wolfman-MC2X - ASO (Advanced Scripting Options) 31 May 2017 (Revision 37)

The purpose of the Advanced Scripting Options is to provide campaign or mission specific features to make:

- mission design more creative
- gaming more challenging and interesting

It is another powerful development tool for MC2 mission designers that only MC2X offers

---

Files involved into the MC2X-ASO:

- mc2xcore.abx
- mc2xplay.abl
- mc2xscript.abl as base of your mission specific script file
-your mission specific warrior brain files

purpose and content of those files:
- mc2xcore.abx contains essential command information
- mc2xplay.abl is the player brain file that the system automatically allocates to each player controlled unit.
- mc2xsript.abl is a script file with standardized structure that carries all the key sections and basic entries for a great mission script;
- it has evolved over time through my campaign and mission design and is a great framework for a game designer's own mission specific script
- warrior brain files contain the mission specific instructions for enemy or friendly units that are not under your direct gaming control.

IMPORTANT:
Note that as a minimum mc2xcore, mc2xplay and some lines of mc2xscript need to be present to use MC2X-ASO.

WARNING:
Changes inside my mc2xcore and mc2xplay files as well as replacing them by files with same names inside or outside the fst files will create inconsistency/incompatibility issues and most likely lead to game crashes.

MC2X-ASO functions:

Inside the mc2xscript file you find the following block of entries

```
        mapsize             = 120;  //300, 240, 200, 160, 120, 100, 80, 60
====================================
        noAttackCode        = FALSE;
        wxmCode             = FALSE;
        fuelCode            = FALSE;
        repairCode          = FALSE;
        jammercode          = FALSE;
====================================
//      wmxid               = 12345;
====================================
```

The entries refer to the functions of MC2X-ASO

---

noAttackCode

... is a boolean indicating if the noAttackCode function will be used in this mission or not; needs to be initialized (set) in your mission script file
because the function is mission specific and not unit specific or campaign specific.
noAttackCode
.. is declared as an eternal boolean in mc2xcore.abx; thus the variable is available in the entire campaign but is set each mission by the script.
Default is FALSE and if no initialization is found (for example: no entry in any script) default is used. This allows backwards compatibility of the ASO to prior mission designs.

For noAttackCode =TRUE:
... the original Microsoft corebrain functionality will be used without the later implemented mechattack command.
This is important for those mission designers that want to code and control units more directly through the original coreattack commands.
The mechattack command improves the corebrain in some ways but takes the coreattack functions out of the mission designer's hands.
Sometimes more interesting warrior brain behavior can be coded without mechattack.
This functionality was used several times in some of my missions to create unique warrior brains that did not work with mechattack active.

---

wxmCode
... is a boolean indicating if the wxmCode function will be used in this mission or not; needs to be initialized (set) in your mission script file
because the function is mission specific and not unit specific or campaign specific.
wxmCode
.. is declared as an eternal boolean in mc2xcore.abx; thus the variable is available in the entire campaign but is set each mission by the script.
Default is FALSE and if no initialization is found (for example: no entry in any script) default is used. This allows backwards compatibility of the ASO to prior mission designs.

For wxmCode =TRUE:
... the functionality is under development right now, not implemented yet

---

fuelCode
... is a boolean indicating if the nofuelCode function will be used in this mission or not; needs to be initialized (set) in your mission script file because the function is mission specific and not unit specific or campaign specific.
fuelCode
.. is declared as an eternal boolean in mc2xcore.abx; thus the variable is available in the entire campaign but is set each mission by the script.
Default is FALSE and if no initialization is found (for example: no entry in any script) default is used.

For fuelCode =TRUE:
... you can use the variable fueloff throughout your mission specific script or inside your warrior brain files to activate or deactivate nits depending on the fuel supply status.
This is a new parameter to make logistics issue during a missions more interesting (for example time without refueling at a base etc.)
fueloff is declared as eternal boolean in mc2xcore as well and by default FALSE.
I have not used that function yet and thus cannot hint towards a mission example.

---

repairCode
... is a boolean indicating if the repairCode function will be used in this mission or not; needs to be initialized (set) in your mission script file
because the function is mission specific and not unit specific or campaign specific.
repairCode
.. is declared as an eternal boolean in mc2xcore.abx; thus the variable is available in the entire campaign but is set each mission by the script.
Default is FALSE and if no initialization is found (for example: no entry in any script) default is used. This allows backwards compatibility of the ASO to prior mission designs.

For repairCode =TRUE:
... your player units repair during a mission constantly without the need to see any repair bay. The units still can be destroyed or lose arms and weapons
but everything that can be repaired is repaired continuously; ammo is filled continuously as well.
The benefit of this function is in the option to game through missions during testing without too many starts from scratch or quicksaves because your units are destroyed.
This is a development tool. Fun but unfair to use it in multiplayer battles what is not part of MC2X.

IMPORTANT:

Works only for mechs and aircrafts but not for other unit types.

---

jammerCode
... is a boolean indicating if the jammerCode function will be used in this mission or not; needs to be initialized (set) in your mission script file
because the function is mission specific and not unit specific or campaign specific.
jammercode
.. is declared as an eternal boolean in mc2xcore.abx; thus the variable is available in the entire campaign but is set each mission by the script.
Default is FALSE and if no initialization is found (for example: no entry in any script) default is used. This allows backwards compatibility of the ASO to prior mission designs.

For jammerCode =TRUE:
... one of your player units can be declared in a mission script file as a unit that jams the electrical system of enemy units; thus they cannot function correctly anymore and show issues to move or operate.
You can code warrior brains for enemy units that will stall them within a specific jamming radius. There is an acoustical signal in case enemy units are inside the area.
The jammer does not impact your player units. It requires the use of 3 additional variables inside your mission specific script and the warrior brain files to use that functionality:
- jammeractive ==> jammerunit is activated (no jammer in case unit is destroyed or powered down etc.)
- jammerID ==> ID of playerunit that carries the jammer
- jammercontact ==> enemy unit inside jammer area

I have used that function is mission 13 ("Sheldon Rim") of my Houdini Project Edition; use the script and warrior brain files as a reference to understand implementation.

---

wmxid
... is an integer indicating if the wmxid of a specific mission and needs to be initialized (set) in your mission script file.
because the function is mission specific and not unit specific or campaign specific.
wmxid
.. is declared as an eternal integer in mc2xcore.abx; thus the variable is available in the entire campaign but is set each mission by the script.
Default is 0 and if no initialization is found (for example: no entry in any script) default is used. This allows backwards compatibility of the ASO to prior mission designs.

For wxmid <> 0 and identified by the gaming system:

... mission specific variants of the pbrain are used. This allows much more interesting player unit behavior that was not programmable without impact on other missions before.
Several of my missions carry a wxmid and use mission specific functionality inside the mc2xplay like shutting units down in some areas of the map or exploding when under deep water or touching lava, etc.
Used wxmids till date: 4712, 4713, 4714, 4715, 4717, 4718, 4719, 4720

mapsize
... is a boolean representing the terrain mapsize of your mission. With the option to create larger maps up to 300x300 now this feature allows two things.
1.) it is used in mc2xranpat brain files to indicate the end of the world to units.
2.)It is required in ASO standard mission objectives to trigger the end of in game mission closing video.
Default is 120 and if no initialization is found (for example: no entry in any script) default is used. This allows backwards compatibility of the ASO to prior mission designs.

ASO standard mission objectives and in game intro/extro videos

The standard mission objectives are used to trigger in game videos at the beginning and at the end of a mission. You can move the camera to different locations to introduce your map to the gamer. It is intended to show important points at the map.

There is always the starting location for your units (startCam) and the end location when the mission is closing (endCam).

You have to select between two locations (startCam and endCam) or five locations (startCam, twoCam, threeCam, endCam, end2Cam) or eight locations (startCam, twoCam, threeCam, fourCan, fiveCam, sixCam, endCam, end2Cam; indicate your selection with variable numCamLocations in the mc2xscript.abl file.

numCamLocations = 1; // 1, 2, 5, 8 Number of camera positions (1 or 2 or 5 or 8: nothing else)

When you selected 2 locations the camera starts at endCam and moves to startCam prior to mission start. startCam should be the location looking at your dropzone. endCam can be any location but best allocation of maximum interest or importance in the mission.
The mission is closing with a camera move towards endCam from wherever it starts at the time it is triggered.

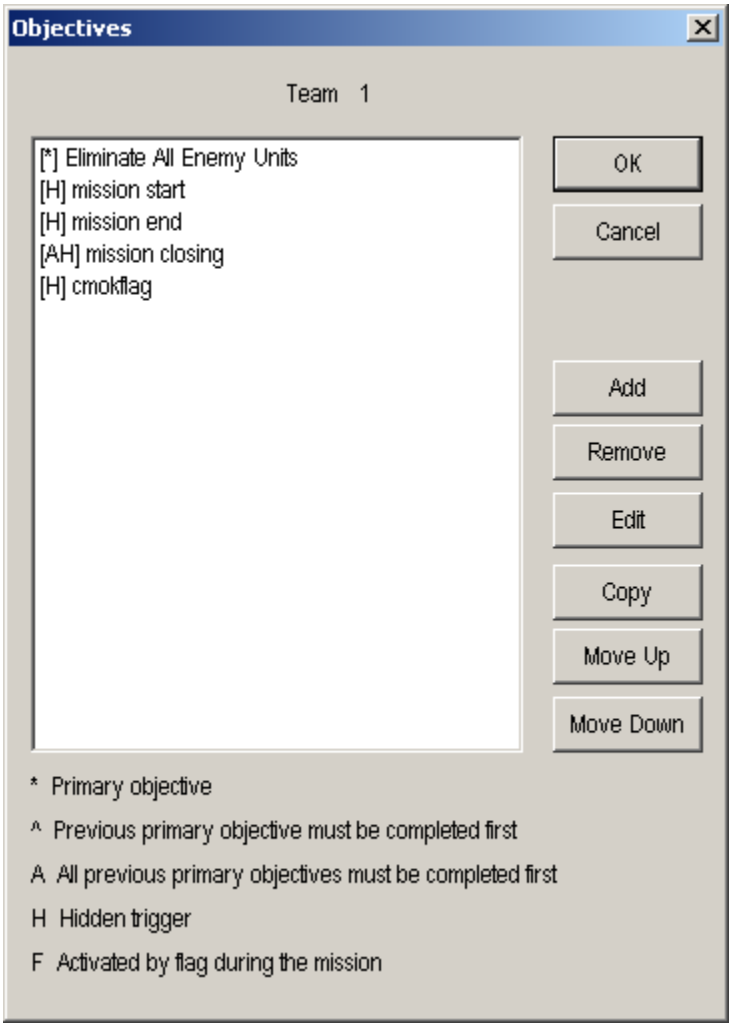
When you selected 5 locations the camera starts at endCam and moves to twoCam and threeCam and endCam prior to mission start. At mission closing it moves to end2Cam from wherever it starts at the time it is triggered. startCam should be the location looking at your dropzone. end2Cam can be any location but best allocation of maximum interest or importance in the mission; in analogy for 8 locations. faceCam allows you to turn the camera at the end of the start movie directly into the player units faces (same location like startCam, but different rotation).

You have the option to move to a new location and then look at the location with specific camera moves.
This option is supported by the “Face” variables in the script file (only available for 8 locations).

You need to declare the camera settings for all locations inside the mc2xscript.abl.file but for your selected number of locations only.

To use these automatic camera movement feature in it is necessary to implement a standardized additional set of objectives in your mission as it is illustrated and outline in the following pages.

ASO standard mission objectives



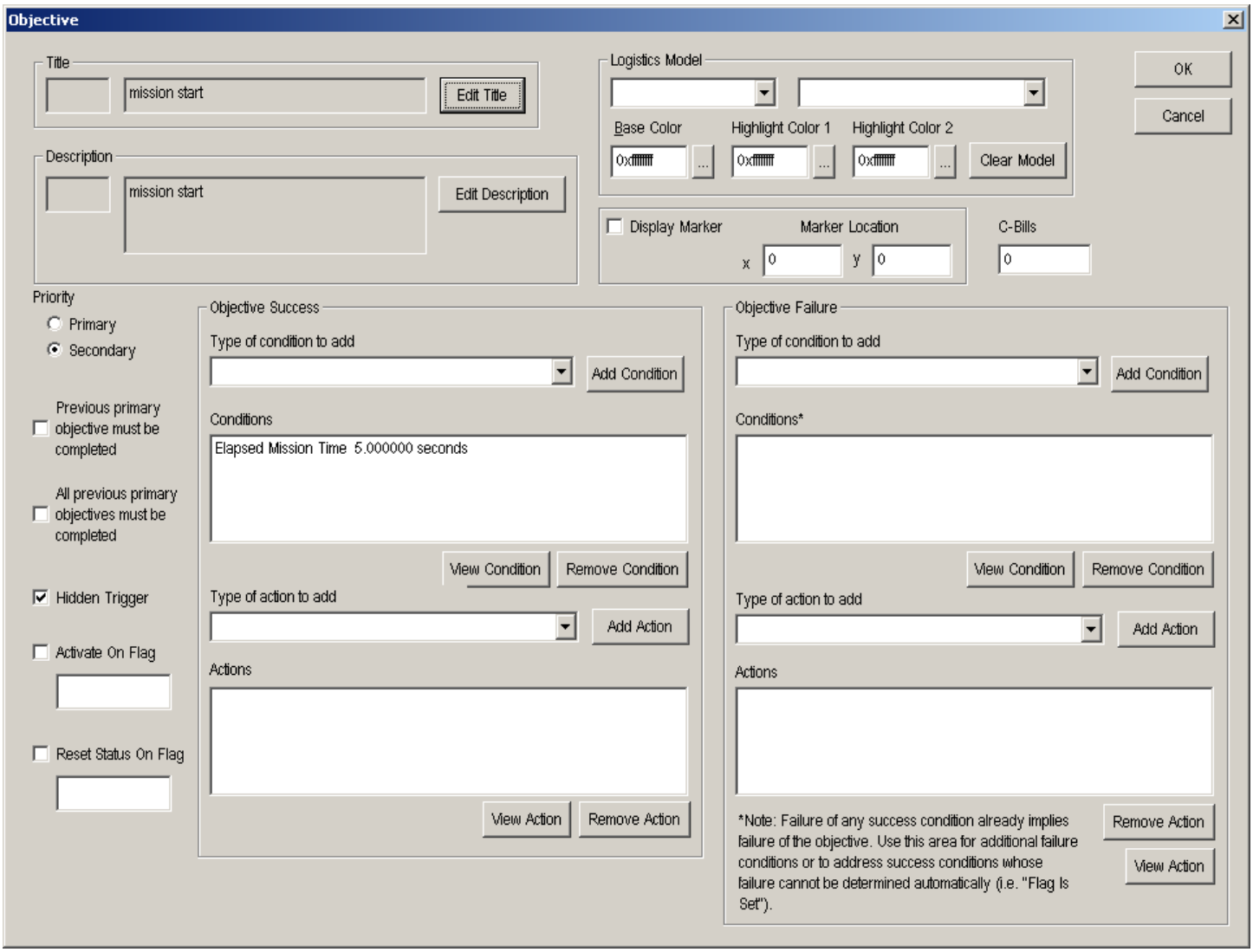
Implement the following objectives in your mission illustrated above :
--mission start
--mission end
--mission closing
--cmokflag

Pay attention to the settings for hidden, secondary, and all previous primary objectives must be completed !
Keep this block of objectives together in the above sequence and place after any mission-defined primary objective.

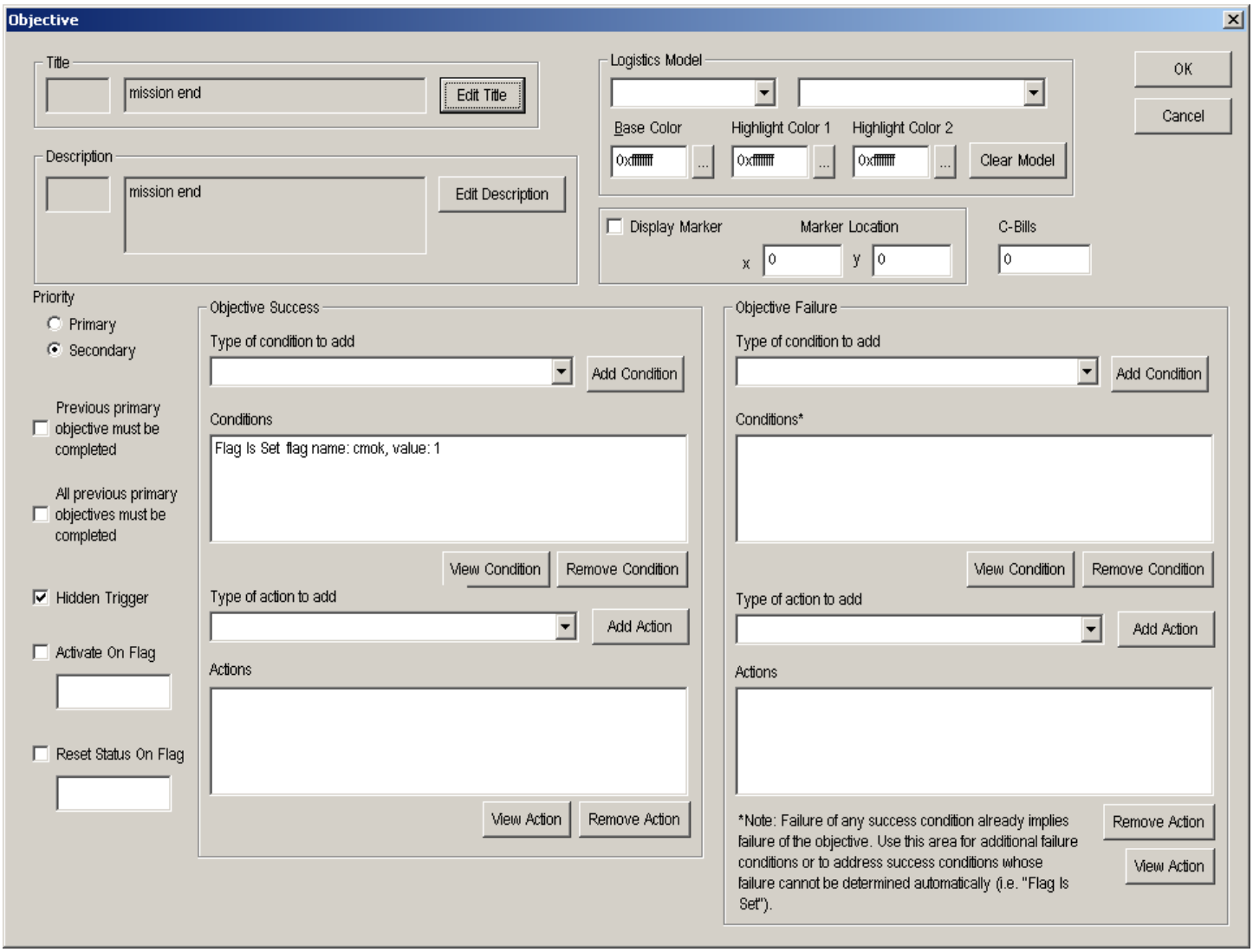
You need to define your objective# for mission start and the objective# for mission closing in the mc2xscript.abl file.
Look for the variable initialization of startobjective and closingobjective and enter value.
Remember that the objective # starts counting at 0.

In the example above the startobjective is the objective# for mission start and in this case 1; closingobjective is the objective# for mission end and in this case 2.

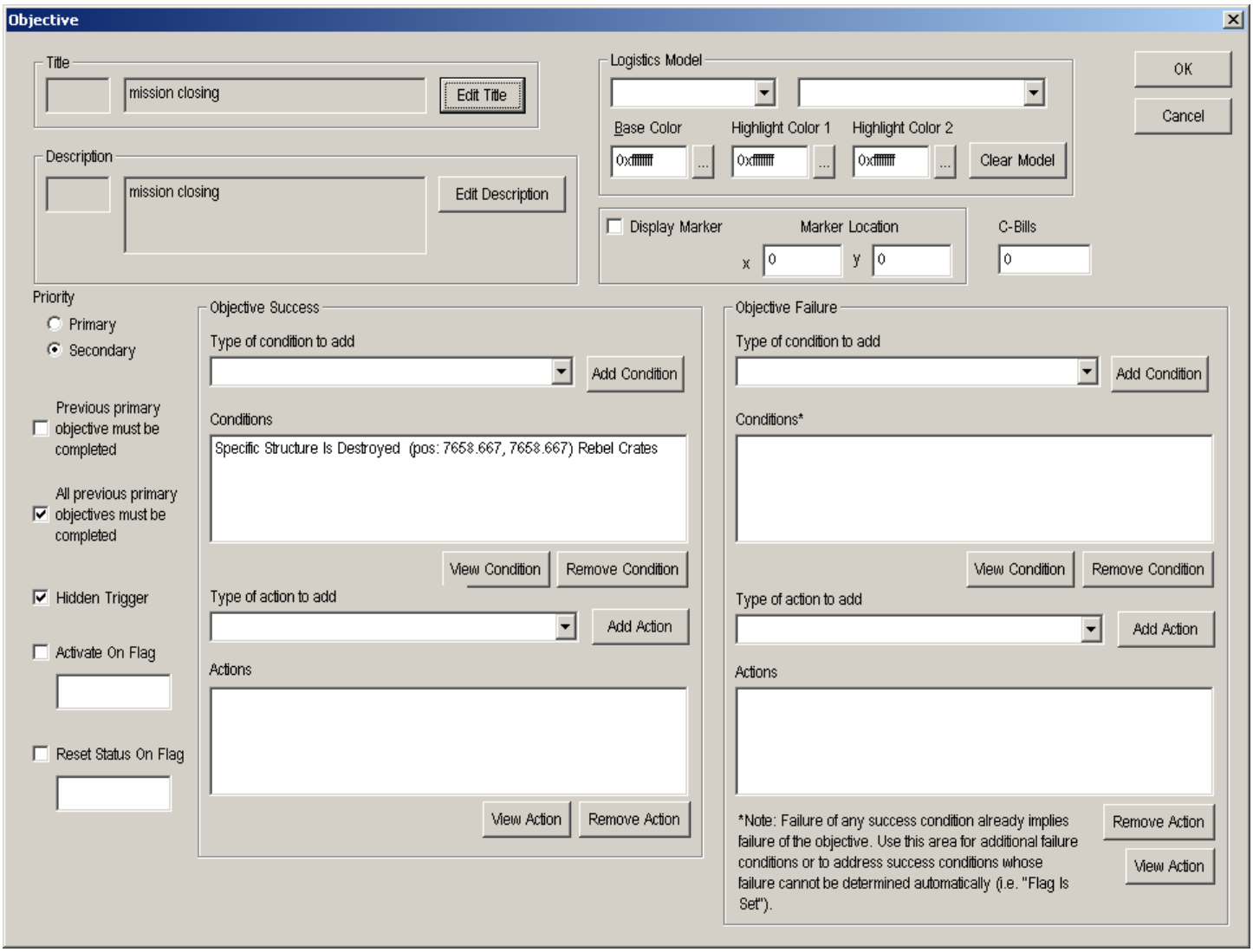
If you do not want to use any intro movie just use 1 location (default). Then no entry of start or end objective nor any camera location is required.



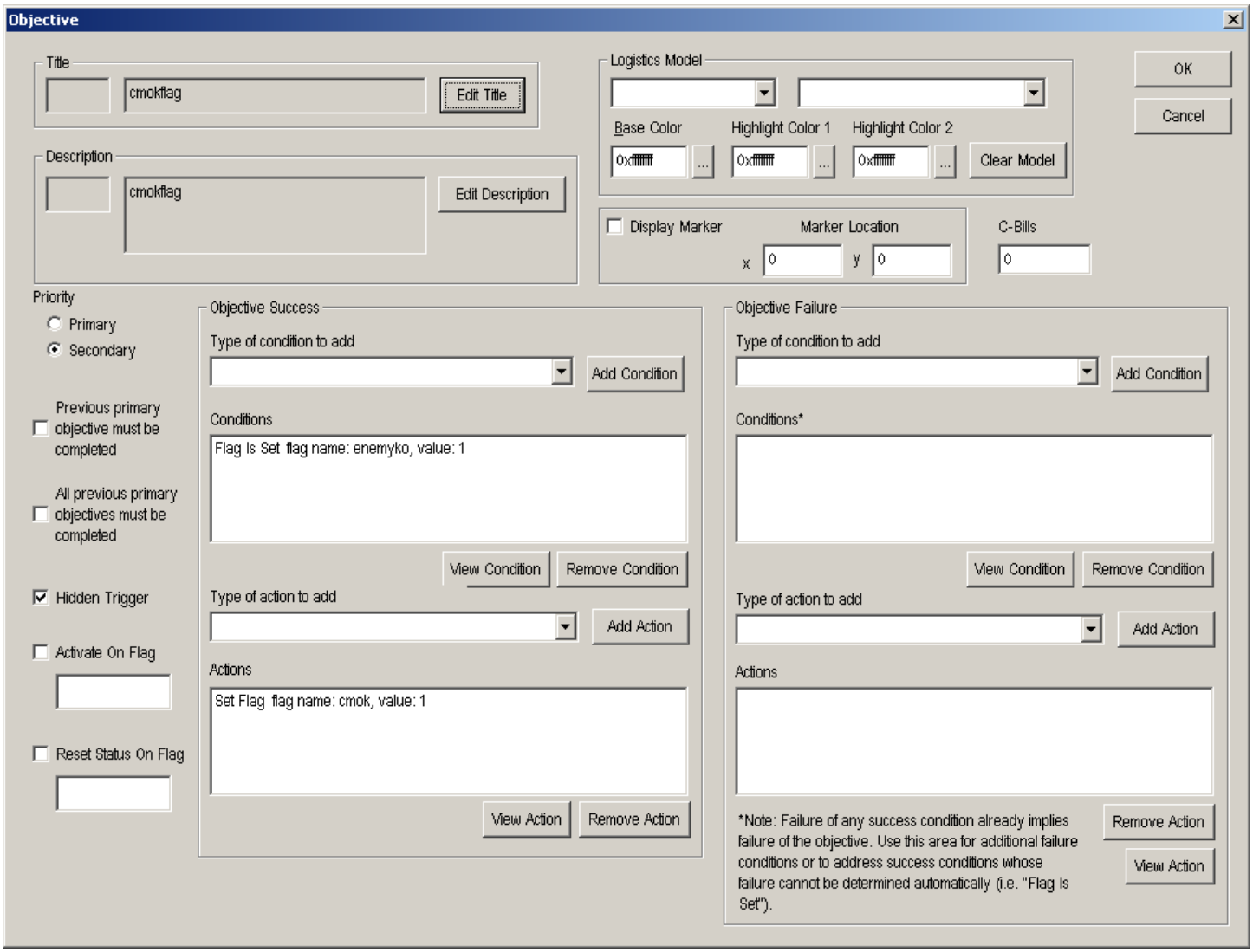
This objective triggers the start of the intro movie after 5 seconds.



This objective indicates to the script that the flag cmok is set and this represents that all primary objectives are fulfilled and the mission can be finished. This means as well that you have to flag all primary objectives and enter the flag status into the cmokflag objective as conditions.



A neutral rebel crates object needs to be placed in the very last square (grid view) on the upper right corner of the map. This will be destroyed automatically by the script as soon as certain conditions are established by the mission objectives and thus the closing movie can be triggered. The script knows by the mapsize automatically the objectID, if positioned correctly in the corner.



This just shows an example that all primary objectives need to be flagged as conditions in cmokflag objective.
The mission primary objective to eliminate all enemy units is flagged with enemyko and this flag became a condition inside the cmokflag objective

//--

MC2X-DSO Declarations in mc2xcore.abx:

```
eternal boolean     repairCode;
eternal boolean     wxmCode;
eternal integer     wmxid;
eternal boolean     fuelCode;
eternal boolean     jammerCode;
eternal boolean     fueloff;
eternal integer     jammerID;
eternal boolean     jammeractive;
```

MC2X-DSO Declarations in mc2xplay.abl:

```
none
```

MC2X-DSO Declarations in mc2xscript.abl:

```
eternal boolean     jammercontact;
eternal boolean     mapsize;        (60, 80, 100, 120, 160, 200, 240, 300)
```

MC2X-DSO Initializations in mc2xscript.abl:

```
mapsize             = 120
noAttackCode        = FALSE;
wxmCode             = FALSE;
fuelCode            = FALSE;
repairCode          = FALSE;
jammercode          = FALSE;
wmxid               = 12345;

jammerID            = 512;
jammercontact       = FALSE;
```

MC2X-DSO Initializations in mc2xplay.abl:

```
fueloff == FALSE)
```

Final Overview: (CB = mc2xcore, PB = mc2xplay, MS = mc2xscript, WB = warrior brain)

Variable	Type	Declared in	Initialized in	Modified in	Used in
mapsize	Bool	MS	MS	MS,WB	MS,WB
jammercode	Bool	CB	MS	MS,WB	MS,WB
jammeractive	Bool	CB	MS	MS,WB	MS,WB
jammerid	Int	CB	MS	WB	
jammercontact	Bool	MS	MS	MS,WB	WB
repaircode	Bool	CB	MS	(MS,WB)	PB
noAttackCode	Bool	CB	MS	(MS,WB)	PB
wmxCode	Bool	CB	MS		
wmxID	Int	CB	MS	(MS,WB)	PB

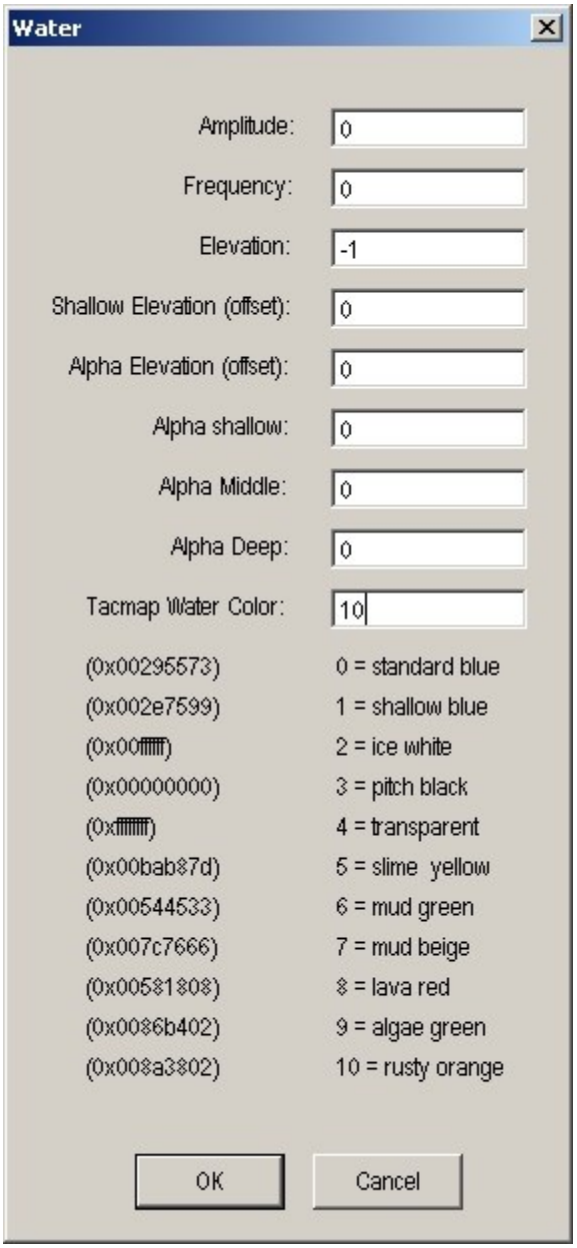
This is a framework rooted deep in some core gaming environment files; it is up to you to create the correct script and warrior brain files to use the potential; there can not be a standard solution for all purposes.

Latest Additions ==========

Water color on tactical map

I always disliked that the water color was always and only blue on the tactical map, even when I use water as lava or mud etc. I coded a solution to be able to select additionally 10 different tacmap water colors directly in the editor.

The former workaround solution using ***shallowDepth*** is now obsolete.



New abl commands: (great new stuff for scripting, most of which based on ***Magic's*** work)

togglesalavagecraft

togglerepairtruck

toggleartillerypiece

togglehover ***(Pollux hovercraft)***

togglespotter ***(Monster vehicle)***

toggleelemental ***(UW_Specialist***

toggleairstrike

togglesensorstrike

togglescoutcopter

toggleminelayer

togglerescue ***(Armored Rescue Vehicle)***

toggledrone ***(Camera Drone)***

add money(Int)

addresourcepoints (Int)

setobjectivestatus(Int, Int)

objectchangesides(PartID, teamID);

PartID - number from the editor for buildings, part id for mover (getunitmates(squadID)) from squad ID
teamID - (0,2 player; 1,3,... enemy)

setBrain(teleunit, "mc2xguard4");

example:
static integer teleunit;
teleSquadID = 3;
getUnitMates(teleSquadID, teleList);
teleunit = teleList[0];
mc2xguard4.abl (warrior brain file)

RemoveMoverFromPlayer(unitID); //512...,2050...

addmovertoplayer(PartID, teamID, commanderID);

PartID - number from the editor for buildings, part id for mover (getunitmates(squadID)) from squad ID.
teamID - (0,2 player; 1,3,... enemy)
commanderID = 0, player is 0 for single player mission

teleporttopoint(PartID, position);

PartID - part id for mover, obtained using (getunitmates(squadID).
position is real position[3], x,y and z coordinates from the editor.
example:
apoint[0] = -504;
apoint[1] = 500;
apoint[2] = 500;
teleporttopoint(2050,apoint);
Note - 2050 is part number of first player unit.
Effect: Teleport unit to position. Works for all movers.

gettargetrelativeposition(TargetID, range, angle);

integer TargetID = gettarget(-1),
real range, angle.
set range and angle from the selected unit to target and store values to
range and angle variables.

getrelativepositiontotarget(TargetID, distance, flag = 2, newPoint);

integer TargetID = gettarget(-1),
real distance - distance you want from target,
real[3] newPoint - coordinates for the new point.

setTextMsg(0,"My Message",x);

first parameter can be 0 or 1.
0 - instant message for the duration of number of seconds (x),
1 - message box with press enter to continue, in this case third parameter is not important.

Effect: display "My Message" in message box for x seconds or with pause.
Uses “Important Message” box formate can be called from scripts and brains
Displays formatted text message using ”&” as formatting character.

To use “Important Message” box format in mission.fit file objective settings
select *Display Important Message* in Editor and fill formatted text into box.
The message be displayed for 8 seconds.

The “Important Message” box format in mission.fit file objective settings can
display a resource string from your resource file as well.
Select *Display Resource Text Message* in Editor and pick existing resource string.
The resource string should be formatted text using “&” as formatting character.
The message be displayed for 8 seconds.

RemoveMoverfromPlayer(unitID, new team, newcommander);

PartID - number from the editor for buildings, part id for mover (getunitmates(squadID)) from squad ID.
newteam: new team 0-7)
Newcommander: new commander (0-7

Effect: moves a player unit to a different team and commander (enemy)

==========

complex example to:
- teleport a player unit to a location,
- set a warrior new brain,
- remove from player forces and
- add to enemy forces

```
teleporttopoint(theunit,camStartPos);
setBrain(teleunit, "mc2xguard4");
RemoveMoverFromPlayer(theunit);
objectchangesides(theunit,1);
```

==========

settxtbuildingname(BuildingID, "New Buildingname");

allows you to rename buildings on the map using scripting
BuildingID - number from the editor for this building
New Buildingname – the new name the building should show

setmechgesture(MechID, gestureID);

MechID - number from the editor for buildings,
part id for mover (getunitmates(squadID)) from squad ID.
gestureID - (0,1,2,3,4,5,6,7,8,..) (gesture will always be "fallen backwards")

Effect: set mech gesture to fallen

destroymechbodylocation(MechID, locationID);

MechID - number from the editor for buildings,
part id for mover (getunitmates(squadID)) from squad ID.
locationID - (0 -HEAD, 1 -CTORSO, 2 LTORSO, 3 -RTORSO, 4 -LARM, 5 -RARM, 6 -LLEG, 7 -RLEG)
Effect: Destroy selected mech body location.

damagemecharmor(MechID);

MechID – number from the editor for buildings,
part id for mover (getunitmates(squadID)) from squad ID.

Effect: Random damage to all unit's front armor locations.

setmissiontune(tuneID);

tuneID – music Id number from sound.snd.

Effect: sets mission ambient tune

setsalvagecraftenabled(state) (state => TRUE or FALSE)

setrepairtruckenabled(state) (state => TRUE or FALSE)

setartillerypieceenabled(state) (state => TRUE or FALSE)

sethoverenabled(state) (state => TRUE or FALSE)

setspotterenabled(state) (state => TRUE or FALSE)

setelementalenabled(state) (state => TRUE or FALSE)

setairstrikeenabled(state) (state => TRUE or FALSE)

setsensorstrikeenabled(state) (state => TRUE or FALSE)

setscoutcopterenabled(state) (state => TRUE or FALSE)

setminelayerenabled(state) (state => TRUE or FALSE)

setrescueenabled(state) (state => TRUE or FALSE)

setdroneenabled(state) (state => TRUE or FALSE)

setinvulnerable(state) (state => TRUE or FALSE)

Effect: makes player units invulnerable

setnewpilot(UnitID, "warriorfile","brainName");

UnitID - unit id for mover (getunitmates(squadID)) from squad ID.
brainName - the name of the brain file without extension,
warriorfile – the pwm file of the new pilot without extension.

Effect: assign new pilot to unit. It also sets the brain file.
If the pilot exists it will overwrite the pilot.
DO NOT add pilots that are used in campaign before, it will erase all accumulated experience.
Use only to add new pilots. Allied team must exist in mission and must have at least one unit.
USE FOR PLAYER UNITS ONLY !

hirePilot("brainName");

brainName - the name of the brain file without extension,
warriorfile – the pwm file of the new pilot without extension.

Effect: adds new pilot in specific mission to be available by salvage craft .

numfriendswithinradius(PartID, radius);

part id for mover
(getunitmates(squadID)) from squad ID (or -1=self for use in warrior brain).
radius - radius around unit to look for friends.
example:

```
num_friends1 = numfriendswithinradius(Commando, 600.0);
```

numenemieswithinradius(PartID, radius);

part id for mover
(getunitmates(squadID)) from squad ID (or -1=self for use in warrior brain).
radius - radius around unit to look for enemies.
example:
    num_enemies1 = numenemieswithinradius(Commando, 500.0);

ablprint("FileName", AnyValue);

For debugging in ABL scripts.
FileName - Name of the file that will be created in data folder.
AnyValue - any variable or value you want ot print to file, chars or
strings must be "example_text".
example:
    ablprint("Friends1", num_friends1);
    ablprint("Enemies1", num_enemies1);

isrefit;

Return value: true if mover is refit vehicle. For use in warrior brain file.

autorepairwithinradius(PartID, radius);

PartID - number from the editor for buildings, part id for mover
(getunitmates(squadID)) from squad ID (or -1=self for use in warrior brain).
radius - radius around unit to look for friendy units that need repairs, if no enemies around.

Repair all damaged friendly units within radius if no enemies around.
example:
autorepairwithinradius(-1, 500.0);

automedicalwithinradius(PartID, radius);

PartID - number from the editor for buildings, part id for mover
(getunitmates(squadID)) from squad ID (or -1=self for use in warrior brain).
radius - radius around unit to look for friendy units that need medical attention, if no enemies around.
Provides medical aid all wounded friendly unit pilots within radius if no enemies around.
example:
automedicalwithinradius(-1, 500.0);

setunitforcegroup(PartID, forcegroup);

part id for mover
(getunitmates(squadID)) from squad ID (or -1=self for use in warrior brain).
forcegroup – forcegroup number between 1 and 9.

autoscan;

True if AutoScan is selected.
Use inside warrior.abl files only!

setBisMsg(0,"My Message",10);

first parameter can be 0 or 1.
0 - instant message for the duration of number of seconds (10),
1 - message box with press enter to continue, in this case third parameter is not important.

Effect: display "My Message" in message box for 10 seconds or with pause.
Uses “Battlefield Information System” box format can be called from scripts and brains
Displays formatted text message using ”&” as formatting character.

To use “Battlefield Information System” box format in mission.fit file objective settings
select *Display Text Message* in Editor and fill formatted text into box.
The message will pause the game and be displayed until RETURN is pressed.

setcampaignglobalvar(VariableNumber, VariableValue);

VariableNumber - number of global variable (5 for now = 0,1,2,3,4).
VariableValue - integer value to be set to selected variable.

example:

```
setcampaignglobalvar(0, 3);
```

getcampaignglobalvar(VariableNumber);

VariableNumber - number of global variable (5 for now = 0,1,2,3,4).
return value - value stored in selected global variable.

example:

```
GlobalVarValue0 = getcampaignglobalvar(0);
```

needsrefit;

True if unit needs refit.
Use in warrior.abl files only!

scanareacapture(radius);

Idle players units will scan area for capturable buildings and capture if any.
Use in warrior.abl files only!

scanarearepair(radius);

Idle players units will scan area for repair bay and repair;
if enemy repair bay it will be captured first;
player units identify and select repair bay types fitting unit type
(Mech Bay, Vehicle Bay, Aircraft Bay, Hospital Bay).
Use in warrior.abl files only!

setfixedbuildingrp(PartID, number_of_resource_points);

PartID - number from the editor for buildings,
number_of_resource_points - integer number of resource points.

Effect: Set Fixed number of RPs to any building and make it capturable.
Works for buildings only .
Any building set with this command will have fixed number of RPs when captured.

setobjectivestatus (objective number, status);

Objective number starts at 0 for first objective.
Status: 0 → undetermined, 1 → success, 2→ failed

getpilotphoto(unitID);

unitID – ID of unit to check on

Effect: returns the picture ID of the pilot of a unit.
thus specific pilots can be addressed directly
Picture ID is found in pmwpilotname.fit file

gettimeoflaststep;

Return: (float) Time of last move step.

iscurtacordermoveorder;

Return: True if current tacorder is move order.

getcurtacordertime;

Return: (float) Time when current tacorder was given.

clearmoveorders;

Effect: Clear all move orders.

sethelpmovetopoint;

NOT for direct scripting use

addmovertoinventory(PartID, unitType, commanderID);

PartID - number from the editor for buildings, part id for mover (getunitmates(squadID)) from squad ID.
unitType - (0 => mech type; 1 => vehicle type)
commanderID = 0, player is 0 for single player mission

effect: adds the selected unit to the player inventory for further use in campaigns

getcurrenttonnage(unitID);

unitID – ID of unit to check on
Return: weight of unit in tons.

setpilot(UnitID, "warriorfile","brainName");

UnitID - unit id for mover (getunitmates(squadID)) from squad ID.
brainName - the name of the brain file without extension,
warriorfile – the pwm file of the new pilot without extension.

Effect: assign new pilot to any unit. It also sets the brain file.
If the pilot exists it will overwrite the pilot.

isrepairbay(partID);

partID – ID of object to check on
Return: TRUE if object is a repairbay

Example: if (isrepairbay(55442)) then…

Rechargerepairbay(partID);

partID – ID of object to check on
Effect: recharges depleted repair bay (can set any selected building to be a repair bay).

Example: rechargerepairbay(55442

hasorderfromplayer;

Return: TRUE if object has player order

Enjoy these features of wolfman-MC2X

wolfman