MechCommander Gold Campaign Builder's Guide

Supplement 1 – Videos

Revision 0.1

By Cmunsta

Revision History

Rev 0.1

- First release of the document.

Introduction

This document is a tutorial on creating and using videos for your MechCommander Gold campaigns. Unlike the first three guides, this document is called a supplement. This is simply because I've sort of given up on the monolithic document idea. Mostly because it became rather unwieldy to maintain, but also this allows me to focus a little more on a specific topic.

Because of this, there will not be a Part 4 and 5 as I originally planned. Not because the information will not come out eventually, but because I'll be creating more of these supplements. So instead of a couple more large guides, I'll be creating several smaller guides, but the content will effectively be the same, just spread out a little more.

Anyway, as for this particular supplement, we'll be covering videos. Both the operation videos, and Warrior pilot videos. We'll also explore some of the things we can do with them. As with part 3 of the guides, I'll be creating simple solo missions to demonstrate things, but it will be easy to see how to extend it to a full campaign.

I will however assume that you have videos already created that simply need to be processed into something that MCG can use. These source videos can be anything at all. Something you recorded with a handheld video camera, some footage from a webcam, a segment from “Debbie does the entirety of Texas” (acquired from some less than reputable bittorrent site perhaps)... well you get the idea. I don't care what the source video is or how you got it. But you'll be needing some, and I'll assume you have it.

More Tools of the Trade

As with the previous guides, we'll be needing a good text editor. Once again, I suggest Notepad++. as it is a very nice, and free text editor.

Since this is a tutorial on making videos for MCG, we'll also need some video tools. On my site are links to one free video editing suite, but anything will do. You may also want to get hold of VirtualDub or one of its variants as it is a rather good video conversion and processing tool. Just google for it.

Another tool we'll use is GIMP. A link to it is on my site.

Since MechCommander Gold uses Smacker as its video codec, we'll need to get hold of it as well. It is available from RAD for free for non-commercial purposes. Though be aware that because this is a rather old game, we'll need an old version of the tool (available from the RAD site). See the forum posts by LouHobbs on the MechCommander.org forums for a link to it (kudos to LouHobbs for digging it up).

Finally, you will also need my simple tools for extracting and packing FST and PKK files as well as my PAK extraction tool and the new (as of this writing) PAK file creation tool. These are available on my site, same place where you got this document.

A Matter of Conversion

To start things off, we'll deal with the operation videos. These are the videos that play in the briefing tab of the logistics screen when we start a mission.

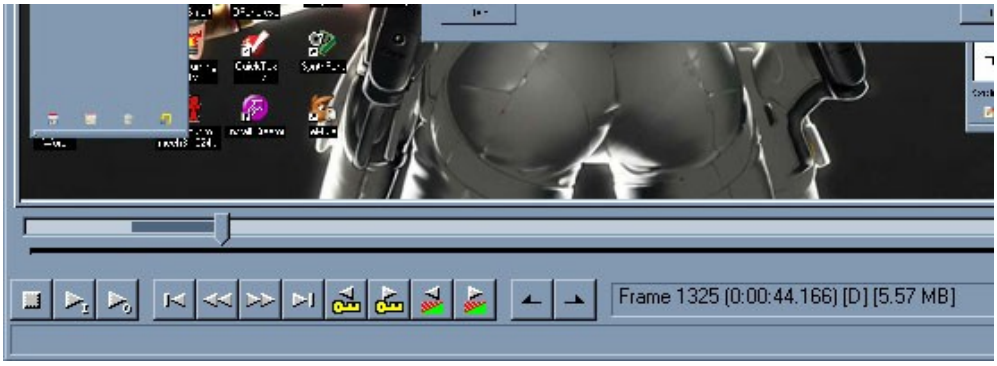
Operation videos should all have the same format. The technical details are as follows:

Video Size	176 x 128 (horizontal x vertical)
Frame Rate	15 frames per second
Audio Track	22050 Hz, 8 bits (16 bits in the expansion), Mono

So the first thing we should do is make sure that our source video abides by the specs above. We don't have to worry about the audio right now since the Smacker tool can handle the conversion, but to get good quality video, it may be best to scale the video and frame rate with something like VirtualDub. This tool can also help select a small section of the source video in case we don't want all of it.

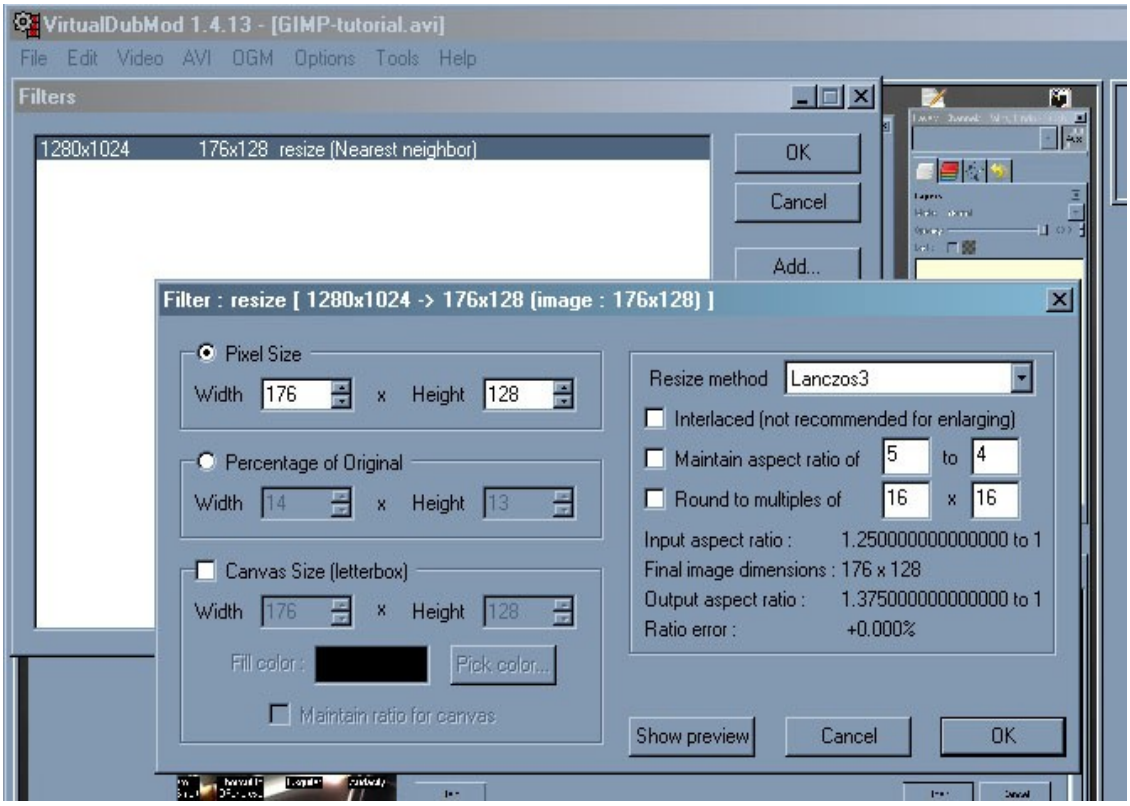
For this example, I'll be using a segment of the Gimp video tutorial I had on my website. Its resolution is 1280x1024 so it is much too big, and it is also at 30 fps so that too needs to be converted. I'll show what I did using VirtualDub, though any tool that can do this will be fine.

Open VirtualDub and load your original video. Now as my video is rather long, I'm going to select only a small portion of the video to process. So I advanced the slider to the start point where I want to convert things, press the little left pointing black arrow button to mark the start point (second button from the right in screen shot), move the slider to the ending point I want then press the right pointing black arrow button (rightmost button in screen shot).

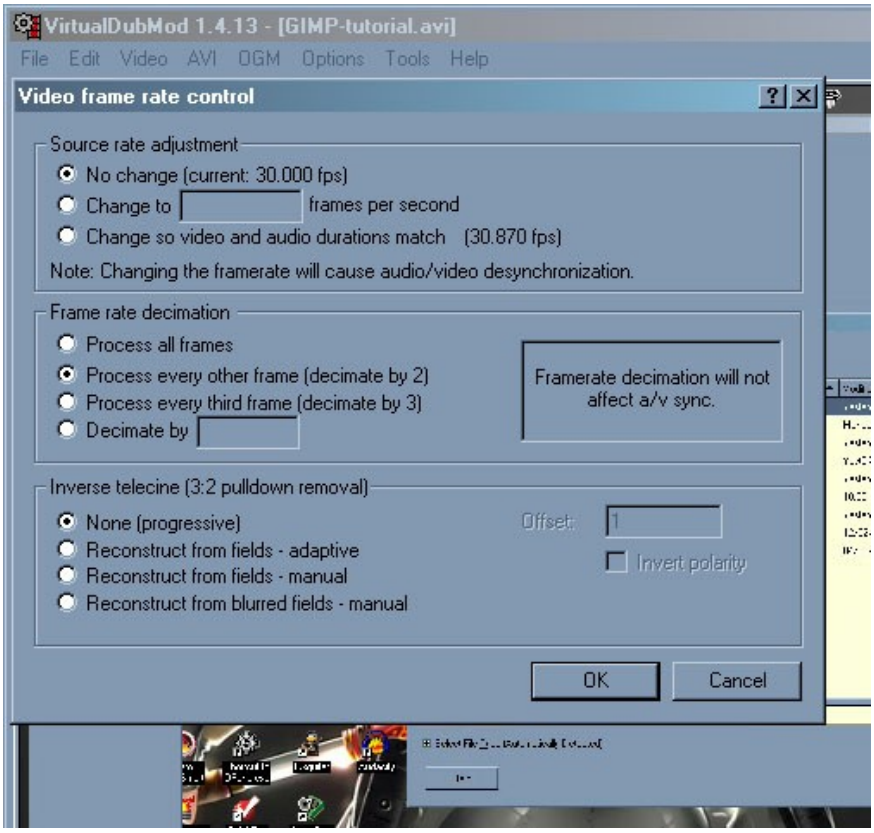


Note the dark blue area shown on the video slider. This is the section I've defined to be processed.

Now I select Video → Filters from the menu and select the Resize filter and configure it to resize the video to 176 x 128. Make sure the “Maintain Aspect Ratio” checkbox is turned off or you will likely not be able to get the correct size. You can also select the resizing algorithm. I chose Lanczos3 as this is a very good algorithm, though you may want to experiment.



Now select Video → Frame Rate to change the frame rate to 15 frames per second in the output by "decimating" the source by 2 (30 / 2 = 15 after all).



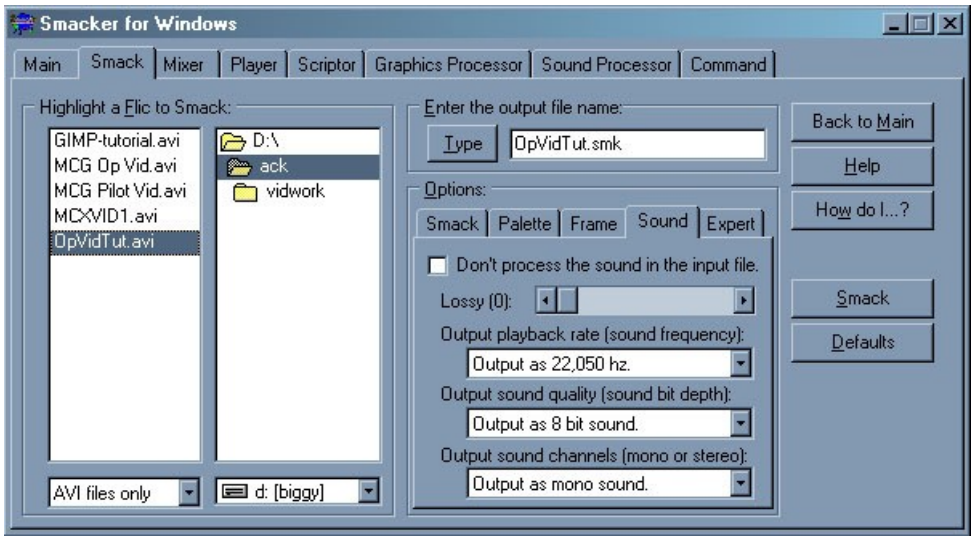
Make sure that “Full Processing Mode” is checked in the Video menu, then select the compression codec via Video → Compression. If the video is short, then your best bet is to simply use Uncompressed RGB frames as this will give the highest video quality (remember we'll be compressing this though Smacker later. That is, we'll be re-re-compressing). If not, then use one of the standard codecs but give it lots of bitrate. This will make a larger file of course, but we just want the output of this step to go into smacker anyway, so this is just temporary.

Under the AVI → Audio menu, make sure to select the audio track and either do a direct stream copy, or do full processing as you wish.

When all settings are satisfactory, select AVI → Save as AVI and wait until it is done.

Now that we have a video file that is of the correct size and frame rate, we can now run it through Smacker and create a .SMK file.

Start the Smacker Tool and go to the Smack tab. Select the video we just made, make sure the output filename is OK, and go through the option tabs. Most can be left at the default values, but on the Sound option Tab, make sure the output rate is at 22050 Hz, mono, as well as either 8 bit or 16 bit data. 16 bit sound will have a higher quality, but will take twice as much space. If you want that same interlaced look that the official campaign operation videos have, under the Smack option tab, select “Interlace” from the Height Compression drop down box.



Press the Smack button on the right to convert to a smacker file. After a bit of processing, we have our video. Copy it into the {MCX}\data\movies folder.

It's in the Game

Now that we have our smacker video, it is time to put it into a mission. For a full blown campaign, we would of course need to do this multiple times, but it is no more complex for one as for ten.

So make a solo mission using the mission editor. Doesn't matter what the mission is since we just want to see our video in the logistics screen. Save this mission as “opvidtut” or something like that.

Once done copy the .SOL file created in the savegame folder and extract all the files (only two in there). The first file, which is what I've been calling the masterdef file (see Guide Part 1), has an entry we need to modify.

Open the file in a text editor and under the [General] section, change the PurchaseFile entry to something other than solopurchase. I changed mine to this:

```
st PurchaseFile="opvidtutpur"
```

Save the file and rebuild the SOL file and copy it back to the savegame folder. (of course, if this was a campaign, then we would be dealing with a PKK instead of a .SOL, but as they are the same thing really, as shown in Part 1 of the builder's guide, this shouldn't be a problem).

Now extract solopurchase.fit from MISSION.FST, put it in your {MCX}\data\missions folder and rename it to opvidtutpur.fit (or whatever name you gave in the masterdef file). Now open it in a text editor.

Here you can set the number of campaign missions to be expected later in this file, though for solo missions this doesn't really have much of an effect. You can also change the cost of mech armor, mech internals, and mech engines, as well as a multiplier for the cost or clan mechs. We can also change the cost of hiring different levels of MechWarriors. But this isn't why we are here.

For each mission in our campaign (in this case, just one as this is a solo mission), we need to define a [Mission***X***] section (where ***X*** is the mission number starting at 0). Now there appears to be a definition here for which purchase file to use (solo missions default to “NetPur” which just gives everything) and also an entry for Operation. We'll get to this last one in a bit.

Last time I tried to change the PurchaseFile entry, it had no effect, but maybe I messed up on that, so you can test it if you wish. But for now leave it alone since this is also not why we are here.

To tell the game we want a given operation video to show up in the logistics screen, we can add a OperationCinema entry. Since my video is called opvidtut.smk, I added this:

```
st OperationCinema = "opvidtut"
```

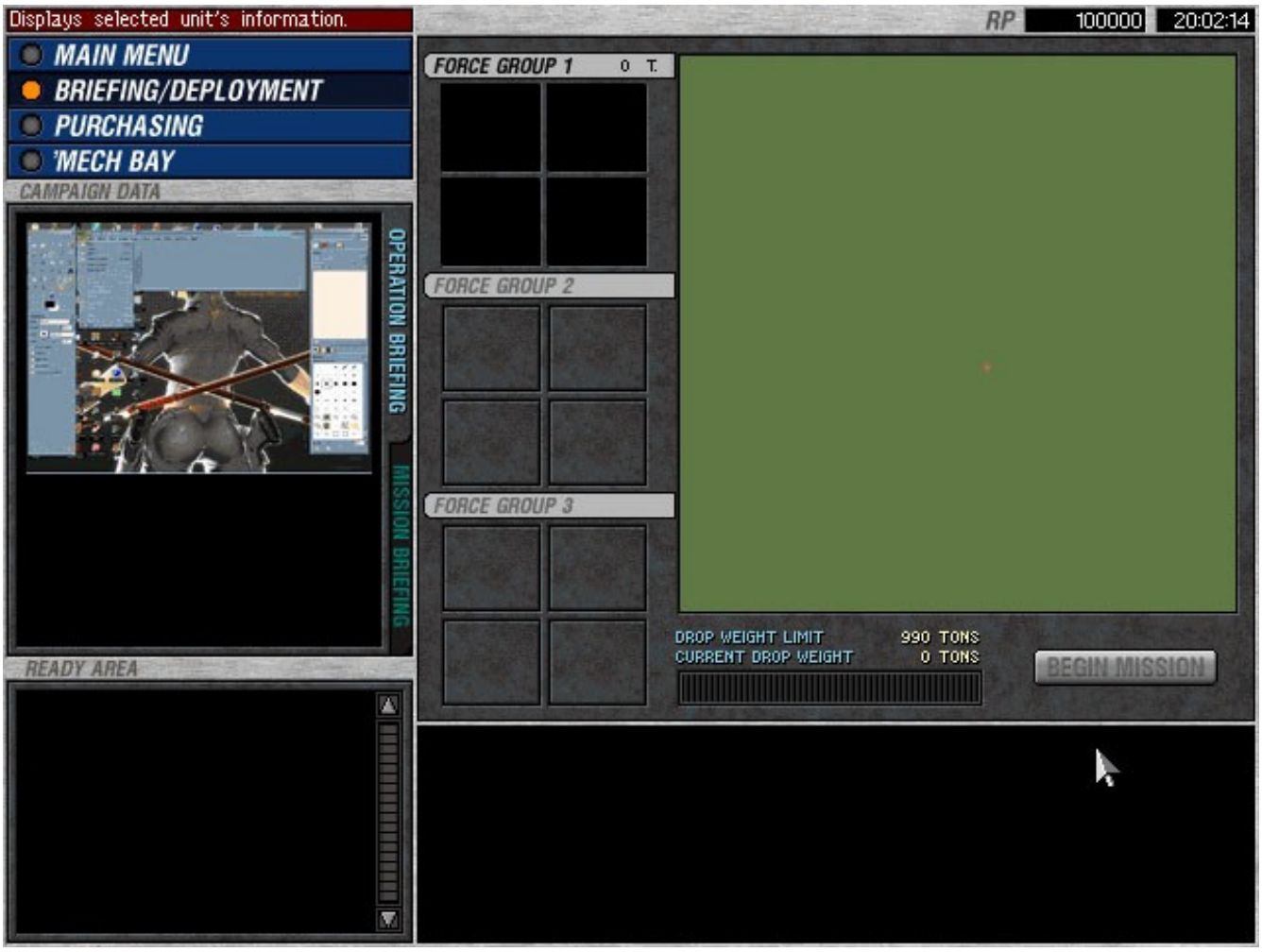
Now the game will know what video to play when the player clicks the briefing tab. But we can go a step further. If we want the video to start playing immediately when the player gets to the logistics screen (just like in the very first mission of the official campaigns) then we can add an AutoPlay entry to the section. I added this:

```
l AutoPlay = 1
```

So now my entire [Mission0] section looks like this:

```
[Mission0]
st PurchaseFile = "NetPur"
l Operation = 1
st OperationCinema = "opvidtut"
l AutoPlay = 1
```

Save the file and try the mission out. Here is my result:



The video auto-plays immediately at mission start just as we wanted.

Everything is working as it should, but there is one small catch. The image that shows up just before the video starts playing (while Betty is saying “Incoming Transmission”) is the one from the official campaign. Hmmmm... Would there be a way to get our own image up there? Indeed yes!

The images in question are, for the original campaign, LSB_OP1.TGA to LSB_OP5.TGA and for the expansion campaign, MCXCard1.TGA to MCXCard3.TGA. There are also two other special images, one is completely blank (no text on it, just the blue background) and is called LSB_OP0.TGA and the other has the “End Transmission” text on it and is called LSB_OP6.TGA.

Now the immediate thought is to simply override one of these images for our own use, but this could cause problems both with the official campaigns, and with any other user made campaign. But there is a bit of a work around.

The Operation entry in our opvidtutpur.fit file actually gets used as the number after the LSB_OP part for the official campaign and after MCXCard if in the expansion campaign. If we used a number other than 0-6, we could make our own images and set the Operation number to tell the game to use it instead AND we wouldn't even have to override the official ones.

So to do this, first extract LSB_OP0.TGA from ART.FST (this is the blank image) and put it in {MCX}\data\art\logart folder then rename it to LSB_OP7.TGA. Load the image up in an image editing tool (I use GIMP) and add any text you want to it then save.

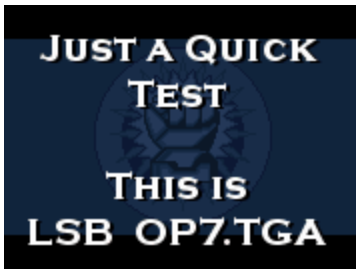
In the opvidtutpur.fit file, change the Operation number to 7 so that it will use our new image. And save the file.

This is what I have:

opvidtutpur.fit file:

```
[Mission0]
st PurchaseFile = "NetPur"
l Operation = 7
st OperationCinema = "opvidtut"
l AutoPlay = 1
```

LSB_OP7.TGA image:



Save everything and try it out again. Now our custom image shows up instead of the old text from the official campaign.

A few extra notes:

For campaigns we already know if we are overriding the original or expansion campaigns. But for solo missions, we use the fact that we are on a Port Arthur or Cermak map to determine this. Though of course, we could also simply change the same flag we do for campaigns to use the other one regardless. See Part 1 of the guide for details on doing this kind of thing.

As to the images that show up just before a video, even though we have a bit of a work around so as not to override the official campaigns, I don't know of any way to specify the prefix of the name and so it is still possible to mess things up for other campaigns/solo missions. The easiest thing to do would simply be to set the Operation number to 0 as this will make it use the blank image (no text). It isn't like we *really* need to have that text there, it's just a bit of extra color. You could even put the image with the text at the start of your video and so have the best of both worlds really. No possible way to stomp on other peoples images, and you would still get the image and text.

When it comes to the videos themselves, you will notice that the colors in the video are somewhat different from what you have when you converted it into a Smacker file. This is because the Smacker videos only have a 256 color palette to them, and the game is using a palette as well. What is happening is that the game (or the smacker player) is converting the colors in the video on the fly as it is playing the video so that it fits with the palette used on the logistics screen. This still seems to give better results than the alternative which would be to force a palette onto the video when you create the SMK file. Though if you find your smacker file looks pretty bad, then that may be an option to try. You can do this in the Palette options tab of the Smacker Tool. For the source palette, you can use one of the default .TGA images that make up the logistics screen. For example, LSB_OP0.TGA. Give that file to the Smacker Tool and it'll convert the video while forcing that palette into the video. But again, this may or may not give better results. Try both ways.

The audio volume may need some adjustment. Since the logistics screen has some background music playing, the video may end up being too low in volume (or too high), in which case some adjustments will be required.

Well that about covers all the ins and outs of operation videos.

Well now that we've seen how to do operation videos, lets try our hand at those little pilot videos that show up in game. You know the ones that appear when one of your guys thinks he's saying something witty, or more usually, when he's in a panic. Those ones.

If you've read the preliminary document I made about creating new MechWarriors, then you already know some of what is involved. But first, let us take a second to check out the technical specs of the pilot videos.

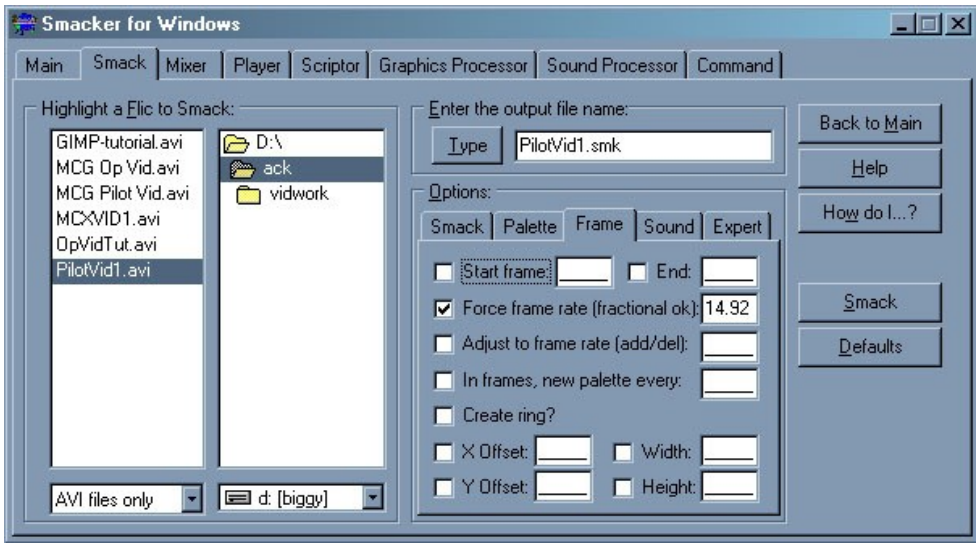
Video Size	48 x 36 (horizontal x vertical)
Frame Rate	14.92 frames per second
Audio Track	No Audio

The first thing to notice is the rather oddball frame rate. I haven't tried to simply set it to 15, but 14.92 fps is what the RadTools file info gives for the pilot videos that come with the game. Though it will likely work at 15 fps anyway since these vids aren't very long.

The second thing to notice is the lack of an audio track. We'll see why in a bit, but for now it means that we should ensure that “Don't process sound in the input file” check box is on when using the Smacker Tool.

Getting the size of the video right is basically the same as for the op videos, so I won't go over it again, though we won't do the frame rate conversion in VirtualDub. Instead, we'll let the Smacker Tool do it. So all VirutalDub needs to do is resize the video to 48x36. We can have it remove the audio if need be, it won't matter since Smacker will be ignoring the audio anyway.

Once we have some source video in the correct size, run it through the Smacker tool making sure that it doesn't process the audio and that we force the frame rate to 14.92 fps. Once set, hit the Smack button to convert it.

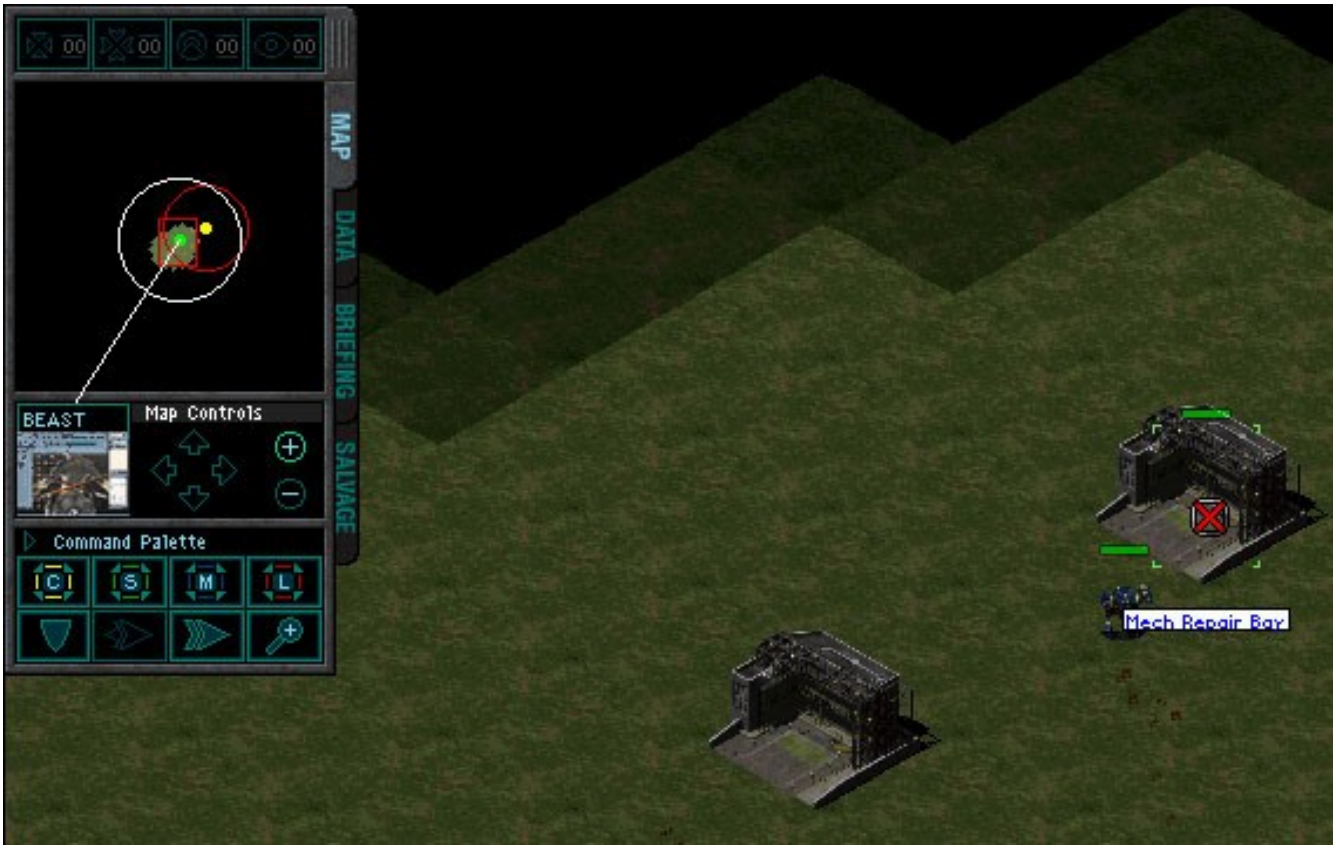


Typically, there are three pilot videos. One for just normal communications, one where the mech is under attack, and one when the things are going *real* bad. As in ejecting etc. We'll see that we can actually have more than this, but as this is simply to demo how to get pilot videos in the game, we will start by simply overriding one of the original pilot videos. Namely Beast's.

Go to your {MCX}\data\movies folder and rename or backup Beast's videos so that we aren't losing them. They are called PILOT00A.SMK, PILOT00C.SMK and PILOT00D.SMK. The 'A' video is the normal one, the 'C' is the video when under attack etc, and of course the 'D' video is the one when he's ejecting. For this test, I'm just overriding the 'A' video, though if you have more than one video ready, you can replace the others as well. Also, if you prefer, go through the MechWarrior Creation tutorial and make a brand new pilot and set the videos on that one instead, doesn't really matter. Once the file is backed up, copy your new video to the {MCX}\data\movies folder and name it PILOT00A.SMK (or the appropriate name(s) for the other vids and/or warriors).

Now create a simple solo mission where there are a few buildings to capture (The mech that captures the building will radio back using the 'A' video so we'll be able to see it). I called my mission pvidtst1 (for pilot video test #1). Yeah, real catchy ain't it? Kind of rolls of the tongue and falls into a puddle of mud... but I digress.

Start the mission, hire Beast (or whichever Warrior has the vids assigned to him) and go capture some buildings. This is what I got:



The 'A' video is playing just fine.

Getting Creative

OK, we can create little videos for the pilots. But can we do more? LouHobbs asked a variant of this question on the MechCommander.org forums and so I decided to delve a little more. Basically, he wanted to be able to play one of those small videos on demand. This presents a couple of problems. First, the pilot videos are used by starting the video and playing a separate audio track. When this audio track terminates, so does the video. Therefore, we'd need a way to get a pilot to use an audio track of our choice, and associate it with a given pilot video. The second problem is getting this information into the game, and lastly, getting the game to play the video/audio.

Now the first thing I checked was the script functions of the game. There are two interesting ones. Namely PlayVideo and PlaySmacker. Unfortunately, both of these functions have been disabled in the game. PlaySmacker doesn't actually call any internal functions, so it doesn't actually do anything, and PlayVideo eventually calls an internal function that would of played a video, but that function has been disabled and simply returns. So PlayVideo also effectively does nothing at all. So that pretty much stopped this method cold.

My next idea was to simply have a pilot have the video/audio we want and play that. This can work, but it creates some problems of its own, though at least it can be done. So this is what we'll be talking about here. The method presented here does have some limitations as we'll see.

First thing we need to do is to get hold of the data file that is used by the game that handles what video and audio file to use when doing the radio messages. This file is radio.csv held in the MISC.FST file. Extract this file and place it in the {MCX}\data\sound folder.

The first line of this file is simply telling you want each column (separated by commas) does. Here's a better description of each:

Column Name	Description
Title	Descriptive name of entry. Ignored by game.
Priority	Priority of entry (1-4). Lower values mean higher priority. If the game tries to play an audio with a higher priority than what is currently playing, the new one can interrupt. Otherwise, the current one continues to play.
Shelflife	In seconds. How long the sound should be allowed to play (see Pilot ID)
Movie	A lowercase X ('x') means no video, any other letter means to play that video. An uppercase X ('X') is allowed. This is used at the end of the pilot video name.
Styles	Number of audio files (1-3) that represent the same thing. Used by the game to create variety in the radio messages. (see Map To)
Style1%	Percentage chance to pick audio file 1 (Style 1)
Style2%	Percentage chance to pick audio file 2 (Style 2)
Style3%	Percentage chance to pick audio file 3 (Style 3)

Column Name	Description
Pilot ID	This is a 'y' or 'n' entry. If 'y' (yes) the pilot will first identify himself before playing the audio (e.g. Beast saying: “Beast Here...”). If the Shelflife entry is too short, this will not be played even though this entry is set to yes. The actual audio file will play in its entirety regardless.
Map To	Audio number to use. The audio number refers to the audio entry in the pilot's audio .PAK file. If the game chose to use Style 2, then it will actually use this number +1. If the game chose to use Style 3, then it will use this number +2.

Now we could add entries to this file, but unfortunately they don't seem to get used by the game so it is somewhat redundant to do so. Also, this file is used globally for all pilots in both the original and expansion campaigns, so we would rather avoid changing this file. However, we can see that there are entries we could use, and we may need to edit this file no matter what.

For this tutorial, I won't change this file at all, but I wanted to point out where the game gets it info from and how it uses it. The interesting data here is the Movie column. As mentioned in the table above, this tells the game what movie to play along with which audio file. The letter is directly related to which video to use. The only special value here is that a lowercase X ('x') is interpreted by the game as meaning no video. Anything else is used as the last letter in the pilot video. So if we changed an 'A' to a 'Q', then the game would look for a video called ***xx***Q.SMK where ***xx*** would be the video prefix defined in the warrior profile. In the case of Beast, this is PILOT00 and so the video would be PILOT00Q.SMK.

So now that we know this, how do we use this information. We can make any pilot radio any of his messages in a script via the PlaySpeech function. This function takes two parameters. The ID of the pilot who is sending the radio message, and message number. The message number is simply which row in radio.csv to use. Since line 1 is merely a column title description, and the valid radio message numbers start at 0, we can figure out the number to use by simply taking the line number and subtracting 2. To test this out, lets make a simple solo mission. We'll make the first player warrior say the he is under artillery fire if he goes to a specific location on the map.

Open the mission editor, put a drop zone and put a small square of concrete nearby and write down the coordinates of the piece of concrete (it is just so we know where to go). Also put a swiftwind in a far corner of the map so there is no chance of seeing it until we go look for it. Save this mission as pvidtst2.

Open pvidtst2.abl in your text editor. Well add code to check if the player is in the designated area, and if so start the radio message. We'll need a flag for this so add one, we'll also need an array or three reals to hold the coordinates we'll be looking for. I added this to the variables:

```
static boolean      StartedRadioMessage;
static real[3]      Pos;
```

We'll also need to initialize these variables, so I added this to the init function:

```
StartedRadioMessage = false;
Pos[0] = 570.0;
Pos[1] = 70.0;
Pos[2] = 0.0;
```

Of course, use the coordinates you got for your mission.

Now in the main code, we'll simply add a check to see if the player reached the area where the concrete is and if so, set the flag and start the radio message. I added this at the start of the code (line 112 or so):

```
if (not StartedRadioMessage) then
    if (InArea(GetVehicleID(PLAYER_FORCE,0,0), Pos, 30.0, -1)) then
        StartedRadioMessage = TRUE;
        PlaySpeech(GetPilotID(GetVehicleID(PLAYER_FORCE,0,0)),24);
        AddStrikes(0,1,1);
    endif;
endif;
```

Note that I'm using radio message 24 which is the RADIO_UNDER_ARTILLERY line from radio.csv. It will play audio file #7, and use video 'C'. The call to AddStrikes here simply gives you a large artillery strike. I did this so that we can see when the code will call the PlaySpeech function in game. If you see the artillery strike added but no speech happens, you'll know something is wrong with the PlaySpeech call.

Save the file and take a mech over to the piece of concrete. When he gets in range, the player will radio you.

Gator thinking he's under attack:

So we can now get one of our guys to radio specific messages, but that is one of our guys. We need a way to get some other pilot to radio us.

We can do this in the same way we add extra drop groups for the player (see Part 3 of the guide for details). Basically, we need to add a friendly mech onto the map and have them radio the player when required. The problem however, and this is where some of the limitations of the method shows up, is that radio messages can only be seen if the pilot doing the calling is on the same team as the player. Otherwise, nothing will show up. This means that we can't have an enemy or even an allied mech radio the player, it *must* be a friendly mech.

Further, if the friendly mech is in a shutdown state, then that mech's radio is also turned off. So we will need to make sure the the radio is on.

So lets modify our sample mission to do this. First, we need to get the friend mech onto the map. We'll do this manually. Open pvidtst2.fit in a text editor and scroll to the bottom of the file. Copy the entire [Part1] section and past it in as [Part2], and change the NumParts entry to 2. Change the [Commander1Group:0] section so that the Swiftwind is now using Part 2 like this:

```
[Commander1Group:0]
l[12] Mates             = 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
```

Now we edit [Part1] to be a friend mech. I turned it into a Cougar-J like this:

```
[Part1]
ul ObjectNumber         = 17
ul ControlType          = 2
b PlayerPart            = True
ul ControlDataType      = 1
c MyIcon                = 0
c TeamID                = 0
c CommanderID           = 0
st ObjectProfile        = "PM201300"
ul Pilot                = 2
f PositionX             = -2496
f PositionY             =  448
f PositionZ             = -1.0
f Rotation              =  45
ul Gesture              = 2
f Velocity              = 0.0
l Active                =  0
l Exists                =  1
```

Note how I changed the pilot entry to 2, and that the Exists flag is on. Also note all the other changes.

Now we need to add a new pilot for this mech. Change the NumWarriors entry to 2, copy the [Warrior1] section and make a new [Warrior2] section. Change the profile to one of the player profiles (I chose Countess), and finally set the brain file to pbrain. This is what I ended up with

```
[Warriors]
ul NumWarriors = 2
uc CaptureChance = 0
st BrainParameterFile = "pvidtst2Params"
[Warrior1]
st Profile = "PMW00025"
st Brain = "DredAttack01"
[Warrior2]
st Profile = "PMW00068"
st Brain = "pbrain"
```

And finally, add a new commander group for this new mech. I added this at the end of the file:

```
[Commander0Group:1]
l[12] Mates             = 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
```

Save the file and open pvidtst2.abl so we can change the code a little. First, we need to change which pilot we are saying needs to send the radio message, and since this will be a friend mech just sitting around, we'll need to also turn it's radio on. The code we added before will now look like this:

```
if (not StartedRadioMessage) then
    if (InArea(GetVehicleID(PLAYER_FORCE,0,0), Pos, 30.0, -1)) then
        StartedRadioMessage = TRUE;
        SetRadio(GetVehicleID(PLAYER_FORCE,1,0),TRUE);
        PlaySpeech(GetPilotID(GetVehicleID(PLAYER_FORCE,1,0)),24);
        AddStrikes(0,1,1);
    endif;
endif;
```

Save everything and try it out. Select a pilot that isn't Countess to run up to the concrete square. When you get there, Countess will say that she is under attack.

Some things to note here, although we do get Countess to say the correct line, there is still a line being drawn onto the TAC map. This is one of the problems with this method. Also, the friend mech we add must exist on the map (we can't have mechs that aren't on the map radioing in) and also, it must be a mech if we want to see the pilot video. Even though vehicle crews also have a definition for a video file in their profiles, it is never used by the game.

To use this technique with our own videos and audio, we could create special pilots whose job is really just to sit around and radio messages. You'd create the small pilot videos for them, make new audio as well, and then have them sit somewhere and radio the player at the appropriate time.

Well that about covers everything I can think of concerning the videos. I hope to see new videos being made for user campaigns.

As usual, the remainder of this document are listings of the files used in the tutorial.

-cmunsta

Listing of OpVidTutPur.fit

```
FITini

[Header]

l NumCampaignMissions = 31
st MultiplayerFile = ""

[PurchaseCosts]
l Armor = 40
l Internal = 50
l Engine = 750
f clan = 4.0

[PilotCosts]

l Green = 3200
l Regular = 5000
l Veteran = 7200
l Elite = 9800

[Mission0]
st PurchaseFile = "NetPur"
l Operation = 7
st OperationCinema = "opvidtut"
l AutoPlay = 1

[FITend]
```

Listing of PvidTst2.fit

```
FITini

// ------------------------------------------------------------------------
//
//      Mission Name: pvidtst2  Creation Date: 9/6/2009
//
// ------------------------------------------------------------------------


[Output]

[GameScale]                     //For 90 Pixel mechs
f WorldUnitsPerMeter = 5.01
f MetersPerWorldUnit = 0.2
ul Duration = 60
f CycleLength = 0.125

[Artillery]
l NumLargeStrikes =  0
l NumSmallStrikes =  0
l NumSensorStrikes =  0
l NumCameraStrikes =  0

[ElementSystem]
ul ElementHeapSize = 358399
ul MaxElements = 2500
ul MaxGroups = 1000

[CraterSystem]
l NumCraters = 100
ul CraterShapeSize = 20479
st CraterFile = "feet"

[ContactManager]
l MaxContacts = 1600

[PotentialContactManager]
l MaxPotentialContacts = 500

[Smoke Manager]
l NumSmokeTypes = 2
l MaxSmokesPerType = 100
ul SmokeSphereHeapSize = 50000

[Smoke0]
l SmokeType = 11

[Smoke1]
l SmokeType = 12

[CameraSystem]
ul CameraHeapSize = 8191
st CameraFileName = "cameras"

[ObjectSystem]
ul ObjectHeapSize = 1535999
ul ObjectTypeHeapSize = 716799
ul NumObjects = 5000
st ObjectFileName = "object2"

[SpriteSystem]
ul SpriteHeapSize = 524287
st SpriteFileName = "sprites"

ul SpriteManagerHeapSize = 2097151
ul SpriteDataHeapSize = 65535
st ShapeFileName = "shapes"

[SpriteManager]
ul LegHeapSize = 1572863
ul TorsoHeapSize = 2097151
```

```
ul RightArmHeapSize = 786431
ul LeftArmHeapSize = 786431
ul TotalMechs = 19
ul Use90Pixel = 1

[TerrainSystem]
st TerrainFileName = "pvidtst2"
st TacMapGifName = "pvidtst2"

[PaletteSystem]
st PaletteSystem = "Palette"

[Music]
uc scenarioTuneNum = 0

[CollisionSystem]
ul XGridSize =24
ul YGridSize =24
ul GridRadius = 200
ul MaxObjects = 300
ul MaxCollisions = 100
ul MaxPending = 40
ul CollisionHeapSize = 16383
f WarningDist = 250.0                          //This is in world Units!!!
f AlertTime = 2.5                                      //This is in seconds
ul NumAlerts = 20

[SensorContactShape]
st shapeName = "blip"

[ABLibraries]
st Library0 = "orders"
st Library1 = "miscfunc"

[Script]
st ScenarioScript = "pvidtst2"

[StatusWindow]
ul PosX = 200
ul PosY = 200
ul Width = 300
ul Height = 100

[Campaign]
st BriefingFile = "pvidtst2.txt"
st MapFile = "pvidtst2"
l MaxTonnage = 990
l NumDropZones = 1

[DropZone0]
l NumSlots = 4

f PositionX = 1216
f PositionY = 64

f OffsetX0 = 0
f OffsetY0 = 0
f Rotation0 = 0

f OffsetX1 = 0
f OffsetY1 = -100
f Rotation1 = 0

f OffsetX2 = 100
f OffsetY2 = 0
f Rotation2 = 0

f OffsetX3 = 100
f OffsetY3 = -100
f Rotation3 = 0

[Objectives]
ul NumObjectives = 1                           //There can be a maximum of eight objectives
```

```
[Objective0]
st Name = "Destroy All Enemies"
ul Type = 0                                  //Types are 0=primary, 1=secondary, 2=other, -1=invisible
f TimeLeft = -1                              //Time to accomplish objective.  -1.0 means forever
ul Status = 0                                //Status are 0=incomplete, 1=success, 2=failed
l Points = 0

[Warriors]
ul NumWarriors = 2
uc CaptureChance = 0
st BrainParameterFile = "pvidtst2Params"
[Warrior1]
st Profile = "PMW00025"
st Brain = "DredAttack01"
[Warrior2]
st Profile = "PMW00068"
st Brain = "pbrain"

[Parts]
ul NumParts = 2
b AlliedTeam = False

[Part1]
ul ObjectNumber         = 17
ul ControlType          = 2                  // player = 1, ai = 2, net = 3
b PlayerPart            = True
ul ControlDataType      = 1                  // vehicle control data = 1
c MyIcon                = 0                  //
c TeamID                = 0    // 0 = IS, 1 = CLAN, 2 = ALLIED
c CommanderID           = 0    // 0 = IS, 1 = CLAN, 2 = ALLIED
st ObjectProfile        = "PM201300"         // Swiftwind
ul Pilot                = 2
f PositionX             = -2496              // Starting position in world.
f PositionY             =  448
f PositionZ             = -1.0               // Elevation automatically set by terrain.
f Rotation              =  45                // Rotation of Object in Degrees
ul Gesture              = 2                  // Initial Sprite Gesture -- Determines velocity in Mechs
                                             // -- non-mechs ignore
f Velocity              = 0.0                // Initial Velocity -- Start velocity for non-mech objects
                                             // -- mechs ignore
l Active                =  0
l Exists                =  1

[Part2]
ul ObjectNumber         = 426
ul ControlType          = 2                  // player = 1, ai = 2, net = 3
b PlayerPart            = False
ul ControlDataType      = 2                  // vehicle control data = 1
c MyIcon                = 0                  //
c TeamID                = 1    // 0 = IS, 1 = CLAN, 2 = ALLIED
c CommanderID           = 1    // 0 = IS, 1 = CLAN, 2 = ALLIED
st ObjectProfile        = "PV20000"         // Swiftwind
ul Pilot                = 1
f PositionX             = -2496              // Starting position in world.
f PositionY             =  448
f PositionZ             = -1.0               // Elevation automatically set by terrain.
f Rotation              =  45                // Rotation of Object in Degrees
ul Gesture              = 2
f Velocity              = 0.0
l Active                =  1
l Exists                =  1

[Teams]
b AlliedTeam = False

[Commander1Group:0]
l[12] Mates             = 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0

[Commander0Group:1]
l[12] Mates             = 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0

FITend
```

Listing of PvidTst2.abl

```
//*****************************************************************************

module MissionBrain_pvidtst2 : integer;

     //---------------------------------------------------------
     // Mission Name: pvidtst2
     // Created: 9/6/2009
     //---------------------------------------------------------

//-----------------------------------------------------------------
//
//     Constant Definitions
//
//-----------------------------------------------------------------
     const

          #include_ "misconst.abi"


//-----------------------------------------------------------------
//
//     Type Definitions
//
//-----------------------------------------------------------------
     type


//-----------------------------------------------------------------
//
//     Variable Declarations
//
//-----------------------------------------------------------------
      var

            #include_ "pvidtst2_VAR.ABI"

            static integer      ScenarioResult;
            eternal boolean     ExitTimerSet;
            static integer      VictoryLevel;
            char[40]            dstring;
            eternal real        gametime;
            static real         nextsecond;
            integer             x;
            integer             y;
            Position            aPoint;
            eternal boolean     PlayerForceDead;
            eternal boolean     ClanForceDead;
            eternal boolean     AlliedForceDead;
            eternal boolean     GeneralAlarm;
            eternal boolean     Flag1;
            eternal boolean     Flag2;
            eternal boolean     Flag3;
            eternal boolean     Flag4;
            eternal boolean     Flag5;
            eternal boolean     Flag6;
            eternal boolean     Flag7;
            eternal boolean     Flag8;
            eternal boolean     Flag9;
            eternal boolean     Flag10;
            static boolean      PlayPASound;
            static boolean      PlayGASound;
            eternal integer     GeneralAlarmCounter;
            boolean             PerimeterBreach;
            static boolean      StartedRadioMessage;
            static real[3]      Pos;

//-----------------------------------------------------------------
//
//     Init Function    (automatically run first time through)
//
//-----------------------------------------------------------------
```

```
function init;

       code

            ScenarioResult = PLAYING;
            PlayerForceDead = FALSE;
            ClanForceDead = FALSE;
            AlliedForceDead = FALSE;
            ExitTimerSet = FALSE;
            VictoryLevel = 0; // New Scheme
            NextSecond = 1.0;
            GeneralAlarmCounter = -1;
            GeneralAlarm = FALSE;
            Flag1 = FALSE;
            Flag2 = FALSE;
            Flag3 = FALSE;
            Flag4 = FALSE;
            Flag5 = FALSE;
            Flag6 = FALSE;
            Flag7 = FALSE;
            Flag8 = FALSE;
            Flag9 = FALSE;
            Flag10 = FALSE;
            PlayPASound = FALSE;
            PlayGASound = FALSE;

            #include_ "pvidtst2_INIT.ABI"

            StartedRadioMessage = false;
            Pos[0] = 570.0;
            Pos[1] = 70.0;
            Pos[2] = 0.0;
endfunction;

//-----------------------------------------------------------------
//
//     Main Code
//
//-----------------------------------------------------------------
      code
            if (not StartedRadioMessage) then
                if (InArea(GetVehicleID(PLAYER_FORCE,0,0), Pos, 30.0, -1)) then
                    StartedRadioMessage = TRUE;
                    SetRadio(GetVehicleID(PLAYER_FORCE,1,0),TRUE);
                    PlaySpeech(GetPilotID(GetVehicleID(PLAYER_FORCE,1,0)),24);
                    AddStrikes(0,1,1);
                endif;
            endif;

            //-------------------------------------------------
            // Debug Window Game Clock Second Counter
            // Note:  This is used by some included functions.
            //-------------------------------------------------
            gametime = gettime;
            If (gametime >= nextsecond) Then
                  nextsecond = gametime + 1;
                  If (GeneralAlarm) then
                       GeneralAlarmCounter = GeneralAlarmCounter + 1;
                  endif;
                  // dstring = "Gametime: ";
                  // concat(dstring,gametime);
                  // Print (dstring);
            endif;
            if ((PlayGASound) and (NextSecond == gametime + 1)) then
                  playSoundEffect(GENERAL_ALARM_SOUND);
            endif;
            if (PlayPASound) then
                  playSoundEffect(PERIMETER_ALARM_SOUND);
            endif;
            PerimeterBreach = FALSE;

            //---------------------------------------
            // Structure
            //---------------------------------------
```

```
#include_ "pvidtst2_STR.ABI"

//----------------------------------------
// Lookout Points
//----------------------------------------
#include_ "pvidtst2_LOP.ABI"

//----------------------------------------
// Force Dead Checks
//----------------------------------------
if (isDeadorFled(PLAYER_FORCE)) then
      PlayerForceDead = TRUE;
endif;
if (isDeadorFled(CLAN_FORCE)) then
      ClanForceDead = TRUE;
endif;
if (isDeadorFled(ALLIED_FORCE)) then
      AlliedForceDead = TRUE;
endif;

//----------------------------------------
// SET EXIT TIMER IF PLAYER DEAD/DISABLED
//----------------------------------------

if ((NOT ExitTimerSet) AND (PlayerForceDead)) then
      // Fail any Undecided Player Objectives
      if (checkObjectiveStatus(0) == INCOMPLETE) then
            setObjectiveStatus(0, FAILED);
      endif;
endif;

//----------------------------------------
// Objective Resolution
//----------------------------------------

if (checkObjectiveStatus(0) == INCOMPLETE) then

     if (isDead(CLAN_FORCE)) then
           SetObjectiveStatus(0,SUCCESS);
           playBetty(BETTY_OBJECTIVE_COMPLETE);
           VictoryLevel = VictoryLevel + 1;
           Objective_0_Decided = TRUE;
           if (VictoryLevel < 1) then
                 PlayDigitalMusic(OBJECTIVE_SUCCESS_MUSIC);
           endif;
     else
           if (gettimeleft == 0.0) then
                 SetObjectiveStatus(0,FAILED);
                 playBetty(BETTY_CANNOT_COMP_OBJ);
                 Objective_0_Decided = TRUE;
           endif;
     endif;

else
      if (NOT ExitTimerSet) then
            //if Failed primary Objective, fail mission
            if (checkObjectiveStatus(0) == FAILED) then
                  setTimer(EXIT_TIMER,2.0);
                  ExitTimerSet = TRUE;
            endif;
      endif;
endif;

if ((NOT ExitTimerSet) AND (Objective_0_Decided)) then
      setTimer(EXIT_TIMER,2.0);
      ExitTimerSet = TRUE;
endif;

//----------------------------------------
// END SCENARIO
//----------------------------------------
```

```
            if ((ExitTimerSet) AND (CheckTimer(EXIT_TIMER) == 0.0)) then
                  if (VictoryLevel >= 1) then
                        ScenarioResult = PLAYER_WIN_BIG;
                        PlayDigitalMusic(VICTORY_MUSIC);
                  else
                        ScenarioResult = PLAYER_LOST_BIG;
                        PlayDigitalMusic(DEFEAT_MUSIC);
                  endif;
            endif;

            //---------------------------------------
            // RETURN RESULT
            //---------------------------------------

            return (ScenarioResult);
endmodule.
//*****************************************************************
```