MechCommander Gold Campaign Builder's Guide

Part 3 – Advanced Features

Revision 0.2

By Cmunsta

Revision History

Rev 0.2

- Added section on giving artillery.
- Added section on pre-damaging buildings.
- Added section on retrieving the ID objects such as bridges using only the map coordinates.
- Added final words section.
- Fixed some typos.

Rev 0.1

- All topics added to the document.
- Some proofreading.

Rev Pre-Release 0.1

- **Very** early edition of the document.

Introduction

Welcome to part 3 of the MechCommander Gold Campaign Builder's Guide. In this guide we will be covering more advanced features such as timed objectives, handing out salvage, capturing vehicles and mechs and much, much more.

The reason these features could not be in Part 1 is that most require ABL script support in some way. This is also why part 2 was strictly about the ABL scripting language itself. With the language now introduced, and the basics of how a campaign is put together, we can concentrate on features that can truly start giving our campaigns that unique feel.

As with part 1 of this series, I'll be adding full listings (where useful) of the files I edited for the examples. However, unlike in part 1, they will most likely not be able to be used directly for any of your campaigns without some form of editing. This is because some of the information needed will rely heavily on where certain objects are placed in the editor. Therefore, it is important to learn what needs to be changed, and where to get the data that is needed from the editor.

In order to isolate each topic, I'll be editing some simple solo missions, but as we've seen in part 1, solo maps are the basis for building campaigns anyway, so you should have no trouble including these into your own campaigns.

Well enough chatter, time to get down to business.

-Cmunsta

Before Starting

I strongly suggest that you review parts 1 and 2 of the builder's guide. Part 1 is of course the basis for building campaigns and so I'll be assuming that you understand what has been discussed there. So if I say a given file is in a certain .FST file, I'll assume that you already know how to extract it.

Part 2 is another story. This is a reference guide to the ABL scripting language. You should at least try to understand the information presented there. Though of course, this can be a little overwhelming if you have never done any programming before. So if not everything in part 2 is clear, don't worry too much about it. But as you read and learn from this document, go back and review some of the things in part 2 as well. This may help make things a little clearer.

Further, as part 2 is a reference guide, if I use a function in a script, try looking that function up in there. This will give you a little extra insight as to both how the language works, and the function itself.

Tools of the Trade

As before, we will be using those command line tools I created and introduced in part 1. The links to download these tools are given in part 1 so I won't bother giving them again here. However, you will only really need them to extract a few files, and that typically will be only for review.

Apart from the tools, we will be using a text editor. I suggest getting a good programming editor (I use Notepad++) but any text editor should do.

That is pretty much all you will need for this guide. A text editor and my tools. Well alright, you will also be using the MechCommander Editor, but I kind of assumed that you would have that tool given the nature of these guides.

Mission: Logic

When dealing with ABL scripts, we need to know where the script is going to be used and for what purpose.

There are effectively three types of ABL scripts that we need to be concerned about. Mission scripts, warrior brains, and libraries. Libraries are the easiest to deal with as they are more or less just a collection of functions that can be used by the other two types of scripts. I've already covered libraries in part 2 so we won't be looking at them much in this guide (though we will be using some). So that leaves mission scripts and warrior brains.

Warrior brain scripts are .ABL files that define the behavior for a given MechWarrior or vehicle pilot. We will not be covering these in any detail in this guide. So as you've likely guessed from the name of this section, we are going to start by looking at mission scripts.

Mission scripts are what controls the logic of what happens in a mission. We've briefly looked at this in part 2, but now we will be looking at them in more detail. But to re-iterate briefly what was in part 2, the mission script has the power of decision over how the mission and/or objectives are completed. To the point that if the mission script says the mission is not over, the game will simply keep going no matter what the player does short of aborting the mission or exiting the game.

So lets start off by creating a very simple solo mission and taking a closer look at the mission script generated by the editor. Yes I know we did this already in part 2, but there I was describing the structure of the file and how it all hangs together. Here I'll be describing the code content of the file. This is one reason why it may be necessary to go back and worth between part 2 and this guide.

For this sample mission, simply make any type of map, put a single unit on the map (as usual, I put a Swiftwind), place a drop-zone somewhere on the map, and save it (the default objective is to “Destroy All Enemies” so that will be the single objective). I called mine “ex1” for example 1.

Once that's done, open `ex1.abl` from the `{MCX}\data\missions` folder into a text editor.

As explained in part 2, you will see that this file is defining a module called MissionBrain_ex1. This is the module that the game will use to run the mission logic for this mission.

Scroll down the file until line 105. This is the main code for this module, and therefore for this mission's logic (everything else is merely support for this section of code).

Now the first chunk of code looks like this:

```
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;
```

The first part of this code gets the current mission time (it calls the internal function GetTime for that), and then checks if it needs to increment a variable called nextsecond. This should clue us to the fact that nextsecond is a variable that is being incremented every second; a useful thing to know.

Inside this if statement, the code also checks to see if the boolean variable GeneralAlarm is true, and if so increments another variable called GeneralAlarmCounter. Therefore, GeneralAlarmCounter will also increment every second, but only if the GeneralAlarm flag is set to true.

Following this, we see three lines of code that have been commented out. These were obviously for debugging purposes, but as we don't have proper access to the ABL debugger (and sadly never will), these lines serve no purpose to us, and as they are commented out, they serve no purpose in the code either. We could delete them if we wanted.

Next we see two if statements, both check to see if they should play a certain sound effect. If so, they play them.

Finally, the code sets a variable called PerimeterBreach to false.

So that pretty much does it for that chunk of code. It is keeping track of time passage, and checking if it needs to play a couple of sound effects for the general and perimeter alarms.

The next few parts of code are interesting (I won't bother showing the code comments here). They are:

```
#include_ "ex1_STR.ABI"
```
and
```
#include_ "ex1_LOP.ABI"
```

These are both include files, which means that what actually goes here is defined in another file. Specifically, `ex1_str.abi` and `ex1_lop.abi`. Try loading those files up in the editor (they too can be found in the `{MCX}\data\missions` folder.

Not much to those files right? These files are created by the editor for special handling of the structures (buildings) and lookout points respectively. It is easier for the editor to deal with the code in this way, and we could also edit or add code in this way if we wanted. Basically anything that goes in these files will be inserted where the #include_ line is in the mission code as if we did a cut and paste job of those files into the code. Of course, the mission we just made is so simplistic, it doesn't *have* buildings or lookout points to deal with, so these files are empty.

The next chunk of code in the mission .ABL file is:

```
if (isDeadorFled(PLAYER_FORCE)) then
    PlayerForceDead = TRUE;
endif;
if (isDeadorFled(CLAN_FORCE)) then
    ClanForceDead = TRUE;
endif;
if (isDeadorFled(ALLIED_FORCE)) then
    AlliedForceDead = TRUE;
endif;
```

This code is doing three checks. Basically looking to see if all units of a given force (or side) are dead or have left the map. If they are, it sets a boolean variable to true to indicate this fact. It does the check by calling a function called `IsDeadOrFled`.

However, this function is not listed in the built-in function list in part 2 of the guide. This is because it isn't a built-in function. It is actually defined in the `miscfunc.abx` library (told you libraries would come into play). Go extract `miscfunc.abx` from `mission.fst` and open that file in a text editor. Do a search for “IsDeadOrFled” and you will find the function itself.

This is something to remember. If you can't find the function listed in part 2 of the guide, then check the libraries (and of course any function defined in the module itself).

The editor always sets things up to load two libraries by default. Namely `miscfunc.abx` and `orders.abx`. But since we can make our own libraries as well, it is best to check with the mission's .FIT file to see what libraries are actually being loaded.

Open `ex1.fit` into the text editor and look for the heading `[ABLibraries]`. This is where the libraries to be loaded are listed. By default, the editor puts this in this section:

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

As can be seen, these are the `orders.abx` and `miscfunc.abx` libraries I mentioned (the game assumes that the libraries have the `.abx` extension)

OK, back to our examination of the code in the `ex1.abl`.

The next section of code is:

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

This first checks that the exit timer has not been set by examining the variable ExitTimerSet and also checks if all player units are dead by examining the PlayerForceDead boolean variable (PlayerForceDead gets set to true in the section of code we just examined previously). If both conditions are true, then it checks if the first objective (the only objective in our case) is incomplete. If it is, then it forces the objective to be failed.

Therefore, this is making sure that the objective (or objectives if we had more than one) has failed if all the player mechs and vehicles have died. If we went and changed the value FAILED to the value SUCCESS in this if statement, we would actually make it so the player would “complete” the mission by killing off his units.

The if statement uses the built-in functions CheckObjectiveStatus and SetObjectiveStatus. If you look those functions up, you will note that it doesn't mention about a value for INCOMPLETE, FAILED or SUCCESS. This is because these are actually constants. These constants are being defined in `misconst.abi` which you will notice is included at the very top of this .ABL file. Open `misconst.abi` into the text editor and take a look. There are all kinds of constants that we can use. We could of course simply use the numbers themselves, but using the names makes it a lot clearer as to what is going on.

We could also define our own constants in the constant block for this module, or even make our own include file and include it the same way `misconst.abi` is.

The next part of the code is:

```
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  // supposedly...
                 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;
```

The large if statement first looks at the state of objective 0 (the first objective) and if it is not yet complete, checks to see if all enemies are dead (uses the IsDead function from `miscfunc.abx`) and if they are, sets the objective as completed (remember, this is a “Destroy All Enemies” objective), plays Betty's audio clip saying the objective is complete, increments a variable and starts playing some music.

If the enemies aren't dead, then it checks if the amount of time left for the mission has reached 0 (this uses the GetTimeLeft built-in function which will return -1 if there isn't a time limit for the mission). If the time has reached 0, then it sets the objective to FAILED and plays Betty's “Objective cannot be completed” quip.

If the objective was in a state other than INCOMPLETE, then it falls to the “else” part of the if statement and does the code in there instead. This part simple checks to see if it should exit the mission if the objective failed.

After all this, there is another if statement that checks if the exit timer needs to be set and if so, starts the exit timer counting down (2 seconds duration).

The next section of code checks if the exit timer has finished counting down its 2 seconds and if so plays the victory or defeat music based on the results of the checks above. It also sets the variable ScenarioResult to something other than PLAYING (which was set in the init function. As explained in part 2 of the builder's guide, the init function is a special function that gets called automatically by the game only once when the module is loaded. Specifically to initialize variables and such.

Finally, the last section of code is quite simply:

```
return (ScenarioResult);
```

This simply returns the result determined previously back to the game.

The game constantly runs the code in this module to check the state of the game. It will end the mission only when this code says it should (or the player aborts the mission via the escape key).

So as seen, this is the default code generated by the mission editor. The code will be slightly different if we have different objective types, or had more than one objective, but the basics would remain the same.

So now it is time to start learning how to add our own code to this file.

As mentioned in the introduction, I'll be creating a solo mission for some of the topics in order to isolate each one and not get bogged down by having code from previous topics cluttering up the files.

Ready? Lets get this show on the road.

Hiding and Revealing Objectives

Lets start things off with something rather simple. Hiding and revealing mission objectives. By this I mean that the objective doesn't show up in the TAC map until we want it to.

Why would we want this? Because we may want a “new” objective to show up after something else has happened in the game. Because we can. Because it is another way to force certain objectives to follow others. Because... well you get the idea.

Now you are probably saying: “Ha! That's way too easy. It already says how to do that in the mission's .FIT file. Just set the objective type to -1”.

Well give it a try then, I'll wait... ...Didn't work eh (oh come on, I'm Canadian, I had to work at least one “eh” in here somewhere).

Turns out that if you attempt to set the objective type to -1, the objective still shows up in the TAC map but will have a description of “invisible” like this:

Not very “invisible” now is it.

To make an objective simply not show up in the TAC map, we need to set it's status to an invalid value. Since normally the status of an objective is 0 (incomplete), 1 (success) or 2 (failed), a value of 3 will do the trick.

Now we could do this either in the .FIT file, or in the code for a mission. As this document is more about scripting things, we'll handle everything through code.

OK, so that handles hiding it. What about getting it back again? Simple. Just set the status to one of the “valid” states again (0, 1 or 2) and it'll magically show up again.

If we want to have hidden objectives, we are going to need a way to reveal them at the appropriate time. Otherwise, how is the player supposed to know there is an objective to complete. If all we ever did with it is keep it hidden, then we wouldn't actually need an “objective” to begin with. As we've seen, the code is what decides anyway, the objectives are just there to tell the player what to do.

Time to make a mission for this example. Open the editor and put a drop-zone on the map. Put a Switfwind somewhere close by, and also put down a building to be captured.

Make two objectives for this mission. The first will be the typical “Destroy all enemies” type, and make the second objective to capture the building (make sure both are primary objectives so the mission doesn't end early on us).

Save the mission as ex2 (for example 2), and open the `ex2.abl` file in a text editor.

Lets make it so that when the player completes the first objective, the second will show up. But not before then.

So the first thing we want to do is hide that second objective. Recall that the init function gets called automatically only once when the module is loaded. Perfect for our needs since we want to hide the objective at the start of the mission.

Scroll down to line 97 (this is the blank line immediately after the `#include_ "ex2_init.abi"` line). This is where we will add our code to hide the objective. But what to add?

Take a look through the functions in part 2 of the guide. Though because there are a lot of functions, we might want to do a search for something. We know we want to do something with an objective status, how about searching for the word “Objective”. Okay, there are a lot of search results, but if we scan these results we find that there is one result that looks promising. Part 2 has a description for a function called SetObjectiveStatus. Looking it up we find that this looks exactly like what we need. We also see that it takes two parameters. An integer saying which objective we want to change, and an integer for what status value we want to set.

Lets try it. Add this on line 97.

```
SetObjectiveStatus(1, 3);
```

We use 1 as the objective number here because the objectives start counting at 0 not 1. So objective 1 here is in fact the second objective. And of course, 3 is the status we want to set to “hide” the objective.

OK, that wasn't so bad. Now how about revealing it again.

Scroll down to line 173 or so. Doesn't this look familiar now? It is where the code checks to see if the first objective (objective 0) is done or not. As we want to reveal the second objective when the first is completed, this looks like a good place to add the code.

So right before the line that says:

```
SetObjectiveStatus(0,SUCCESS);
```

Add the line:

```
SetObjectiveStatus(1,INCOMPLETE);
```

So now it looks like this:

```
SetObjectiveStatus(1,INCOMPLETE);
SetObjectiveStatus(0,SUCCESS);
```

Save the file and try the mission out. Switch to the briefing tab in game and you'll see that only the first objective shows up. Go kill the Swiftwind and all of a sudden the second objective will show up. Hooray!

The briefings tab at the start of the mission:

And after killing the Swiftwind:

One thing you may notice is if you capture the building *before* you kill the Swiftwind, the building will be captured, but you won't get credit for it. Oops!

This brings up a few things that must be learned. When scripting, you will have to start thinking of all the possible outcomes that something can do. You probably won't think of them all, and someone will either find a way to exploit it to their advantage, or most likely it will simply bomb or otherwise make the mission unplayable.

This is true even of the original campaigns. While researching this document, I often went through the original and expansion campaigns. Once, while playing the original campaign mission where you first encounter Falcon and have to escort her to the extraction zone (she starts on the other side of a river). I was rushing to try to get Falcon across the first bridge to join up with the others as I normally do. Unfortunately I messed up and she was on the bridge when it blew. My bad, but hey. Now she didn't actually get hit by the air strikes so no problem there, but the bridge had blown both in front and behind her. Since her mech doesn't come with jump jets, the mission suddenly could not be finished. I doubt very much that having Falcon stuck on a bridge like this was a possibility even remotely foreseen by the developers.

But enough of that. We should see if we can fix *this* problem.

Lets review the bug. If the player captures the building before completing the first objective, then though the building will be captured, the player won't get credit for it and so will not be able to finish the mission.

Well there are a number of ways we could fix this, but since the methods to do many of them will be described elsewhere in this document, we will restrict ourselves to simply revealing the hidden objective and giving credit to the player for it.

Now looking at the code as it currently is, we see around line 200 an if statement that is checking if objective 1 (the second objective) is complete, and if not, checks the captured status of the building and if it is, sets the objective as complete. Looks fine, but obviously this code is never running because the player never gets the credit.

Spot it yet? If you said it's the line:

```
if (checkObjectiveStatus(1) == INCOMPLETE) then
```

Give yourself a pat on the back. But do you see why?

We changed the status of the objective in the init function to 3, so it is not INCOMPLETE (0) anymore. Hence the if statement will always fall into the else part.

But we can't just make it look for 3 either since when the first objective is completed, we *then* set the status to INCOMPLETE.

This is where the logic operators come in handy. We can check for both conditions. We'll want the if part of this statement to execute if the status is 3 or when it is INCOMPLETE, so we'll need an OR statement in there.

Change the line to this:

```
if ((checkObjectiveStatus(1) == INCOMPLETE) or
           (checkObjectiveStatus(1) == 3)) then
```

I had to break the line into two for this document, but you could actually use this as two lines in the code too. The number of spaces and carriage returns between keywords isn't important in ABL scripts (the parser just throws them away).

So the OR statement in here will make the “if” true if either the first check or the second check is true. If they are both true it would also work, though in this case that is a logical impossibility since the status of the objective can't possibly be two different values.

Note the use of extra parentheses to force the precedence.

So this would seem to be the solution right? Not so fast.

Remember that when the first objective is completed, we made it change the status of this objective to INCOMPLETE. If we leave it as is, it is going to reset the status of objective 1 even if we originally completed it. What we really need to do is set it to incomplete if the status is still hidden.

So go back to the line that says:

```
SetObjectiveStatus(1,INCOMPLETE);
```

(It's around line 173 or so)

We need to put this line in an if statement of its own that checks the status of objective 1. Since the code around here is peppered with these types of checks, it shouldn't be too hard to figure out what to do.

Change that one line to become these three lines:

```
if (CheckObjectiveStatus(1) == 3) then
    SetObjectiveStatus(1,INCOMPLETE);
endif;
```

Now we should be done. Try this new version.

I know this section has been going on for quite a while for what is effectively a very simple thing, but I wanted to show an example of some of the things you will need to start thinking about while making up your scripts and also debugging those scripts.

How to debug is an important thing to learn, so make the effort. Unfortunately, I can only scratch the surface here or I'd need a whole tutorial just for that.

Having said this, the rest of this document will not delve into so much detail but will instead concentrate on getting the information across. This means that you may need to think about how to implement all the details yourself.

Further, what I'll show in the following sections may not completely be what you need, but something similar. Therefore, treat this as the basics of how to do these features and modify them to your needs.

Timed Missions and Objectives

In this section I'll be talking about all the mission and objective timers that can be used for your campaigns. Mission and objective timers are different beasts and so need to be handled differently. We'll start with the easy one.

A timed mission is one where the mission as a whole is on a timer. You must complete all primary objectives before the time runs out or you fail the mission. You've likely seen these in the official campaigns. They are the ones where you see the time remaining while viewing the map panel on the TAC map.

Unfortunately, there doesn't appear to be a way to set this timer programmatically (that is via a script function), so we must set the mission timer within the mission's .FIT file.

So once again, lets make a simple example mission and set the mission time for it.

Open the editor and make a simple mission with a single objective (what it is doesn't really matter) and save it as “ex3” (example 3).

Now in a text editor, open the mission's .FIT file (`ex3.fit`) and search for the heading `[Objectives]`. To this heading section, add a TimeLeft entry which will represent the duration of the mission, in seconds. So for a 4 minute mission, we would need to set this value to 240. My objectives section looks like this:

```
[Objectives]
ul NumObjectives = 1
l TimeLeft = 240
```

Save the .FIT file and open the .ABL file (`ex3.abl`). As this is just an example mission we don't actually need to change anything here as the code created by the editor already checks for timed missions. But I want to point out how the code is checking for the time remaining so that you can add or change what you need.

Scroll down to line 178 or so. This should be inside the else part of the if statement that is looking to see if the player has completed the objective. You will notice that the if inside this else is checking GameTimeLeft being equal to zero. In fact, GameTimeLeft isn't a variable but a Built-in function that returns the timer value remaining for the mission timer.

This is the type of thing we need in the code, a way to make the mission objectives fail if the time runs out. If we don't, the timer will count down to zero and nothing else will happen.

As I said, we don't actually need to change anything in this code, I just wanted to point out where it was checking the timer.

Try the mission out.

Some things to note. When the mission time hits 2 minutes remaining (or immediately entering the mission if the timer is 2 minutes or less), the game will play Betty's audio clip saying that the “mission time will expire in 2 minutes”. When it hits 30 seconds, she will say her “mission critical: mission time will expire in 30 seconds” line. This is done internally by the game, not in a script. So there is no way to turn this off.

Because she also says these lines if the mission timer is set to 2 minutes or less, missions should not have timers set to less than 2 minutes or the player may get confused. It would be annoying to have Betty say you got 2 minutes when you only actually have a minute and a half on entering the mission.

As the only change was one line in the .FIT file, I won't be listing the file in the appendices.

So that is how we create a timed mission, but what about individually timed objectives.

Timed objectives can be set both in the .FIT file and from a script. The editor however doesn't add appropriate code for this so we will have to edit the .ABL file as well.

Lets make yet another simple mission called “ex4” for example 4. Create a new mission and put a couple of buildings down. Make them different buildings so that we know which one will be which. Now make two mission objective. The first, will be to capture the first building. Make this a primary objective. For the second, make it a secondary objective to capture the other building (doesn't really have to be secondary, but we'll use that first objective as a way of keeping the mission from ending as we test the timer on the secondary objective). Put a Swiftwind on the map somewhere so that the editor is happy and save the mission.

Now open "ex4.fit" in a text editor and look for the section heading [Objective1]. Change the TimeLeft entry to 60 (which is one minute, since this entry is also in seconds). And save the file.

Open "ex4.abl" into the text editor. As mentioned, the editor doesn't add code to handle timed objectives, so we need to modify a couple of things. Scroll down to line 211. It looks like this:

```
if ((gettimeleft == 0.0) OR (getObjectDamage(x) > 99)) then
```

This is in the else part of the if statement that is checking if objective 1 (the secondary objective) is complete or not. We need to make sure that the objective will fail when the timer runs out.

Recall that GetTimeLeft used in the above line is a function that returns the mission time. As this isn't a timed mission, we don't really need this check anymore (we would if we were planning to make this a timed mission of course). However, we do want to check the time remaining, but only for this objective. So we should be able to replace the GetTimeLeft function with something else.

Looking through the built-in functions, we find a function called CheckObjectiveTimer. Bingo. That is what we need. This function has one parameter which is the objective number to check. So lets modify line 211 to look like this:

```
if ((CheckObjectiveTimer(1) == 0.0)
        OR (getObjectDamage(x) > 99)) then
```

(again, I had to break the line up to display it in this document)

So now, if the objective timer has reached 0, it will mark the objective as failed.

Save the .ABL file and try it out.

The objective clock is ticking:

After letting the clock run out gives:

Capturing the building before time runs out gives:

Note how the timer is still running on this objective even though we completed it. This is because we haven't actually stopped the clock. The game happily keeps ticking time off the clock.

Now if we look through the built-in functions, we can't find a function to stop the objective timer. What to do. The above images give a hint.

The first idea would be to try to set the timer back to -1 so that it is infinite again. This actually doesn't work, though you are welcome to give it a try when you see how to set the timer.

Look at the objective failed image. There is no clock display on it even though the objective failed because we let time run out. Hmmm. It looks like we could set the clock to 0 and the clock will go away. Lets try that.

At line 203 (just above where we were editing) is the line:

```
SetObjectiveStatus(1,SUCCESS);
```

This is the line that sets the objective to SUCCESS in the if statement that checks this secondary objective. So just before this we could add a line to set the objective clock.

Looking through the ABL reference once again, we come up with the function SetObjectiveTimer. Perfect.

Add this new line immediately before line 203 so that they look like this:

```
SetObjectiveTimer(1, 0.0);
SetObjectiveStatus(1,SUCCESS);
```

Save this new version and try it. Now the clock will go away when the secondary objective is completed.

This also shows how we can set the objective timer in a script. By calling the SetObjectiveTimer function. Lets try making the first objective also on a timer, but we'll do it from within the ABL script. We need to decide where to do this though.

We could of course do the obvious. Set the timer in the init function and so the timer will start when the mission does, but that isn't very fun. Instead, let's make it so that the timer will start if the player captures that secondary objective. This way, the player can grab the primary objective at his leisure, but if he grabs the secondary objective first, he suddenly finds he's got 45 seconds to capture the other one (lets pretend capturing that secondary objective causes explosives to start ticking on the primary objective or something like that).

Now the first change we'll need to do to our `ex4.abl` file is to start the timer on the other objective. But we should check to make sure that objective hasn't already been completed. This check is actually a little redundant here because objective 0 is of course the only primary objective, but lets play it safe anyway; we may want to add new objectives later.

So we go back to line 203, this is where we added the timer reset. Modify this section so that it checks if objective 0 is completed. If not, then it starts the timer on objective 0. This part of the code will now look like this:

```
if ((IsCaptured(x) == 1) AND
                ( objectSide(x) ==  INNER_SPHERE))  then
      if(CheckObjectiveStatus(0) == INCOMPLETE) then
          SetObjectiveTimer(0, 45.0);
      endif;
      SetObjectiveTimer(1, 0.0);
      SetObjectiveStatus(1,SUCCESS);
```

This checks to see if objective 0 has been completed, and if not starts the timer on objective 0 with a 45 second countdown.

That takes care of starting the timer, what about marking objective 0 as failed when this timer runs out.

First, much like we did with objective 1, we are going to need to check the timer. However, there is a potential problem here. If we simply put the check in for objective 0's timer, the objective will fail immediately on mission start because the timer normally starts with a value of -1, meaning infinite time. When CheckObjectiveTimer sees that value, it always returns 0 as the result for the timer. So we are also going to have to check to see if we actually started this objective timer. For this, we will need to flag this fact with a boolean variable.

So first, lets define this boolean variable. Scroll up to line 64. This is the blank line at the end of all the other variables already being declared. We'll add our boolean here. But as we want this variable to keep its value every time the game runs this module during the mission, we also need to make this variable static. If we don't the value of the variable will always get reset to 0 (or false in the case of boolean variables) each time the game runs this module.

Add this as the new line 64:

```
static boolean StartedObjective0Timer;
```

Now because this is a static variable, we will also need to initialize its starting value (we would also need to do this for eternal variables). So in the init function, initialize the value to false. I added this as the new line 97:

```
StartedObjective0Timer = false;
```

Excellent. Now we have our flag that will tell us if we've started the timer or not. Now to set this variable to true when we actually do start the timer, go back to where we started objective 0's timer (line 204 or so) and modify the code so that we also set our flag to true when we start the timer. I changed this part to look like this:

```
if(CheckObjectiveStatus(0) == INCOMPLETE) then
    StartedObjective0Timer = true;
    SetObjectiveTimer(0, 45.0);
endif;
```

Now all we need is to make sure we check both the timer and this flag for objective 0.

Scroll up to line 182 or thereabouts. This is where the code is checking if the mission time has expired for objective 0. The line looks like this (note how similar it is to the original line that did the same job for objective 1):

```
if ((gettimeleft == 0.0) OR (getObjectDamage(x) > 99)) then
```

Once again, as this isn't a timed mission, we don't really need the GetTimeLeft check. But we need to replace that part with a check for both the objective 0 timer reaching 0 and checking that our flag is set to true. If either of these conditions are false then we don't want to fail this mission.

By the way, the “GetObjectDamage” part of this if statement is checking that the building hasn't been destroyed. We'll want to keep that part so that if the player is dumb enough to blow up the building he should be trying to capture, he will fail the objective.

So what we need to do is replace the (GetTimeLeft == 0.0) part of this if statement with the two checks we want.

I should point out something about boolean values here. The if statement actually expects a single boolean value in the condition. The logic operators also expect boolean values, but do their operation and return a single value. So in the if statement shown, it will get a true or false boolean value for the (GetTimeLeft == 0.0) part, get another boolean true or false value for the (GetObjectDamage(x) > 99) part, perform the OR logical operator on these two results giving a final boolean true or false value. This last is what the if statement will use as its condition value.

Well we can use something like this as well, we want our two checks to boil down to a single boolean true or false value so that it can be used with the OR logical operation with the result of the (GetObjectDamage(x) > 99) part. Since we want our two checks to come out true if and only if both conditions are met, we will need to use the AND logical operator.

If you actually figured out what I've been trying to say here, you'll see that the if statement will need to be modified like this:

```
if ((StartedObjective0Timer and (CheckObjectiveTimer(0) == 0.0))
        OR (getObjectDamage(x) > 99)) then
```

The StartedObjective0Timer variable here is already a boolean value, so we don't need to compare it to anything as it is already giving a true or false value. The (CheckObjectiveTimer(0) == 0.0) part will give a value of true if the objective 0 timer has a value of 0. These two values are AND'ed together which will give a result of true if and only if both parts were also true. Finally, this result is OR'd with the check to see if the building has been destroyed.

To put it in words, the condition can be read as: if the objective 0 timer was started and the objective 0 timer has reached 0, or the building was destroyed, then do the code inside this if.

This also demonstrates something I believe you should try to practice. Try to actually “read” the code. I don't mean just say the keywords written, but read the meaning behind these keywords. This will greatly help you understand the code that is written, as well as help you write your own.

Well that was a long digression for what boils down to a simple check after all. Save the .ABL and try this new version out.

After capturing the second building:

And letting the timer run out gives:

Though objective 0's timer does indeed get started when the other building is captured, there are still two problems with this.

The first is rather easy as we've already dealt with the same thing. If we capture the primary building when the timer is ticking, the timer will still be there. Not much of a problem for this example as the mission will end anyway, but we should set this timer to 0 when we mark the objective as complete. We should also do it for when it fails since is is technically possible that the player will capture the second building, then go destroy the first one. So we want to get rid of the timer in that case as well. I'll let you figure out how to do that on your own (check the listing in the appendices to check your results).

The second is actually quite the problem. Unlike the first which is a cosmetic thing as it doesn't really matter if the timer keeps ticking. If we let some time pass before capturing that second building, the timer for the first objective doesn't actually end up starting at 45 seconds, but starts with even less time.

What is actually happening is that the game always keeps track of how much time has gone by in the mission, and calculates all its timers based on this. This means that when we set the time for an objective, it is based on the start of the mission, not when we set the timer itself. So to make it always count 45 seconds, we also need to add the current game time. Thankfully, we've already seen that the code keeps track of this for us in the variable GameTime. So all we need to do is add this variable to the timer as well.

Go back to where we set objective 0's timer to 45 seconds (around line 205 or so) and change the line to look like this:

```
SetObjectiveTimer(0, (GameTime + 45.0));
```

Save the file and try it again. Let some time count down and then grab the second objective. That did the trick.

Now how about we fancy this up a little more. One thing you may have noticed is that Betty doesn't say her warnings for timed objectives. Typically this is a good thing, but since we only have one primary objective and we do start a timer on it, it would be nice to have Betty give her 30 second warning at the appropriate time. This is a rather easy thing to add to our code.

Around line 126 or so (right after that first if statement that is tracking time, we can add our own if statement that will check the objective 0 timer (if it is started) and check to see if it has hit 30 seconds yet. If so, we'll make Betty play her 30 second warning audio cue.

Getting Betty to play an audio cue is easy. The code already has some examples of that so it should be obvious that we need the PlayBetty function. Looking this function up in the reference, we find that there is a table that defines all the audio cues we can use, and which number they happen to be. We see that audio clip number 7 is her “Mission Critical: Mission time will expire in 30 seconds” line.

Back in the first section of this document I mentioned about a bunch of constants being defined in `misconst.abi`, and that this file was included in the code by the editor. Well if we take a look through this file, we see that there is a constant already defined called BETTY_THIRTY_LEFT and it is given the value 7. Not too hard to see that this is exactly the one we want. So instead of using the value 7, we could also use this constant. It'll help keep things clear in the code.

So all we need to do now is add an if statement that checks to see if the timer has started, and checks if the timer is at 30 seconds.

Well yes and no. Though the above is technically correct, there is a little something that you should be aware of. Note that the time value returned by the CheckObjectiveTimer function is a real value. There is a known oddity with all real or floating point values and computers. Sometimes, the value used inside the computer will be slightly off the expected value. This deviation may be so incredibly slight that during calculations this deviation doesn't really affect us, but in terms of comparisons, this can come back to haunt us. This is because when a computer does a comparison of two numbers, they *must* be identical or the computer will reject them for being un-equal. We may think that 30 and 30.000000000001 are effectively the same thing, but the computer won't. So when dealing with real values and variables, don't assume that the values are exact.

Up until now, we've been able to get away with it because we've been comparing to 0, and the timers will always hit 0 perfectly. The game will always make the timer hit 0 if it tries to go below 0 so we can be sure that this value *will* be set exactly. But for arbitrary values, this may not always be the case. The trick around this is to compare a range of values instead of a single one.

So lets trigger Betty's audio clip if the timer hits any value from 30 to 31 seconds. But we don't really know how often the game calls this module, so we may end up trying to play Betty's audio clip multiple times. So to avoid this, we will also need another flag to say if we've started Betty's audio or not.

First, lets add another static boolean to the module's variables. I added this:

```
static boolean      StartedBettyWarning;
```

And of course, as it's a static, we need to initialize it in the init function as well. Like this:

```
StartedBettyWarning = false;
```

Now, we need to add our if statement. We can also split it into an if inside another if. This can actually help speed things up a bit if we make the outside if check for conditions that don't happed too often. I added the following after that initial if in the code.

```
if(StartedObjective0Timer and not StartedBettyWarning) then
    if( (CheckObjectiveTimer(0) >= 30.0) and
                    (CheckObjectiveTimer(0) <= 31.0)) then
        PlayBetty(BETTY_THIRTY_LEFT);
        StartedBettyWarning = true;
    endif;
endif;
```

See if you can follow the logic here. The first if (the outside if) checks that the objective 0 timer was actually started and that we haven't yet started Betty's warning. If either of these cases are false, then the if is skipped. Otherwise, we enter this if block and do the code inside here. Which immediately does another if statement. This one of course is checking if the objective timer for objective 0 is greater or equal to 30 and less than or equal to 31. If so, then it starts Betty's audio warning and sets our flag saying that we have started it.

Try it out. Go capture the second building to start the primary objective timer, then wait for the timer to count down. When it is down to 31 seconds or less, it will start Betty's audio clip.

That about covers all the ins and outs about timed missions and objectives.

You may be wondering if it is possible to have an objective timer show up the way the mission timer does on the TAC map. Unfortunately, this is a “feature” of the mission timer only, and it is handled by the game. And since there is no known way to set the main mission timer other than through the .FIT file, there is no way to fake it either.

Using General Timers

So we've seen how to do mission and objective timers. But you may have noticed the game code also sets a timer called EXIT_TIMER using a function called SetTimer. It also checks the value of this timer via the function CheckTimer.

This section will be a brief discussion about using these general timers. I won't bother actually creating any file listings for this section as they work pretty much like the objective timers, so I will leave it as an exercise for the reader.

In total, the game has eight general timers that may be used with scripts. Though the game actually allows up to 32 timers in all, some of the timer functions we can use in our scripts only allows for eight of them. All these timers start from a given time value (in seconds) and count down to 0.

Accessing these timers is done with the three functions SetTimer, CheckTimer and EndTimer.

SetTimer will create and start one of these eight timers. It takes two parameters. The timer number to be created and the amount of time, in seconds, it is to count.

CheckTimer takes one parameter, the timer number, and returns the current countdown time. It functions pretty much like the objective timers we saw earlier.

EndTimer also takes a single parameter which is the timer number. When this function is called, the timer is stopped and removed, regardless whether the timer actually reached 0 or not.

So all we need to know in order to use these timers are the timer numbers. They are not 1 to 8, or 0 to 7 as you may expect. The timer values are in fact numbered 7 to 14. Thankfully, we don't have to deal directly with these numbers. The include file `misconst.abi` has some constants already defined for use with these timers.

Open `misconst.abi` and look at the constants defined starting at line 124. You will see the following:

```
SCENARIO_TIMER_1            = 7;
SCENARIO_TIMER_2            = 8;
SCENARIO_TIMER_3            = 9;
SCENARIO_TIMER_4            = 10;
SCENARIO_TIMER_5            = 11;
SCENARIO_TIMER_6            = 12;
SCENARIO_TIMER_7            = 13;
EXIT_TIMER                  = 14;
PERIMETER_ALARM_TIMER       = 13;
```

Note that the first seven timers are defined as values 7 to 13 inclusive, and that timer number 14 is defined as EXIT_TIMER. Further, note how the timer PERIMETER_ALARM_TIMER is also defined as 13 which is the same value as SCENARIO_TIMER_7. This means that missions that use perimeter alarms will also be using timer 13 and hence SCENARIO_TIMER_7 should not be used in the code of these missions or you may confuse things.

With this in mind, load up that first example program (`ex1.abl`). That ABL file was completely unmodified so we can examine it again without having to deal with our modifications.

Looking through the code we see this on line 189.

```
setTimer(EXIT_TIMER,2.0);
```

and Line 204 reads:

```
if ((ExitTimerSet) AND (CheckTimer(EXIT_TIMER) == 0.0)) then
```

These are examples of how to use these timers. Of course, EXIT_TIMER here is being used as a two second delay before the mission ends. This is done since normally the mission will be ending because a primary objective failed, or all the primary objectives have completed. In either case, the code that sets that last objective status is going to play one of Betty's lines. So the 2 second delay allows a little time to pass in order for the player to actually hear the line. Don't believe me? Try changing all the places where it sets the EXIT_TIMER so that the delay is only half a second. Play the mission and see what happens.

And to prove that the EXIT_TIMER isn't really special, try changing all the places it uses EXIT_TIMER to one of the other 7 timers. Say SCENARIO_TIMER_4. As long as you properly change all the places it refers to the EXIT_TIMER to this new one, it will run smoothly.

One thing that makes these timers different from the objective timers is that they do *not* assume that the time is from the beginning of the mission. Therefore, we don't need to add the current game time to them like we had to do for setting the objective timer part way through the mission.

Though you can in fact use EXIT_TIMER for whatever purpose you choose, it is better to leave that one alone because it is used by default as the exit delay timer. Also I'd suggest not using timer 13 either since that is also used for the perimeter alarm as we've seen.

Other that that, you should check that one of the other timers isn't being used somewhere else in the code or in the pilot brain files. The libraries should be safe as I don't think any of the functions defined there use timers, though this could change if you start using a library made by someone else.

Moving Objective Positions

What if we wanted to create a mission where the player has to capture a vehicle, but the vehicle is constantly moving. Or maybe we want to do something like have a capture or destroy objective where there is more than one thing to destroy and they are on completely different sides of the map. It would be nice if we could change the position of the objective shown in the TAC map.

Well we can. And in fact it is so simple to do that I'm not going to make an example mission. Everything we've discussed so far should be more than enough for you to figure out how to do most of this.

All that is required to set the position of an objective is to call the function SetObjectivePos. This function takes four parameters. The first is the objective number we want to set, and the others are the map coordinates we are trying to set. These coordinates are the X, Y and Z coordinates that we can get from the editor.

But it is even easier. The function actually ignores the Z coordinate even though it is needed as a parameter, so we can simply pass 0 as a value and it will work fine.

So lets say we want to set the position of objective 1 to coordinates -3427.0, 1889.0, 0.0 (again, I'm using a Z coordinate of 0 since it doesn't matter). We would simply call the function like this:

```
SetObjectivePos(1, -3427.0, 1889.0, 0.0);
```

If this was the position of the second object to capture or destroy, we could put the above line when the player captures or destroys the first object and so the player would then be presented with the location of the second object etc.

This is fine for static objects such as buildings, but what about vehicles, mechs, or trains that actually move. We don't know in advance what the coordinates are going to be.

In this case we can simply ask the object itself. All we need for that is the object's ID, an array of three real values and the function GetObjectPosition (we'll see later how to get the IDs of these objects, just assume that we currently have them).

To define the array, we put this in the variable block of our code.

```
Real[3] CurrentPos;
```

We could also have an integer defined for holding the ID of the object like so:

```
integer ObjectID;
```

Then, assuming that the variable ObjectID contained the correct ID of the object we want the position of, we would call GetObjectPosition like so:

```
GetObjectPosition(ObjectID, CurrentPos);
```

The function will actually fill our array CurrentPos with the coordinates of the object identified by ObjectID. Then, as before, we could simply set the location of the objective to these coordinates like this:

```
SetObjectivePos(1, CurrentPos[0], CurrentPos[1], CurrentPos[2]);
```

The above will work for any object as long as you have the correct ID for it. However, if the object is moving, then we have to do this all the time. But since the module code is also being called by the game all the time, we simply have to put the code in there. Though you will likely want to make sure that the objective isn't hidden or completed already.

This brings up a second point. How can we *hide* the objective marker from the TAC map.

This too is extremely easy. Simply give coordinates that are far off the map. The game simply won't draw it in this case. Coordinates of -10000.0, -10000.0, 0.0 will do just fine. They are extremely far off the map so there is no chance that it will be drawn.

To show it again, simply set it to a valid spot on the map.

Required Vehicles/Mech and Forcing A Given Unit To Capture Etc

Here's another topic that has mostly easy answers. But mainly because in this case the answers are negative.

First up is trying to force a specific player unit (vehicle or mech) to capture a given building (as they did in the second mission of the original campaign) or some other activity similar to this.

The sad news is that it simply cannot be done. If we take a closer look at that original campaign mission (`mis0102`), we will see that even the game developers cheat. Though the mission objective states that the player is supposed to capture the barn with the APC, what the mission is really doing is simply checking to see if the APC is within range of the barn when it does get captured. If not, it doesn't give you credit for the objective until it is.

When the APC does get in range of the barn, it checks to see if the barn is captured. If so, it then gives credit for the objective. You can try it out for yourself. Load up the original campaign and start the second mission. Take all your mechs through the mission leaving the APC behind at the starting point (make sure to kill all the guys just north east of the starting point as they periodically move a few of these by that start location).

When your mechs reach the barn, note how you are still allowed to capture it (the cursor still turns into the capture icon). Capture it and move your mechs to the extraction zone (after clearing out any stragglers). Now move the APC up next to the barn. Only now will the objective be marked as complete.

So really, the only way to “force” a unit to capture a building (or do some other task) is to fake it. Actually, this may not be completely true, I can think of one way this may be accomplished, but I'll leave that for another time.

Well that covers “forcing” units to capture a building, what about required vehicles or mechs.

What this is all about is forcing the player to use a given vehicle or mech for a mission. Unfortunately, this trick only works for the very first mission of the campaign because it requires editing a file in the PKK (this trick also works for solo missions).

Basically, we must give the player a mech or vehicle as part of his starting inventory and in the profile embedded in the PKK, make sure to set the Required flag to true (see part 1 for details of editing PKK files and adding starting equipment). Because this only works for the first mission, it is easier for us to simply give the player the vehicle or mech when he enters the mission.

We'll see how to do that a little later.

Creating Mechs with No Crews/Pilots

Imagine this scenario. You are creating an old abandoned base. In this base you want the player to find a couple of old mechs hanging around. There is nobody around to pilot them so they just hang about. If the player attacks them, they'll just stand there. Further, we want it to say no crew on the mech just like you can for vehicles. How to do this.

Well believe it or not, I found out how to do this from the official campaigns themselves. In operation 2 mission 5 of the original campaign (the one where you must intercept and stop a train, there is a single Uller-A that hangs out in the southeast corner of the map that has no pilot in it.

To go find the mech in question, start up this mission from the official campaign, run up and intercept the train but only destroy the train engine (the first car of the train). This way the train won't leave the map and so after killing the train guardian mechs, you can run around the map at your leisure. Just run back to where the train tracks cross the paved road and follow that road eastward. You'll find the mech in question on the side of the road near the southeast corner of the map.

As you can see, though it doesn't say “no crew”, it does say “no pilot” which in the end is actually more accurate since mechs don't have crews but a single pilot.

Now it is most likely the developers never actually intended this mech to be just sitting there without a pilot, but it did allow me to examine how to do this on purpose.

In short, all we have to do is to open the mission's .FIT file and set the Active flag to 0. This is enough to mark the mech as having no pilot.

But because of this simple discovery, I've experimented a bit and found a few other interesting things we can do with “pilot-less” mechs. So to show these, lets create another simple example mission.

Create a solo mission with a single mech (doesn't matter what mech it is) and also put a building down somewhere on the map. Put a drop-zone somewhere as well, and make the single objective of this mission to capture the building. The capture the building objective is so that even if we kill the mech on purpose, or by mistake, the mission won't end immediately.

Save this mission as “ex5” (example 5), and load the mission's .FIT file into a text editor.

Scroll to the bottom of the file and look for the section called `[Part1]`. This is where the mechs placed on the map are defined (one “part” for each mech or vehicle on the map). As we only have one mech placed on the map, there is only one part section.

In this section change the Active entry to 0 like this:

```
l Active = 0
```

Save the file and try out the mission now.

In game, place the mouse cursor over the mech and you will see that it now says it has no pilot. You can walk in front of it, even attack it and it won't budge an inch.

This is what you should see:

Well that was easy enough, but that was only a small change in the .FIT file. So how about doing things from a script.

We can change the active state of an object with the function SetObjectActive. This function needs two parameters. The ID of the object to be changed, and the new state of the Active flag to be set. This value is a boolean value with FALSE meaning 0 and TRUE meaning 1. So with this one function we can activate and deactivate any object, and therefore make a mech have, or not have, a pilot.

There is one catch though. How do we get the object ID for the mech.

Back when I discussed moving objectives, I said that I'd be showing how to get object IDs. Here is where I'll introduce how to get the ID of a vehicle or mech (getting the ID of a building is a little different so I'll leave that for the next section when we talk about creating salvage).

There is a function defined in the `miscfunc.abx` library that can help us get the ID of the mech. The function is called GetVehicleID, but don't let the name fool you. It'll work for mechs too.

The parameters for GetVehicleID are Force, Group, and Member. The Force parameter is an easy one. The values to use have constants defined for them already in `misconst.abi`. So open that file in a text editor.

Starting at line 168, we see the following three constants being defined.

```
PLAYER_FORCE                    = 500;
ALLIED_FORCE                    = 502;
CLAN_FORCE                      = 501;
```

These are the values we can use for the first parameter of GetVehicleID. So what about the other two parameters.

For these, we will have to do a little investigating. The answers however are held in the mission's .FIT file. So go back to `ex5.fit` and open it up. Go all the way to the bottom of the file. There you will see the following:

```
[Commander1Group:0]
l[12] Mates             = 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
```

Here, we see a section heading like we haven't seen before. It looks a little odd, but whatever. The section heading is telling us some information that we need though. Now as this is merely a simple map with a single mech on it, there aren't many section headings here, but there could be. So this is how to decode the section heading:

The “Commander1” part of the section heading name is actually telling us which force this section belongs to. There are only three possibilities here. “Commander0”, “Commander1” or “Commander2”. These correspond to the `misconst.abi` defined constants PLAYER_FORCE, CLAN_FORCE, and ALLIED_FORCE respectively.

So this heading is for a CLAN_FORCE, and as the rest of the section heading name implies, this is group 0.

Now as we can see there is a list of 12 numbers here. The first number in the list is 1, the rest are all 0s. For this list, treat 0 as meaning “nothing defined here”. But what about the 1. This number actually refers back to the part sections in this .FIT file. As we only have one mech, we only have one “part” which is called `[Part1]`. The 1 in this list refers to the 1 in the part section heading. So if there was a second mech, it would be defined in a section called `[Part2]` and the entry in these group lists would be 2 as well.

So now we know what the numbers in this list mean, but how do we use them. Simple. All you have to do is count. Just remember to start counting at 0 not 1. So since the mech we are interested in is the very first mech in this list, it would be number 0.

Now we have all the information we need to call the GetVehicleID function. The parameters we'll want are a force of CLAN_FORCE, a group number of 0, and a member number of 0. So our function call will look like this:

```
GetVehicleID(CLAN_FORCE, 0, 0)
```

Now remember this was all done simply to get the ID of the mech so that we can call SetObjectActive with this ID. So our actual function call will be:

```
SetObjectActive(GetVehicleID(CLAN_FORCE, 0, 0), FALSE);
```

Take a second to see what I'm doing here. I'm actually calling a function within the parameters of another function. This is possible because GetVehicleID returns an integer which is the ID of the mech or vehicle asked for. Since SetObjectActive wants an integer that is the ID of the object to set as active, this will work perfectly fine.

Of course, the above call will deactivate the mech in question, which, as we've already deactivated it in the .FIT file, won't really do much. But to set it to active again, all we need to do is change the FALSE in the above call to TRUE.

Try adding this line to the init function of `ex5.abl`:

```
SetObjectActive(GetVehicleID(CLAN_FORCE, 0, 0), TRUE);
```

After adding above line to init function:

Well that was rather underwhelming. It no longer says “no pilot” but the mech isn't budging. What gives.

This is something we all have to get used to. The order the game does things in. Remember, we are turning off the active flag in the .FIT file, then turning it back on in the mission's init function. The problem here is that we've set the state twice in two different ways, and the game has yet to even run the AI code of the mech itself. So things are a little confused. It has a pilot, but it doesn't. We just need a little time delay for the game to start things up properly first. Note that this only applies to mechs that we disable via the .FIT file.

So to fix this little bump, lets put a little time delay on the re-activation of the mech. Remove the line we added in the init function and put this in the main code. I put mine around line 129:

```
if(gametime >= 3.0 and gametime <= 3.9) then
    SetObjectActive(GetVehicleID(CLAN_FORCE, 0, 0), TRUE);
endif;
```

Note that this calls the activation of the mech 3 seconds after the mission has started.

With a small delay, the mech gets re-activated properly:

Again, I want to re-iterate that this applies only to mechs or vehicles that have been disabled via the .FIT file, and only if trying to reactivate them very early in the code. If we disabled the mech in the script, this would not be a problem.

In fact, the delay needed seems to be around 3 seconds, but I haven't done extensive tests on this.

There is one last thing I'll mention here. If we deactivate a mech while it was attacking a target, it will *not* lose that target. So if it is reactivated, it will attempt to go attack that target even if it is on the other side of the map. So we should clear out any targets it has either before deactivating it, or immediately after reactivating the unit.

Creating Salvage & Capturing Buildings

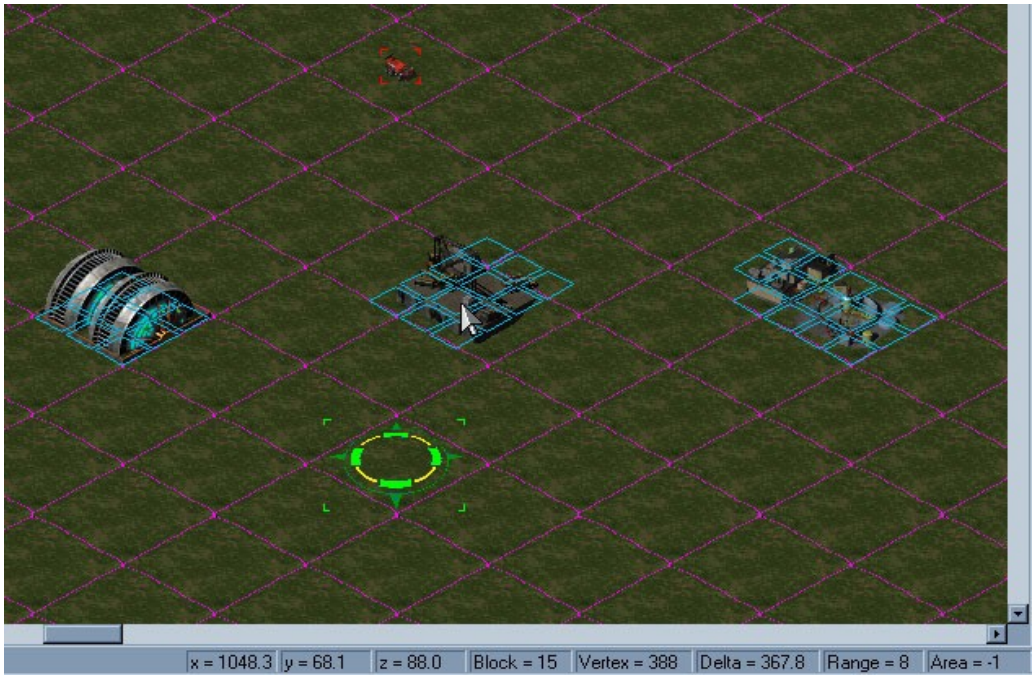
When creating maps in the editor, there are a couple of things we can place on the map that the player can capture and get salvage. However, when we play the official campaigns, we see that they they have all kinds of buildings and objects that can be captured and give out salvage. This section will show how we can do the same for our own campaigns.

Basically, we need to do three things in order to create a building that can be captured and give salvage. First, we must find a way to get the ID of the building we wish to affect, second, we must tell the game that this building should be flagged as available for capture, and third, we must tell the game what salvage items should be given to the player if and when they do capture the building.

So to start things off, create a new solo mission and place a drop zone and a single Swiftwind on the map. Leave the default objective as destroy all enemies as this will allow us to run around the map for test purposes without ending the mission until we kill off the Swiftwind.

Now on this map place three buildings. Use buildings that cannot normally be captured as well. This will help demonstrate that we can do this for almost any object we can place down in the editor (not everything can be, but most).

Now before we save this mission, we are going to need some information from the editor so that we can get the IDs of the buildings in the script. So for each of the buildings, hover the mouse over the building and write down the Block and Vertex numbers that are shown on the status bar at the bottom.



Once you have the block and vertex numbers for all three buildings, save the map as “ex6” (example 6). Then close the editor and open `ex6.abl` in a text editor

Now we are going to need three variables to hold the object IDs of these buildings. A little later in this section we will be doing some other things to these buildings, and therefore we should also make these variables static so that we don't have to constantly try to get the IDs of these buildings. In the modules variable block, add three static integers like this:

```
static integer SalvageObj1;
static integer SalvageObj2;
static integer SalvageObj3;
```

I just gave generic names for these variables, but you could give them more descriptive names to help identify which variable goes with which building.

Now once again, as these are static variables, we need to initialize their values, but the value we need is the object ID of these buildings. We can get this with the built-in function GetTerrainObjectPartID. This function takes two parameters which are the block and vertex numbers that we got from the editor. So for each variable, we will initialize them with a call to this function with the correct block and vertex numbers. Add the following to the init function, but make sure that you use *your* block and vertex numbers.

```
SalvageObj1 = GetTerrainObjectPartID(21,26);
SalvageObj2 = GetTerrainObjectPartID(15,388);
SalvageObj3 = GetTerrainObjectPartID(15,350);
```

That is all we need to get the object IDs of the buildings. Now to set these buildings as available for capture. Since we want them available right from the start of the mission, we'll do this in the init function as well. We will use another built-in function named SetCaptureable (note the spelling) to do this. This function takes two parameters. The first is the ID of the object we want to set, and the second is a boolean value to say if the capture state should be on or off. Well we know the state, we want it set to TRUE. And as we just found what the IDs of these buildings are, we also have that information.

So to set these buildings as available for capture, all we need is these three additional lines of code in our init function:

```
SetCaptureable(SalvageObj1, TRUE);
SetCaptureable(SalvageObj2, TRUE);
SetCaptureable(SalvageObj3, TRUE);
```

All that is left is to tell the game what items should be handed out for each building. We do this with the built-in function SetSalvage. This function takes three parameters. The first is again the ID of the object to affect, the second is the ID of the component to be handed out, and the third is how many of these components should be given. The object ID we already have, and we can decide how many to hand out too. The component ID is nothing more than the component number from the table that can be found in part 1 of the guide.

So all we need to do is look through the component table and decide which component to give out, and use that number as the second parameter to this function.

If we want to give more than one component type on the same object, we can call this function multiple times on the same object and it will add the new items to it. This does mean however that we can't remove individual components from an item once they are set.

I decided that the first building will hand out two Clan ER-PPCs and three Large X Pulse Lasers. The second building will give a single Clan Ultra AC/20 and the third will give 4 Clan Pulse Lasers, a Clan Heavy Flamer, and a Rail Gun (Hey, it's an example mission. It isn't like they'll be using it to power through a campaign or anything).

So to do the above, still in the init function, we would add the following code:

```
SetSalvage(SalvageObj1, 154, 2);      // 2x Clan ER-PPC
SetSalvage(SalvageObj1, 139, 3);      // 3x Large X Pulse Laser

SetSalvage(SalvageObj2, 112, 1);      // Clan Ultra AC/20

SetSalvage(SalvageObj3, 153, 4);      // 4x Clan Pulse Laser
SetSalvage(SalvageObj3, 155, 1);      // Clan Heavy Flamer
SetSalvage(SalvageObj3, 98, 1);       // Rail Gun
```

I added the comments to make things a little clearer.

Save the .ABL file and try it out.

Capturing the three buildings gives:

As mentioned earlier, not everything you can place down in the editor can be set as being available for capture. Or more precisely, the game won't give the capture icon for all items that can be placed down, so be realistic. Though the above code will execute fine, don't go expecting to be able to capture an extraction marker or some land mines. The game simply won't give a capture icon for these objects.

Earlier, I mentioned that you cannot remove items from the salvage list of an object once it is set. This is true, but you *can* remove all items that were salvaged from that object from the player's salvage list.

We do this via the built-in function SetSalvageStatus. This function takes two parameters. The first is once again the ID of the object to be affected, and the other is a boolean value saying if the items on this object can be salvaged (even if the object can't be captured).

So to remove all the items that were on SalvageObj1 from the player's salvage list, we can simply call SetSalvageStatus like this:

```
SetSalvageStatus(SalvageObj1, FALSE);
```

We could also re-instate them like this:

```
SetSalvageStatus(SalvageObj1, TRUE);
```

What this function is actually doing is saying that the object itself is handing out items. It doesn't even matter if the object has been captured or not. If the object's salvage status is set to true, it will give salvage on that object no matter what.

To test this, lets remove the capture flag from the second building and make it so that when the first building is captured, it gives out the salvage for the second building as well.

So go back to the .ABL file and remove or comment out the line that says:

```
SetCaptureable(SalvageObj2, TRUE);
```

(to comment the line out, just put a double slash ('//') at the start of it).

Now, to check if the first building has been captured. Add this into the main code of the module around line 147 or so (this should be between the first if that keeps track of time and the `#include "ex6_str.abi"` line).

```
if((IsCaptured(SalvageObj1) == 1) and
                (ObjectSide(SalvageObj1) == INNER_SPHERE)) then
    SetSalvageStatus(SalvageObj2, TRUE);
endif;
```

Note how this if statement works. We are checking if the object (in this case the first building) has been captured, and if so which side has captured it (INNER_SPHERE is the player side). If this is true, then it sets the salvage that was on the second building as part of the salvage list even though we haven't captured that building.

Save the .ABL and try it out.

Got the salvage of the loading facility:

Now you will notice that the second building is still red to us. This is because that second building hasn't changed at all. We only said that the building's salvage should be given to the player. So if we wanted to make it so that building also was tagged as belonging to the player side, we need to call the built-in function ObjectChangeSides. But even that will only tell the building which side it is on. What if we also wanted that building to be tagged as being captured. In short, capturing the first building also captures the second one. We can do this with the built-in function SetCaptured.

Lets try that.

In the code we just added to give the salvage of the second building to the player, we now add the calls to the functions ObjectChangeSides and SetCaptured. The SetCaptured function is the easiest to deal with. It only takes one parameter and that is the ID of the object that is to be set as captured. For the ObjectChangeSides function we need two parameters. The first is again the ID of the object that is to change sides, and the second is which side to assign it to. The if statement we are adding these calls to gives a hint as to what we can use here. The call to ObjectSide, we now know, returns which side an object is on. And our if statement is looking for the constant INNER_SPHERE. Well that is also the value we will need for the call to ObjectChangeSides. Change the if statement we just added so that it now looks like this:

```
if((IsCaptured(SalvageObj1) == 1) and
                  (ObjectSide(SalvageObj1) == INNER_SPHERE)) then
    ObjectChangeSides(SalvageObj2, INNER_SPHERE);
    SetCaptured(SalvageObj2);
    SetSalvageStatus(SalvageObj2, TRUE);
endif;
```

Save the file again and try it out.

Capturing two buildings at once:

As we can see, the second building is now also tagged as being on the player's side, and though we can't see it, it is also marked as captured.

Note that just saying that the building is captured and/or marking it as belonging to the player's side doesn't actually hand out the salvage on that building. We still need that call to the SetSalvageStatus function.

If the player destroys a building that is giving him salvage (even if set to the player's side), he will automatically lose all the salvage from that building. We don't actually have to do anything at all. The game will handle this for us. This does mean however that we should set these building to at least be on the player's side so that they don't go destroying them by accident (if they do it on purpose however, then they are getting what they deserve).

You will notice that when you hand out salvage, either via capturing the building normally or through the SetSalvageStatus function, Betty will chime in with her “Enemy components captured” line. This is done by the game itself so there isn't anything we can do about it. A shame really, it would have been nice to have the option of playing a different Betty clip or no clip at all.

One last note. If you place a building that the editor normally sets as being able to be captured, or the building in question is one you made part of a “Capture the building” objective, then the editor will generate code to set the object's side and capture flags. This code is actually placed in the *xxx*`_init.abi` file that you can see included in the init function (in the case of this example, the file is `ex6_init.abi`). We could of added the example code to this file instead of directly in the init function, it would have had the same effect. But that is the file you should look at if you do want to change a few things from what the editor makes by default.

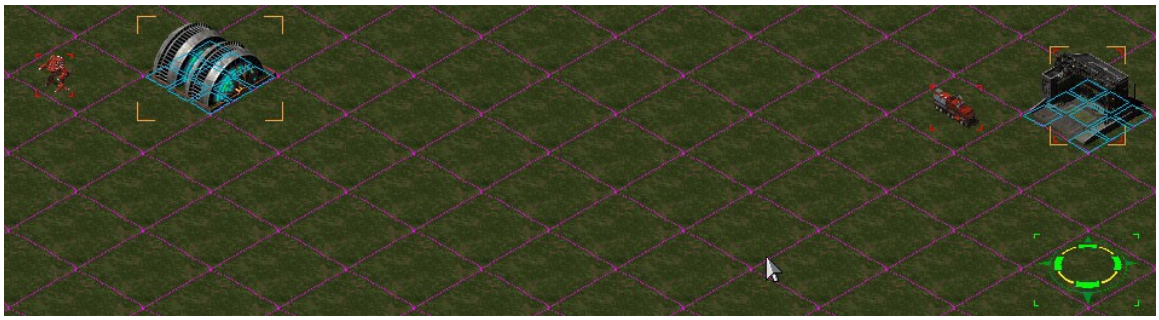
Capturing and Salvaging Vehicles and Mechs

So we've now seen how to capture a building and set the salvage on it. With that under our belts, lets now see about capturing and salvaging vehicles and mechs. The easiest to deal with are vehicles, so lets start with them.

To make a vehicle that can be captured, we do the exact same thing as with buildings. Except of course, we use the ID of the vehicle. We've already seen how to get a vehicle ID when we discussed how to make a mech with no pilot. We use the same method for vehicles.

So all that needs to be done is to call GetVehicleID to get the vehicle's ID, then call SetCaptureable with this ID.

Lets try this out. Load up the editor and create a new solo mission. Place a vehicle such as a MobileHQ, somewhere on the map. Also place two buildings on the map. The first, place it near the vehicle, and make it the objective of the mission (capturing this building). The second building, place it far enough away so that we could capture it without interference from the vehicle. Place the drop zone near the vehicle. Now place a mech near the second building. This mech should be close enough that it will block the capturing of the second building (this setup is done on purpose so that I can demonstrate something when we get to capturing mechs). Finally, make the second building (the one near the mech) be a second objective. You should have something like this:



When done, save it as ex7, close the editor and load up `ex7.abl` into the text editor.

Now, we are going to need the ID of the vehicle and hence a static variable to hold this value in as well. So in the module's variable block, add the following:

```
static integer MobileHQID;
```

And we'll initialize it in the init function with a call to GetVehicleID. Again, use the same technique we used for finding the ID of the mech when we made the pilot-less mech.

I added this (though of course, your parameters may be different):

```
MobileHQID = GetVehicleID(CLAN_FORCE,0,0);
```

Now, just as we did for buildings, we call SetCaptureable to allow this vehicle to be captured (do this in the init function as well).

```
SetCaptureable(MobileHQID, TRUE);
```

We can also assign some salvage on the vehicle exactly as we did for buildings. Lets add a couple of Clan Pulse Lasers.

```
SetSalvage(MobileHQID, 153, 2);
```

Save the .ABL file and try it out.

Capturing the Mobile HQ gives:

Though we get the salvage we assigned to the vehicle, note how the vehicle itself hasn't changed sides. This is something that is a little different from capturing buildings. Vehicles don't automatically get assigned to the player's side.

To change this, just like we did when setting which side a building is on, we can use the ObjectChangeSides function with the vehicle's ID. I added this around line 135 in the main code block:

```
if((IsCaptured(MobileHQID) == 1) and
               (ObjectSide(MobileHQID) <> INNER_SPHERE)) then
    ObjectChangeSides(MobileHQID, INNER_SPHERE);
endif;
```

Note the use of the “not equal” operator in the if condition. Save and try the new version out.

Capturing the Mobile HQ now puts it on the player's side:

Though I won't demonstrate it here, the SetSalvageStatus function also works for vehicles exactly as it does for buildings.

That takes care of capturing vehicles, so you may be asking yourself what it is about mechs that they need a separate description.

Unfortunately, though we can call the SetCaptureable function on mechs as well, the internals of this function will actually ignore mechs. This means that there is no way for us to actually set a mech as capturable.

Therefore, the only recourse we have is to make the capture by proxy. That is, much like we did with capturing two buildings at once, we must use some other event in order to make the mech captured. Lets do this when we mark the Mobile HQ as being on the player's side.

But first, we are going to need another static integer to hold the mech's ID. So once again, go up to the module's variable block and add:

```
static integer MechID;
```

And in the init function, we once again must get the ID of this mech with the GetVehicleID function. I added this:

```
MechID = GetVehicleID(CLAN_FORCE,0,1);
```

(don't forget to change the parameters to what you need).

And now, in the if statement where we change the Mobile HQ side, we set the mech as captured as you'd expect:

```
SetCaptured(MechID);
```

Save this new version and try it out.

Mech is now captured:

After capturing the Mobile HQ, we can go over to where the mech is and as it is captured, it doesn't attack. But there are a few things to take note of here.

First, like the vehicle, setting the mech as captured doesn't switch which side the mech is on. Further, again as with vehicles, the mech hasn't actually been added to the salvage list. Finally, the mech is blocking the capture of the building.

The first two problems are easy enough to fix. All we need to do is change which side the mech is on via the ObjectChangeSides function, and we can add the mech to the salvage list by calling the SetSalvageStatus function. So lets add these calls where we set the mech as captured. I added this:

```
SetSalvageStatus(MechID, TRUE);
ObjectChangeSides(MechID, INNER_SPHERE);
```

Save and try this new version out.

The mech is now added as salvage:

Hooray! The mech is added to the salvage list, and it is on our side too. But wait a minute, the building the mech is standing next to still can't be captured even though the mech is on our side. Why is that.

The game seems to determine if something can be captured or not based on weapons being close by. As the mech obviously has weapons, it blocks the capture. An easy solution to this is to remove the pilot from the mech as this shuts the mech down, hence no weapons. We already know how to make pilot-less mechs, so lets try that. I added the following line:

```
SetObjectActive(MechID, FALSE);
```

Save the file and try it out now.

Mech no longer blocks the capture:

Finally, the mech isn't stopping us from capturing the building. Note that we didn't really need to change which side the mech was on to be able to capture the building. It was the mech's active state that did the trick for us.

There is one other difference between mechs and vehicles that I should mention. You can't set salvage items on a mech. Though the script won't crash if you try, the function itself will simply do nothing and so when you give the mech as salvage, these items won't show up as they were never actually set.

So now we can capture and/or give vehicles and mechs as salvage. But his actually brings up one more point that we should explore a little. Since we can give out mechs and vehicles even if the player hasn't “captured” them (that is, via the SetSalvageStatus function), is there a way to hide a mech or vehicle from the map so that the player never sees it, but that we can still give away as special salvage etc.

The answer is yes, there is. And it is very easy to do as well.

Simply change the Exists flag in the .FIT file (the one right after the Active flag). When this flag is set to 0, the game will not place the mech or vehicle on the map. But we can still hand them out as salvage etc. Very convenient.

Lookout Points

Time for another easy topic: Lookout points. You may be wondering what exactly a lookout point is. Simply speaking, lookout points are areas of the map that get revealed to the player when some event happens in game. You have seen this in the official campaigns where capturing a Mobile HQ or some building reveals a few areas on the map. These revealed areas are lookout points.

The basic lookout point is nothing more that a single call to the function SetRevealed. This function takes three parameters. The first is which side we are revealing the map for (very nice to have for multiplayer games), the second is the radius of the area to be revealed, and the third is an array of three real values containing the map coordinates that should be revealed.

So if we wanted to reveal a circular area of 100 meters around point X = 245.0, Y = 532.0, Z = 12.0, and we had an array of 3 reals called Point, we would call this function like this:

```
Point[0] = 245.0;
Point[1] = 532.0;
Point[2] = 0.0;
SetRevealed(INNER_SPHERE, 100.0, Point);
```

Two things to note here. First, the Z coordinate for the point is set to 0. This is because the function actually ignores any Z value passed to it anyway, so it doesn't really matter what is placed here. The second thing to note is that the constant INNER_SPHERE is being used to tell the function which side to reveal for (of course, we would likely want to put this when some other event happens, such as when a given building or vehicle is captured)

To reveal multiple points, it simply requires multiple calls to this function. For example:

```
Point[0] = 245.0;
Point[1] = 532.0;
Point[2] = 0.0;
SetRevealed(INNER_SPHERE, 100.0, Point);
Point[0] = -1045.0;
Point[1] = 1099.0;
Point[2] = 0.0;
SetRevealed(INNER_SPHERE, 100.0, Point);
```

We can also reveal the entire map by realizing that the center of the map is at coordinates 0,0,0 and that a radius of 1500 meters will cover the map. Like so:

```
Point[0] = 0.0;
Point[1] = 0.0;
Point[2] = 0.0;
SetRevealed(INNER_SPHERE, 1500.0, Point);
```

Giving In-Mission Drop Groups

Back when I discussed required mechs and vehicles for missions, I suggested that the best way to do this was to simply give the mech or vehicle inside the mission. This section is where we will learn to do this.

We are going to have to do a fair bit of editing for what is actually not that hard a thing to do. This is due to two factors. First, the editor does not allow us to place friendly mechs on the map. We can place allies, but not friends. Therefore, we are forced to place the desired mechs on the map, then manually edit the .FIT file in order to change them into friendlies that can be given out.

The second factor is when attempting to identify the mech or vehicle IDs. We are going to have to jump through a couple of hoops just to get the proper IDs.

But apart from this, the handing out of friendlies inside a mission is simple and straightforward. In fact, we've already dealt with all the functions needed.

Turns out that the game handles friendlies in a different way than the other groups. So some of the features discussed earlier actually act a little differently when talking about friendly mechs and vehicles. The largest difference is in how the game treats the active flag in the .FIT file. If we attempt to place a mech or vehicle into a PLAYER_FORCE group (Commander 0) and this vehicle or mech is set as active, then it will *not* put this mech on the map. There actually is a logical reason for this, though it is beyond the scope of this document. Suffice it to say that it is due to a special mode the game can run in.

Therefore, if we are to give a mech to the player in-mission, we *must* set the active flag to 0. This means that giving the mech to the player is nothing more than setting the active flag of the mech to true (1) in game. We've already seen how to do this via the SetObjectActive function. The function needs the ID of the mech or vehicle to be set and the state of the active flag we wish to set. But this brings us to another problem caused by this same special treatment of PLAYER_FORCE mechs and vehicles.

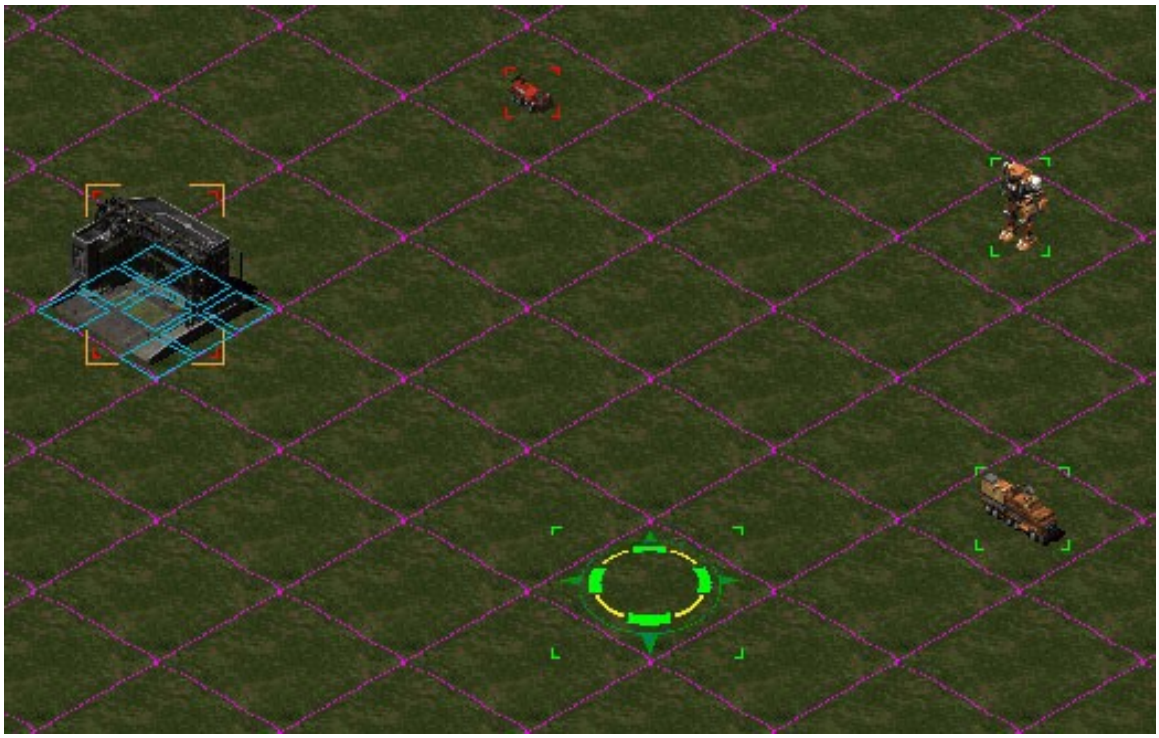
The game not only treats the active flag differently, it also re-arranges the order of the vehicles and mechs. This is in part due to that same special mode I mentioned, and partially because the game does not know in advance the number of mechs, and in which drop zone group, the player will bring into the mission from the logistics screen. Because of this, the game is actually forced to re-sort the list of mechs and vehicles on the PLAYER_FORCE. The sort order appears to place all mechs first, then vehicles.

Knowing this however, we can manually arrange the PLAYER_FORCE groups so that this re-sorting will re-create the same list and therefore, not change things on us. Another method would be to do a search through all available mechs and vehicles to find the one we are interested in. In this case, the order wouldn't actually matter anymore.

Now that we know this, lets create a mission to demonstrate how to give extra drop groups to the player.

Open the editor and create a new solo mission. Place a single drop zone on the map and also a single Swiftwind. Place a building somewhere and make the objective of this mission to capture the building (we use this so that killing the Swiftwind won't end the mission). Now to place a vehicle and a mech to give to the player. Select a mech (doesn't really matter which one) and place it on the map as an "Ally" mech, and also set this mech's default attack orders to none. We do this so that it will be easier to find the mech when we go edit the .FIT file. Do this again for a vehicle.

This is what my map looks like:



Save the map as ex8 (example 8) and open the .FIT file (`ex8.fit`) into a text editor.

First, scan through the list of warriors (look for the `[WarriorN]` headings) and mark down which ones have a brain file identified as `"Drednull01"`. These are the ones we set as having no attack orders. We'll come back to edit these in a bit, but right now we just need to know which warriors are the ones we set as having no orders. So write down the number identified in the heading so we know which pilots we will need to change. In my case, it is warriors 1 and 2 that had the `"Drednull01"` brain.

Scroll a little further down in the file and find the [Part*N*] headings. Find all the ones that have a TeamID of 2 and a CommanderID of 2 and also have a Pilot entry that is one of the pilots we identified in the warrior list as having a "Drednull01" brain file. Write down which part number these are. In my case, I placed an Atlas and a MobileHQ that were to be the ones to give to the player. The Atlas is Part1 and the MobileHQ is Part2, so once again, I want numbers 1 and 2.

For each of the Part sections now identified, change TeamID and CommanderID to 0, change the Active flag to 0, and lastly, change PlayerPart to TRUE. This is what my Atlas mech entry looks like after these changes (I removed the comments to make it fit in this document):

```
[Part1]
ul ObjectNumber         = 51
ul ControlType          = 2
b PlayerPart            = True
ul ControlDataType      = 1
c MyIcon                = 0
c TeamID                = 0
c CommanderID           = 0
st ObjectProfile        = "PM109300"
ul Pilot                = 1
f PositionX             =  960
f PositionY             =  576
f PositionZ             = -1.0
f Rotation              =  45
ul Gesture              = 2
f Velocity              = 0.0
l Active                =  0
l Exists                =  1
```

Now for each of these parts, look at the Pilot entry, and go back and edit the warrior section that is for this pilot. For example, the Atlas entry shown here has Pilot 1, so I go edit the [Warrior1] section. For mechs, set the Profile entry to that of one of the pilots that the player can hire. I decided that I wanted Lynx so I set the Profile entry to "PMW00066" (see part 1 of the guide for details about warrior profiles). Also change the Brain entry to "pbrain". This is the default brain for all player controlled mechs (and many others too) . Though of course, you could use a different brain file if desired.

For vehicles, again the Brain file should be set to "pbrain", and the Profile should be set to one of the generic vehicle crew profiles. I chose "PCREWB" for the profile of the Mobile HQ. After these changes, My two warrior sections affected look like this:

```
[Warrior1]
st Profile = "PMW00066"
st Brain = "pbrain"

[Warrior2]
st Profile = "PCREWB"
st Brain = "pbrain"
```

Scroll to the bottom of the file and find the `[Commander2Group:0]` heading. Recall that Commander2 is the ALLIED_FORCE groups. If we had many more allies on the map, we would need to remove the mechs we identified earlier as being the ones we want and move those into their own group, but as we only have these two and they both need to move to the PLAYER_FORCE, we just need to change the heading to `[Commander0Group:1]` instead of `[Commander2Group:0]`. This also shows another little quirk with PLAYER_FORCE groups. Any group we create to be given to the player needs to start in group 1, not group 0. Group 0 has special meaning for the PLAYER_FORCE, so don't use it. After the change, my section now looks like this.

```
[Commander0Group:1]
l[12] Mates             = 1, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
```

As I mentioned at the start of this section, the game will re-order the entries in this list. So make sure that any mechs come first, then vehicles. I go so far as to keep them in ascending numerical order as well. My list is already in the correct order so I had nothing to change here.

Well That should do it for the .FIT file. I did say there were a lot of edits to be done, but that this was mainly due to the fact the editor doesn't allow us to place friendly mechs and vehicles.

Save the .FIT file and open the .ABL file (`ex8.abl`). It is time to actually assign these guys to the player.

First thing we need to do is decide how and when we are going to give these mechs and vehicles to the player. We can of course simply assign them at the start of the mission, but that isn't a whole lot of fun. Instead, lets make it so that they are given when all enemies are destroyed. As we only have a single Swiftwind on the map for the enemy forces, this shouldn't take very long.

So to begin, we need to declare a couple of static variables to hold the IDs of the vehicles in question. So in the variable block, I added:

```
    static integer Atlas;
    static integer MobileHQ;
```

And of course, in the init function I initialize these variables with the IDs. I once again use our friend GetVehicleID like this:

```
Atlas = GetVehicleID(PLAYER_FORCE,1,0);
MobileHQ = GetVehicleID(PLAYER_FORCE,1,1);
```

Note the use of PLAYER_FORCE here, as well as using group 1 as we should.

Now all that is left to do is check to see if all clan forces have been destroyed, and if so, activate the mech and vehicle via the SetObjectActive function and they will be added to the player controlled forces.

Now at this point you may be asking yourself, won't we need some kind of flag to see if we've already activated the mech and vehicle? You are right, but we don't actually need to make a variable for it. We can use SetObjectActive's counterpart function GetObjectActive. It takes one parameter, the ID of the object to be checked, and returns the current active state of that object. Therefore, we can use the unit's own active state as the flag. Just note that GetObjectActive doesn't return a boolean value, but an integer. So we'll need to compare it to 0 or 1 instead of false or true.

We know the code created by the game already keeps track whether or not all the forces of a given group are dead or not, and keeps this fact in the boolean variables PlayerForceDead, ClanForceDead, and AlliedForceDead. So we have access to all the information we need. Time to make the code. I added the following around line 133 (after the starting if statement and before the `#include_ "ex8_STR.ABI"` line):

```
if (ClanForceDead and (GetObjectActive(Atlas) == 0)) then
    SetObjectActive(Atlas, TRUE);
    SetObjectActive(MobileHQ, TRUE);
endif;
```

That's all there is to it. Pretty straightforward really. Note that I don't bother checking the state of the Mobile HQ. This is because I'm making it active at the same time I make the Atlas active, so its active status will follow that of the Atlas.

Save the .ABL and try it out. Kill the Swiftwind and you will have a Mobile HQ and an Atlas with Lynx in it to help you out.

Beast is all alone at the start:

Starting the mission, the mech and vehicle are shutdown:

Note how the mech and vehicle are in a “Shutdown” state when you first start the mission. Also note how they are colored blue and gold even though we didn't use these colors in the editor.

After killing the Swiftwind:

There is only one thing left to mention. You must leave at least one drop-group free in order to allow one of these groups to be added to the player. Even though in-game there can be four groups and the logistics screen only allows for a maximum of three, the game will still check if there is a free drop-zone at mission start and if not, will actually bomb out with an error and exit. Therefore, never have more that two drop zones in any mission you intend to give mech and/or vehicle groups in game.

Controlling Units from the Mission ABL

Up to this point, we've been dealing mostly with map and mission features. But that isn't the only thing we can do in the mission's .ABL file, we can also control individual units and even groups so that they can perform special activities.

As an example, I'm going to re-create some code that I was asked to make for a campaign being made by LipSheZ (online name on the MechCommander.org forums). What we will be doing is creating a group of friend units. That is, when one unit decides to attack, he will “radio” his friends for help. These friends will come over to see what is happening and if they see an enemy, they will also attack.

So lets first examine what we will need from a scripting point of view. First, we are going to need to maintain a list of unit IDs for all the units that are friends (that is, that will come to help any of the other members of this group). We will also need a way to detect when any member of this group is attacking something, and of course a way to order a unit to come to the help of another.

As we can see, this is going to require a little more than a line or two of script to achieve.

So first, lets make yet another solo mission. In it, we'll place a couple of Ullers and a couple of tanks as well. We'll make all four part of the same friend group. Place one Uller in the northwest corner of the map, and the other in the northeast corner. The tanks I'm placing near the southeast corner of the map, and the drop zone will be somewhere in the middle. Leave the objective as “Destroy all enemies”.

Save the map as ex9 (example 9), and open the .ABL in a text editor.

Now since we need to see what the units are doing and they are far away, lets reveal the entire map. This will allow us to see the units on the TAC map. So add an array of three reals to the module's variable block like this:

```
real[3] Position;
```

We don't need to make it static since we only need it to hold coordinates that we will set prior to calling the functions that need it.

Now to reveal the entire map, as mentioned in the section on lookout points, we reveal from the center of the map in a radius of 1500 meters. Since the center of the map is at coordinates 0,0,0 we put the following in the init function:

```
Position[0] = 0.0;
Position[1] = 0.0;
Position[2] = 0.0;
SetRevealed(INNER_SPHERE, 1500.0, Position);
```

Remember, we are putting this in simply so we can see what is going on. In a real mission you would likely never use this.

Alright. With that out of the way, it is time to really start looking at the code. To make this easier to deal with, we are going to create a couple of functions.

As mentioned a little earlier, we need a way to make one unit, be it a vehicle or a mech, come to the help of another unit. So lets make a function that will do exactly that. But we should decide what parameters are going to be needed for this function, and also what the return value will be, if any.

Since we need to make one unit come to the aid of another unit, it should be obvious that what we will need as parameters are the ID of the unit that should come help, and the ID of the unit that is requesting help. We don't really need a return value for this function so lets not bother with one. Now we need a descriptive name for this function as well, so that we know what it is trying to do. How about "RequestHelpFromFriend", that seems descriptive.

We now have enough information to create the shell of the function. That is, we can define the function itself, but of course, we don't yet have any code to put in it, but that'll be next. So after the init function, but before the the main code, add the following.

```
function RequestHelpFromFriend(integer RequesterID, integer FriendID);
code
endfunction;
```

Now for the code to put in here. We need to get the friend unit to come help the "requester" unit. How to do this. Well, one simple way is to simply order the friend unit to go to the location of the requester. If there are any enemies nearby he'll attack according to his normal orders anyway. Sounds like a sound plan. How to do that then.

Looking through the built-in functions, we find one called OrderMoveToObject. That's what we need, but wait a minute. That function's parameters are the ID of the object to move to, and a flag to say if we should walk or run. How does it know which unit to send the order to?

This is where we must introduce the concept of currently selected objects. Normally, these functions are for use inside a given unit's brain file. Therefore, the currently selected object is the unit that is running that brain file code. But we are doing this from the mission's brain file (the mission's .ABL file) so we need to explicitly select the object. There are various functions to allow us to do that, so lets take a closer look at the OrderMoveToObject function and see what it actually wants. From the description in the function definition, we see that this function actually sends its orders to the currently selected pilot. So to select a pilot, we are going to need the SelectWarrior function. But that function wants the ID of the pilot to be selected, so we also need the function GetPilotID which will retrieve the ID of the pilot of a vehicle or mech.

Sounds a little complicated, but the whole thing really boils down to two simple lines of script like this:

```
    SelectWarrior(GetPilotID(FriendID));
    OrderMoveToObject(RequesterID, true);
```

Note what is being done here and how. We first set the current pilot as being the pilot of the friend unit, and then order that unit to run to the location of the requesting unit. Doesn't seem that complex when you see the script, does it.

Well that should work, but we can do a little better than this. What would happen if the Friend mech was already in the middle of fighting when we called this function. Whoops, it would still make that unit run to the requester. So before we order the friend unit, we should check to see if he doesn't already have a target. If not, then we can order the friend to come help, otherwise we'll leave him alone.

We can check if a given unit has a target by calling the GetTarget function. This function requires the ID of a unit that has a pilot, so it only works for vehicles and mechs. But that's OK as we want this for mechs and vehicles anyway. The function's return value is the ID of the unit being targeted, or 0 if he doesn't have a target. Just what we need. So the above two line should be inside an if statement like so:

```
    if(GetTarget(FriendID) == 0) then
        SelectWarrior(GetPilotID(FriendID));
        OrderMoveToObject(RequesterID, true);
    endif;
```

So that is what is needed as the code for the function, so add those four lines to the function code to get this:

```
function RequestHelpFromFriend(integer RequesterID, integer FriendID);
code
    if(GetTarget(FriendID) == 0) then
        SelectWarrior(GetPilotID(FriendID));
        OrderMoveToObject(RequesterID, true);
    endif;
endfunction;
```

Now we have a function that we can call to make one unit go to the aid of another unit if the friend isn't already busy attacking something.

But we want more than this. We want to check to see if a unit is attacking and if so, tell its friends to come help. We know how to check to see if a unit is attacking, We'll check if is has a target. But which units should we check, and which are its friends.

We can maintain a list of units that are friends with each other. We can then periodically run through this list and see if any of them are attacking. If so, we can then tell all the other members in the list to come help. Lets once again put all this is a function.

So first, we are going to need to keep a list of all the units in the group. As we are going to be sending this list to a function, we are going to need to create a special variable type (seems that the ABL language doesn't like array declarations in the parameter list). This also means we need to decide the maximum number of friend units we can have in a friend list. Lets say six to start, we can always change it later. However, changing it could be troublesome because it is possible that we might forget to change something. This is where constants can come in handy. So lets start by making a constant for the maximum size of the friend list. Lets call our constant MAX_FRIEND_LIST_SIZE, and as mentioned, lets make it the value 6. Scroll all the way up to the top of the file and on line 18 (immediately after the `#include_ "misconst.abi"`) add the following:

```
MAX_FRIEND_LIST_SIZE  = 6;
```

Now as I said, we will be passing the list to a function, so we need to define a user variable type to do so. Since this is going to be a list of unit IDs that are friends with each other, they are going to be integers. Lets call our variable type FriendList as this represents a list of friends. Add the following around line 28 (immediately after the keyword "type").

```
FriendList = integer[MAX_FRIEND_LIST_SIZE];
```

So now, we can use FriendList as a variable type, and each one will be an array of integers of size MAX_FRIEND_LIST_SIZE.

OK, lets actually declare our list. In the variable block, add the following:

```
FriendList FriendGroup1;
```

Note that I don't give an array size. This is because our variable type FriendList is already an array, so we don't need to.

You will also notice that I named the variable itself as FriendGroup1. This is to imply that we could have more lists if we wanted, we'd just need to declare a FriendGroup2, FriendGroup3 etc.

Now to start thinking about the function itself. Once again we'll start with a function shell. So lets think of what information we need to send to this function. Obviously, it will need the list of friends to check, and since the list may not be full, we will also need to tell the function how many are in the list. We won't need any other information since the function will be able to get this information itself. So all we need to do now is decide on a name for this function. Once again, we should use a name that is relevant to what the function is actually doing. As this function checks if a unit is attacking and request help if so, lets call it CheckAndRequestHelp. The parameters, as mentioned, will be the list of friends and the number of friends in the list.

So immediately after our RequestHelpFromFriend function, we setup out new function like this:

```
function CheckAndRequestHelp(FriendList List, integer ListSize);
code
endfunction;
```

Now for the code. The first thing we should do here is verify that the parameter ListSize isn't over our maximum size nor less than one. This would cause an error if it was. So a little error checking will help. I added the following:

```
    if ((ListSize > 0) or (ListSize <= MAX_FRIEND_LIST_SIZE)) then
    endif;
```

We'll put the rest of our code inside this if statement so that it is only executed if the ListSize parameter has a legal value for our list.

Now we need to loop through all the units in this list, We can do this with a for loop, however we are going to need a variable to hold the loop's index. Now as we only need this variable for the loop inside this function, we can define this variable locally to this function. This means that only this function actually knows about this variable. We do this by adding a variable block to the function itself, and declaring the variable there. The variable is going to be an integer, and we'll give it the name of UnitIndex since it will be an index into the list of units in the friend list. Adding this, our function now looks like this:

```
function CheckAndRequestHelp(FriendList List, integer ListSize);
var
    integer UnitIndex;
code
    if ((ListSize > 0) or (ListSize <= MAX_FRIEND_LIST_SIZE)) then
    endif;
endfunction;
```

Now we add the for loop itself. We'll be going from 0 to the size of the list minus one. We use minus one here because we are starting to count from 0. If we didn't, we would actually end up going one past the end of the list. Count them and see. Adding the for loop inside the if statement, we get:

```
    if ((ListSize > 0) or (ListSize <= MAX_FRIEND_LIST_SIZE)) then
        for UnitIndex = 0 to (ListSize - 1) do
        endfor;
    endif;
```

So now we need to check each of these units to see if they are attacking. As with the previous function, we will be using the GetTarget function to see if the unit in question has a target. The IDs of the units are held in the friend list passed into this function as the parameter List and which one in the list we should look at (the index) is held in UnitIndex.

So the call to GetTarget will look like this:

```
GetTarget(List[UnitIndex])
```

And of course, this will be in the condition of an if statement and we want to do the code inside it only if the unit actually has a target (that is, the target ID is not 0). The for loop then becomes:

```
for UnitIndex = 0 to (ListSize - 1) do
    if(GetTarget(List[UnitIndex]) <> 0) then
    endif;
endfor;
```

So now, if we are executing inside this if, we know that this unit has a target and therefore it needs to ask its friends for help. The friends are all the other units in this same list. Therefore, we are going to need another for loop, and hence another local integer variable for index of this new for loop. So in this function's variable block, make a new integer called FriendIndex and the new for loop itself will also go from 0 to the size of the list minus one. Adding both of these our entire function now looks like this:

```
function CheckAndRequestHelp(FriendList List, integer ListSize);
var
    integer UnitIndex;
    integer FriendIndex;
code
    if ((ListSize > 0) or (ListSize <= MAX_FRIEND_LIST_SIZE)) then
        for UnitIndex = 0 to (ListSize - 1) do
            if(GetTarget(List[UnitIndex]) <> 0) then
                for FriendIndex = 0 to (ListSize - 1) do
                endfor;
            endif;
        endfor;
    endif;
endfunction;
```

For each unit in the list indexed by FriendIndex, we need to ask it to come help. We can do this by calling the other function we made, RequestHelpFromFriend. But there is a catch. We are running through the same list we did in the other for loop, so it is possible that the unit indexed by FriendIndex is the same as that indexed by UnitIndex. If this is the case, we don't want to do anything since it would be pointless for the unit to ask itself for help. So inside this second for loop, we will need yet another if statement to make sure the two indexes are not equal (i.e. pointing to the same unit), and if not, call the RequestHelpFromFriend function. This if will look like this:

```
if(UnitIndex <> FriendIndex) then
    RequestHelpFromFriend(List[UnitIndex], List[FriendIndex]);
endif;
```

The now completed function looks like this:

```
function CheckAndRequestHelp(FriendList List, integer ListSize);
var
    integer UnitIndex;
    integer FriendIndex;
code
    if ((ListSize > 0) or (ListSize <= MAX_FRIEND_LIST_SIZE)) then
        for UnitIndex = 0 to (ListSize - 1) do
            if(GetTarget(List[UnitIndex]) <> 0) then
                for FriendIndex = 0 to (ListSize - 1) do
                    if(UnitIndex <> FriendIndex) then
                        RequestHelpFromFriend(List[UnitIndex],
                                                List[FriendIndex]);
                    endif;
                endfor;
            endif;
        endfor;
    endif;
endfunction;
```

(once again, I had to split one of the lines so that it would fit in the document).

Alright! We got that done. We've only got a couple of easy things to do now. Initialize the list, and call the function.

Go back up to the init function, and initialize the list. We've got four units to add to the FriendGroup1 list. We've already declared this list variable, we just need to initialize it. So in the init function add these four lines:

```
    FriendGroup1[0] = GetVehicleID(CLAN_FORCE,0,0);
    FriendGroup1[1] = GetVehicleID(CLAN_FORCE,0,1);
    FriendGroup1[2] = GetVehicleID(CLAN_FORCE,0,2);
    FriendGroup1[3] = GetVehicleID(CLAN_FORCE,0,3);
```

Of course, you should use the appropriate values for your mission.

Now to actually call the function itself. In the main code, before the line `#include_ "ex9_STR.abi"`, add this simple call to our check function:

```
    CheckAndRequestHelp(FriendGroup1, 4);
```

Note how we call this function. We provide the list that it will check and the size of the list. Save the file and try it out.

When entering the map, we see that the entire map is revealed. Remember we did this so we could see what is going on. Pick any of the enemies and try attacking it. You will suddenly see all the others start running towards the one you attacked. If you are fast enough, you can leave and if the one you attack decides to break off, you will see the others run to that location.

The map as you first enter:

Starting an attack on the tanks:

The mechs arrived too late:

Restart the mission and try attacking one of the mechs instead. Suddenly, all the others will run to that location.

Something I didn't add to the code is a check to see if the unit that is being requested for help is alive. If the unit is dead, obviously it won't be coming to anyone's aid. See if you can figure out how to do that.

Making Units Enter The Map

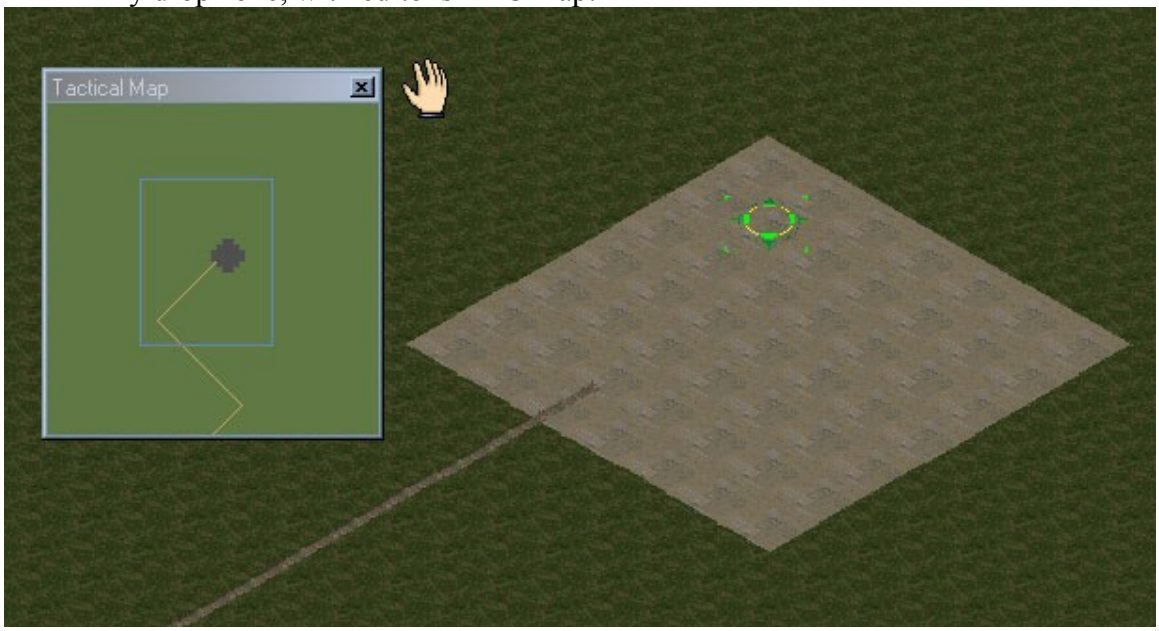
I hope your head isn't spinning too much from the previous section. I admit that it was a bit code heavy if you've never programmed before (those that have likely snored through the whole thing, but that is what happens when you try to write tutorials for a broad audience). Anyway, we are getting back to something a little easier to handle. Getting one or more units to enter the map at a given time.

The example mission we are going to build here will have three mechs entering from the bottom of the map 20 seconds after the mission starts, then proceed to move to some location we designate. We'll put the drop-zone near the end point so that we don't even have to move from there. They'll come to us.

Sound complicated? It's easier than you think; you'll see. Lets get started.

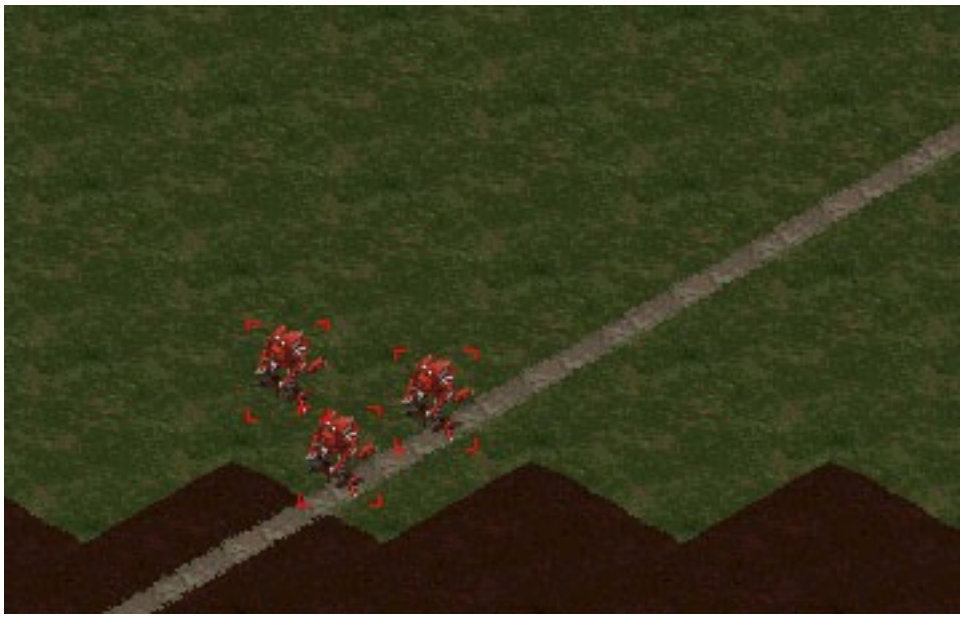
Start up the mission editor and create a simple solo mission map. Somewhere in the middle put a drop-zone. I made the area have some concrete on the ground, but that is just to easily identify the spot on the TAC map. Now from this location, place a road that goes all the way to the south edge of the map. Well OK, it doesn't really matter where it goes or which side of the map you use, I just decided I wanted to use the south side.

My drop zone, with editor's TAC map:



Now on the south side of the map, right at the edge, but *not* in the darkened area, place three mechs. For each of them, make them move along a path that follows the road right up to the drop zone. Leave the mission objective as the default destroy all enemies.

The mechs of south end:



Save the map as ex10 (example 10), and close the editor.

Now as mentioned, we want the mechs to enter the map 20 seconds after the mission starts. But right now the mechs will be present right from the start. The solution is to leverage the trick we discussed about hiding mechs and vehicles. We set the Exists flag to 0 in the .FIT file (actually, it is more like that hiding trick is leveraging this particular feature, but in that trick, we never bother to bring the mech onto the map). So open the .FIT file in a text editor, since we need to set those flags for these mechs.

For each `[PartN]` where our mechs are defined, set the Exists flag to 0 like this:

```
1 Exists = 0
```

The rest of the entries can stay as they are.

Save the .FIT file and open the .ABL file.

Once again, we are going to be in the middle of the map, while the mechs will be entering from the south. So in order to see what is going on, we will once again reveal the entirety of the map. Again, this is not recommended for actual missions, this is just for us to see what is happening in this example.

As before, create an array of three reals to hold the position of the map to be revealed like so:

```
    real[3] Pos;
```

And again, in the init function add the code to reveal the map like so:

```
Pos[0] = 0.0;
Pos[1] = 0.0;
Pos[2] = 0.0;
SetRevealed(INNER_SPHERE, 1500.0, Pos);
```

So all that is left is to make the mechs appear after 20 seconds. We can easily do this by using the GameTime variable. Well check if this variable is greater than 20.0 (and less than 21.0) and if so, we will make the mechs appear on the map. We do this last by calling the function ObjectCreate. This function simply needs the ID of the unit to be created.

Now that we have all the info we need, lets add the code. Around line 133 or so (again this should be right before the `#include_ "ex10_STR.abi"` line), add the following:

```
if((gametime >= 20.0) and (gametime < 21.0)) then
    ObjectCreate(GetVehicleID(CLAN_FORCE,0,0));
    ObjectCreate(GetVehicleID(CLAN_FORCE,0,1));
    ObjectCreate(GetVehicleID(CLAN_FORCE,0,2));
endif;
```

Note how we get the vehicle IDs here. It is the usual method, but as we only need to do the ObjectCreate once, we don't really need to put the IDs in a variable so we simply call the GetVehicleID function as the parameter to the ObjectCreate function.

Save the file and try it out.

We are alone at the start:

20 seconds later however...

Well that was fairly easy to do, how about modifying the code so that they enter one at a time, at 5 second intervals. At the same time, I'll introduce a different way to get the vehicle IDs as well. This second method of getting the vehicle ID is what is usually seen in the scripts created by the editor, and in the official campaigns.

Instead of calling GetVehicleID, there is another function, also in the `miscfunc.abx` library, called VehicleID (note how it doesn't have the word "Get" at the start of its name). The parameters to this function are a little different from those of GetVehicleID. The first parameter is a constant representing which side *and* group the vehicle is on, the second is the member index (it acts exactly like the third parameter of GetVehicleID) and the third parameter I'm still not sure about. Typically it is simply set to 0, as setting it to anything other than 0 will subtract 1 from the value of the ID this function returns (see the function itself in the `miscfunc.abx` library). Why anyone would want that to happen is the great mystery. Leave it as 0.

So what is the constant to use with this new function? Open `misconst.abi` and have a look. Starting at line 138, we see a series of constants like this:

```
    PLAYER_LANCE0               = 1;
    PLAYER_LANCE1               = 2;
```
... and continues on to ...
```
    ALLIED_LANCE0               = 329;
    ALLIED_LANCE1               = 330;
```
... and finally into ...
```
    CLAN_STAR0                  = 165;
    CLAN_STAR1                  = 166;
```

Of course, there are many more entries than I'm showing here.

Each of these represents a combination of Commander and Group from the .FIT file. Since we were using CLAN_FORCE and a group of 0 in the parameters to GetVehicleID, for VehicleID, we'd instead use the constant CLAN_STAR0.

If we had a second group, we'd use CLAN_STAR1 etc. So in the case of our code, we can replace all the GetVehicleID function calls with calls to VehicleID. The three ObjectCreate lines would then look like this:

```
ObjectCreate(VehicleID(CLAN_STAR0,0,0));
ObjectCreate(VehicleID(CLAN_STAR0,1,0));
ObjectCreate(VehicleID(CLAN_STAR0,2,0));
```

Not really very different is it. It doesn't really matter which one you use, more of a preference really. But I wanted to show how this version works so that if you saw it in other code, you wouldn't be at a loss. Just remember that the CLAN_STAR*x*, PLAYER_LANCE*x* and ALLIED_LANCE*x* constants are for VehicleID, and the *xxx*_FORCE constants are for GetVehicleID. You can't use the constants for one function with the other.

Anyway, enough with the digression. Lets make these mechs come into the map one at a time. The first thing to change will be the if statement condition since we will want to be creating the objects more often than between the 20th and 21st second. As I said, we'll make them show up at 5 second intervals. So the first mech will show up at the 20 second mark, the second mech at the 25th second and the third on the 30th second. So change the if so that it can execute through seconds 20 to 31 like this:

```
if((gametime >= 20.0) and (gametime < 31.0)) then
```

Now we need to put each of the calls to ObjectCreate in their own if statements that will check the time and create the mechs when needed. So now our entire if statement becomes:

```
if((gametime >= 20.0) and (gametime < 31.0)) then
    if(gametime >= 20.0) then
        ObjectCreate(VehicleID(CLAN_STAR0,0,0));
    endif;
    if(gametime >= 25.0) then
        ObjectCreate(VehicleID(CLAN_STAR0,1,0));
    endif;
    if(gametime >= 30.0) then
        ObjectCreate(VehicleID(CLAN_STAR0,2,0));
    endif;
endif;
```

Save the file and try it out.

If you look through some of the official campaigns, you will find their code is usually placed in an include file. For instance, Op 2 mission 1 of the original campaign brings in a lot of tanks onto the map. The code that handles this is in the file `2_1CRE.ABI`.

The observant may note that the above code will actually attempt to create the mechs even though they are already created (for instance, on the 27th second, it'll try to re-create the first and second vehicle). This apparently is not a problem, though may cause a little slow down. But since this is exactly what the official campaigns are doing, I wouldn't worry about it too much.

Making Units Leave the Map

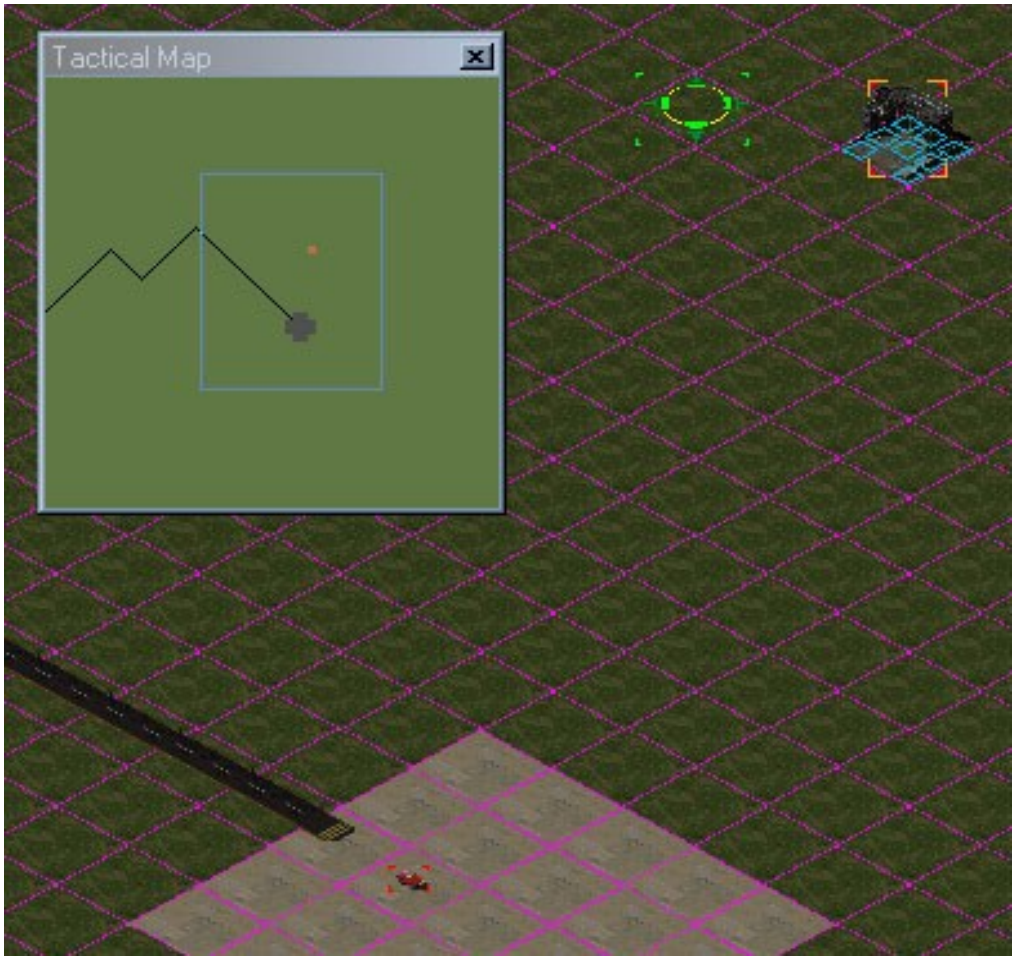
Now that we've seen how to make units enter the map, we should tackle its counter feature. Getting them to leave again. As we'll see, it really isn't any more difficult then getting them to enter.

So for this example, we'll put an APC on the map and have it move along a road and leave the map. We'll modify it later to make capturing the vehicle part of an objective, but lets start by simply getting the vehicle off the map in the first place.

So once again, we load up the editor and create a mission. As we will later be making an extra objective to capture the vehicle that will be leaving, we'll put a building on this map and make the primary objective to capture this building. Now draw a road for the vehicle to travel on (not strictly required of course, but as we'll be giving this vehicle a path to follow, it is easier to make the path follow the road). Place the drop zone far enough away from the vehicle and its path so that we aren't really interfering with it.

Make the path of the APC actually go into the darkened area off the side of the map.

I made this:



Make sure that you write down the coordinates at the end of the path the vehicle is going to follow. We'll need to know them. I had coordinates X = -3540.0, Y = -4148.0. I rounded the numbers a bit, and didn't bother writing down the Z coordinate as it will be ignored anyway.

Save the mission as ex11 (example 11), and open the .ABL file in a text editor (we don't have to change anything in the .FIT file at the moment).

We will once again want to reveal the entire map so that we can see the vehicle leave. So as we've done a few times now, declare an array of three reals like this:

```
real[3] Pos;
```

And now in the init function, we reveal the map like we have before with:

```
Pos[0] = 0.0;
Pos[1] = 0.0;
Pos[2] = 0.0;
SetRevealed(INNER_SPHERE, 1500.0, Pos);
```

Now to get the vehicle to leave. All we need to do is call the function OrderWithdraw. This is another of those functions that work on currently selected objects, in this case, it uses the current pilot. We've seen how to deal with that already, so this is a simple matter. All we need to do is find out when the APC has reached the correct spot.

We can do that with the InArea built-in function. This function takes the ID of the unit to check (it can also be a group ID), an array of three reals that contain the map coordinates for the area to check, the radius, in meters, of the area, and finally, the minimum number of units that must be in this area for this function to return true. This last parameter is needed because as mentioned, we can pass the ID of a group (such as CLAN_STAR0 etc) and it will check that the specified number of units of that group are in the given area. If we give it -1 as a minimum, this means that all units in the group must be present in the given area. Quite a useful function.

So given the above, we'll put our check in our usual spot around line 133 (right above the `#include_ "ex11_STR.ABI"` line). We want to check only a small area (say 30 meters or so), so our if statement becomes:

```
Pos[0] = -3540.0;
Pos[1] = -4148.0;
Pos[2] = 0.0;
if(InArea(GetVehicleID(CLAN_FORCE,0,0), Pos, 30.0, 1)) then
endif;
```

You will notice three things here. First, since InArea wants the coordinates in an array, I reuse the Pos array we declared for revealing the entire map. But as it isn't a static, we must reset the value every time.

The second thing to notice is that I haven't declared a variable to hold the ID of the vehicle either. Therefore, I must call the GetVehicleID function here instead. Of course, I could of used VehicleID as I explained in the previous section, but I prefer GetVehicleID.

Lastly, as the InArea function returns a boolean, we don't need to actually compare it to some other value, we use the return value directly.

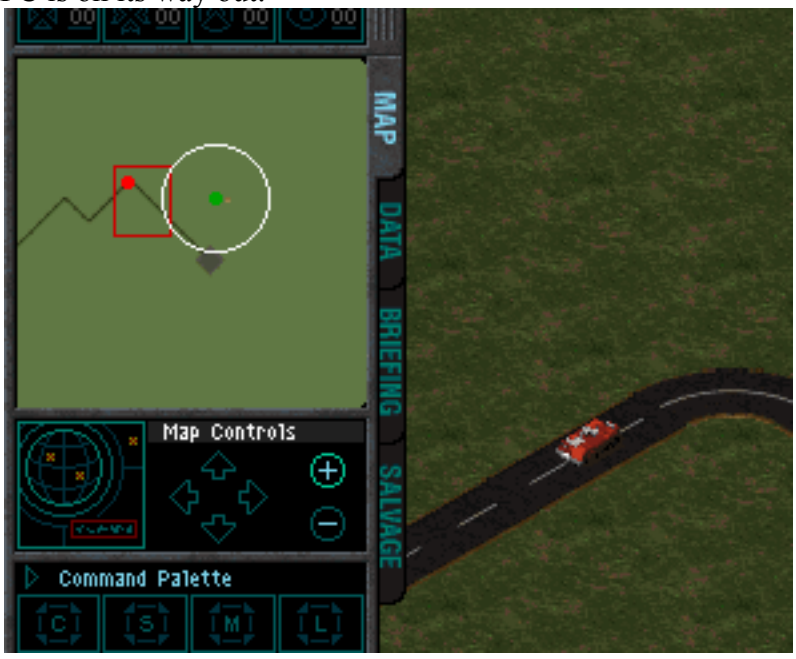
So now we simply need to add the code inside this if statement that will order the vehicle to “withdraw”. As explained, the OrderWithdraw function uses the current pilot. So we first must set the current pilot, then order it to withdraw. We do it like this:

```
SelectWarrior(GetPilotID(VehicleID(CLAN_STAR0,0,0)));
OrderWithdraw;
```

Note how I'm selecting the current pilot. As I don't have a variable with the ID of the APC, I must call either GetVehicleID or VehicleID to get it (I purposely chose VehicleID here to demonstrate that it really doesn't matter which one you use). From that ID I get the ID of the pilot, and finally set that pilot ID as the current pilot. The whole thing is done on a single line of code. Finally, I order the current pilot to withdraw. This will make the APC disappear from the map when off the normal map area (which it now is).

Save the file and try it out.

The APC is on its way out:



And now it's gone:

Now it is time to do something a little more. As I said at the beginning, we will make the player capture the APC. However, we don't want to try to do this in the editor or it will mess with the code we've already written. Further, the editor doesn't normally allow us to use APCs as capturable vehicles. Instead we'll manually add the secondary objective to the mission and make it so we can capture the APC. This will involve a fair bit of editing though because we are doing it all manually.

First, open the .FIT file in a text editor and find the `[Objectives]` heading. Since we are adding a new objective, change the NumObjectives entry to 2 like this:

```
ul NumObjectives = 2
```

Now copy the entire `[Objective0]` section and paste it back as a second objective (don't forget to call it `[Objective1]`). Change the name entry to “Capture APC”, and change the type to 1 since this will be a secondary objective. The new section should now look like:

```
[Objective1]
st Name = "Capture APC"
ul Type = 1
f TimeLeft = -1
ul Status = 0
l Points = 0
```

Now as mentioned, the editor doesn't normally allow APCs to be captured. This is because APCs have a laser weapon on them. However, In operation 2, mission 4 of the expansion campaign there is an APC that the player must capture. If we open the .FIT file for that mission (`mcx0204.fit`). We see that the APC is using a different vehicle profile (the APC is defined in `[Part1]`). If we open this profile (`PV20650.FIT`), we see at the bottom of this profile that the laser weapon has been removed. Since that APC has no weapons, it can be captured without blocking itself. So lets use that profile too. Change the APC's profile in `ex11.fit` to use this same vehicle profile. The ObjectProfile entry should now be:

```
st ObjectProfile        = "PV20650"
```

Save the .FIT and open the .ABL file now. It's time to write some more code.

First, we are going to need a flag to tell us if our new objective has failed (I'll explain why in a bit). Add the following to the variable block for the module:

```
static boolean ObjectiveOneFailed;
```

Again, it doesn't really matter what the variable is called, but giving it a name like OogaBooga doesn't really tell us what the variable is for now does it.

Now as it is a static, we need to initialize it so in the init function, set this variable to false:

```
ObjectiveOneFailed = false;
```

As this new objective is going to be about capturing a vehicle, we need to make the vehicle capturable obviously. We've already seen how to do this, so it should come as no surprise that in the init function we also need the following:

```
SetCaptureable(VehicleID(CLAN_STAR0,0,0), TRUE);
```

Note again how I used the VehicleID function instead of GetVehicleID. Again, use the one that you are most comfortable with. The end result is the same.

Now in the if statement we added that checks and removes the vehicle, set the ObjectiveOneFailed flag to true, also add a check in the if condition to ensure this flag is not true. This will ensure we do this if only once (right now it actually will get called constantly, hence why we use this flag). The if now looks like this:

```
if(InArea(GetVehicleID(CLAN_FORCE,0,0), Pos, 30.0, 1) and
                                  not ObjectiveOneFailed) then
    SelectWarrior(GetPilotID(VehicleID(CLAN_STAR0,0,0)));
    OrderWithdraw;
    ObjectiveOneFailed = true;
endif;
```

Note the use of the NOT logical operator here. This is because we want to execute inside the if statement when our flag is false, but since it is part of an AND operator, this would make the whole AND be false, so we use the NOT to reverse the true/false state of our flag.

Now immediately after this if statement, we'll add our checks to see if the objective has completed or failed. We've already dealt with objective code throughout most of the rest of this document, so it shouldn't be too hard to figure out what to do. Basically we want an if to first check if objective 1 is still INCOMPLETE, if so we then check our flag to see if it is set to true. If it is, we set the objective to FAILED (and play Betty's audio clip to tell the player so) since the APC is now gone from the map. We also want to check to see if the APC has been captured. If so, then we want to set the objective to SUCCESS (and play the correct Betty clip). See if you can work out the code. It should be something like this:

```
if(CheckObjectiveStatus(1) == INCOMPLETE) then
    if(ObjectiveOneFailed) then
        SetObjectiveStatus(1, FAILED);
        PlayBetty(BETTY_SECONDARY_FAILED);
    endif;
    if ((IsCaptured(VehicleID(CLAN_STAR0,0,0)) == 1)) then
        SetObjectiveStatus(1, SUCCESS);
        PlayBetty(BETTY_OBJECTIVE_COMPLETE);
    endif;
endif;
```

Finally, in the code that forces the objectives to FAILED status if all the player units have died, add a call to make objective 1 fail as well (you should already know how to do that by now). That's it. Save the file and try it out.

We can capture the APC now:

Note how the APC is still red to us. This is of course because we haven't bothered to set it to our side (see the section on making vehicles and mechs as salvage).

When the APC has left the map:

And when we let the APC leave the map, our objective fails as it should. In both cases, Betty says the appropriate message.

If we wanted to make this a primary objective instead of a secondary one, we would of course need to change the objective type in the .FIT file, but we would also need to change a couple of things in the code so that if objective 1 failed, it would force the mission to fail as well. I'll leave that as an exercise for the reader.

Giving Artillery Strikes As Salvage

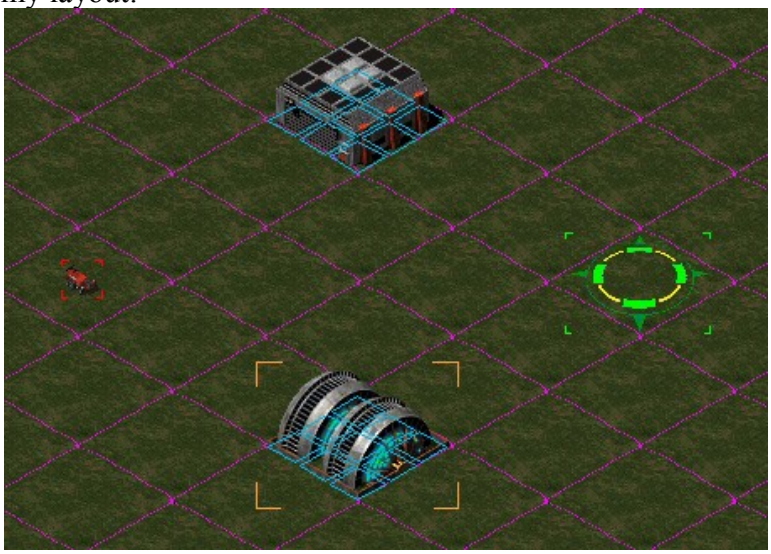
We've seen how to give salvage items to the player via buildings and vehicles, as well as giving mechs out as salvage as well. But there is still something we could give the player that we haven't seen yet. Namely artillery strikes, camera drones and sensor probes.

As these aren't standard items that can be placed on an object to be picked up, nor are they objects seen on the map itself, we will need to rely on functions and hand out the items to the player manually.

At this point some of you may be asking “Why bother?”. Yes, we can specify the artillery strikes etc in the mission's .FIT file, but that will be a fixed number for the duration of the mission. What we are after here is the ability to add and/or change the number of strikes the player has within the mission, during the course of the mission.

Open the editor and create a solo mission. The objective of this mission is not really important, but make it something like capturing a specific building or reaching an extraction point. We just don't want the mission to end before we have a chance to test things. On this map, place one or two buildings or objects that we will capture in order to give out the artillery strikes. Place a Swiftwind (or whatever) somewhere so an enemy is present on the map and also place a drop zone. Do not add any artillery strikes to this mission. We'll be giving those out when the building is captured. Save this mission as `ex12` (example 12).

This is my layout:



The main power object will serve as the mission's objective. The other building will be the one to capture to receive the artillery strikes.

Open `ex12.abl` in a text editor and set things up so that the buildings or objects that will give out the artillery are capturable, but do not have any salvage items on them (well OK, we could have salvage items on them too if we wanted, but lets keep things separate for now).

We'll need two variables for this example. The first is just a static integer to hold the ID of the building that will give out the artillery. Since we already need this to make the building capturable in the first place, you likely already have that setup (I hope). The second variable is a static boolean that will be used to flag if we've given out the artillery strikes (we don't want to be repeatedly handing these out). In my code, I added the following:

```
static integer ArtyBuilding;
static boolean GotBuilding;
```

And in the init function, I get the ID of the building, set it as capturable, then set the flag to false like this:

```
ArtyBuilding = GetTerrainObjectPartID(21,4);
SetCaptureable(ArtyBuilding, TRUE);
GotBuilding = FALSE;
```

Nothing we haven't already done right?

So now in the main code, we'll check to see if the building has been set to the player's side and also if our flag is still false. If both conditions are true, then obviously the player just captured the building so this will be where we give out the artillery. But I haven't mentioned what function we need to do this.

Looking at the functions listed in part 2 of the guide, we find the function AddStrikes. This function takes a Commander ID, the type of artillery strike to add, and the number of them to be added to the player.

The first parameter to this function is a type of ID that we haven't much dealt with yet; at least not directly. Although the Commander ID is similar to the alignment or side, it does not follow the same actual values. Therefore, we can't use the PLAYER_FORCE, CLAN_FORCE and ALLIED_FORCE constants for it, nor can we use the INNER_SPHERE, CLAN, and NEUTRAL constants. What we need are the values that we've seen from the unit group lists from the .FIT file. That is, 0 is the player, 1 is the enemy, and 2 is the allied side. Since we wish to add the artillery strikes to the player side, we need a Commander ID of 0.

The second parameter is the type of artillery strike to be added. Here, 0 means small strikes, 1 means large strikes, 2 is for sensor drones, and finally 3 is for camera drones.

The last parameter is merely how many strikes we want to add. I decided on five.

I'm going to be a little fancy here and add a call to play Betty's audio clip that says “Enemy Resources Captured”. This clip doesn't have a constant defined for it in `misconst.abi`, but from the table in the function description in part 2 of the guide, we see that it is clip 6.

So the entire if statement I added becomes:

```
if ((ObjectSide(ArtyBuilding) == INNER_SPHERE) and
                                  not GotBuilding) then
    GotBuilding = TRUE;
    AddStrikes(0, 0, 5);
    PlayBetty(6);
endif;
```

Save the file and try it out.

At the start of the mission, we have no artillery:

But after capturing the building, we now have 5 small strikes:

And Betty says her “Enemy Resources Captured” line right on cue.

If a second building had been set up to also give five more artillery strikes, then we would end up with 10 of them. This is because the AddStrikes function does exactly that. It adds the given number of artillery strikes. If instead we simply wanted to set a given number, we could use the SetStrikes function. This version simply sets the number of artillery strikes directly. So if we had 10 strikes and then did a SetStrikes with only 1, the net result would be the player would have only one artillery strike.

There is also a GetStrikes function that returns the number of artillery strikes of the given type that the side has. With this and the SetStrikes function, we could also simulate removing strikes (a SubtractStrikes function if you like) by first getting the current number the player has, subtracting the desired number, then calling SetStrikes to set the new value. I'll leave this last as an exercise for the reader.

Creating Pre-Damaged Buildings

One thing that we often see in the original and expansion campaigns are buildings that are already partially damaged. We see this mostly with mech repair bays because the less health the repair bay has, the less amount of repairs the bay can give out. This makes for a convenient way to limit the amount of repair a player can get in the mission. But this trick could also be used to create weak buildings for enemy units to attack and destroy in a short amount of time.

As the method to do this is very simple, I won't bother creating an example mission.

The trick simply relies on the function SetObjectDamage. This function takes the ID of the terrain object we wish to damage, and the amount of damage, as a percentage, that is to be applied. Note that it is a percentage of **damage** to the object and not a percentage of health. Therefore, a value of 100 would mean to destroy the object. It should also be noted that once the damage has been applied, it cannot be recovered as there is no function to repair a building.

So if we had a mech repair bay that we wanted to only have a quarter of its normal health (in other words, only 25% of its repair ability), and the ID of this repair bay was in a variable called MechRepairBayID, then we would pre-damage the repair bay in the .ABL's init function like this:

```
SetObjectDamage(MechRepairBayID, 75);
```

Again, the 75 here is the percentage of damage to be applied to the object and thereby only leaving 25% of its health.

That's all there is to it. If however you still want an example of the use of this function, see the next section.

Getting IDs of Terrain Objects Using Map Coordinates

This section came into being because I was once asked if it was possible to have an enemy unit start firing at a section of bridge.

Now the first thing to cross my mind was to simply do what we do for any other terrain object. That is, get the block and vertex number from the editor, call GetTerrainObjectPartID with those values and then have the enemy unit in question fire at the object whose ID we just got.

The only hitch with the above is that the editor doesn't give a block and vertex number for bridges. This is a problem.

Well I got curious. If we play a mission with bridges, we can clearly see that the bridge pieces all have health bars and so internally seem to be treated the same as any other building. If this is correct, we should be able to get the ID of the object itself, and then do whatever we wish to the object. The trouble is getting the block and vertex numbers.

So I fired up the editor and started toying around to see how exactly these block and vertex numbers work in relation to the map itself. I managed to figure out a formula that, given some map coordinates, can calculate a block and vertex number we can use with GetTerrainObjectPartID and get the correct ID.

Now I should mention here that this is a bit of a hack, and so should only be used for objects whose block and vertex numbers aren't given by the editor. Further, only one terrain object should be on a given tile or the function could return the ID of the wrong object.

The first thing we need to know is how big each tile on the map is. This is fairly easy to get since we just need to put the mouse cursor at one corner of a map tile and write down the coordinates, then do the same for the opposite corner of the same tile. Subtracting the numbers tells us that each tile is 128 map units wide and high.

Next, we need to figure out how big a “Block” is in terms of map tiles. We do this by simply placing an object on the map and moving it around until it has a vertex value of 0, then moving it in a direction to find when the “Block” value changes. Doing this tells us that a “Block” is 20 tiles wide by 20 high.

Now we move our object around in the block to figure out how the vertex numbers change within it. Doing this tells us that the vertex number starts in the top left corner of the block and increases by one for each tile as we move to the right. Once completely on the right, we go to the next line of tiles on the left and continue, like we would reading this text.

Lastly, we move our object until it is on the tile that is at map coordinates of X = 0.0 and Y = 0.0, and we write down which block number this is (Block 15) and also note that moving up or down to the next block changes the block number by 6.

We now have all the information we need to convert from map coordinates to a block and vertex number. First we take the X map coordinates, divide it by 128, then again by 20 to get the block coordinate and then add 15 to it to get the base block value.

We then take the Y coordinate, also divide it by 128 then again by 20 and multiply this number by -6 (we use a negative number because the block values decrease as the Y coordinates increase). This value is then added to the base block value to get the actual block number. We also need to do a little extra footwork if the coordinates are negative instead of positive, but that is a little extra detail that I won't go into here, but is taken into account in the final version presented.

For the vertex number, we again divide the coordinates by 128, but then do a modulo 20 on the number instead of dividing by 20. This will give us the tile position inside the block. We then simply multiply the Y part by 20 and add the X part and we have the vertex number. We again need to do a little extra when the map coordinates are negative, but again this is merely a little extra detail.

So here I present a function that will do all of the above calculations, call GetTerrainObjectPartID with these calculated block and vertex numbers and return the ID back to the caller.

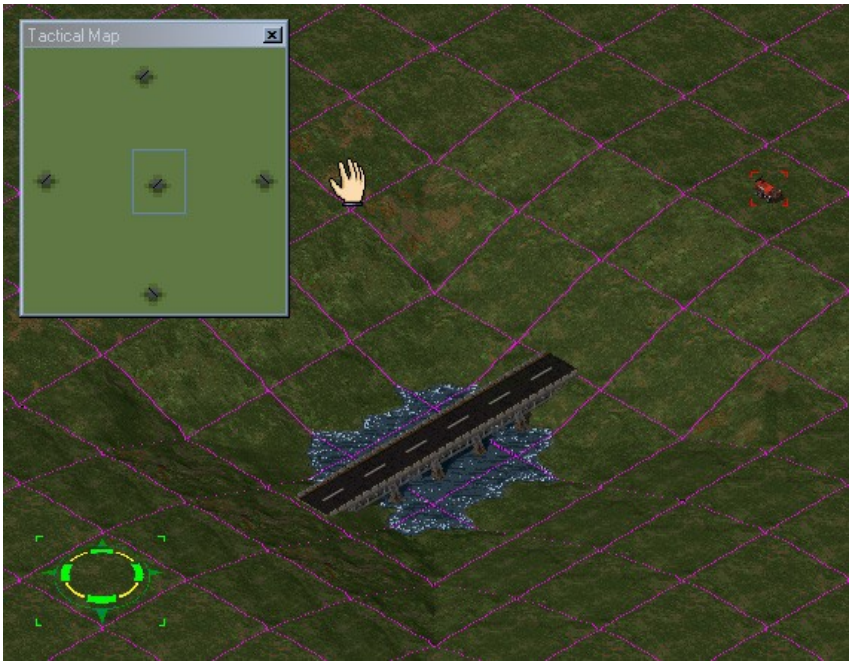
```
function GetObjIDFromCoords(real xcoord, real ycoord) : integer;
var
    integer blk;
    integer vert;
    integer itmp;
    integer itmp2;
    integer xneg;
    integer yneg;
    integer mult;

code
    xneg = 0;
    if (xcoord < 0.0) then
        xneg = -1;
    endif;

    yneg = 0;
    mult = -6;
    if(ycoord < 0.0) then
        yneg = 1;
        mult = 6;
    endif;

    ycoord = abs(ycoord);

    // Compute the block number for these coordinates
    itmp = trunc(xcoord / 2560.0) + xneg + 15;
    itmp2 = mult * (trunc(ycoord / 2560.0) + yneg);
    blk = itmp + itmp2;
```

```
    // Now compute the vertex number
    itmp = trunc(ycoord / 128.0) mod 20;
    if (yneg <> 1) then
        itmp = 19 - itmp;
    endif;

    itmp2 = trunc(abs(xcoord) / 128.0) mod 20;
    if (xneg == -1) then
        itmp2 = 19 - itmp2;
    endif;

    vert = (itmp * 20) + itmp2;

    // Get the ID of the object at this block and vertex (if any) and return it.
    return (GetTerrainObjectPartID(blk,vert));
endfunction;
```

Now that the function itself is given, lets use it in an example to pre-damage a section of a bridge.

Open the editor and start a new solo mission. Lower the ground in a few places on the map and draw a bridge across these gaps. Hover the mouse over at least one of the bridge pieces and write down its map coordinates. It doesn't need to be perfectly centered on the tile, but don't go choosing a spot at the edges of a map tile. Choose a piece of bridge that you can easily find when playing so that you will know it is affecting the correct section of bridge. Place a Swiftwind on the map and leave the mission objective as “Destroy All Enemies”. Place a drop zone somewhere and save the map as ex13 (example 13).

Here's what I did:



The five bridges are placed so that the X and Y coordinates alternate positive and negative. The middle bridge is nearest to the middle of the map as possible.

Open `ex13.abl` in your text editor, copy the above function and paste it into the script right before the init function (if we placed it afterwards, we would not be able to call it from the init function).

In the init function itself, reveal the entire map in the usual way, and now, for each bridge piece you want to pre-damage, we call the SetObjectDamage function (as described in the previous section) but for the object ID, we make a call to the GetObjIDFromCoords function we just pasted in with the map coordinates we wrote down. I added this to my init function:

```
Pos[0] = 0.0;
Pos[1] = 0.0;
Pos[2] = 0.0;
SetRevealed(INNER_SPHERE, 1500.0, Pos);

SetObjectDamage(GetObjIDFromCoords(0.1, -0.1), 75);
SetObjectDamage(GetObjIDFromCoords(2995.0, 3148.0), 75);
SetObjectDamage(GetObjIDFromCoords(3003.0, -3251.0), 75);
SetObjectDamage(GetObjIDFromCoords(-3404.0, 2629.0), 75);
SetObjectDamage(GetObjIDFromCoords(-3287.0, -3138.0), 75);
```

Pos is declared as a local variable to the init function. Note how all of these bridges are being damaged by 75%, so they will only have a quarter of their full health.

Save the file and try the mission out. You should find that all the bridges specified have been damaged appropriately. We could of course have used the ID to have an enemy attack the bridge, or some other activity.

Here's what my middle bridge looks like. The other bridges are similarly affected as they should be.

Final Words

I hope this document has provided valuable information and isn't too overwhelming. I admit that some of the information here can be a little hard to digest. Further, much of the information is somewhat hidden in the document. Though it wasn't my intent to hide it, but as I had to give simple examples in order to demonstrate the topics, the implications of some of what is given may not be immediately apparent and hence requires the reader to investigate a little further.

For example, the section on creating salvage for the player to capture explains that you cannot remove individual items from the player's salvage list, but you can remove all items that was salvaged from a given object. One implication of this that isn't explained specifically is that we can actually take advantage of this by removing any salvage retrieved from certain buildings if some event happens in game. This could be useful for certain situations. Therefore, it is important to try to go beyond just what is explained to see what else can be done with what is shown here.

Of course, this document cannot hope to cover all possible topics. I may release a few supplements at a later time when new topics need to be addressed and are discovered. But that will be for the future.

As with part one of the guide series, the rest of this document is the appendices with file listings for the examples covered throughout the document.

-Cmunsta

Appendix 1: File Listing for Hiding and Revealing Objectives

Listing of ex2.abl

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

module MissionBrain_ex2 : integer;

     //---------------------------------------------------------
     // Mission Name: ex2
     // Created: 9/21/2008
     //---------------------------------------------------------

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

          #include_ "misconst.abi"


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


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

            #include_ "ex2_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;
```

```
//--------------------------------------------------------------------
//
//     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_ "ex2_INIT.ABI"
            SetObjectiveStatus(1, 3);
endfunction;

//--------------------------------------------------------------------
//
//     Main Code
//
//--------------------------------------------------------------------
      code

            //-------------------------------------------------
            // 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_ "ex2_STR.ABI"
```

```
//--------------------------------------
// Lookout Points
//--------------------------------------
#include_ "ex2_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;
      if (checkObjectiveStatus(1) == INCOMPLETE) then
            setObjectiveStatus(1, FAILED);
      endif;
endif;

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

if (checkObjectiveStatus(0) == INCOMPLETE) then

     if (isDead(CLAN_FORCE)) then
           if (CheckObjectiveStatus(1) == 3) then
               SetObjectiveStatus(1,INCOMPLETE);
           endif;
           SetObjectiveStatus(0,SUCCESS);
           playBetty(BETTY_OBJECTIVE_COMPLETE);
           VictoryLevel = VictoryLevel + 1;
           Objective_0_Decided = TRUE;
           if (VictoryLevel < 2) then
                 PlayDigitalMusic(OBJECTIVE_SUCCESS_MUSIC);
           endif;
     else
           if (gettimeleft == 0.0) then  // supposedly ...
                 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 ((checkObjectiveStatus(1) == INCOMPLETE)
          or (checkObjectiveStatus(1) == 3)) then

     x = 71336;
```

```
            if ((IsCaptured(x) == 1) AND ( objectSide(x) ==  INNER_SPHERE))  then
                  SetObjectiveStatus(1,SUCCESS);
                  playBetty(BETTY_OBJECTIVE_COMPLETE);
                  VictoryLevel = VictoryLevel + 1;
                  Objective_1_Decided = TRUE;
                  if (VictoryLevel < 2) then
                        PlayDigitalMusic(OBJECTIVE_SUCCESS_MUSIC);
                  endif;
            else
                  if ((gettimeleft == 0.0) OR (getObjectDamage(x) > 99)) then
                        SetObjectiveStatus(1,FAILED);
                        playBetty(BETTY_CANNOT_COMP_OBJ);
                        Objective_1_Decided = TRUE;
                  endif;
            endif;

      else
            if (NOT ExitTimerSet) then
                  //if Failed primary Objective, fail mission
                  if (checkObjectiveStatus(1) == FAILED) then
                        setTimer(EXIT_TIMER,2.0);
                        ExitTimerSet = TRUE;
                  endif;
            endif;
      endif;

      if ((NOT ExitTimerSet) AND (Objective_0_Decided)
                        AND (Objective_1_Decided)) then
            setTimer(EXIT_TIMER,2.0);
            ExitTimerSet = TRUE;
      endif;

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

      if ((ExitTimerSet) AND (CheckTimer(EXIT_TIMER) == 0.0)) then
            if (VictoryLevel >= 2) then
                  ScenarioResult = PLAYER_WIN_BIG;
                  PlayDigitalMusic(VICTORY_MUSIC);
            else
                  ScenarioResult = PLAYER_LOST_BIG;
                  PlayDigitalMusic(DEFEAT_MUSIC);
            endif;
      endif;

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

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

Appendix 2: File Listing for Timed Missions and Objectives

Listing of ex4.abl

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

module MissionBrain_ex4 : integer;

     //--------------------------------------------------------
     // Mission Name: ex4
     // Created: 9/22/2008
     //--------------------------------------------------------

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

          #include_ "misconst.abi"


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


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

            #include_ "ex4_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      StartedObjective0Timer;
            static boolean      StartedBettyWarning;
```

```
//-------------------------------------------------------------------
//
//     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_ "ex4_INIT.ABI"
            StartedObjective0Timer = false;
            StartedBettyWarning = false;
endfunction;

//-------------------------------------------------------------------
//
//     Main Code
//
//-------------------------------------------------------------------
      code

            //-------------------------------------------------
            // 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;
```

```
if(StartedObjective0Timer and not StartedBettyWarning) then
    if( (CheckObjectiveTimer(0) >= 30.0) and
                  (CheckObjectiveTimer(0) <= 31.0)) then
        PlayBetty(BETTY_THIRTY_LEFT);
        StartedBettyWarning = true;
    endif;
endif;

//----------------------------------------
// Structure
//----------------------------------------
#include_ "ex4_STR.ABI"

//----------------------------------------
// Lookout Points
//----------------------------------------
#include_ "ex4_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;
      if (checkObjectiveStatus(1) == INCOMPLETE) then
            setObjectiveStatus(1, FAILED);
      endif;
endif;

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

if (checkObjectiveStatus(0) == INCOMPLETE) then

     x = 71656;
     if ((IsCaptured(x) == 1) AND ( objectSide(x) == INNER_SPHERE)) then
           SetObjectiveStatus(0,SUCCESS);
           SetObjectiveTimer(0, 0.0);
           playBetty(BETTY_OBJECTIVE_COMPLETE);
           VictoryLevel = VictoryLevel + 1;
           Objective_0_Decided = TRUE;
           if (VictoryLevel < 1) then
                 PlayDigitalMusic(OBJECTIVE_SUCCESS_MUSIC);
           endif;
     else
           if ((StartedObjective0Timer and (CheckObjectiveTimer(0) == 0.0))
                           OR (getObjectDamage(x) > 99)) then
                 SetObjectiveStatus(0,FAILED);
                 SetObjectiveTimer(0, 0.0);
                 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 (checkObjectiveStatus(1) == INCOMPLETE) then

                 x = 55200;
                 if ((IsCaptured(x) == 1) AND ( objectSide(x) == INNER_SPHERE)) then
                        if(CheckObjectiveStatus(0) == INCOMPLETE) then
                            StartedObjective0Timer = true;
                            SetObjectiveTimer(0, (GameTime + 45.0));
                        endif;
                        SetObjectiveTimer(1, 0.0);
                        SetObjectiveStatus(1,SUCCESS);
                        playBetty(BETTY_OBJECTIVE_COMPLETE);
                        SetObjectiveStatus(1,SUCCESS);
                        Objective_1_Decided = TRUE;
                        if (VictoryLevel < 1) then
                              PlayDigitalMusic(OBJECTIVE_SUCCESS_MUSIC);
                        endif;
                 else
                        if ((CheckObjectiveTimer(1) == 0.0) OR
                                          (getObjectDamage(x) > 99)) then
                              SetObjectiveStatus(1,FAILED);
                              playBetty(BETTY_SECONDARY_FAILED);
                              Objective_1_Decided = TRUE;
                              PlayDigitalMusic(OBJECTIVE_FAIL_MUSIC);
                        endif;
                 endif;

            endif;

            if ((NOT ExitTimerSet) AND (Objective_0_Decided) AND
                                (Objective_1_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.
//*****************************************************************
```

Appendix 3: File Listing for Creating Mechs with No Crew/Pilots

Listing of ex5.abl

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

module MissionBrain_ex5 : integer;

     //---------------------------------------------------------
     // Mission Name: ex5
     // Created: 9/22/2008
     //---------------------------------------------------------

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

          #include_ "misconst.abi"


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


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

            #include_ "ex5_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;
```

```
//--------------------------------------------------------------------
//
//     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_ "ex5_INIT.ABI"

endfunction;

//--------------------------------------------------------------------
//
//     Main Code
//
//--------------------------------------------------------------------
      code

            //-------------------------------------------------
            // 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;

            if((gametime >= 3.0) and (gametime <= 3.9)) then
                SetObjectActive(GetVehicleID(CLAN_FORCE, 0, 0), TRUE);
            endif;
```

```
//--------------------------------------
// Structure
//--------------------------------------
#include_ "ex5_STR.ABI"

//--------------------------------------
// Lookout Points
//--------------------------------------
#include_ "ex5_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

     x = 71328;
     if ((IsCaptured(x) == 1) AND ( objectSide(x) ==  INNER_SPHERE)) 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) OR (getObjectDamage(x) > 99)) 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.
//*****************************************************************
```

Appendix 4: File Listing for Creating Salvage & Capturing Buildings

Listing of ex6.abl

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

module MissionBrain_ex6 : integer;

     //---------------------------------------------------------
     // Mission Name: ex6
     // Created: 9/24/2008
     //---------------------------------------------------------

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

          #include_ "misconst.abi"


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


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

            #include_ "ex6_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 integer SalvageObj1;
            static integer SalvageObj2;
            static integer SalvageObj3;
//--------------------------------------------------------------------
//
//     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_ "ex6_INIT.ABI"

            SalvageObj1 = GetTerrainObjectPartID(21,26);
            SalvageObj2 = GetTerrainObjectPartID(15,388);
            SalvageObj3 = GetTerrainObjectPartID(15,350);

            SetCaptureable(SalvageObj1, TRUE);
            // SetCaptureable(SalvageObj2, TRUE);   // Commented out
            SetCaptureable(SalvageObj3, TRUE);

            SetSalvage(SalvageObj1, 154, 2);      // 2x Clan ER-PPC
            SetSalvage(SalvageObj1, 139, 3);      // 3x Large X Pulse Laser

            SetSalvage(SalvageObj2, 112, 1);      // Clan Ultra AC/20

            SetSalvage(SalvageObj3, 153, 4);      // 4x Clan Pulse Laser
            SetSalvage(SalvageObj3, 155, 1);      // Clan Heavy Flamer
            SetSalvage(SalvageObj3, 98, 1);       // Rail Gun

endfunction;

//--------------------------------------------------------------------
//
//     Main Code
//
//--------------------------------------------------------------------
      code

            //-----------------------------------------------
            // 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;

if((IsCaptured(SalvageObj1) == 1) and (
               ObjectSide(SalvageObj1) == INNER_SPHERE)) then
    ObjectChangeSides(SalvageObj2, INNER_SPHERE);
    SetCaptured(SalvageObj2);
    SetSalvageStatus(SalvageObj2, TRUE);
endif;

//---------------------------------------
// Structure
//---------------------------------------
#include_ "ex6_STR.ABI"

//---------------------------------------
// Lookout Points
//---------------------------------------
#include_ "ex6_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  // supposedly will not ...
                        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.
//*****************************************************************
```

Appendix 5: File Listing for Capturing and Salvaging Vehicles and Mechs

Listing of ex7.abl:

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

module MissionBrain_ex7 : integer;

     //-------------------------------------------------------
     // Mission Name: ex7
     // Created: 9/24/2008
     //-------------------------------------------------------

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

          #include_ "misconst.abi"


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


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

            #include_ "ex7_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 integer MobileHQID;
            static integer MechID;

//--------------------------------------------------------------------
//
//     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_ "ex7_INIT.ABI"

            MobileHQID = GetVehicleID(CLAN_FORCE,0,0);

            SetCaptureable(MobileHQID, TRUE);

            SetSalvage(MobileHQID, 153, 2);

            MechID = GetVehicleID(CLAN_FORCE,0,1);

endfunction;

//--------------------------------------------------------------------
//
//     Main Code
//
//--------------------------------------------------------------------
      code

            //-------------------------------------------------
            // 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;

if((IsCaptured(MobileHQID) == 1) and
              (ObjectSide(MobileHQID) <> INNER_SPHERE)) then
    ObjectChangeSides(MobileHQID, INNER_SPHERE);

    SetCaptured(MechID);
    SetSalvageStatus(MechID, TRUE);
    ObjectChangeSides(MechID, INNER_SPHERE);
    SetObjectActive(MechID, FALSE);
endif;

//----------------------------------------
// Structure
//----------------------------------------
#include_ "ex7_STR.ABI"

//----------------------------------------
// Lookout Points
//----------------------------------------
#include_ "ex7_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;
      if (checkObjectiveStatus(1) == INCOMPLETE) then
            setObjectiveStatus(1, FAILED);
      endif;
endif;

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

if (checkObjectiveStatus(0) == INCOMPLETE) then

     x = 54552;
     if ((IsCaptured(x) == 1) AND ( objectSide(x) ==  INNER_SPHERE))  then
           SetObjectiveStatus(0,SUCCESS);
           playBetty(BETTY_OBJECTIVE_COMPLETE);
           VictoryLevel = VictoryLevel + 1;
           Objective_0_Decided = TRUE;
           if (VictoryLevel < 2) then
                 PlayDigitalMusic(OBJECTIVE_SUCCESS_MUSIC);
           endif;
```

```
      else
            if ((gettimeleft == 0.0) OR (getObjectDamage(x) > 99)) 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 (checkObjectiveStatus(1) == INCOMPLETE) then

      x = 71456;
      if ((IsCaptured(x) == 1) AND ( objectSide(x) ==  INNER_SPHERE))  then
            SetObjectiveStatus(1,SUCCESS);
            playBetty(BETTY_OBJECTIVE_COMPLETE);
            VictoryLevel = VictoryLevel + 1;
            Objective_1_Decided = TRUE;
            if (VictoryLevel < 2) then
                  PlayDigitalMusic(OBJECTIVE_SUCCESS_MUSIC);
            endif;
      else
            if ((gettimeleft == 0.0) OR (getObjectDamage(x) > 99)) then
                  SetObjectiveStatus(1,FAILED);
                  playBetty(BETTY_CANNOT_COMP_OBJ);
                  Objective_1_Decided = TRUE;
            endif;
      endif;

else
      if (NOT ExitTimerSet) then
            //if Failed primary Objective, fail mission
            if (checkObjectiveStatus(1) == FAILED) then
                  setTimer(EXIT_TIMER,2.0);
                  ExitTimerSet = TRUE;
            endif;
      endif;
endif;

  if ((NOT ExitTimerSet) AND (Objective_0_Decided) AND (Objective_1_Decided))
then
      setTimer(EXIT_TIMER,2.0);
      ExitTimerSet = TRUE;
endif;

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

if ((ExitTimerSet) AND (CheckTimer(EXIT_TIMER) == 0.0)) then
      if (VictoryLevel >= 2) then
            ScenarioResult = PLAYER_WIN_BIG;
            PlayDigitalMusic(VICTORY_MUSIC);
      else
            ScenarioResult = PLAYER_LOST_BIG;
            PlayDigitalMusic(DEFEAT_MUSIC);
      endif;
endif;
```

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

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

Appendix 6: File Listings for Giving In-Mission Drop Groups

Listing of ex8.fit

```
FITini

// ------------------------------------------------------------------------
//
//      Mission Name: ex8  Creation Date: 9/25/2008
//
// ------------------------------------------------------------------------


[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 = "ex8"
st TacMapGifName = "ex8"

[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 = "ex8"

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

[Campaign]
st BriefingFile = "ex8.txt"
st MapFile = "ex8"
l MaxTonnage = 990
l NumDropZones = 1

[DropZone0]
l NumSlots = 4

f PositionX = 1088
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 = "Capture Buildings"
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 = 3
uc CaptureChance = 0
st BrainParameterFile = "ex8Params"
[Warrior1]
st Profile = "PMW00066"
st Brain = "pbrain"

[Warrior2]
st Profile = "PCREWB"
st Brain = "pbrain"

[Warrior3]
st Profile = "PMW00025"
st Brain = "DredAttack01"

[Parts]
ul NumParts = 3
b AlliedTeam = True

[Part1]
ul ObjectNumber         = 51
ul ControlType          = 2                 // player = 1, ai = 2, net = 3
b PlayerPart            = True
ul ControlDataType      = 1                 // mech 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        = "PM109300"        // AS7-J
ul Pilot                = 1
f PositionX             =  960              // Starting position in world.
f PositionY             =  576
f PositionZ             = -1.0              // Elevation automatically set by terrain.
f Rotation              =  45               // Rotation of Object in Degrees
ul Gesture              = 2           // Initial Sprite Gesture -- Determines ...
f Velocity              = 0.0         // Initial Velocity -- Start velocity ...
l Active                =  0
l Exists                =  1

[Part2]
ul ObjectNumber         = 414
ul ControlType          = 2                 // player = 1, ai = 2, net = 3
b PlayerPart            = True
ul ControlDataType      = 2                 // 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        = "PV20200"        // Mobile HQ
ul Pilot                = 2
f PositionX             =  1216             // Starting position in world.
f PositionY             =  320
f PositionZ             = -1.0              // Elevation automatically set by terrain.
f Rotation              =  45               // Rotation of Object in Degrees
ul Gesture              = 2               // Initial Sprite Gesture -- Determines ...
f Velocity              = 0.0             // Initial Velocity -- Start velocity ...
```

```
l Active                =  0
l Exists                =  1

[Part3]
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                = 3
f PositionX             =  576              // 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...
f Velocity              = 0.0               // Initial Velocity -- Start velocity...
l Active                =  1
l Exists                =  1

[Teams]
b AlliedTeam = True

[Commander1Group:0]
l[12] Mates             = 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0

[Commander0Group:1]
l[12] Mates             = 1, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0

FITend
```

Listing of ex8.abl

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

module MissionBrain_ex8 : integer;

     //-------------------------------------------------------
     // Mission Name: ex8
     // Created: 9/25/2008
     //-------------------------------------------------------

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

          #include_ "misconst.abi"


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


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

```
            #include_ "ex8_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 integer Atlas;
            static integer MobileHQ;

//--------------------------------------------------------------------
//
//     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_ "ex8_INIT.ABI"

            Atlas = GetVehicleID(PLAYER_FORCE,1,0);
            MobileHQ = GetVehicleID(PLAYER_FORCE,1,1);
endfunction;

//--------------------------------------------------------------------
```

```
//
//     Main Code
//
//-------------------------------------------------------------------
      code

            //-----------------------------------------------
            // 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;

            if (ClanForceDead and (GetObjectActive(Atlas) == 0)) then
                SetObjectActive(Atlas, TRUE);
                SetObjectActive(MobileHQ, TRUE);
            endif;

            //---------------------------------------
            // Structure
            //---------------------------------------
            #include_ "ex8_STR.ABI"

            //---------------------------------------
            // Lookout Points
            //---------------------------------------
            #include_ "ex8_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

                  x = 55168;
                  if ((IsCaptured(x) == 1) AND ( objectSide(x) ==  INNER_SPHERE)) 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) OR (getObjectDamage(x) > 99)) 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.
//****************************************************************
```

Appendix 7: File Listings for Controlling Units from the Mission ABL

Listing of ex9.abl

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

module MissionBrain_ex9 : integer;

     //---------------------------------------------------------
     // Mission Name: ex9
     // Created: 9/25/2008
     //---------------------------------------------------------

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

        #include_ "misconst.abi"

        MAX_FRIEND_LIST_SIZE  = 6;

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

        FriendList = integer[MAX_FRIEND_LIST_SIZE];

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

            #include_ "ex9_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;
```

```
            real[3] Position;
            FriendList FriendGroup1;

//--------------------------------------------------------------------
//
//     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_ "ex9_INIT.ABI"

            Position[0] = 0.0;
            Position[1] = 0.0;
            Position[2] = 0.0;
            SetRevealed(INNER_SPHERE, 1500.0, Position);

            FriendGroup1[0] = GetVehicleID(CLAN_FORCE,0,0);
            FriendGroup1[1] = GetVehicleID(CLAN_FORCE,0,1);
            FriendGroup1[2] = GetVehicleID(CLAN_FORCE,0,2);
            FriendGroup1[3] = GetVehicleID(CLAN_FORCE,0,3);

endfunction;


//====================================================================
// RequestHelpFromFriend
//
// Asks a friend unit to come help the requester unit. If the friend
// is busy, the function does nothing.
//====================================================================
function RequestHelpFromFriend(integer RequesterID, integer FriendID);
code
    if(GetTarget(FriendID) == 0) then
        SelectWarrior(GetPilotID(FriendID));
        OrderMoveToObject(RequesterID, true);
    endif;
endfunction;
```

```
//=====================================================================
// CheckAndRequestHelp
//
// Runs through a list of friendly units and checks if any of them
// are currently attacking something. If so, it will request help
// from all the other units in the friend list.
//=====================================================================
function CheckAndRequestHelp(FriendList List, integer ListSize);
var
    integer UnitIndex;
    integer FriendIndex;
code
    if ((ListSize > 0) or (ListSize <= MAX_FRIEND_LIST_SIZE)) then
        for UnitIndex = 0 to (ListSize - 1) do
            if(GetTarget(List[UnitIndex]) <> 0) then
                for FriendIndex = 0 to (ListSize - 1) do
                    if(UnitIndex <> FriendIndex) then
                        RequestHelpFromFriend(List[UnitIndex], List[FriendIndex]);
                    endif;
                endfor;
            endif;
        endfor;
    endif;
endfunction;


//---------------------------------------------------------------------
//
//     Main Code
//
//---------------------------------------------------------------------
      code

            //-----------------------------------------------
            // 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;

            CheckAndRequestHelp(FriendGroup1, 4);

            //---------------------------------------
            // Structure
            //---------------------------------------
            #include_ "ex9_STR.ABI"

            //---------------------------------------
            // Lookout Points
            //---------------------------------------
            #include_ "ex9_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  // supposedly will not ...
                 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.
//*****************************************************************
```

Appendix 8: File Listing for Making Units Enter The Map

Listing of ex10.abl

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

module MissionBrain_ex10 : integer;

     //---------------------------------------------------------
     // Mission Name: ex10
     // Created: 9/27/2008
     //---------------------------------------------------------

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

          #include_ "misconst.abi"


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


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

            #include_ "ex10_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;

            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_ "ex10_INIT.ABI"

            Pos[0] = 0.0;
            Pos[1] = 0.0;
            Pos[2] = 0.0;
            SetRevealed(INNER_SPHERE, 1500.0, Pos);
endfunction;

//--------------------------------------------------------------------
//
//     Main Code
//
//--------------------------------------------------------------------
      code

            //-------------------------------------------------
            // 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;
```

```
if((gametime >= 20.0) and (gametime < 31.0)) then
    if(gametime >= 20.0) then
        ObjectCreate(VehicleID(CLAN_STAR0,0,0));
    endif;
    if(gametime >= 25.0) then
        ObjectCreate(VehicleID(CLAN_STAR0,1,0));
    endif;
    if(gametime >= 30.0) then
        ObjectCreate(VehicleID(CLAN_STAR0,2,0));
    endif;
endif;

//----------------------------------------
// Structure
//----------------------------------------
#include_ "ex10_STR.ABI"

//----------------------------------------
// Lookout Points
//----------------------------------------
#include_ "ex10_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  // supposedly will not happen ...
                 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.
//*****************************************************************
```

Appendix 9: File Listings for Making Units Leave the Map

Listing of ex11.fit

```
FITini

// ---------------------------------------------------------------------------
//
//      Mission Name: ex11  Creation Date: 9/27/2008
//
// ---------------------------------------------------------------------------


[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 = "ex11"
st TacMapGifName = "ex11"

[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 = "ex11"

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

[Campaign]
st BriefingFile = "ex11.txt"
st MapFile = "ex11"
l MaxTonnage = 990
l NumDropZones = 1

[DropZone0]
l NumSlots = 4

f PositionX = -192
f PositionY = 1344

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 = 2                              //There can be a maximum ...

[Objective0]
st Name = "Capture Buildings"
ul Type = 0                                               //Types are 0=primary...
f TimeLeft = -1                                           //Time to accomplish ...
ul Status = 0                                             //Status are 0=incomplete...
l Points = 0

[Objective1]
st Name = "Capture APC"
ul Type = 1
f TimeLeft = -1
ul Status = 0
l Points = 0

[Warriors]
ul NumWarriors = 1
uc CaptureChance = 0
st BrainParameterFile = "ex11Params"
[Warrior1]
st Profile = "PMW00025"
st Brain = "DredGoto01"

[Parts]
ul NumParts = 1
b AlliedTeam = False

[Part1]
ul ObjectNumber         = 402
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        = "PV20650"        // APC
ul Pilot                = 1
f PositionX             =  832             // Starting position in world.
f PositionY             = -192
f PositionZ             = -1.0             // Elevation automatically set by terrain.
f Rotation              =  225             // Rotation of Object in Degrees
ul Gesture              = 2                // Initial Sprite Gesture ...
f Velocity              = 0.0              // Initial Velocity ...
l Active                =  1
l Exists                =  1

[Teams]
b AlliedTeam = False

[Commander1Group:0]
l[12] Mates             = 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0

FITend
```

Listing of ex11.abl

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

module MissionBrain_ex11 : integer;

     //--------------------------------------------------------
     // Mission Name: ex11
     // Created: 9/27/2008
     //--------------------------------------------------------

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

          #include_ "misconst.abi"


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


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

            #include_ "ex11_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;

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

```
//--------------------------------------------------------------------
//
//     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_ "ex11_INIT.ABI"

            Pos[0] = 0.0;
            Pos[1] = 0.0;
            Pos[2] = 0.0;
            SetRevealed(INNER_SPHERE, 1500.0, Pos);

            SetCaptureable(VehicleID(CLAN_STAR0,0,0), TRUE);

            ObjectiveOneFailed = false;
endfunction;

//--------------------------------------------------------------------
//
//     Main Code
//
//--------------------------------------------------------------------
      code

            //-------------------------------------------------
            // 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;
```

```
Pos[0] = -3540.0;
Pos[1] = -4148.0;
Pos[2] = 0.0;
if(InArea(GetVehicleID(CLAN_FORCE,0,0), Pos, 30.0, 1) and
                                    not ObjectiveOneFailed) then
    SelectWarrior(GetPilotID(VehicleID(CLAN_STAR0,0,0)));
    OrderWithdraw;
    ObjectiveOneFailed = true;
endif;

if(CheckObjectiveStatus(1) == INCOMPLETE) then
    if(ObjectiveOneFailed) then
        SetObjectiveStatus(1, FAILED);
        PlayBetty(BETTY_SECONDARY_FAILED);
    endif;
    if ((IsCaptured(VehicleID(CLAN_STAR0,0,0)) == 1)) then
        SetObjectiveStatus(1, SUCCESS);
        PlayBetty(BETTY_OBJECTIVE_COMPLETE);
    endif;
endif;

//----------------------------------------
// Structure
//----------------------------------------
#include_ "ex11_STR.ABI"

//----------------------------------------
// Lookout Points
//----------------------------------------
#include_ "ex11_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);
            SetObjectiveStatus(1, FAILED);
      endif;
endif;

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

if (checkObjectiveStatus(0) == INCOMPLETE) then

     x = 53376;
     if ((IsCaptured(x) == 1) AND ( objectSide(x) ==  INNER_SPHERE))  then
           Obj_0_Action_1_Success = TRUE;
     else
```

```
                  if ((gettimeleft == 0.0) OR (getObjectDamage(x) > 99)) then
                        SetObjectiveStatus(0,FAILED);
                        playBetty(BETTY_CANNOT_COMP_OBJ);
                        Objective_0_Decided = TRUE;
                  endif;
            endif;


            if ((Obj_0_Action_1_Success) AND (Obj_0_Action_2_Success)) then
                  SetObjectiveStatus(0,SUCCESS);
                  playBetty(BETTY_OBJECTIVE_COMPLETE);
                  VictoryLevel = VictoryLevel + 1;
                  Objective_0_Decided = TRUE;
                  if (VictoryLevel < 1) then
                        PlayDigitalMusic(OBJECTIVE_SUCCESS_MUSIC);
                  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.
//*****************************************************************
```

Appendix 10: File Listing for Giving Artillery Strikes as Salvage

Listing of ex12.abl

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

module MissionBrain_ex12 : integer;

     //-------------------------------------------------------
     // Mission Name: ex12
     // Created: 4/10/2009
     //-------------------------------------------------------

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

          #include_ "misconst.abi"


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


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

            #include_ "ex12_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 integer      ArtyBuilding;
            static boolean      GotBuilding;
```

```
//-------------------------------------------------------------------
//
//     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_ "ex12_INIT.ABI"

            ArtyBuilding = GetTerrainObjectPartID(21,4);
            SetCaptureable(ArtyBuilding, TRUE);

            GotBuilding = FALSE;
endfunction;

//-------------------------------------------------------------------
//
//     Main Code
//
//-------------------------------------------------------------------
      code

            //-------------------------------------------------
            // 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;
```

```
if ((ObjectSide(ArtyBuilding) == INNER_SPHERE) and not GotBuilding) then
    // Set our flag to true to indicate the building has been captured
    // (and hence don't keep calling this code)
    GotBuilding = TRUE;

    // Add five small artillery strikes to Commander 0 (that's the player)
    AddStrikes(0, 0, 5);

    // Play Betty's "Enemy Resources Captured" line
    PlayBetty(6);
endif;

//--------------------------------------
// Structure
//--------------------------------------
#include_ "ex12_STR.ABI"

//--------------------------------------
// Lookout Points
//--------------------------------------
#include_ "ex12_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

     x = 71832;
     if ((IsCaptured(x) == 1) AND ( objectSide(x) ==  INNER_SPHERE))  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) OR (getObjectDamage(x) > 99)) 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.
//****************************************************************
```

Appendix 11: File Listing for Getting IDs Using Map Coordinates

Listing of ex13.abl

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

module MissionBrain_ex13 : integer;

     //---------------------------------------------------------
     // Mission Name: ex13
     // Created: 4/10/2009
     //---------------------------------------------------------

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

          #include_ "misconst.abi"


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


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

            #include_ "ex13_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;
```

```
function GetObjIDFromCoords(real xcoord, real ycoord) : integer;
var
    integer blk;
    integer vert;
    integer itmp;
    integer itmp2;
    integer xneg;
    integer yneg;
    integer mult;

code
    xneg = 0;
    yneg = 0;

    if (xcoord < 0.0) then
        xneg = -1;
    endif;

    mult = -6;
    if(ycoord < 0.0) then
        yneg = 1;
        mult = 6;
    endif;

    ycoord = abs(ycoord);

    // Compute the block number for these coordinates
    itmp = trunc(xcoord / 2560.0) + xneg + 15;
    itmp2 = mult * (trunc(ycoord / 2560.0) + yneg);
    blk = itmp + itmp2;

    // Now compute the vertex number
    itmp = trunc(ycoord / 128.0) mod 20;
    if (yneg <> 1) then
        itmp = 19 - itmp;
    endif;

    itmp2 = trunc(abs(xcoord) / 128.0) mod 20;
    if (xneg == -1) then
        itmp2 = 19 - itmp2;
    endif;

    vert = (itmp * 20) + itmp2;

    // Get the ID of the object at this block and vertex (if any) and return it.
    return (GetTerrainObjectPartID(blk,vert));
endfunction;


//-----------------------------------------------------------------------
//
//     Init Function    (automatically run first time through)
//
//-----------------------------------------------------------------------
function init;
var
    real[3] Pos;

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_ "ex13_INIT.ABI"

    Pos[0] = 0.0;
    Pos[1] = 0.0;
    Pos[2] = 0.0;
    SetRevealed(INNER_SPHERE, 1500.0, Pos);

    SetObjectDamage(GetObjIDFromCoords(0.1, -0.1), 75);
    SetObjectDamage(GetObjIDFromCoords(2995.0, 3148.0), 75);
    SetObjectDamage(GetObjIDFromCoords(3003.0, -3251.0), 75);
    SetObjectDamage(GetObjIDFromCoords(-3404.0, 2629.0), 75);
    SetObjectDamage(GetObjIDFromCoords(-3287.0, -3138.0), 75);
endfunction;

//---------------------------------------------------------------------
//
//     Main Code
//
//---------------------------------------------------------------------
      code

            //-----------------------------------------------
            // 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_ "ex13_STR.ABI"

            //---------------------------------------
            // Lookout Points
            //---------------------------------------
            #include_ "ex13_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  // supposedly will not happen unless
time limit is set
                              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.
//*****************************************************************
```