when you reach its position you will see nothing there! (This could have its uses, I suppose. What about a Coastal Command game, in which ships or U-boats at sea appear on your map, but are not always in that position when you reach it?)

The structure of *griddata.dat* is quirky, to say the least. I suggest you read Dom's account of it several times, and then you still won't understand it! Briefly, however, the EAW world is divided into a series of squares, each containing a number of tiles, and each square is represented by an 8-byte record in *griddata.dat*. Because the squares overlap a lot, there are usually several different records in which you could put a given target (but you must never put a given target into any more than one record).

In each record, the first byte (byte 0) gives the number of targets to which that record applies. (This is often zero). Bytes 4 and 5 give the T-number of the *first* target to which that record applies. If the record applies to more than one target, the game engine assumes that the other targets follow in consecutive T-number order.

For example: suppose a record looks like this:

03 00 00 00 01 01 00 00

This tells us that this grid square refers to three targets (first byte = 3), and the first of these targets has the T-number 0101 (low byte first). The other two targets referred to must therefore be 0201 and 0301.

The practical result of all this is that if you put several targets close together (within a few tiles of each other), you would be wise to make sure they have consecutive T-numbers, or you may have difficulty fitting them into the structure of *griddata.dat*. This is probably the hardest single part of making a target set!

How, then, do you edit the position of a target?

First, use your working map to decide where you want to put the target. I use a separate layer of the map for this, and I mark each target with a single pixel. Remember that because your bitmap is 640 x 320 pixels, one pixel represents one tile, so this shows you

which tile the target will occupy.

Read off the (X,Y) co-ordinates of the target's tile from the map, note them down, and turn to one of Dom's programs, called VERTCON. Then proceed as follows:

1. Open my EAW spreadsheet to the Targets page. Using CTRL-F, search for the target you want. (Important: for this to work, you must select the search settings *Columns* and *Values* in the search dialog box).

2. Note down for future reference (a) the T-code number of the target, (b) its string value (in hex notation), and (c) the address of its record in *targets.dat*.

3. Run Dom's VERTCON program in a DOS window. When prompted, enter the chosen line (Y co-ordinate) and column (X co-ordinate) for your target's position. Very important: note that for this purpose, Dom's line numbering starts from 1, not zero! This means you must add 1 to the line number taken from your map, in order to get Dom's line number. For example, if the Y co-ordinate of your target (taken from your map) is 213, you must enter 214 as the line number in VERTCON.

4. Make a note of the results given by VERTCON:

   1 the X & Y values for entry in *targets.dat*, and

   2 the suggested addresses to use in *griddata.dat*

5. Using a hex editor, open the current copy of *targets.dat*. (I strongly suggest that you work with a "stripped" copy of the file: that is, a copy with the 4-byte file header removed. This makes it a lot easier to display the data, by viewing the file in a 16-column window.)

6. Using your hex editor's GOTO function, go to the address of the target's record (found in step 2 above). Examine bytes 16-17 (counting from zero) and check the string value (step 2) to verify that you are looking at the correct target record.

7. At the beginning of the target's record, enter the X and Y co-ordinates (each of 4 bytes) which you found in step 4, over-writing the existing values.

8. Save the file as something like *targets.stripped.dat*.

9. Using the hex editor, open the current copy of *griddata.dat*. (This file has no header.) This also views best at 16 columns' width.

10. Go to the target's first suggested address (see step 4 above). If this 8-byte record is empty (all zeroes), then at the first byte of the record, enter 01 to denote a single target. At the fifth and sixth bytes, enter the target's T-code (see step 2 above). Save *griddata.dat*.

11. If the first record you examine in *griddata.dat* is not empty, you have to make a choice. If your new target's T-code follows consecutively on from the target or

targets already shown there, you just need to alter the first byte to show the new number of targets held in that record. If it doesn't, you will need to use a different record in *griddata.dat*. Try the next suggested address (from step 4). Keep trying till you find an empty one.

12. When you have finished editing, put the file header back into *targets.dat*, and put both *targets.dat* and *griddata.dat* into your working copy of EAW. Use a single mission to check out the target. (If you get a CTD at this point, you probably forgot to put the 4-byte file header back into *targets.dat* !)

7. Target names

This is an easy bit for a change. You need Paulo Morais's string utilities *dumpstr.exe* and *buildstr.exe*. They should be in the same folder as a working copy of the target names file, *tarnames.str*.

Run *dumpstr.exe* in a DOS window. The syntax is:

dumpstr.exe tarnames.str > tarnames.txt

Edit the resulting text file *tarnames.txt*, and then run *buildstr.exe* using the syntax:

buildstr.exe tarnames.txt

My EAW spreadsheet shows all the default target names, and associates them with their correct target identity numbers (T-codes).

8. Air bases

On one level, air bases are just another kind of target, but they have additional characteristics to enable players to take off and land there.

Each air base is assigned a specific identity number, known (using Dom's convention) as its "A-number". The A-number appears in the air base's record within *targets.dat*, and is apparently used by it to cross-refer to another data file, *airfield.dat*.

Airfield.dat: this file is fully described by Dominique and Moggy. Much of its contents are not yet understood; but in outline, its structure is as follows:

1 The file consists of 7,132 bytes, including a 4-byte file header, and 162 records of 44 bytes each.
2 In each record, bytes 36-37 (counting from zero) contain the target's T-code, as a sort of cross-check.
3 Bytes 20 to 27 contain "time codes", which determine the appearance of the air base in each of the four EAW time periods: 1940, 1943,

1944 and 1945 (see below).

4 Byte 42 contains the "location code", which specifies the country in which the air base lies: see below.

Time codes: Each of the four time periods is coded for by two bytes. (For example, 1943 is coded by bytes 22-23.) If the base is not active in a given period, the code used is FF FF; otherwise, it takes the form XX 00, where XX is a number denoting the type of 3D model used to draw the airbase. Moggy's notes explain this code system in more detail.

These codes can be freely edited, but if you change the type number of an airbase, you will usually find that the other obejcts present at that base no longer appear in the right place (hangars on the runway, and that sort of thing) and need to be re-arranged.

Location: When you play a campaign, a black screen displays the name of your base and its country of location (for example: "Biggin Hill, England"). Byte 42 of the *airfield.dat* record codes for this. Possible values range from 00 to 05, and the corresponding texts are found in the file *briefing.str*. In the default EAW world, the codes are as follows:

00	England
01	Germany
02	France
03	Belgium
04	Holland
05	Luxembourg

However, it appears that only code 00 works properly for Allied forces in user-made campaigns. If an Allied base is given any other code, it will appear in the EAW world, but it will not be available for selection as a home base (at least in a single mission). It is not known if this problem can be solved.

9. 3D objects

In this section and the next we deal with "target objects": those 3D objects deliberately placed around a specific target location. It is important, first of all, to understand that:

1 *Targets.dat* establishes the position of the centre point of the target area; and
2 *tardata.dat* specifies which objects will appear in that target area, and where they are placed in relation to its centre point.

The more editing you do on any target's objects, the longer your whole project will take. Before you start editing *tardata.dat*, consider (changing 3D models)

10. Layout of target objects

Tardata.dat: The structure of this file was mainly uncovered by Moggy, and briefly it is as follows:

1 The file consists of 122,436 bytes, comprising a four-byte file header, and 3,826 records each of 32 bytes.
2 Within each record, byte 4 (counting from zero) contains the number of the 3D object to be used (and hence defines the type of object). Appendix 2 lists all the known object types.
3 Byte 8 contains the T-code for the target, as a sort of cross-check.
4 Bytes 12-19 give the (X,Y) co-ordinate of the object, relative to the centre of the target area. Note that this is a *different system* from the one used in *targets.dat*: see below for further explanation.
5 Bytes 24-25 determine the orientation of the object: see below.

Note that the number of objects present at a target is actually specified in *targets.dat* rather than *tardata.dat*.

If you just want fewer objects to appear, rather than changing the types of objects, simply change the 3DZ number of any object that you don't want, to read 00.

Changing the objects

To change the identity of an object, just change its object code (byte 4 in the record). Remember to use the hexadecimal code, or you will get strange results!

Experiments have shown that Axis radar stations only appear in certain years (they never appear in 1940; and basically, the later in the war, the more likely they are to appear). It is not known if any other objects behave in this way.

The Object Co-ordinate system

Although superficially similar to the system used in *target.dat*, the *tardata.dat* co-ordinate system works differently, and the two systems should not be confused. Rather than describe the system in detail, I would just point out the following:

1 The centre of the target area corresponds to Object Co-ordinates (0,256) rather than (0,0) as you might expect. Nobody knows why.
2 Each tile measures 1024 x 1024 co-ordinates in the Object Co-ordinate system.
3 My EAW spreadsheet contains (on the TargetDiag page), a system for translating the 4-byte co-ordinates used in *tardata.dat* into straightforward numbers, and vice versa. It wil also draw you a rough diagram of the target area, showing where the objects lie. I suggest this is all you really need to know!!

Object Orientation

Each object has a "front end" which points in a direction specified by bytes 24-25 of the record in *tardata.dat*. The main values are as follows:

0000	North	0020	North-west
0040	West	0060	South-west
0080	South	00A0	South-east
00C0	East	00E0	North-east

The orientation does not usually matter except in the case of vehicles or ships.

11. **Railways**

The railway lines in EAW are handled quite differently from any other terrain feature. Each stretch of track is drawn as a textured straight line between two end points in the EAW world. The file *EAW_RRD.dat* contains a complete list of all these end points, in the following format:

1 The file consists of 21,460 bytes, including a 4-byte file header, and 894 records each of 24 bytes.
2 In each record, the first 16 bytes are organised in 4 groups of 4. These are two 4-byte X co-ordinates, followed by two 4-byte Y co-ordinates. Each pair of (X,Y) marks one end of a stretch of railway line.

There is a complete analysis of *EAW_RRD.dat* in my EAW spreadsheet. By trial and error, I have worked out the towns where most of the lines begin and end.

To edit the railways, you just need to change the points listed in the file. If you do not need as many end points as are contained in the default world, simply change the rest of the file entries to zeroes.

One additional complication is that most (perhaps all) of the railways begin and end at stations, and run across bridges. It can be very tricky to position the station or bridge object so that it sits on the railway line (or alternatively, to position the railway line so that it runs under the bridge or the station.) One method of simplifying this might be to place the station or bridge at the exact centre of the target area, and then make sure that the railway line also runs through the centre (by making the first 2 bytes in each co-ordinate equal to zero).

12. **Interdiction files**

Interdiction missions are a complicated feature of the EAW system, and in user-made campaigns they do not always work properly. Missions can be directed against road traffic, rail traffic, or convoys at sea, but it is not known how the system chooses which type of target it will assign you.

There are four “interdiction files”, which come in two pairs:

1 *Roadlist.dat* and *roadwpt.dat*; and

2 *Raillist.dat* and *railwpt.dat*

Each pair works in the same way, so we will look at rhe two road files.

Roadlist.dat contains 1,168 bytes. After the usual 4-byte header, there are 97 records, each of 12 bytes. Each record corresponds to a different road-interdiction mission flight path. The game appears to chose a record at random, and uses the information contained in it to look up the waypoints for that mission in *roadwpt.dat*.

Roadwpt.dat is therefore the more important of the two files. It contains 3,568 bytes, consisting of a 4-byte header, and 297 records each of 12 bytes. Each record begins (bytes 0-7) with the (X,Y) co-ordinate of the waypoint, using the Target Co-ordinate system explained in section 6 above. The only other important byte is byte 10, which is always 01 except in the last record of a set of waypoints, when it is 00.

Raillist.dat and *railwpt.dat* work in exactly the same way, although they contain different numbers of records.

My EAW spreadsheet analyses the two waypoint files. It is not necessary to analyse the two "list" files.

Editing the four files

I strongly suggest not editing the two "list" files. Just accept that the number of waypoints on each flight path must remain the same as in the default EAW.

Instead, just edit the (X,Y) co-ordinates of each set of waypoints. Set to zero any waypoints you don't want.

Problem areas

You will see from all this that ship convoys at sea are not controlled by these four files. In fact, we still do not know what controls the convoys, and they may actually be hard-coded into the .exe file.

Even with edited interdiction files, problems still arise. For example, in DAW we ran into a "phantom convoy" problem. Interdiction missions in North Africa were occasionally sent against a convoy which was sailing off the coast but was out of range of almost all the aircraft used. This was clearly one of the default convoys from the original EAW, and we were never able either to get rid of it, or to stop the program from targetting it.

An alternative approach to all this would simply be to have no campaign interdiction missions at all. By eliminating interdiction missions from the squadron campaign files, you eliminate the need to edit these files. Missions against ground transport could still be created, provided that you set up vehicles or railway lines around a named target area.

13. **The briefing map and other screens**

This is a very big area, and another one in which a considerable amount of artistic ability might be useful. Making new aircraft selection screens, barracks screens and so on is far from easy, and the results do not always look good unless you learn to manipulate the EAW palattes (which I have not).

One screen which you will certainly need, however, is the briefing map. Without it, a lot of the immersion factor will be lost from your new world.

Screen name:	Europe1.mpc
Text files:	Tarnames.str

The map measures 1536 x 1280 pixels. However, the area with actual targets on it is only about 1140 x 620, so you can get away with a smaller map graphic as long as you don't mind there being some blank areas around it.

To make a basic map screen, open the map in EAW and take screenshots of the various areas. You will need about 9 to cover the whole map. In your graphics program, open a new file 1536 x 1280 and paste the screen shots into it, using a montage to make the whole map. Although laborious, it is a good idea to remove all target symbols from the map in case they don't quite match the ones that EAW lays over it.

Alternatively you can use the new PIC to PCX utility to convert the briefing map into a .pcx file. Of course if you want a completely new map, just start with the blank 1536 x 1280 file.

Once your map is finished, use PICPAC to convert it into a suitable form for EAW to use.

<u>Coverage and distortion</u>

The briefing map does not bear a straightforward relationship to the tile and target layout. If you look at the default briefing map, you will see that part (but only part) of it is enclosed in grid lines. This is the part that represents the actual flyable EAW world. The rest of the briefing map is just there for show.

Not only that, but notice how the lines of longitude diverge, and the lines of latitude are curved. In the game world, however, these lines would be straight and parallel.

The briefing map is, therefore, a distorted representation of the actual "world". To get around this, my approach in DAW was as follows:

1. I did no work on the briefing map until most of the targets had been placed into *targets.dat*. (Ideally you should wait until they are *all* placed.)
2. I made a blank .pcx file 1536 x 1280 pixels (which I coloured brown, for better visibility in the game!), converted it using PICPAC, and put it into EAW as a

substitute for the briefing map.

3. I ran EAW using the new *targets.dat* file, and made a map out of screenshots as described above.
4. This gave me a blank .pcx map with all the new targets marked on it.
5. I then drew my briefing map on the blank .pcx map, making sure it fitted correctly around the targets, and filled it in with colours taken from the default map.
6. Finally I deleted all the targets from the briefing map, and converted it using PICPAC.

Briefing-room screens

As well as a new briefing map, you really need to make new briefing-room screens, because they use the same map. Here is the method I have used (instructions refer to PaintShop Pro):

Screen name:	Brief**.mpc (** = ge/uk/us)
Text files:	Briefing.str

1. Take a screen shot of the briefing-room screen (or convert using the PIC-to-PCX utility) and convert it to 16-million colours.
2. Fit your new map onto it. The map area is 510 x 424 (approximately 1.2:1) , and its top LH corner is at (64,24). IMPORTANT: make the map into a separate layer.
3. Copy the background (the whole screen). Paste it in three times, each time as a new layer, all of them above the map.
4. Working on each of these three layers in turn, add the shadows on the map edges. Use the rounded rectangle setting to select an area starting at (57,16) and covering the whole map including its frame. Apply the Cutout effect, with the following settings:

First layer:	V25 H20 Opacity 66 Blur 15
Second layer:	V25 H -20 Opacity 66 Blur 15
Third layer:	V-15 H0 Opacity 66 Blur 15

5. Flatten all the layers. If necessary, use the Push tool to fill in the corners of the map with shadow (usually only the bottom left and top right will need work).
6. From the original screen shot, “lassoo” the blackboard, amend it if necessary, then paste it into the new screen and give it a drop shadow using each of these settings in turn:

   (a) V5 H-5 Op61 Blur 9.9
   (b) V-2 H-5 Op61 Blur 9.9

7. To make a new flag, try applying the Texture-blinds effect to your flag graphic, and then using triangular selections from the existing flag as masks or guides to selecting suitably shaded regions of the new one to use instead.
8. The flag shadow is made by “lassooing” the flag and applying the following drop shadow: V56 H78 Op50 Blur14.8.

APPENDIX 1: EAW TILE SET

In each half of the table, the first two columns list the tile number (hex first, then decimal). Column 3 gives the file name (approximately) and column 4 a brief description.

00	0	Field8	Fields:grass (3:1)	20	32	Coast4	Straight coast, fields (2:2)
01	1	Grass	Grass	21	33	Coast5	Rounded O/s corner coast (1:3)
02	2	Sea	Sea	22	34	Road1	Bend, edge:edge, fields
03	3	Coast1	Straight coast, fields (2:2)	23	35	Road2	Straight, fields
04	4	Coast3	Shallow O/s corner coast (1:3)	24	36	Road6	Straight, fields
05	5	Coast2	Inside corner coast (3:1)	25	37	Road4	Diagonal, fields
06	6	Field1	Fields	26	38	Field7	Fields:grass (1:3)
07	7	Field2	Fields	27	39	Road8	Curve, edge:corner, fields
08	8	Field5	Fields + 2 small woods	28	40	Road9	Curve, edge:corner, fields
09	9	Field4	Fields	29	41	Forest7	Fields with small wood
0A	10	City1	City	2A	42	City5	Fields with small town
0B	11	City2	City edge: fields (3:1)	2B	43	River ND	River end, fields
0C	12	City3	City edge: fields (1:3)	2C	44	Coast6	Straight coast, fields (2:2)
0D	13	City4	City edge: fields (2:2)	2D	45	Road10	3-way junction, fields
0E	14	River1	Straight, fields	2E	46	City9	City edge: fields (2:2), diag road
0F	15	River2	Straight, fields	2F	47	Field6	Fields:grass (2:2)
10	16	River9	River bend (edge:edge), fields	30	48	City7	Fields with small town + lake
11	17	River4	River bend (edge:edge), fields	31	49	City8	Fields with scattered village
12	18	River5	Straight, city	32	50	Coast U1	Straight coast, bay, fields (2:2)
13	19	River6	River bend (edge:edge), city	33	51	Coast U2	Straight coast, port, fields (2:2)
14	20	River7	Straight, city edge:fields (2:2)	34	52	Coast BR	Straight coast, bridge, fields (2:2)
15	21	River8	Straight, estuary (2:2)	35	53	River BR	Straight, bridge, fields
16	22	Mts1	Mountain	36	54	River FK	Fork, fields
17	23	Mts4	Mountain:forest (3:1)	37	55	Coast RV	I/s corner coast (3:1), estuary
18	24	Mts3	Mountain:forest (2:2)	38	56	River TR	Straight, gap for railway, fields
19	25	Mts2	Mountain:forest (1:3)	39	57	Coast CT	Straight coast, town, fields (2:2)
1A	26	Forest4	Forest:fields (2:2)	3A	58		r
t (	3 :1	), estua	ry	18	24		
oun	t ai	n:forest	(2:2) 38 56 River T	R S	tra	i	g
gap	fo	r railwa	y, fields 19 25 Mts2 Mnta	in:	f	o	
(1	: 3)	39 57	oast CT Straight coast, town, f	iel	ds	(	2
	1A	26 For	st4 Forest:fields (	2:2	) 3	A	

APPENDIX 2: OBJECT CODES (decimal)

0	*?*
1	Oil plant
2	Ball bearing plant
3	Chemical plant
4	Aircraft factory
5	Airfield
6	Armaments factory
7	RR station
8	Shipyard
9	U-boat pen
10	V-1 site
11	Road Bridge Stone
12	Road Bridge Wood
13	Road Bridge Iron
14	Rail Bridge Stone
15	Rail Bridge Wood
16	Rail Bridge Iron
17	Flak
18	Light flak
19	Oil tank
20	Radar station
21	Radar tower
22	3D forest
23	Fort
24	Airfield fuel dump
25	Control tower
26	Airfield ammo bunker
27	Hangar
28	Airfield barracks
29	Airfield mess
30	Airfield O-club
31	Briefing hut
32	Tree
33	Trees
34	Tree
35	Trees
36	Tree
37	Trees
38	Tree
39	Trees
40	3D forest
41	Warehouse
42	3D forest
43	Train Engine
44	Train Tender
45	Train Boxcar
46	Train Oil Tanker
47	Train Flatbed
48	Train Flatbed Tanks
49	Train Flatbed Guns
50	Fuel Truck
51	Truck
52	Truck

55	Panther Tank
56	Sherman Tank
57	Freighter
58	Church
59	City1
60	City2
61	City3
62	City4
63	Cathedral
64	Farm
65	Manor house
66	Chateau
67	Windmill
68	?Haystack
69	Parachutist
70	?
71	?
72	Destroyer