Fastplay Replacement Tutorial Intermediate

By Milo Christiansen

Before Starting this tutorial you should have completed the basics part of this tutorial. This assumes you have the mod you made when you did that tutorial, if not you can use the “FastplayTutBasics.wz” file that came with that tutorial.

This tutorial details how to make an away mission style fastplay replacement by building off the scripts you made in the basics tutorial.

Centering the Camera

When you started the mod you made in the basics tutorial all you could see was black, right? This is because the camera focuses on the center of the map by default. Luckily this is an easy fix.

First a word about coordinates. Warzone uses two coordinate systems for specifying spots on the map, world coordinates and tile Coordinates. Tile coordinates are simply the tile count from the top left corner, for example the top left corner is z0 x0 one tile right and one tile down is z1 x1. World coordinates are the tile coordinates multiplied by 128, for example the middle of the top left corner tile is z64 x64. Some script functions use tile coordinates and some use world coordinates, although world coordinates offer greater precision it is almost never needed. For my scripts when I use a function that needs world coordinates I just use tile coordinates * 128.

Open “tutorial.slo” and “tutorial.vlo” in your text editor.

Add this to the vlo:

```
// tile to center view on
ViewCoords[0]  int      0         //tile X
ViewCoords[1]  int      0         //tile Z
```

This declares a two element array named ViewCoords to store the view coordinates in.

Now add this to the slo:

```
public         int            ViewCoords[2];
```

Now open flaME. Load your test map and place your cursor over the tile you want to center the view on when the mission starts. At the bottom of your screen are three sets of numbers, Tile is the tile coordinates, Pos is the world coordinates. For this tutorial we are interested in the tile coordinates only. Set the first ViewCoords entry to the number by “x:” set the second entry to the number by “y:”.

Now add this to the GameInit event in the slo:

```
centreViewPos(ViewCoords[0] * 128, ViewCoords[1] * 128);
```

centerViewPos takes world coordinates but ViewCoords is in tile coordinates so to make it work the value of ViewCoords must be multiplied by 128.

You can repackage your mod and give it a test run if you like. A lot nicer now that the camera starts off looking at your units, isn't it?

The Transport

The transport is arguably the most important unit in any away mission. Your mission would be rather boring if you only had the units you started with. Setting up the transport is the second hardest part of the tutorial scripts.

The first thing to do is set up the unit list. The unit list is a series of arrays that the transport event uses to decide what units to load.

Add this to the vlo:

```
// Unit List
UnitTemp[0]    TEMPLATE  "BarbarianTrike"
UnitFlag[0]    BOOL      TRUE

UnitTemp[1]    TEMPLATE  "BarbarianTrike"
UnitFlag[1]    BOOL      TRUE

UnitTemp[2]    TEMPLATE  "BarbarianBuggy"
UnitFlag[2]    BOOL      TRUE

UnitTemp[3]    TEMPLATE  "BabaFireCan"
UnitFlag[3]    BOOL      TRUE

UnitTemp[4]    TEMPLATE  "BabaRKJeep"
UnitFlag[4]    BOOL      FALSE

UnitTemp[5]    TEMPLATE  "BabaJeep"
UnitFlag[5]    BOOL      FALSE

UnitTemp[6]    TEMPLATE  "BabaJeep"
UnitFlag[6]    BOOL      FALSE

UnitTemp[7]    TEMPLATE  "BabaJeep"
UnitFlag[7]    BOOL      FALSE

UnitTemp[8]    TEMPLATE  "BabaJeep"
UnitFlag[8]    BOOL      FALSE

UnitTemp[9]    TEMPLATE  "BabaJeep"
UnitFlag[9]    BOOL      FALSE
```

UnitTemp is the name of the template (from templates.txt) to use for this unit, UnitFlag is a flag to tell the transport event if it should add this unit to the next load or not. There are ALWAYS ten entries in the unit list, one for every spot in the transport.

The reason for having entries you might not actually use is so that you never have to modify the slo for a new mission. Actually you can use the same slo with many missions, all you need to do tho make a new one is edit the vlo and map. There are a few exceptions to this but most of the time you can circumvent them with a little work, therefor making your scripts more versatile.

Now for the slo:

```
// Unit List
public          TEMPLATE  UnitTemp[10];
public          BOOL      UnitFlag[10];
```

Add this to the vlo:

```
// LZ coords, Must describe 3X3 tile area
LZCoords[0]    int       0         //tile X upper left corner
LZCoords[1]    int       0         //tile Z upper left corner
LZCoords[2]    int       0         //tile X lower right corner
LZCoords[3]    int       0         //tile Z lower right corner

// entrance/exit coords for transport should be on map edge
EnterCoord[0]  int       0         //tile X
EnterCoord[1]  int       0         //tile Z
```

Fill in the values the same way you did for centreViewPos using the comments for guidelines. For LZCoords the “upper left” and “lower right” part are suggestions, you can flip them or twist them however you want as long as the are diagonally opposed from each other.

Now add this to the slo:

```
// Transport Values
private         BOOL      TransOnMap;
private         DROID     TransportDroid;
private         int       countTrans;
public          int       LZCoords[4];
public          int       EnterCoord[2];
```

Now you need to add the triggers for the transport events to the slo:

```
trigger PlayerTransTrig(every, 600);   //every 60 secs
trigger OffMapTrig(CALL_TRANSPORTER_OFFMAP, player);
```

Next you need to add the events themselves. Add two new events one named “PlayerTransEvnt” using the “PlayerTransTrig” and one named “TransOffMap” using the “OffMapTrig” trigger.

Put this in the body of the TransOffMap event:

```
TransOnMap = FALSE;
```

And this in the body of the “PlayerTransEvnt” event:

```
if (not TransOnMap)	//make sure transporter not still on map!
{
	countTrans = 0;
	while ( countTrans < 10)
	{
		if (UnitFlag[countTrans] == TRUE)
		{
			addDroidToTransporter(TransportDroid, 
addDroidToMissionList(UnitTemp[countTrans], player));
		}
		countTrans = countTrans + 1;
	}
	//call in transport
	setTransporterExit(player, EnterCoord[0], EnterCoord[1]);
	flyTransporterIn(player, EnterCoord[0], EnterCoord[1], false);
	TransOnMap = TRUE;
}
```

And add this to the beginning of the GameInit event:

```
//stop player building on LZ
initAllNoGoAreas();
setLandingZone(LZCoords[0], LZCoords[1], LZCoords[2], LZCoords[3]);

//add player transport
TransportDroid = addDroidToMissionList(transporter, player);
```

Go ahead and test your mod again. If everything worked a transport should arrive carrying all the units whose UnitFlag is set to true once every 60 seconds.

Artifacts

Artifacts in this tutorial will work a little different from artifacts in the campaign. Instead of giving you technologies to research artifacts will give you units for the next and future transport loads. Artifacts are the most complex part of this tutorial.

First you need to make the artifact list. The artifact list is to artifacts what the unit list is to units. There is always one entry in the artifact list for every entry in the unit list, and like the unit list, the redundant entries are so the slo never needs modifying.

First add this to the vlo:

```
// Artifacts
```

```
ArtSnd              SOUND     "pcv352.ogg"  //artifact sound

// Artifact List
// ArtType:
//   0 = Null Entry (corresponding unit marked true from start)
//        Set all other values except ArtValStruct or ArtValFeat to
0
//   1 = artifact placed by script
//   2 = get from STRUCTURE
//   3 = get from FEATURE
// ArtValStruct: ID of structure to get artifact from MUST BE VALID
// ArtValFeat: ID of feature to get artifact from MUST BE VALID
// ArtX, ArtY: artifact coordinates

ArtType        [0]  int            0
ArtValStruct   [0]  STRUCTURE      0
ArtValFeat     [0]  FEATURE        1
ArtX           [0]  int            0
ArtY           [0]  int            0

ArtType        [1]  int            0
ArtValStruct   [1]  STRUCTURE      0
ArtValFeat     [1]  FEATURE        1
ArtX           [1]  int            0
ArtY           [1]  int            0

ArtType        [2]  int            0
ArtValStruct   [2]  STRUCTURE      0
ArtValFeat     [2]  FEATURE        1
ArtX           [2]  int            0
ArtY           [2]  int            0

ArtType        [3]  int            0
ArtValStruct   [3]  STRUCTURE      0
ArtValFeat     [3]  FEATURE        1
ArtX           [3]  int            0
ArtY           [3]  int            0

ArtType        [4]  int            0
ArtValStruct   [4]  STRUCTURE      0
ArtValFeat     [4]  FEATURE        1
ArtX           [4]  int            0
ArtY           [4]  int            0

ArtType        [5]  int            0
ArtValStruct   [5]  STRUCTURE      0
ArtValFeat     [5]  FEATURE        1
ArtX           [5]  int            0
ArtY           [5]  int            0

ArtType        [6]  int            0
ArtValStruct   [6]  STRUCTURE      0
ArtValFeat     [6]  FEATURE        1
ArtX           [6]  int            0
ArtY           [6]  int            0
```

```
ArtType         [7]  int            0
ArtValStruct    [7]  STRUCTURE      0
ArtValFeat      [7]  FEATURE        1
ArtX            [7]  int            0
ArtY            [7]  int            0

ArtType         [8]  int            0
ArtValStruct    [8]  STRUCTURE      0
ArtValFeat      [8]  FEATURE        1
ArtX            [8]  int            0
ArtY            [8]  int            0

ArtType         [9]  int            0
ArtValStruct    [9]  STRUCTURE      0
ArtValFeat      [9]  FEATURE        1
ArtX            [9]  int            0
ArtY            [9]  int            0
```

The header for the artifact list is very important, not to Warzone, but to you. If you ever want to reuse these scripts it would be a real pain to figure out the acceptable values for ArtType, for example, without this header.

Now for the slo:

```
// Artifacts
public          int             ArtType[10], ArtX[10], ArtY[10];
public          STRUCTURE       ArtValStruct[10];
public          FEATURE         ArtValFeat[10];
private         BASEOBJ         ArtOBJ[10];
private         BOOL            ArtPlaced[10];
private         BOOL            ArtCollected[10];
private         FEATURE         ArtID[10];
private         int             ArtCollectedCount;
public          SOUND           ArtSnd;
public          int             ArtDefaultVal;
private         int             ArtLoopCount;
private         BOOL            ArtLoopState;
```

The private values are all for use in keeping track a which artifacts are placed and/or collected or for use in the “InitArtifacts” event.

Now for the artifact events.

First add this trigger to the slo:

```
trigger everysecond(every, 10);
```

This is the trigger the main artifact event will use.

Now add a new event named “ArtifactEvnt” using “inactive” as its trigger.

Now put this in the body of the “ArtifactEvnt” event:

```
// all artifacts collected?
if (ArtCollectedCount == 10)
{
     setEventTrigger(ArtifactEvnt, inactive);
}
else
{
     ArtLoopCount = 0;
     while (ArtLoopCount < 10)
     {
          if (ArtPlaced[ArtLoopCount] == FALSE)
          {
               ArtLoopState = FALSE;

               // scripted placement
               if (ArtType[ArtLoopCount] == 1 and
ArtPlaced[ArtLoopCount] == FALSE)
               {
                    ArtLoopState = TRUE;
               }

               // From Structure
               if (ArtType[ArtLoopCount] == 2)
               {
                    // is object dead and realated artifact not
placed?
                    if (ArtValStruct[ArtLoopCount] == NULLOBJECT and
ArtPlaced[ArtLoopCount] == FALSE)
                    {
                         ArtLoopState = TRUE;
                    }
               }

               // From Feature
               if (ArtType[ArtLoopCount] == 3)
               {
                    // is object dead and realated artifact not
placed?
                    if (ArtValFeat[ArtLoopCount] == NULLOBJECT and
ArtPlaced[ArtLoopCount] == FALSE)
                    {
                         ArtLoopState = TRUE;
                    }
               }

               // OK to place crate?
               if (ArtLoopState == TRUE)
               {
                    // place artifact crate, and allow check for prox
                    ArtID[ArtLoopCount] = addFeature(crate,
ArtX[ArtLoopCount] * 128, ArtY[ArtLoopCount] * 128);
                    ArtPlaced[ArtLoopCount] = TRUE;
               }
          }

          if (ArtPlaced[ArtLoopCount] == TRUE and
ArtCollected[ArtLoopCount] == FALSE)
          {
               if (droidInRange(player, ArtX[ArtLoopCount] * 128,
ArtY[ArtLoopCount] * 128, 200))
               {
                    ArtCollectedCount = ArtCollectedCount + 1;
                    ArtCollected[ArtLoopCount] = TRUE;
                    playSoundPos(ArtSnd, player,
ArtID[ArtLoopCount].x, ArtID[ArtLoopCount].y,
ArtID[ArtLoopCount].z);
                    destroyFeature(ArtID[ArtLoopCount]);
                    UnitFlag[ArtLoopCount] = TRUE;
               }
          }

          ArtLoopCount = ArtLoopCount + 1;
     }
}
```

Take a little while to study the above code. This is the code that puts every thing together as far as artifacts are concerned. It is more or less divided in three parts, The initial condition check to see if the artifact needs to be placed, the actual placement, and the part that handles recovery by a player unit.

Careful study will reveal that the above code will not work with out either a great deal of changes or some supporting code some where, and that is where the next part comes in.

Add this line between the last trigger and the GameInit event:

```
event ArtifactEvnt;
```

This is a forward declaration of ArtifactEvnt. A forward declaration lets you use an event before it is created, in this case although ArtifactEvnt is declared after GameInit you will be able to Reset the trigger for ArtifactEvent in GameInit.

Add this to the “GameInit” event:

```
// Init Artifacts
ArtLoopCount = 0;
while ( ArtLoopCount < 10)
{
     if (ArtType[ArtLoopCount] == 0)
     {
          ArtCollectedCount = ArtCollectedCount + 1;
          ArtPlaced[ArtLoopCount] = TRUE;
          ArtCollected[ArtLoopCount] = TRUE;
     }
     ArtLoopCount = ArtLoopCount + 1;
}
setEventTrigger(ArtifactEvnt,everysecond);
```

This code takes care of any Null entries in the artifact list and makes sure that ArtCollectedCount will work OK.

There is one thing you should make a note of: all maps that you want to use these scripts with MUST have at least one building and one feature for the artifact list to work.

Congratulations on finishing this tutorial! If you have further questions about Warzone fastplay scripting after finishing this a good place to look would be the original fastplay scripts. To get a list of these scripts read the original “fastdemo.wrf”.

Happy Scripting!!!!