NEXUS

Modding Documentation

by

Zoltán Motyán

András Tímár-Geng

Version 1.01

October 19, 2004
Steam Workshop Added in November 2016

1 MOD STRUCTURE

1.1 MOD TYPES

1.1.1 STANDARD MULTIPLAYER GAME

This does not truly create a new mod, but expands the existing multiplayer game set with a new game, using existing multiplayer ship and device types.

To do this, create a new ***.mpgame** file in the *NEXUS "Standard Multiplayer Battles/universe/MOD_MULTIPLAYER"* library. The structure and ship types of this are detailed in the Mission Scripting chapter. We've provided two standard game type scripts as examples.

1.1.2 SINGLEPLAYER MOD

Enables adding of new models, textures, effects, sounds etc to the system as well as redefinition of the existing ones.

It disposes a separate mission set, startable from the main menu individually (Meaning that no campaigns can be created, only standalone missions).

1.1.3 MULTIPLAYER MOD

Similar to the above, but contains multiplayer games instead of missions.

It's only possible to join multiplayer servers running the same MODs as the local machine. If this condition is not met, the game will display the required MODs name.

1.2 EXAMPLE MODS

Two examples for creating multiplayer games are provided in the *mods/NEXUS Standard Multiplayer Battles/universe/mod_multiplayer* library – a deathmatch and a VIP escort game.

Three example mods are provided to exemplify MOD structure.

SAMPLE 01 A Complete Mission

Template for mission scripting, and includes the entire script of the official demo mission. It´s also a good example for creating own missions without actually making any changes to the system, just using existing ships and devices.

SAMPLE 02 Everything Change

Examples for all modification possibilities.

SAMPLE 03 Multiplayer

Example for a multiplayer game system using custom ships.

SAMPLE 04 Champaign Changes

Example for overwriting main campaign elements.

SAMPLE OPT 01 Changed Texture

This is an example of an activation mod that can be activated on top of every mod. This example changes some texture.

SAMPLE OPT 02 Changed NPC Face

This is an example of an activation mod that can be activated on top of every mod. This example changes NPC faces.

1.3 MOD DIRECTORY STRUCTURE

Each MOD is a sub-directory in the *mods* library. The name of the subdirectory will be the name of the MOD as well. The composition of the directory is as follows.

1.3.1 WORKING FOLDER

__work – All source files containing model, texture and parameter files to be converted to for the game are placed here.

Art

converter.tpr – Data-file of the converter, see below

Meshes - Lightwave models. For its structure, see the Modelling chapter.

Textures – the textures of models and planets as well as icons, effects, NPC images and animations.

Params

efx.xls – Effect parametric table. See the Effects chapter.

tactics.xls – Tactical parameter table. See the Data tables chapter.

1.3.2 MODELS

Meshes – this is where all models used in the game are to be converted. Subdirectory structures can be created if the need arises, and the models can be denoted together with ship type and device table paths.

1.3.3 TEXTURES

Textures

Meshes – the textures used by the models; must by strictly placed into this library fused.

FX – Effect animation textures.

Icons – Ship class icons, as they appear on the ship lists in the game.

Commands – Icons of the newly scripted ship commands, as they appear on the lower command line. See the Command Scripting chapter.

Planets – planet textures. See the Solar system editing chapter.

1.3.4 NPC-S

Npc – all NPC faces have an own directory, named after the face's three-syllable ID. This directory contains the animation image sequence for a mute and talking face. The animation plays at 15 frames / second.

000-029.tex – Mute state animation (repeatable)

030-069.tex – Talking state animation (repeatable)

still.tex – a larger still image of the NPC.

1.3.5 MUSIC & SOUNDS

Music – all music should be placed here, in **Ogg Vorbis** format. For more info, see, the Music and Mission scripting chapters.

Sound – sounds should be in **wav** format and can have any directory structure. See the Sounds and Data tables chapters.

1.3.6 PARAMETER FILES

Universe

main.ini – the main title and main menu – see the Main Menu chapter.

mod_consts.ini – constant definitions, see the Constants chapter.

engine – visual effects and sounds definitions, see the Effects and Sounds chapters.

mod_ani.ini – definitions for all animations used in effects.

mod_efx.ini - visual effects can be generated from the *_work/params/efx.xls* sheet.

mod_sound.ini – sound and music definitions

mod_particles – definitions of all named particles, in any number of files.

systems – solar systems. The stellar bodies´ parameters can be found in the **.system* files, while the **.system.bg* files contain the galactic background elements. See the Solar system editor chapter.

tactics – tactical parameter files, see the Data tables chapter

races.ini – definitions for all races present in the game

tacticsbase.ini – the operational parameters of the tactical systems

tacticsai.ini - AI parameters

tacticstypes.ini – ship classes, devices, aiming parameters; all can be generated from the *work/params/tactics.xls* sheet.

mod_commands.ini – inserts own ship commands to the orders bar.

mod_commands – scripts for player issued ship commands

mod_aicommands - AI-controlled ship command scripts

mod_missions - mission scripts and include files. See the Mission scripting chapter.

texts – text definitions, localization

texts – simple text and title definitions

dialogs – MPC dialogues and answering windows

1.4 CREATING A MOD – STEP BY STEP

1.4.1 CREATION

Create a subdirectory in the mods library. The directory's name will be the mods name as well. In the beginning, it's easier to just copy existing necessary directories from an example mod, and modify them.

In the root of the directory if you place a file named *optional.ini* (ideally please enter in the file Title "Put the actual mod name here" inside this file) the mod launcher will recognise it as an activation mod that can be activated on top of every mod or the main game.

1.4.2 MOD TOOLS

Click the *Nexus / Modding / Modification Tools* menu item in the start menu. This starts a small menu from where you can launch all modding tools.

Expanding the *Select Modification* menu will show all subdirectories of the mods library, including the new mod. Select it.

1.4.3 MODELLING, TEXTURING

Use Lightwave to design the desired models and textures. Place them into the *_work/art* subdirectory. (For precise locations, see the Modelling chapter).

The *Model viewer* program will be of considerable aid your design, as it shows all changes in textures or the model instantly, as it will appear in the game.

1.4.4 CONVERTING

The finished models should be converted with the help of the *Model / Texture Converter* program and placed into their destinations (the *meshes* and *textures/meshes subdirectories*).The names must be given in the ship and device table's.

1.4.5 EFFECTS

The definition of the visual effects is in the *_work/params/efx.xls* sheet. The table contains a macro which writes the sheet contents in the form required by the game into the *universe/engine/mod_efx.ini* file. The macro is launched by pressing **Ctrl+Shift+X**.

The table can refer to particle effects, the definitions of these are done in the *universe/engine/mod_particles* library.

New animation sequences can be added to the system in the form of textures, but the existing ones can be used as templates as well for creating new effects.

The newly defined effects will be **added** to the existing effect set in the game.

1.4.6 MUSIC, SOUNDS

These are placed in the *music*, and *sound subdirectories* respectively, then listed in the *universe/engine/mod_sound.ini* file. The must be given identifiers. They can then be used with the ID´s in mission scripts and effect-tables.

The newly defined sounds and music will be **added** to the existing effect set in the game.

1.4.7 PARAMETERS

Ship classes, devices and aiming data can be defined in the *_work/params/tactics.xls* table. The table contains a macro which writes the sheet contents in the form required by the game into the *universe/tactics/tacticstypes.ini* file. The macro is launched by pressing **Ctrl+Shift+T**.

The newly defined objects **will replace** the original objects, meaning that the original ship types and devices will not be available in the MOD using this .ini file.

1.4.8 MISSION SCENES

Entire solar systems can be built using the Solar sytem editor. Mission scenes can be selected to take place within these, then use the Mission Editor to place all objects)ships, asteroids, radiation fields etc.) to be included in the mission.

1.4.9 MISSION SCRIPTING

Mission play must be written in the scripting language, no editor is provided for that. However, running the script in the Mission Editor will provide a lot of debugging options or display detailed error messages respectively.

1.5 MOD SPREADING

Standard multiplayer games can be spread simply by handing on the mpgame file; copying it into the target machine's *NEXUS Standard Multiplayer Battles/universe/MOD_MULTIPLAYER* library.

The other types of MODs are spread by handing on the entire MOD library, **excluding** the mods **__work** directory. Copying the MOD directory into the target machine's *mod* directory will make it instantly playable. Special care must be given that the directory's name (that is the mods name as well) should be the same on the target computer – otherwise the two machines will not be able to connect in multiplayer.

1.5.1 STEAM WORKSHOP TOOL

Once the files and folders for a mod have been created start the Steam Workshop Tool via the Modding tools launcher application. In the list of mods your mod should now appear. Please enter a description and set up visibility and tags. Now you need to create a preview image. To do this just name a JPG file with 1280x720 as dimensions in the *mods__steam_workshop* directory and name it the same as your mod. Now you are

ready to publish your Workshop item. You can do this hitting the “Create Item”. If you have done changes just do the changes, start up the Workshop Tool, select your mod and hit the “Update Item” button.

2 MODELING

2.1 INTRODUCTION

Models are made with Lightwave 3D.

The models should be placed in the *_work/art/meshes* directory, in any subdirectory structure, but all models must include the following subdirectories.

2.2 DIRECTORY STRUCTURE

All models have three subdirectories.

2.2.1 SCENES

If a model contains animations, they have to be assembled in an LWS file, which is placed in the *scenes* library.

2.2.2 OBJ

The directory of the LWO file(s). The objects should be loaded to the LWS from here.

If the model is not animated, then it has to be placed in a single LWO. Layer parenting must be done in the Modeler.

2.2.3 TEXTURES

All the models textures related to this model only must be placed here.

Textures shared by multiple models should be placed in the *_work/art/textures/meshes* directory.

The textures´ names must be **unique** in both cases.

2.3 TEXTURES

All textures must be in TGA format.

The textures´ names must be **unique**!

2.3.1 COLOUR MAP

24 bit RGB or 32 bit ARGB in the case of alpha textures.

Width and height can differ, but both factors must be powers of two, and cannot be larger than 2048. The figures can therefore be: 4, 8, 16, 32, 64, 128, 256, 512, 1024, and 2048.

2.3.2 LUMINOSITY MAP

Grayscale.

Its size and projection must be identical to the surface colour map.

Can only be used together with a colour map, and only if the colour map doesn't include alpha, since the luminosity will be loaded into the colour map's alpha channel.

Its name will be assembled form the colour map's, adding *__lum*. Example:

colour map:	cruiser_side.tga
luminosity map:	cruiser_side__lum.tga

If the converter finds such a texture, it **will use it regardless** if the luminosity map is added to the surface in the Modeller.

2.3.3 BUMP MAP

Grayscale.

Its size and projection must be identical to the surface colour map.

Its name will be assembled form the colour map's, adding *__bump*. Example:

color map:	cruiser_side.tga
bump map:	cruiser_side__bump.tga

If the converter finds such a texture, it **will use it regardless** if the bump map is added to the surface in the Modeler.

2.3.4 SPECULAR MAP

Grayscale.

Its size and projection must be identical to the surface colour map.

Can only be used together with a bump map (specular will be loaded into the bump map's alpha channel).

Its name will be assembled form the colour map's, adding *__spec*. Example:

color map: cruiser_side.tga

specular map: cruiser_side__spec.tga

If there is a bump map and the converter finds such a texture, it **will use it regardless** if the specular is added to the surface in the Modeler.

2.3.5 REFLECTION MAP

Separate reflection maps cannot be defined.

If the surface **Reflection** value is other than zero, and there is a given specular map, then it will function as a **reflection map.**

If no specular map is given, then the mirroring value of the surface will equal the **Reflection** value.

2.3.6 TRANSPARENT SURFACES

If **Layer Opacity is less then 100** in the texture options, then the default alpha values of the colour map will be used as transparency. (If Transparency equals 100, than the alpha channel will be considered as luminosity map by the engine, see above) This means that transparent surfaces cannot have **luminosity maps**.)

At the same time, the handling of such surfaces is not fully supported by the engine, thus embedded transparent parts and transparent parts next to each other should be avoided on all models. The engine supports single, isolated transparent surfaces for modelling purposes.

2.3.7 ALPHA CLIPPING

If the **Blending Mode** is set to **Subtractive** in the texture options, than the **colour map's alpha** channel will show up in a different way: for alpha values less than 128 the surface will be **wholly transparent**, while with larger values it will be **completely opaque**. Of course, luminosity maps cannot be used in this case.

This form of clipping can be used indefinitely. It's of especially good use for dense, thin frameworks which use a lot of polygons - (for example contraptions extruding from wrecks, construction frames etc).

2.4 LIGHTWAVE OBJECTS

2.4.1 LAYERS

Parts not moving relative to each other should be placed on the same layer.

They must have a main part, placed on the 1st layer. All other layers must me parented to this layer. This can either be done in the modeller or the Scene editor.

2.4.2 LOD PHASES

All layers require separate LOD-phase-series.

Individual phases must be placed in **separate layers**, which must be **parented** to the original layer to which they belong, and their names must be:

#red d#

where “d” is the distance from the camera where the LOD phase will be actual.

2.5 LIGHTWAVE SCENES

2.5.1 HIERARCHY

Models containing animations must be assembled from LWS scenes.

Care must be taken that the objects in the scene should bring their LOD phases. The LOD´s must not be tampered with; the object hierarchies of the animation must be built from the **original objects.**

Cloning can be used. In this case only the main object needs to be cloned, not the LOD phases.

Nullobjects can also be used to create animations.

2.5.2 ANIMATIONS

A model can contain several independently playable animations.

Separation of those is done by giving each animation a time sequence. The keys in the time sequence make up the animation.

The identification of an animation is done by creating **Nullobjects**, named the following:

#anim name beginframe endframe#

So **all animations include a null-object**, containing the animation name and time sequence begin and endframe it its name.

The game uses the following animations (it's not necessary for all ships to feature every type):

> **#anim def ... #**

Plays continuously when the ship is active, so as such must be loopable.

> **#anim attacked ... #**

Plays once when the ship is attacked. Plays reverse when the attack is over.

> **#anim attackedloop ... #**

Plays continuously when the ship is under attack, so as such must be loopable.

> **#anim dock**n **... #**

Opens the dock denoted with the #**dock** n# parameter (see below). When a ship wants to enter or leave the dock, the animation plays once, and when docking is complete it plays once more reverse.

2.6 SHIP LABELS

For **human ships**, an area must be selected on the hull where the ship's ingame name will be displayed as a painted label.

To do this, the selected area's surface must be given a second texture layer, where the **__inscription__.tga** texture must be loaded. This contains a test label, to aid in setting the texture projection correctly.

It's important to give this layer an **Opacity value of less than 100** – 99 is perfectly OK.

2.7 GAME PARAMETERS

The model must be given a row of special areas which are used by the game for devices and effects positioning.

These must be given by adding polygons to specially named surfaces. The surface's name should have the following syntax:

#parametername ... # note

Sometimes material settings contain information as well as surface names (these have usually nothing to do with the original Lightwave representation.

These surfaces must always be placed on the **1. layer** of the object.

2.7.1 PARAMETERS FOR LARGE SHIPS

> #blink#

To place flashing lights, single-point polygons must be added to the surface, where each poly will represent the location of a single light. Multiple surfaces can be created if we want to add differently coloured and sized lights.

Denotation of surface parameters:

- **color**: light colour
- **luminosity**: light size
- **diffuse**: Duration of switched on lights, 100 = 1 second
- **specularity**: Duration of switched off lights, these two repeat periodically.

> #boosters#

The place for engine flares. Two-point polygons must be added. The first point of every polygon is the flare's centre; the second point is its vector. During ship movement, the flares will follow the correct direction with correct intensity.

There should be at least 2 engines on each of the ship's six sides. The rear engines should be larger and there can be more of them.

Some of the rear engines should be assigned to the **Reserve engines** (if the reserve engines are active, only these will flare, otherwise only the others will).

Of the **rear engines** nozzles, some should be assigned to drag plasma trails (the others will just flare). There should not be more than 4-6 of them, depending on ship size).

Denotation of surface parameters:

- **luminosity**: the maximum size of the flare
- **diffuse**: if less than 100, it's a reserve-nozzle
- **double sided**: if checked, it will drag a trail

> Shining engine parts

Engine operation can not only be shown by flares. Shining surfaces can be placed in the nozzles which adapt the colouration of the actual engine, and will shine and pulse according to performance.

These should be placed on the surfaces so that the polygons of nozzles pointing in different directions should be different surfaces. The @**boosters** ending must be added to the surface's name. The game engine will ascertain the direction of the engine from the normals of all polygons.

To denote a reserve nozzle, the **Reflection** parameter of the surface setting should be set to **1**. All other nozzles should be set to **0**.

The surface textures should contain luminosity maps, with the parts assigned to shine painted white on the luminosity map to allow proper engine colouring.

> **#slot** n#

Gun placement, belonging to the “n” weapon slot.

A single ship can mount 4-8 normal weapons (slots 1-8). At least two rotary weapons hould be assigned to each of these, so that they cover the entire field. A gun will roughly cover the half-circle arc above it, meaning two guns usually suffice, but care must be taken so they don´t fire through a ship´s hull.

For larger ships, a **ninth slot** must be denoted where a large, fixed siege weapon can be placed.

All gun placements are denoted by a right-angled triangle. The first point of the triangle (the right angle) is the origo of the gun, the second pointis it´s X-axis and the third is it´s Z-axis (it´s resting position).

Flak is placed in slots**10 - 13.** These are laser-half-spheres, able to fire within their entire fire arc without rotation. The places of these must be denoted differently; here, the **Z-axis must point upwards to the half-sphere´s top** and the X axis doesn´t matter at all.

Denotation of surface parameters:

- **diffuse**: gun size scaling

> **#angle** n#

Place for “n” slot not containing weapons.

Other devices do not have a visual body on the ship, but places and shooting directions must be denoted to them on the hull. Single-or two-point polygons must be given (meaning a device can be targeted from either one or more directions). The first point (the top of the cone) is the device´s place. The other point gives the axis of another cone, inside which the device is targetable. A cone´s angle an range from 90 to 120 degrees, depending on the device´s type.

Slot distribution:

- **14 shield generator**
- **15 main engine**, denoted with the **#boosters#** prefiz to be reserve nozzles; they should be denoted with **#angle 15#**, so that ships attacking the main engine will target all engines randomly.
- **16 reserve engine**, denoted with the **#boosters#** prefiz to be reserve nozzles; they should be denoted with **#angle 16#**, so that ships attacking reserve nozzles will target all nozzles randomly.
- **17-18 special engines**, these should be placed by one of the side or braking nozzles.
- **20-23 support devices**, should be placed on a ship´s sides or tabernacles.

> **#dock** n#

Two-point polygons must be denoted, showing dock entry and exit paths for the small ships. The first point must be inside the hanger and the other must be a bit from the hull at the appropriate distance. The docking small ships will travel along this path.

"n" denotes which docking bay door will open when someone wants to dock along a path. During docking, the above stated **#dock anim**n **....** # animation will play.

Multiple paths can be given to a single docking bay door, enabling several ships to dock at once through large doors.

> **#breach#**

These denote breaching points for enemy commandos. Two-point polygons must be denoted. The first point is the breach point on the ship hull, the other is placed away from the hull and shows the path on which the commando's approach and leave.

All ships must be given 4-6 such paths to enable multiple commandos to enter.

2.7.2 PARAMETERS FOR SMALL SHIPS

Under "small" ships", we mean all non-controllable craft transported in larger ships. These are fighter, gunboats, bombers, commando ships, rescue pods etc.

> **#boosters#**

Same as with larger ships, but they require less. No reserve engines are necessary, and one **trail-dragging** nozzle is enough for the rear.

> **#slot** n#

Generally there are **2-3 weapons** on a small ship, all containing a **gun**. There are **no rotating guns** at all, so triangles must be placed differently: the right-angle should be placed at some of the ship's parts denoted as weapons and the Z-axis should point forward. The direction of the X-axis is negligible.

2.7.3 GUN PARAMETERS

The models of rotating guns should consist of two or three layers.

The first layer contains the gun's body, rotateable in the X-Z plane, set to Z direction.

The second layer contains the traversable barrel, in a full rest (Y-axis) position. This layer must be parented to the first.

The optional third layer is for adding an un-animated (for example a plinth) to the weapon. It this layer exists, the first layer must be parented to this.

> **#center#**

Single-point polygon denotes the point around which the gun body and barrel rotate. (Meaning these two points must be together).

> **#fire#**

Single-point polygon, showing the point from which the gun's projectile is fired, so it should be placed in front of the barrel.

The barrel must lie along the rectitude connecting the #center# with the #view#-.

3 THE MODEL VIEWER

3.1 INTRODUCTION

The Viewer program serves the purpose of letting the designer to see the ingame outlook of the model during debugging and modelling.

3.2 FUNCTION

To view the model, it and its textures must first be converted to engine format. This process is automatic, and the converted files will be placed in the *temp* directory. If we've converted a model, only parts that have changed will be converted subsequently.

If we switch to a different program, then the Viewer will not continue to render the image, saving processor time, making it operateable with Lightwave and painting programs.

When we switch back, the Viewer will check the model and textures for changes. If it detects changes, those parts will be re-converted and it will render the altered model.

This means that the results of changed textures and modified models can be seen instantly!

3.3 USAGE

3.3.1 MODEL SELECTION

Start the Viewer from the Modification Tools program. A file selection window will appear, where you select the LWO or LWS file you want to view.

If several models are in the same directory, you can select between them using the **PgDn, PgUp, Home, End** keys, like in ACDSee.

Pressing F3 will bring up the file selection window, from which you can select any other file.

If you work on several models at once, several Viewers can be launched since only one is active a time.

3.3.2 BUMP MAP QUALITY

Full size 32 bit, half-size 32 bit or full-sized compressed bump map views of the model can be selected with the F9, F10, F11 keys. It enables visual decision making about the format the model's textures should be converted to.

3.3.3 GUN SELECTION

You can select which guns to view on the model. If we edit their structures or meshes, the changes will be displayed in the Viewer just like with ships.

F5 normal guns

F6 siege guns (**#slot 9#**)

F7 flak guns

3.3.4 VIEW CHANGE

The camera wil lalwyas point to the model´s centre, and can be rotated by pressing the **Right Mouse Button and moving the mouse.**

Zoom In/Out with the **mouse wheel.**.

3.3.5 SPECIAL EFFECTS

Pressing **E** will turn the engines On/Off and switches between main and auxiliary engines, enabling simple control of all **#boosters#** positions and sizes.

Numpad 2 / 8	rear / forward engine
Numpad 4 / 6	left / right engine
Numpad 7 / 9	lower / upper engine
Numpad 0	increase intensity
Ctrl + Numpad 0	decrease intensity

D will display different damage conditions of the ship.

L switches from one to more light sources.

3.3.6 ANIMATIONS

Animations can be played by pressing keys 1-9, playing the assigned animation.Ctrl´1-9 will play the animation in reverse. If the animation has a “def” name, than all “def” animations will be played and looped.

3.3.7 EXIT

By pressing Ctrl+Q Viewer can be closed.

3.4 ERROR MESSAGES

All errors during converting, modelling as well as lack of game parameters are displayed in a separate message window.

All messages are saved in the viewer.log file.

With some messages, the erroneous polygons or edges will flash on the model. Pressing **R** will switch this notification ON/Off.

3.4.1 COVERTER ERRORS

```
***** cannot find texture "név"
```

The used texture cannot be found in neither the model´s *textures* directory nor the *art\ingame\12_textures\meshes* directory.

```
***** "objektumnév": LOD layer cloned
```

```
A lOD-phase object is cloned in the Scene editor.
```

```
***** "objektumnév": LOD layer not parented
```

A layer is denoted as a LOD-phase (**#red ...** #) but is not parented to anything.

```
***** "objektumnév": LOD layer parented to a null object
```

A layer is denoted as a LOD-phase (**#red ...** #) but is parented to a null object.

```
***** "objektumnév": LOD layer parented to a LOD layer
```

A LOD phase is parented to another LOD phase.

3.4.2 MODELLING ERRORS

```
***** untextured polygon (yellow flashing polygons)
```

There are untextured polygons in the model. These polygons will flash yellow in the viewer.

```
***** bad texture projection: identical UV-s (red flashing polygons)
```

Two tops of a polygon have been given the same texture UV coordinates. This can happen at an erroneous Planar projection (the direction of the projection is parallel to the plane of the polygon). The polygons are flashing red in the Viewer.

```
***** more than two polygons shares the same edge (yellow flashing edges)
```

Two polygons share the same edge. This causes shadowing problems with the game engine. The polygons will flash in yellow in the Viewer. There can be two reasons to this problem:

- Modelling error, such as polygon duality or forgotten extra polygons. These can be simply removed.
- Two elements are joined so they share the same edge. If this is the case, one of the elements end points should be moved – one cm is sufficient. This will not influence any visuals significantly, and the converter will now be able to add the two elements without error.

3.4.3 GAME PARAMETER ERRORS

```
***** #....#: invalid keyword
```

Unknown parameter

```
***** #slot n#: slot number must fall into 1 - 13

***** #angle n#: slot number must fall into 14 - 28 or 20 - 26
```

Wrong slot number

```
***** #slot n#: slot must have at least 2 cannon places
```

At least two guns must be placed at each weapon slot on large ships.

```
***** #slot n#: slot can have only one cannon place
```

There can only be one gun at each weapon slot on small ships.

```
***** #slot n#: cannon pair must be placed on opposite sides
```

If a slot contains two guns, their tops should face into the opposite direction to enable them to cover the entire field.

```
***** unrecognized mesh type: no parameters defined
```

No parameter is added in the modeler, so the Viewer cannot see if it´s a ship or gun model.

```
***** #....#: parameter invalid on boats
```

Small ships cannot have this parameter.

```
***** no weapon slots defined
```

The ship doesn´t have a weapon #slot# .

```
***** no flak slots defined
```

The ship doesn´t have flak #slot#.

```
***** no shield angle defined
```

No shield angle is defined.

```
***** no main engine angles defined
```

No main engine #angle# is defined.

```
***** no aux engine angles defined
```

No reserve engine #angle# is defined.

```
***** no special engine angles defined
```

No special engine #angle# is defined.

```
***** no docking points defined
```

No #dock# is defined.

```
***** no breaching points defined
```

No #breach# is defined.

```
***** dock n has no door
```

The “n” docking point (#dock n#) doens not have a bay door animation (#anim dockn ... #).

```
***** dock %u door has no dock points defined
```

The “n” bay door animation (#anim dockn ... #) does not have a docking point (#dock n#).

```
***** non-main booster defined as emitter
```

A non-rear enigne #booster# is set to drag trails.

```
***** no emitting main boosters defined
```

No main engine nozzles #booster# are denoted to drag trails.

```
***** no emitting aux boosters defined
```

No auxilary engine nozzles #booster# are denoted to drag trails.

```
***** number of right side boosters less than 2
```

```
***** number of left side boosters less than 2
```

```
***** number of top side boosters less than 2
```

```
***** number of bottom side boosters less than 2
```

The given side contains less then 2 nozzles #booster#

```
***** no brake boosters defined
```

No braking nozzles #booster# are defined.

```
***** no main boosters defined
```

no rear nozzles #booster# are defined.

```
***** no lamps defined
```

No flashing light are defined.

```
***** no inscription added
```

There is no label on the ship.

```
***** cannon must have 2 parts
```

A gun must have two layers

```
***** center point not defined
```

No gun rotation centre (#center#) is defined.

```
***** fire point not defined
```

No firing point (#fire#) is defined for the gun.

3.4.4 PLANET TEXTURE TESTING

If we choose to load a TGA file, than it (and its __bump, __lum, __night textures, see the solar system editor chapter) will be interpreted as a planet texture by the viewer and will show it streched to a sphere.

Numpad 7,8,9	Increase surface RGB colour components
Ctrl + Numpad 7,8,9	Decrese the same
Numpad 4,5,6	Increase surface specular RGB colour components
Ctrl + Numpad 4,5,6	Decrese the same
Numpad 1,2,3	Increase atmosphere RGB colour components
Ctrl + Numpad 1,2,3	Decrese the same

4 CONVERTING

4.1 INTRODUCTION

The converter is for converting all designed models, icons and textures to a format interpretable by the game.

All models and textures created for the MOD can be organised into converter tasks and saved as such, so that later modifications do not require the re-setting of parameters. All changes can be converted with the single press of a button!

The Converter is launched from the Modification Tools program. It will load all *__work/art/converter.tpr* files, which contain the assembled tasks.

4.2 TASKS

Models and textures using the same parameters and same target directory should be assembled into one task. The source directory does not bneed to be the same; a task can contain any number of source files from different directories.

The *Pixel Format* and *Texture Options* parameters apply to the textures in the task, while the *Mesh Options* parameters apply to the models. For the sake of simplicity, textures and models should not be mixed within a single task.

It deserves mentioning thta converting models will not automatically convert their textures, this must be done in a separate task.

Creating a task is done with the **New** button. Enter the destination directory of the conversion into **Destination Directory**, with a path relative to the MOD´s root directory (**Destination Root**). Add files using the **Add** button.

For parameter settings of texture groups, see the *SAMPLE 02 Everything Change* example MOD.

4.3 MODEL PARAMETERS

Model conversion is possible from LightWave LWO or LWS files.

Generate Overlay – needs to be set for all ships which should display damage charring on their surface.

The other parameters are inopportune.

4.4 TEXTURE PARAMETERS

Textures must be in TGA or BMP format. Textures with alpha channel can only be produced from 32 bit TGA format.

From the textures selected from the **Add** window, only those which do not contain __lum, __bump, __spec endings are added; these will be automatically converted with their basic texture (if this is set in the Options).

Texture Pixel Format - 32 bit, 16 bit, or compressed textures can be made. Within this, the converter will choose from two sub-formats, depending on whether the source file contains an alpha channel or not.

See below about the sequence menu item.

Bump/Specular Format - bump textures look best in 32-bit format, but for the sake of memory saving, their size may need cutting down. The 32-bit half size and compressed full size options reduce memory usage to a quarter. The best results depend of course on the model; you can inspect all three options using the Model Viewer.

Texture Options

Generate Mipmap – tells the converter to generate a mipmap. The mipmap demands more memory, but increases render speed. It´s worth the use it for model and effect textures, but it´s redundant for example for icons, since they are generally not scaled.

Search Bump Map – search for a party with __bump ending for the texture and convert it to bump map. If this otopn is On, and no __bump file is found than the basic texture will be used as bump.. **Bump Scale** – the measurement of the bump effect.

Search Luminosity Map – if there is a __lum part for the texture, then it will be placed in the basic texture´s alpha channel as a grayscale image, used by the engine as luminosity map.

Search Specular Map – Only worthwhile if BUMP is On. If there is __spec part for the texture, then it will be placed in the basic texture´s alpha channel as a grayscale image, used by the engine as specular map.

4.4.1 ANIMATED SEQUENCES

The converter aids in the merging of animated sequence frames into a single texture as well.

Name the frames as namexxxx.tga, or BMP format, where xxxx is the frame´s serial number in four-digit form. The numbering should start from 1. Then create a task in the converter and add the first frame to it. Set the **sequence** in the Texture Pixel Format field. Starting the task will create a BMP containing all frames.

This is just a middle tool, the created BMP must be converted further normally.

4.5 STARTING THE CONVERSION

Converting will begin when you press the **Process All** button (all tasks) or **Process Task** button (only the selected task). If the **Process Changed Only** option is On, than only files changed since the last conversion will be processed. In the case of LWS files, converting will occur even if just a single component of the file has changed.

Incidentally missing target folders will be automatically created during the conversion.

5 EFFECTS

5.1 EFX TABLE

5.1.1 USAGE

Visual effects in the game are defined in the__*work/params/efx.xls* Excel-sheet.

The *SAMPLE 02 Everything Change* mod has good examples of usage; when creating a mod, it´s recommended to copy this file and modify it.

The table contains a macro, which exports the sheet´s content into a format readable by the game into the *universe/engine/mod_efx.ini* file. The macro is launched with the **Ctrl+Shift+X** combination.

The macro has to be run after each parameter modification to make the change in the game. You can view the result when using the Mission Editor without restarting it, just press the **Ctrl+Shift+X** combination (in the editor!), when it will re-read the file and the effects will be modified.

5.1.2 IDENTIFIERS

All effects have a unique identifier. The built-in effects in Nexus are placed in the 1-299 id-range. Effects defined in mods should be in the 300-399 range.The new effects are added to the existing ones.

The identifiers are mainly used by the tactical tables, (see tactics.xls, Data tables chapter), but the mission script can launch effects as well. In addition to the identifier, a size factor must be given in the tables, meaning that the same effect can be used in different sizes. The size factor will affect the sound volume of the effect as well.

All effects can be supplied with a fix or random sound effect as well.

5.1.3 EFFECT-GROUPS

The effects are grouped according to usage (it is exemplified in the example table).

> Ships

All effects concerning ship hulls. All ship classes can be denoted with a smaller and larger impact effect (**ImpactEfx**, and **BigImpactEfx**), a destruction effect (**FinalEfx**), an energy channel which shows when the ships are assuming formation. (**ChannelEfx**). These are set in the SHIPCLASS table in the *tactics.xls* sheet.

> Weapons

Weapon effects. All weapons can be denoted with three effects: fire (*tactics.xls WEAPONS sheet* **Start**), travel (**Travel**) and impact (**End**) effect. The fire effect will be displayed at the weapon's muzzle, like a flash and flare. The travel effect is a constant glowing light effect (could drag a trail) with projectile weapons, and a laser beam with lasers.

> Engines

All engine can be given two effects. The first is the movement light and flame effect, displayed at the boosters, the second defines a sound to the engine.

> Other Devices

Shields and other support deives can be denoted an On / Off effect (*tactics.xls* table **OnEfx, OffEfx**). These can only be sound denoting effects.

5.1.4 PARAMETERS

List of denotable parameters:

> Basic data

- effect identifier, 300-399

Description – the effect name, any text, without reference

Link – the number of another effect; denoting this will launch the second effect simultaneously with the first, producing complex effects. Any number of effects can be combined uisng the Link parameter.

NoModColor – the colours of most effects in the game are modified based on which race´s ships they are launched from (projectile, enigne, impact). The modifying colours are denotable in the *universe/tactics/races.ini* file, see the Data table.chapter. Placing a * in this column disallows colour modification for the effect.

Particle – particle effect name, which launches together with the effect. The second parameter is the size factor, modifying the particle speed and size, thereby affecting it´s enitre size. See more about particle effects below.

> FLARE

Lens flare.

Type -

1	star + glow + blue anamorphic stripe
2	star + glow
3	as 1, but with a “burn in” effect
4	star + glow + blue anamorphic stripe+ diagonal blue stripe
5	as 1 + lens reflections (diagonal row of circles on the screen)
6	as 5 + red circle around the centre
7	as 1 + red glow
8	only glow
9	star + horizontally streched glow and circle + blue anamorphic stripe
10	as 9, but with a “burn in” effect
11	glow with “burn in” effect
12	glow with black hole inside

Size - size

Colour - RGB colour, with components ranging from 0-1.

ColorB – Secondary RGB colour. If denoted, this is the colour of the secondary glow around the flare, creating a colour transition.

Heat, HeatB – alternative colour denotaion. Used instead of **Colour**. A single number decides the colour on a heat scale. 0 - black, 0.25 - red, 0.5 - yellow, 0.75 - white, 1 – blue-white, 1.25 - violet, 1.5 - black. An additional attribute of this denotation is the fade effect produced when running the effect – it will run through the scale until it reaches the denoted colour.

RFactor – lense reflection strenght

The four parameters below denote the flare´s time interval changes. With continous effects as projeectiles, these need not be filled in.

Delay - delay. This is the time in seconds after which the flare is revealed.

Attack - run. The flare reaches its full size and brightness after this amount of seconds.

Sustain - sustainment. This is how long the flare stays at maximum size and brightness.

Decay – run down. This is the time in seconds under which the flare winds down in size and brightness. In not given, it will stay constant indefinitely (until it´s switched off).

PulseFactor – vibration. The effect´s size alternates between the **Size** and **PulseFactor*Size**.

ShakeFactor – the camera shake factor measurment, applied when it´s close to the effect. The vibration will lessen squaredly relative to the distance.

Multiply - flare multiplication. Four parameters must be given:

1. duplicate number
2. radius of position spread.
3. size randomization measurement.
4. delay randomization measurement.

> ANI

Animated sprite.

Index – two animation identifiers must be given, from whose range one will be seleected when an effect launches. If the second identifier is 0, then no randomization will occur. The selectable identifiers are the following:

1	explosion A, plays once
2	explosion B, plays once
3	explosion C, plays once
4	explosion D, plays once
11	energy sphere, loops
12	mechanoid, loops
13	ionization, loops
14	stealth field (black wreathing), loops
15	fireball, still image
16	smoke wisp, still image
17	spark (small), still image
30	wreck A, still image
31	wreck B, still image
32	wreck C, still image
33	wreck D, still image

Sprites with the “still image” attribute are principally usable for particle effects (see below) rather as spinning parts and are listed here for the sakes of completeness. Additionally, own animations can be added to the system, see the next chapter.

Speed – animation speed factor. Generally, animations play at 25 fps. If the parameter is given, it will serv as an FPS multiplier.

Size - size

Colour - RGB colour, multiplying the sprite texture colour

Delay – see flare

Multiply - see flare

> STRIP

Strip-like effect with start and end point. The end points are set by the game depending on effect usage mode.

For weapons, the start point will be the gun nozzle, end point will be a point along the shot´s line, depending on the **Length** parameter.

As traveling effect for the projectile, it will be projected in the reverse direction of the projectile´s travel at "**Length"** length.

As engine effect, it will be projected from the booster nozzle opposite to travel direction for **Length** length.

As laser beam, it starts from the gun muzzle. The **Length** parameter does not have any effect, the point of impact will decide the end point.

Texture – the stripe´s texture

11 beam

12 flame

Length – length, see above

Radius – stripe width

Colour, ColorB, Heat, HeatB – see flare

Additive – giving a " * " will draw the effect with burn in effect.

Delay, Attack, Decay. Sustain, PulseFactor, Multiply – see flare

> SNAKE

Plasma trail dragged by the effect. Will project a node at set intervals. These nodes are connected with a STRIP-like effect beam. All nodes inherit the source´s colour and size when projected, but will have an own lifetime after.

If the **Speed** is set, than this is the speed at which nodes are projected from the effect´s actual position. If no speed is given, beams are only generated when the effect changes posiotion, with still nodes.

In the case of projectiles, the beam will be projected opposite to the travel direction, and in case of engines, in the booster´s direction.

Texture - the stripe´s texture

11 beam

13 smoke-ish beam

TexScale – texture streching/comprimation along the beam.

CrSect - if a "*" is given, then when a strip part is nearly parallel with the camera's looking direction, at the nodes of the strip smoke circles appear, rising the quality of the effect (for example when the camera goes through a ship's booster plasma). It's not worth to apply with small projectiles, because it is a resource consuming effect.

Period – period time between node projection. The basic unit is 0.1 seconds.

Speed – node projection speed.

SRadius, ERadius – beam start and ending width in the nodes, see**, NodeDec**

Colour, ColorB, Heat, HeatB – source colour (this is the colour inherited by the nodes, see flare).

Planck – alternative heat scale usage, closer to the real Planck-scale.

Additive – “burn-in” effect

Delay, Attack, Decay. Sustain – source colour and size change interval, see flare

NodeAtt – the amount of time in seconds when the node´s reach **SRadius** thickness and actual source colour.

NodeSus – the amount of time the node keeps the previous values.

NodeDec – the time after which the source fades or rather its thickness will **Eradius**.

> SPARK

Thin, beam-like spark mass, fbursting out in all directions evenly. This is an obsolete effect, it´s recommended to use particles instead.

Count – No. of sparks

Heat – beginning temperature of spark colour

HeatDecay – temperature decay speed

MinSpeed, MaxSpeed – the speed of the sparks will be randomized from this range.

> RING

Ring-like shockwave

Colour - a ring colour

Delay – time delay after start when the effect appears

Speed – spread speed

> FLASH

Flash covering the entire screen, if the camera is near the effect.

Type -

1 the flash colour is added to the image (burn-in effect)

2 the flash colour intesifies

3 the flash colour fades

4 inverting

Radius – if the camera is within this range, the effect´s measure will be maximal. It will decrease squaredly with distance.

Colour, Heat - RGB, or temperature colour

Delay, Attack, Decay. Sustain, PulseFactor – see flare

> SOUND

Effects can be given sounds, which are played from the start of the effect from the effect´s position. If the sound is looped, it will be played from the effect´s actual position.

Index – two sound identifiers should be given, from which range the sound effect should be randomized. If the second identifier is 0, no randomization will occur. For selectable indexes, see the Sounds chapter.

Level – Volume factor

Delay – Delay time until the sound plays.

5.2 ANIMATED SEQUENCES

Own sprites can be added to the effect system. To do this, the frames of the animation should be placed in a single texture, in a grid, starting from left to right, going top to bottom. Then define the animation in the *universe/engine/mod_ani.ini* file in the following syntax:

Ani identifier "texture name" blending playmode numframes framesperline numlines

The identifier can be in the 40-59 range. The texture´s name must be given with full path and without extension.

The blending will denote background interaction:

#BL_NONE
#BL_ALPHA
#BL_COLOR
#BL_ADD
#BL_ALPHAADD
#BL_MULTIPLY
#BL_FILTER
#BL_AMPLIFY
#BL_INVERT
#BL_ALPHAINVERT
#BL_INVISIBLE

#BL_INVALPHA
#BL_SUBTRACT
#BL_ALPHASUBTRACT
#BL_INVALPHAADD

The playmode set playing mode:

#APM_ONCE	plays once
#APM_LOOP	loops forwards
#APM_BDONCE	plays once forward, then in reverse
#APM_BDLOOP	plays continously back and forth

numframes – the number of frames in the animation

framesperline – the number of frames per line in the texture

numlines – the number of frame lines in the texture

5.3 PARTICLE EFFECTS

5.3.1 INTRODUCTION

The particle effects should be defined in the *universe/engine/mod_particles* directory. You can create any number of *.prt* extension files, which can contain any number of effects each.

The particle effect are not usable on their own, they have to be assigned to an effect in the table listed above.

5.3.2 PARTICLE

The basic element of an effect is the particle, a traveling imaginary point with a given lifetime and velocity.

A particle can be assigned a lens flare, animated sprite and a plasma trail each.You can define complex behaviours to each of their parameters.

A particle can in addition carry any number of emitters, meaning that any still or traveling particle can emit further particles. If no effects are assigned to the particle, then an invisible emitter will be created.

All particle effects are in fact particles with assigned emitters.

5.3.3 EMITTER

Emitters are projection sources for a given type of particle. The projection can be made in a geometrical formation, with an assigned period time or stepping time.

5.3.4 PARAMETER TYPES

A particle and its assigned effects have the following parameters:

scalar – any floating point value

3D vector – value denoting either position in space or velocity vector; it´s syntax is <x, y, z>, where all components are floating point values.

RGB colour - RGB colour component definition, syntax <r, g, b>, where each component is a floating point value between 0-1. Values less then 0 or greater then 1 will be considered as either 0 or 1 respectively when showing the colour.

Temperature colour – heat defined colour, known from the EFX-table. Syntax <temperature, brightness, 0>, where the temperature is a floating point value between 0-1.5 (0 - black, 0.25 - red, 0.5 - yellow, 0.75 - white, 1 – blue-white, 1.25 - value, 1.5 - black), brightness is a floating point value between 0 and 1 – the third component is not used.

5.3.5 RANDOM PARAMETERS

To make the effects more versatile, you can randomize effect parameters. This means that the particles projected by the emitter will have different parameters from a defined range. The following syntaxes can be used:

[min, max] – the value is randomized evenly from the value range, for example. [0.5,1.5], [<1,0.5,0>,<1,1,1>]

[min, max, power] – the probability depends on the given exhibitor. If the power > 1, then values closer to „min" will be selected with higher probability, for power < 1 the case is reversed. The further the power is from 1, the greater the shift.The power has to be a positive value such as [50,100,2]

[/min, max] – even probability selection, but the half the results will be inverted. This is useful for sprite rotation, when the rotation speed cannot go below a minimum value, while the random rotation direction gives a more spectacular result. Example: [/30,50]

5.3.6 PARAMETER RECEPTION

Some Read-Only variables are defined in the system for the case when you need to adjust a parameter to the parameters of an emitter or given particle. For example, you want to adjust the initial colour of a particle´s flare to an emitter´s flare, or you want the sprite´s colour to be identical to the flare´s.

The variables below should be used with the **e.**, or **p.** prefix depending if you want to refer to the particlés emitter or the particle itself. Example: e.flare.color, p.flare.color.

position

velocity

flare.size

flare.color

sequence.size

sequence.color

There is another emitter-variable, set by the game:

e.wind

This considers the ship´s velocity vector at projectile impact, so when used as a particle position modifier, you can make the effect move with the ship for a short time.

The values of these variables can be multiplied with a constant or random value. Vector or colour values can be multiplied with both scalar or vector values. Vector multiplication should be treated per component. Example:

e.flare.color*<0.8,0.5,0.1>

e.flare.color*[0.5,1]

p.flare.size*4

5.3.7 EFFECTORS

The parameters of effects assigned to particles are changelable with time. Effectors fulfil this purpose, and any number of effectors can be assigned to a paramtere. The syntax is:

{duration effect}

the duration can have the following forms:

intv start, end – the duration of the effect, in seconds. The two time values can only be constants.

dur duration – counted from the previous effector´s end time or rather the time during the effect is active from start. The duration can be a constant or random value.

cont – the effect shows under the entire duration of the particle.

Possible effect forms:

lerp value1,value2 – the parameter is fluctuated evenly from the beginning to the end within the range between value1 and value2.

lerp value1,value2, power – the parameter changes according to the exhibitor function in the range between value1 and value2 from the beginning time to its end. If the power > 1, the change will be slow in the beginning and rapid in the end, if the power < 1 then the opposite is true. The greater the difference from 1, the greater the speed change. The power has to be positive.

lerpto value – the parameter fluctuates from the start to its end time, from either the previous effector´s end value or the parameter´s beginning value respectively in an even fashion, until it reaches the given value.

vel speed – the parameter will fluctuate between the start and end time with the given speed.

The values given for effects can either be constants, random values or variables.

Examples:

The colour of a flare starts with white, then fades to red between the 2nd and 3rd seconds, then fades to black (disappears) after four seconds:

color <1,1,1> {intv 2,3 lerpto <1,0,0>} {dur 4 lerpto <0,0,0>}

The size of a flare will fluctuate randomly between 10 and 20, then will reach 50 under 5-8 seconds at a squared pace:

size {dur [5,8] lerp [10,20],50,2}

The colour of a sprite will begin with the flare´s colour, then will fade to the orange-filtered flare colour under 5 seconds.

color p.flare.color {dur 5 lerpto p.flare.color*<1,0.5,0>}

5.3.8 PARTICLE-EFFECT DEFINITION

Detailed description of parametering.

> particle-effect

PARTICLE "name"

particle-description

END

> particle-description

duration duration – the particle´s lifetime, after which it will vanish. Can be a constant or random value.

position vector – the particle´s position. Never give it a concrete value, since it´ll be decideed by the emitter´s position or the particle´s projection speed. You can use effectors on it, especially the **e.wind** parameter detailed above:

position {cont vel e.wind}

adds exactly the aforementioned speed to the particle´s movement..

Anyone of the blocks below can be ignored. If you only denote emitter blocks, then you create an invisible emitter.

FLARE

flare-description

END

SEQUENCE

sequence-description

END

SNAKE

snake-description

END

You can define any number of EMITTER blocks, enabling multiple particle types can be projected from the same source.

EMITTER

emitter-description

END

> flare-description

type type - lens flare type, see the effects table

behind – if given, no covering test will be made and the flare will only show in covered form. Useful for effects projecting a lot of particles, since it lessens render time considerably.

size, pulse, color, colorb, heatcolor, heatcolorb, reflectf – see description in the effect table. Value denotation is the following for each type::

[begining value] [effector]...

> sequence-description

Animated or still sprite.

type animation – animation-identifier, see the effect table. Can be a random value.

fps speed – play speed for animated sprites, frames per second. Basic value: 25. Can be a random value.

startframe picture frame – play begins from the denoted frame. Can be a random value.

size beginning size [effector]... - sprite size

heightf beginning factor [effector]... – the vertical-to-horizontal ratio of the sprite. The sprite can be distorted to a rectangle.

rotation beginning speed [effector]... – sprite rotation speed

color beginning colour [effector]... - sprite RGB colour

alpha beginning value [effector]... - sprite opacity. Only sensible with BL_ALPHA blending mode animations. From the built-in animations, this is true for animations 30-33 (wreckage).

heatcolor beginning colour [effector]... - a sprite colour on a temperature scale.

> snake-description

The efx table contains despription of the trails which projectile leave as they travel. This effect can be assigned to any particle.

type texture – stripe texture identifier, see efx table

period, attack, sustain, decay value – similar to the efx table´s **Period, NodeAtt, NodeSus, NodeDec** parameters. Can only be constants.

nodevelocity speed [effector]... – node projection speed. The projection is done in the opposite direction of the particle´s travel. Basic value is 0, which means the particle „draw2 the strip (trail) behind it. Fluctuation with effectors only affects node beginning speed and not already projected nodes.

startradius, endradius, color, colorb, heatcolor, heatcolorb value [effector]... – see the efx table. Fluctuation only affect nodes not projected yet.

texturescale strech – texture streching in the stripe´s vector. Can only be a constant.

crosssectcircles – see the **CrSect** parameter in the efx table.

> emitter-description

The three parameters below decide projection geometry. Only one of them is used. The centre of the forms is always the particle's actual position.

sphere radious – the particles will be distrubuted randomly inside the sphere with the given radious, and will shoot out at the radious vector.If the radious = 0, than they will shoot out from one point.

cone directional_vector angle – the particles shoot out from a cone pointing in the given direction, in random directions. The direction can be modified by the program. For ship and projectile impact, the particle effect's Z axis is rotated to the opposite direction of the impact. So, if we define such an effect that the cone point to Z´s direction, (for example cone <0,0,1> 90), then the effect will lash back to the opposite direction the shot came from.

cylinder directional_vector radious – the particles shoot out from a cylinder with the given radious and direction, traveling parallel to each other from random points inside the cylinder's base circle. The direction is modified by the program like with the cone.

The two parameters below decide the projection frequency. Use one of them, or neither.

period – particles will be released with this time interval, in seconds. Can be a constant or random value.

distanceperiod – if the emitter is moving, then particles will be released for each set distance traveled. For example, useful for effects dragging a smoke trail, since it produces an even trail regardless of emitter speed. If the emitter is still, no projection will occur. Can be a constant or random value.

maxcount – the maximum value of particles that are emitted. If it´s not given, the emitter wil project particles for it´s entire lifetime. If neither period or distanceperiod are given, then you MUST give this parameter – if this is the case than this is the amount of particles that´ll be emitted at once. Can be a constant or random value.

velocity speed – the speed of the projected particles. Can be a constant or random value.

Now comes the definition of the projected particles. A complete particle definition can be given in a block, so emitters can be defined as well. In this case the emitted particles will emit further particles themselves. This can be assigned to any depth.

EMIT

particle-description

END

6 MUSIC & SOUNDS

6.1 MUSIC

Own music should be placed in the *music* subdirectory in Ogg Vorbis format. Secondly, they must be defined in the *universe/engine/mod_sound.ini* file´s *MUSIC* section in the following syntax:

identifier volume_factor "music_filename"

The identifier can be any number between 30-49. The filename must be given as it is,with The filename must be given as it is, with extensions listed and without path references.

The music file can be launched from the mission script with the **playMusic** command by referring to its identifer.

6.2 SOUNDS

Sound effects should be placed in the *sound* directory in any directory structure. The format must be mono WAV, and you should strive to have as samll files as possible (for example, small noises can be saved in 8kHz format).

Secondly, they must be defined in the *universe/engine/mod_sound.ini* file´s *MUSIC* section in the following syntax:

identifier group flags "filename"

The identifier can be any number between 200-299. The filename must be given as it is, with extensions listed and without path references.

The game can only play a limited number of sounds at once. To enable different sounds to be played simultaneously, (so they cannot override each other), the sounds are sorted in groups and each group has a maximum number of sounds that can be played at once. The groups are the following:

1 big explosions (ship destruction, bombs), max. 2 sounds

2 explosion, max. 4 sounds

3 laser, max. 4 sounds

4 projectile shot, max. 4 sounds

5 engine and rumble, max. 2 sounds

6 other effects, max. 2 sounds

7 environmental sounds, max. 4 sounds

8 2D effects, max. 2 sounds

9 special effects, max. 1 sound

Further, defined sounds can only be played one at a time. For looped sounds, such as engines, only the sound source closest to the camera will sounds.

Sounds in groups 2,3,4 will played with different pitch at each playing to make the sound image more versatile.

Engine sound pitches are also altered, paralell with the current engine performance.

The flags should be given as a string of letters, or given a "-" sign respectively if we don´t want a sound to have a flag. The letters denote:

N non-3D sound, not placed anywhere in space so it will play normally. These sounds (and only these) can be in stereo format. If it´s not denoted, than the sound will be placed relative to the camera and sound source, using the actual sound system´s possibilites (4 speakers, Dolby Sorround etc.).

L looped sound. The WAV has to contain a loopable sound. If not denoted, it only plays once.

C the sound will only play if the camera is really close to the sound source (the volume will change more faster with the distance). Such is the rumble noise of the ships.

D the volume is not distance specific but is maximal at all times. It will still be directioned to the sound source.

6.3 DIALOGUES

These are the lines spoken by NPC´s. They have to be placed in the *sound/dialogs/xx* subdirectory, where xx is the language´s code:00 - english, 01 – german.

Their name shave to be the same as the dialogue´s defined name (see the localisation chapter). Their format is mono WAV. Their size should be as small as possible, since they will be loaded only prior to playing from the disc. Most of the dialoghues in the game are in 8kHz/8bit format.

6.4 BUILT IN MUSIC AND SOUNDS

The original music in the game is accessable with the following prefixes (the names show basically where we used them in the game):

1	menu- and solar system music
10	asteroid filed
11	empty space
12	mist
13	nexus
14	general danger
15	gorg threat
16	unknown danger
20	general battle
21	ghost
22	gorg battle
23	raptor battle
24	vardrag
25	mechanoid

The original sounds (in .ini format, see above):

```
 1 2 - "explosions\explosion_01.wav"
 2 1 - "explosions\explosion_02.wav"
 3 2 - "explosions\explosion_03.wav"
 4 2 - "explosions\explosion_04.wav"
 5 2 - "explosions\explosion_05.wav"
 6 2 - "explosions\explosion_06.wav"
 7 2 - "explosions\explosion_07.wav"
 8 2 - "explosions\explosion_08.wav"
 9 2 - "explosions\explosion_09.wav"
10 2 - "explosions\explosion_10.wav"
11 2 - "explosions\explosion_11.wav"
12 2 - "explosions\explosion_12.wav"

13 4 - "weapons\launch_01.wav"
14 4 - "weapons\launch_02.wav"
15 4 - "weapons\launch_03.wav"
16 4 - "weapons\launch_04.wav"
17 4 - "weapons\launch_05.wav"
18 4 - "weapons\launch_06.wav"
22 7 - "communication\radio_01.wav"
```

```
 23 7 - "communication\radio_02.wav"
 24 7 - "communication\radio_03.wav"
 25 7 - "communication\radio_04.wav"
 26 3 - "weapons\l_siege.wav"
 27 3 - "weapons\formation_attach_01.wav"
 28 3 - "weapons\formation_attach_02.wav"
 29 1 - "explosions\expl_ebomb.wav"
 30 3 - "weapons\l_tactical_01.wav"
 31 3 - "weapons\l_tactical_02.wav"
 32 3 - "weapons\l_tactical_03.wav"
 33 3 - "weapons\l_astro_01.wav"
 34 3 - "weapons\l_astro_02.wav"
 35 3 - "weapons\l_military_01.wav"
 36 3 - "weapons\l_military_02.wav"
 37 3 - "weapons\l_military_03.wav"
 38 3 - "weapons\l_military_04.wav"
 39 3 - "weapons\l_heavy_01.wav"
 40 3 - "weapons\l_heavy_02.wav"
 41 3 - "weapons\l_heavy_03.wav"
 50 3 - "weapons\flak_01.wav"
 51 3 - "weapons\flak_02.wav"
 60 4 - "weapons\plazma_01.wav"
 61 4 - "weapons\plazma_02.wav"
 62 4 - "weapons\plazma_03.wav"
 63 4 - "weapons\plazma_04.wav"
 64 4 - "weapons\plazma_05.wav"
 65 4 - "weapons\plazma_06.wav"
 66 4 - "weapons\blinder_launch.wav"
 70 4 - "weapons\photon_bomb_01.wav"
 71 4 - "weapons\photon_bomb_02.wav"
 72 4 - "weapons\photon_bomb_03.wav"
 73 4 - "weapons\gun_magnetic_01.wav"
 74 4 - "weapons\gun_magnetic_02.wav"
 75 4 - "weapons\atom_gun_1.wav"
 76 4 - "weapons\atom_gun_2.wav"
 78 2 - "explosions\expl_vacuum_bomb.wav"
 80 4 - "weapons\anti-g_gun_01.wav"
 81 4 - "weapons\anti-g_gun_02.wav"
 82 3 - "weapons\l_battle_01.wav"
 83 3 - "weapons\l_battle_02.wav"
 84 3 - "weapons\l_battle_03.wav"
 85 3 - "weapons\l_st2_heavy_01.wav"
 86 3 - "weapons\l_st2_heavy_02.wav"
 87 3 - "weapons\l_st2_heavy_03.wav"
 90 4 - "weapons\plazma_bomber_01.wav"
 91 4 - "weapons\plazma_bomber_02.wav"
 92 4 - "weapons\plazma_bomber_03.wav"
 93 3 - "weapons\energy wind_01.wav"
 95 2 - "weapons\blinder_01.wav"
 96 2 - "weapons\blinder_02.wav"
 97 3 - "weapons\mechanoid_ray.wav"
 98 2 - "explosions\expl_etorpedo.wav"
 99 1 - "explosions\Cataclism_expl.wav"

100 5 LC "engines\melyduborog 1.wav"
101 5 LC "engines\melyduborog 2.wav"
102 5 LC "engines\melyduborog 3.wav"
103 5 L "engines\drive_plasma_01.wav"
104 5 L "engines\drive_fusion_01.wav"
105 5 L "engines\drive_chemical_01.wav"
```

```
106 5 L "engines\drive_tactical_01.wav"
107 5 L "engines\drive_anti-g_01.wav"
108 5 L "engines\drive_ion_01.wav"
109 5 L "engines\drive_photon_01.wav"
110 5 L "engines\drive_small_01.wav"
111 5 L "engines\drive_a-mat.wav"
112 5 L "engines\drive_stealth.wav"
113 5 L "engines\drive_ghost.wav"
114 5 L "engines\drive_mechanoid.wav"
115 5 L "engines\drive_combat.wav"
116 6 LD "engines\drive_ip_warmup.wav"
117 9 D "engines\drive_ip_jump.wav"
118 9 D "engines\drive_ip_arrive.wav"

120 8 - "weapons\Datascanner_ON.wav"
121 8 - "weapons\Datascanner_OFF.wav"
122 8 L "weapons\Datascanner_LOOP.wav"

126 7 D "jumpgate\Gate Lock.wav"
127 7 D "jumpgate\Gate Unlock.wav"
128 7 F "jumpgate\jumpgate_open_01.wav"
129 7 F "jumpgate\jumpgate_closing_01.wav"
130 7 LF "jumpgate\jumpgate_openstate_01.wav"
131 7 - "jumpgate\jumgate_belep_kivulrol_01.wav"
132 7 - "jumpgate\jumgate_belep_kivulrol_01.wav"
133 8 NF "jumpgate\wormhole_enter_01.wav"
134 8 NF "jumpgate\wormhole_exit_01.wav"
135 7 L "communication\alarm4.wav"
136 9 NF "communication\trans_beeplead.wav"
137 9 NF "communication\trans_lead.wav"
138 9 NF "communication\trans_data_2.wav"
139 9 NF "communication\trans_beep.wav"

140 6 - "shields\shield_basic_up.wav"
141 6 - "shields\shield_basic_down.wav"
142 6 - "shields\shield_bomber_up.wav"
143 6 - "shields\shield_bomber_down.wav"
144 6 - "shields\shield_mask_up.wav"
145 6 - "shields\shield_mask_down.wav"
146 6 - "shields\shield_integrity_up.wav"
147 6 - "shields\shield_integrity_down.wav"
148 6 - "shields\shield_fort_up.wav"
149 6 - "shields\shield_fort_down.wav"
150 6 - "shields\shield_gorg_up.wav"
151 6 - "shields\shield_gorg_down.wav"
152 6 - "shields\shield_ghost_up.wav"
153 6 - "shields\shield_ghost_down.wav"
154 6 - "shields\shield_cloaking_up.wav"
155 6 - "shields\shield_cloaking_down.wav"
156 5 LC "shields\shield_cloaking_active.wav"

160 5 LC "engines\Locust Mother LOOP.wav"
161 5 L "engines\Locust_flying_warriors.wav"
162 4 - "weapons\Locust_ionballs_SHOT.wav"
163 2 - "explosions\Locust_ionballs_EXPLOSION.wav"
164 7 - "weapons\Locust Ionising 1.wav"
165 7 - "weapons\Locust Ionising 2.wav"
166 4 - "weapons\Locust Kamikaze.wav"
167 8 LC "weapons\Locust ShieldSucking.wav"
```

7 MISSION EDITING

7.1 SOLAR SYSTEM EDITOR

You can assemble solar systems and mission scenes with this editor.

7.1.1 LAUNCHING

If you wish to create a new solar system, create an accordingly names *system* file in the *universe/systems* directory, then copy the header parameters from another existing solar system. Enter the system´s name in the **Name** parameter. Changing the **Position** parameter will change the system´s location on the starmap. A new, empty solar sytem is now created.

Start the Solar System Editor from the Modification Tools program. You´ll see the galaxy map, from which you can edit the system by double-clicking it.

If the system in question is empty, the first step has to bee adding a Sun using the (**Ctrl+Shift+U**, see below) function.

7.1.2 PLANET CLASSES

When creating planets or selecting planet textures respectively, you´ll have to take into account the planet´s class, since the texture palette and model selection depends on it.

A	habitable planet/moon
B	non-habitable planet/moon with atmosphere
C	non-habitable planet/moon without atmosphere
D	gas giant, possibly with rings
E	asteroid, small moon
F	comet

7.1.3 BASIC OPERATIONS

Ctrl + S Save the system. Apart from the .system file,a.system.bg file will be created, saving the backgrounds and star patterns. Both have to be checked-out, or if the .bg didn´t exist, it has to be imported.

If the system has a nexus gate that has been changed, than the entire nexus will be rolled back, so it has to be checked out as well.

Changes to objects from the .scene files will **NOT** be saved, so permanent objects must sooner or later be placed in the .system file (objects created with the editor will be placed there automatically).

Ctrl + L reloads the data in the .system file, so all modifications done can be viewed from the text editor. Warning: All changes made on the screen will vanish, so ii is recommended to press **Ctrl+S** before using the text editor.

Ctrl + C Defines the system´s basic camera seting to the current one. The operation is only sensible if the camera points to the central sun.

Left Click selects stellar body. Further functions concern celestials only.

Left DoubleClick directs the camera to a stellar body. This means the celestial will become selected!

7.1.4 ADDING CELESTIAL BODIES

Ctrl + Shift + U,A-F adds a sun (**U**), or new stellar body from the given class (**A-F**) to the system, orbiting the selected stellar body. The celestial body becomes selected and the camera will jump to it. It´s parameters will be according to generation parameters.

U can be pressed in an empty system, creating a central sun.

Ctrl + Shift + Del deletes the celestial from the system. Warning: all childer objects of the celestial will be deleted, including stations and nexus gates!

7.1.5 LEVEL DATA

PgUp, PgDn semiMajorAxis – major orbit axis

Ctrl + PgUp, PgDn eccentricity - orbit excentricity (ellipse elongation)

The next three denote different rotation types, no need to be scared off by their names!

up, down inclinaction – orbit angle

Shift + up, down argOfPericenter – vector of sun-centre-point

Shift + PgUp, PgDn asendingNode – vector of ascending node

←, → meanAnomaly – orbit start position (orbit speed will be decided by the mother celestial´s mass, see below)

7.1.6 ROTATION DATA

Ctrl + up, down obliquity – angle of rotating axis

Ctrl + Shift + up, down longOfRotationAxis – other angle of the rotating axis

Ctrl + ←, → rotationOffset - rotation starting position

Ctrl + Shift + ←, → rotationPeriod – rotation speed (time starts, see above)

7.1.7 LOOKS

Shift + A-F	change the class of the celestial. Pressing the same key will generate random planet types for the class.
Numpad +, -	size, and brightness as well for suns
Ctrl + Numpad +, -	mass, influences orbit speed for all celestials around it (time starts, see above)
Numpad 4, 6	switching between textures/meshes for the class
Ctrl + Numpad 5	toggle key; changes whether the keys below set texture or glossiness colour. Glossiness is only good for textures with specular maps, otherwise they just look weird.
Numpad 1, 2, 3	texture filter / glossiness colour compentent increase (RGB)
Ctrl + Numpad 1, 2, 3	texture filter / glossiness colour compentent decrease
Ctrl + Numpad 6	toggle key; changes whether the keys below set atmosphere or ring colour.
Numpad 7, 8, 9	atmosphere / ring colour compentent increase (RGB)
Ctrl + Numpad 7, 8, 9	atmosphere / ring colour compentent increase (RGB)
Numpad *, /	temperature colour, only for suns

7.1.8 RINGS

R	selects ring texture or switch rings On/Off
Home, End	outer radius
Ctrl + Home, End	inner radious
Shift + Home, End	opacity

7.1.9 GAME OBJECTS

Ctrl + Shift + I	creates an asteroid field by the selected object. The field becomes selected as well.
Ctrl + Shift + T	creates a station by the selected object. The station becomes selected. It will be of the basic type, and assigned to Race 0. There is no way to edit these here, you have to do it from the Text editor in the corresponding .system file after saving.
Ctrl + Shift + N	creates a navigation point by the selected object. The point becomes selected as well.
Ctrl + P	relocates the object. The new location is denoted by a new selection (Left click).

Changing the **orbital data** of the above objects is done the same way as with celestials.

7.1.10 COSMIC BACKGROUND

The backgound colour is generated by randomizing four colours with different temperature and brightness Each change generates a new unique pattern, but the current pattern will be saved when saving. The same is true for stars.

F1-F4, Ctrl + F1-F4 changing the four components´ colours on a temperature scale (Ctrl-clicking decreases the same).The actual values will be displayed on the upper left part of the screen:

0 – black, 25 – red, 50 – yellow, 75 – white, 100 – blue-white, 125 – violet

F5-F8, Ctrl + F5-F8 changing the brightness of the colours, 0 – 100

F9, Ctrl + F9 number of stars.

7.1.11 STELLAR MISTS

Ctrl + Tab switching stellar mist On/Off. When displaying stellar mist,some of the above keys change subjects and will modify the actual stellar mists instead of celestials.

Ctrl + Ins insert new mist. The mist becomes selected

Ctrl + Del delete mist

1 - 9 selects numbered mist

←, →, up, down movement

PgUp, PgDn rotation

Ctrl + ←, →, up, down sizing

Numpad 4, 6 texture selection

Numpad 1, 2, 3 increase filter colour components (R, G, B, 0 – 100)

Ctrl + Numpad 1, 2, 3 decrease filter colour components

7.1.12 ADDING CUSTOM PLANET TEXTURES TO THE SYSTEM

Planet textures must fulfill the following criteria. The name must follow the:

planet_name_classes**.tga**

syntax. The name can be anything, but cannot contain „_-t”. Class is one or more letters, denoting to which planet classes the texture can be used. Example: planet_vulcan_A.tga.

In addition to the basic texture, you can create bump maps, (xxx__bump), specular maps (which have to be named xxx__lum due to technical reasons, since unlike models, it´s placed in the alpha channel of the basic texture). For habitable planets, a separate night texture can be created (xxx_night).

The planet textures should be converted to the *textures/planets* directory, when they will be added to the basic set in the solar system editor.

7.1.13 MISSION SCENE DENOTATION

Create a navigation point by the selected planet (**Ctrl+Shift+N**). You can set the scene´s position by adjusting the nav point´s orbital data.

To get an overview of the area from the selected point, just zoom in fully with the camera and rotate it around.

7.1.14 EXIT

By pressing Ctrl+Q the editor can be closed.

7.2 MISSION SCENE EDITOR

7.2.1 INTRODUCTION

The editor can create, delete and place all mission **SHIP, PIPE, EFFECT, OBSTACLE, PLACE, FLEETENTRY** objects. The parameters of these should be set as ususal from the text editor.

When creating a new mission, make an empty mission file in the *universe/mod_missions* directory, containing only the mission´s identifier and name. Add the scene´s selected nav point name to the **DefLocation**..

The Editor is launched from the Modification Tools program. The missions list will be displayed and you can select which mission to edit.

The SceneInit event will not run, so object generating equations such as **addNavigPoint**, **genAsteroidField** should not be used; such object(groups) can be created in the editor. If we want a randomly placed asteroid field in a mission, then you have to use the **genAsteroidField**-but this field cannot be handled by the editor

Asteroid fields placed incidentally in the **CREATEASTEROIDFIELD** szekcióban sections will only be shown as a background, since the asteroids themselves are generated randomly, so editing them is pointless.

When using the shortcuts listed below, make sure the 3D view is the active window when pressing the keys. Just moe the cursor above the 3D window to make it active.

7.2.2 SAVING

The saving process works like this: the original mission file will be saved unchanged in its entirety to the first **FLEETENTRY, POSITIONS, ENTITIES** or **DYNAMIC** block. The remaining part will be **deleted in its entirety** and **replaced by new data**.

Objects brought in from the **Solar system** will only have their positions saved to the **POSITIONS** section.

To save, press the corresponding button on the dialogue box, or press **Ctrl+S**.

7.2.3 CAMERA

Left DblClick binds camera to object

Left DblClick on empty space free camera

Right Button + mouse move. Rotate camera

Wheel, or Left + Right + mouse move. Zoom IN/OUT

You can jump to an object by double-clicking its name in the objects list.

7.2.4 SELECTION

In 3D view:

LeftClick select single object

Ctrl + LeftClick add or delete object from selection

LeftClick on empty space delete selection

In the dialogue box:

Use common Windows grouping commands (Left, Ctrl+Left, Shift+Left) in the object list.. Clicking **Select All** selects all objects.

With a **PIPE,** the entry and exit sphere appear as two different objects (**entry, exit**).

The Object properties window in the dialogue box displays the most important parameters of the selected object, or in case of groups, the first selected object. If we move the cursor over a different object, then the window will display the distance between the two objects as well as the objects´s heading and pitch from the selected object´s view.

7.2.5 CREATING NEW OBJECTS

Create new objects using the **Add Objects** button of the dialogue box or press **Ctrl+A.** The Add Objects dialogue box will appear. Select the desired object tyoe, or in the case of ships, race, ship typa and class.

Unlike the above cases, even objects that are not ships are given a name in order to make them identifiable in the object list. The only exception to this rule is the **FLEETENTRY**, since it´s identified with a number. There can only be five FLEETENTRIES..

Clicking the Add button will create the object in the origo and it will become selected as well.

7.2.6 ASTEROID FIELDS

Created from the Add Objects dialogue box as well. You have to denote:

- a Name Prefix, (the asteroids will get this name plus a three-digit number)

- three radii of an an ellipsoid space,

- a ship Class Interval, from which the generator will choose randomly, a Class Power, which will affect the randomization spread: the lesser the power, the larger the classes from which the randomization will occur.

- the maximum number of asteroids. The actual asteroids created can be lesser than this number if the space is too small, since the generator takes care not to meld objects together.

The group will be generated around an origo, but since all of it´s elements are selected, it can simply be moved to the desired location. Thanks to the common name tag, the entire group is easily handled (use Shift-grouping in the objects list).

7.2.7 DELETING

Selected objects can be deleted with either the **Delete** button of the dialogue box, or by pressing **Ctrl-Delete**. Objects imported from **strategic systems** cannot be deleted

7.2.8 HIDDEN SHIPS

For ships and asteroids, you can use the Dynamic switch in the dialogue box to set if the object should be in the **ENTITIES** or the **DYNAMIC** section. Groups can be configured as well.

7.2.9 AID GRAPHS

The aid graphs show object positions with lined graphs. These can be switched On/Off in the Display block of the dialogue box. In case of group selection, the same options can be applied to multiple objects.

Object	The visibility of the object (engine load can be decreased by turning objects off there is a lot of them)
Axis's	local coordinate axes and connecting lines
Bounds	**SHIP, OBSTACLE**: repelling circle, **PIPE**: entry and exit areas, **EFFECT**: radius
Combat	
Artillery	
Bomber	range around the objects

7.2.10 MOVEMENT

Holding down the Shift key enables you to move selected objects in a simple way, similar to the absolute movement in the game.

Since the amount of asteroids can be large, the circles denoting them will only become visible if the distance of the moved object is close to theirs.

7.2.11 ROTATION

Rotation is done by pressing down the **R** key and moving the mouse. The vertical and horizontal movements of the mouse will rotate the object accordingly. To rotate the object around the camera independently, press and hold the Left mouse button, then rotate move the mouse.

If more objects are selected, rotation will be centred in the middle of the selected object group, making rotation of groups simple and easy.

7.2.12 SIZING

Modify the scope of the **PIPE entry & exit, EFFECT, OBSTACLE** objects by pressing down the **S** key and moving the mouse horizontally. In the case of **PIPE exit** and **EFFECT**, holding down the left mouse button changes the diameter of the outer limits.

7.2.13 INTENSITY

Modify the **INTENSITY EFFECT** by pressing down the **I** key and move the mouse horizontally.

7.2.14 FORMATIONS

Ctrl + F the selected ships assume formation; the first ship selected becomes the leader. The ships must belong to the same race!

If a **FLEETENTRY** object is selected as well, the ships will assume positions as if they've arrived from this entry point, giving a primary picture of the situation after the jump.

The operation will not be completed if any other objects are in the selection.

7.2.15 EXIT

By pressing Ctrl+Q the editor can be closed.

8 SCRIPTING

8.1 THE SCRIPT LANGUAGE

8.1.1 GENERAL

The script language works with real numbers, arbitrary numbers of terms can be described in it, it knows the arithmetic (+-*/) and logical (&|) operators, it handles the operators' precedence and the parentheses accurately. You can define user functions in it. The language is insensitive to the differences between small, and capital letters, ItDoesn'TmAtter is the same as iTdOESN'tMaTTER. You can only write space in the command, if it is in quotation marks.

If you give a script command bad parameters (e.g. not the right type of object, bad race ID, not the correct shipclass, etc.), the will stop running with an error message. The message contains:

- in which line of which file the erroneous statement is,
  - the function call itself,
  - which parameter contained the error, and why. The 0. parameter in the object:command(…) type calls means the object itself
  At the same time the console window remains active, and in addition to the above mentioned ones, the detailed evaluation of the erroneous parameter gets listed here too. If this wasn't enough to spot the error, than arbitrary script commands can be given out for query. All this remains like this until you click in the dialogue box:
  - Abort, because you leave Nexus than
  - Ignore, because the not correctly called function returns with a 0, and the script will keep on running.

8.1.2 RULES

The script rests on the **events** and the **rules** that handle them which are sent by the system. These two things make a **ruleset**. The program sends events about the more important actions in the game that can be handled by the rules that are defined in the script. The rules can be named, so that they could be allowed or disabled, a **condition** can also be set, under which conditions should the rule run. The calling of events can be initiated by script commands too, that way a chain of events can be brought about, even timed too.

8.1.3 AUTOMATONS

The **ruleset** can contain a **state-machine**. The state-machine can handle several groups of rules. It arranges them in **states**, and only the rules in the same group can run simultaneously. Through this, it can be achieved that different rules run for the same events in different situations. Several state-machine can run simultaneously, they can be embedded in each other.

8.1.4 VARIABLES

The variables of the language are floating-points, but they can store the references of certain objects. All variables are identified by their name, this can contain letters, numbers, and almost any kinds of characters that are not the operators of the language. Everything can have variables, the universe, the mission, the tactical orders, concrete game objects, dialogues, selections, and events. These variables can be distinguished by prefixes, e.g. U.energy E.location. The prefixes are:

U – universe
I - episode
M – mission
O – command
E – event
P – Previous event. This always means the directly previous valid E. variables.
E.g. at the parametering of an event call, P. means the inbound variables of that event, from where we sent the call.
(*LocalEvent(DoSomethingWithShip, E.theship := P.ship);*)
This concept can be used in multiple depths, e.g. when we issue another call from a delay:
(*Delay(0, 5,*
Call(DoSomethingWithShip, E.theship := P.ship), E.ship := P.ship);
// here we give the ship of the delay into the subroutine's parameter
// and than we give the Caller's ship into the delay's parameter
....., 0);)
C – caller's variable
S – the currently selected object.
D – the variables of the dialogue. These can be set in the message() command, and the dialogue's expire event and the possible scripts of the options' can reach it by this prefix.
T – variables of the tactics

You don't always have to use the prefixes, at this point, your own variables of the current ruleset can be reached. So, in the universe it is enough to write npc1:rank, and not U.npc1:rank. Because the event and the object do not have their own rules, they can only be reached by prefixes. A few special variables in the objects just to show an example:

npc – is 1, if the object is NPC
ship – is 1, if the object is ship
effect – is 1, if the object is effect
this – reference to the object. It makes sense, when (e.g.) you want to execute a command on all the elements of a list, and for that you need the reference of the object. E.g.:
(*081___demo.mission, line 122*)
M.guiLevel – 0, all the guis are visible, by 1 and 2 fewer and fewer of them are visible

Some new, useful data types are introduced in the script language that are, similarly to the variables in the numeric values, can be stored in either in lists, or several different operations can be performed by them. Such is the string, the color, the rotating, and the space vector complex data type.

Constants can be added into the universe\constants.ini, and they can be reached from the script by the # character.

8.1.5 OBJECTS AND COMMANDS

All the ships, npcs, devices, etc. that occur in the game are **objects**, they can be reached as objects. Script commands can be defined onto these objects. That is the **obj:command** (*081___demo.mission, line 145*) format, or these can be submitted as parameters by the **command(obj)** format (*081___demo.mission, line 30*). The variables of the objects can be reached by the **obj.variable** or the **obj:variable** format (*081___demo.mission, line 29*). The objects are often in parent-child relation, in this case, the chain can be visited in **obj:1obj2:variable** mode. But in this case, the **obj1:obj2.variable form cannot be used.**

8.1.6 SELECTIONS

The lists, that is the **selections** are created in order to refer more easily to the whole of the objects that have common features. The list contains those elements that met all demands during the collate.

All selection has to possess a serial number (selection ID) for the sake of the references. **All ruleset has different selections**, similarly to the variables. Therefore, you can have for example a list of 312 elements in the mission, and in let's say six simultaneously running state-machines too. None of the rulesets can refer to the other's lists, they can only handle their own lists. At the checking of a list, or at the executing of a command that was issued on it, you have to pay attention whether that list is still valid (wasn't it deleted accidentally, or there is an element added that was not meant to be there). That is why it is practical to do all work that is related to the list right after the collating.

In the script language the **array management** can be carried out through the lists. A selection's dimension can be maximized by command, and certain elements can be reached by different commands. Such data that is not an object can be placed into such lists. The general selection handles only objects.

8.2 MISSION SCRIPT SYNTAX

Bulk text key word

[] optional part

<> part that must be filled in

8.2.1 SYNTAX

MISSION <num>

name <the mission of John Doe>
[description y **]**

//the rest of the line is comment

```
/*
the things that are between such opening and closing marks are comments
*/

[#include "filename"]    //we would paste the given file's content in one piece in here.
        //Nesting is also possible, so the included file can also
        //contain #include.
        //With this we can create e.g. such generally useful script-parts
        //that cannot be created by virtue of a subroutine.
        //But naturally not just the RULEs can be included, but arbitrary
        //parts of files too, except the parts within an action.
        //If we give a filename without a path, than it will search for the file in the same
        //folder, from where we gave it out.
        // I recommend the INC extension.

CREATELOCATION <celestial>                  //related celestial body
        semiMajorAxisKM <dist>              //distance from the celestial body
        inclination <axis>                  //dip angle of the orbit
        meanAnomaly <axis>                  //the angle on the orbit
END

//or

CREATEASTEROIDFILED <celestial>             //related celestial body
        semiMajorAxisKM <dist>              //distance from the celestial body
        inclination <axis>                  //dip angle of the orbit
        meanAnomaly <axis>                  //the angle on the orbit

        // comet
        fogDensity <density>   //the density of the fog 0-100, default: 20
        fogHiding <strength>   //the hiding effect of the fog, it has the same effect as
                               //if we would create a HIDE EFFECT that spreads over
the whole area                 //with this Strength. If it is not give, than
                               //it is the same as the FogDensity.
                               //This fog's color here will always be yellowish-purulent.

        // asteroid field

        classes <c1> <c2> <weight>   //the choosable asteroids and their weight
        density <num>                //the number of asteroids
        background "pic"             //background
        fogDensity                   //as above, but the default is 0.
        fogHiding                    //as above, but the default is 0.
        fogColor                     //the color of the fog (the density and the color
                                     //of the sun(s) also influence the outcome

        // nebula

        density 0                    //the number of asteroids

        //There is no separate nebula object, we have to create a foggy asteroid field,
        //where there is no defined background, and the Density is 0 (not the
FogDensity!!!).
        //At that time there won't be any asteroids, it is just fog.

END

//or
```

```
CREATESTATION
        Name "stat_Izebize"
        Race 15

        // central part
        SHIP
                StationMain
                name "ship_Izebize"
                class #cls_stat_CombatStation          //this is only an example!
        END

        // modules
        [SHIP
                StationBlock
                class #cls_sblk_Sensor                 //this is only an example!
        END

        SHIP
                StationBlock
                class #cls_sblk_Shipyard               //this is only an example!
        END]

        //platforms
        [SHIP
                name "ship_Plat1"
                class #cls_Platform                    //this is only an example!
                position <x> <y> <z>
                orientation <angle1> <angle2> <angle3>

        END]
END

HideLocation "name"   //The given background object (generally a planet)
                      //will be invisible during the mission.

RULES

        RULE  event <event>
              [name <nName>]              //it is only needed for enable/disable
              [condition <condition>
              //or
              :condition
                      <condition1>
                      <condition2>
              :end]

              :action
                      Command(par1, par2, par3, …);
                      Command(par1, par2, par3, …);
              :end
              [:else
                      Command(par1, par2, par3, …);
                      Command(par1, par2, par3, …);
              :end]
        END

        [RULE event cheat0
                      // (events that can be activated by ctrl + numpad keys from 0-9)
                      // it only works in the editor
```

```
            :action
                    Command(par1, par2, par3, …);
                    Command(par1, par2, par3, …);
            :end
        END]

END


[FLEETENTRY <index>                                 //(1-5)
        gate   posx posy posz  dirx diry dirz
        …
END]

[ENTITIES

        EFFECT
            category <category>
            center <x> <y> <z>
            radius <x>
            decayType  <type>
            decayRad <r>
            strength <strength>
            …
        END

        SHIP
            name <name>
            class <#shipclass>
            //or
            shipType <#shiptype>
            race <race>
            position <x> <y> <z>
            orientation <angle1> <angle2> <angle3>
            color <r> <g> <b>
            devices
                    <#dev1> <#dev2>
            ;
        END

        PLACE
            name <name>
            position <x> <y> <z>
            orientation <angle1> <angle2> <angle3>
        END

END]

[DYNAMIC

        FLEET
            name <name>
            race <race>

            SHIP
                    name <name1>
                    …
            END
```

```
            SHIP
                name <name2>
                …
            END
        END

        SHIP
            name <name3>
            …

        END

    END]

END
```

8.3 STATEMACHINE SYNTAX

8.3.1 HOW TO APPLY

The MACHINEs are named objects that can be reached in the script. With the

GetMachine("name")

Function we get the automaton's identifier, and this has to be used for the commands. The automaton's own states can naturally be manipulated by plain objectless calls from its inside:

ChangeState(state_name, 0)

The GetMachine command finds the automatons only that were directly in the given ruleset. Since from now on every STATE is a separate ruleset (see below), if it is called there, the GetMachine will search between those sub-automatons that were defined in the STATE. That is why you can set the neighboring automaton's state with the qualified calling marking the parent ruleset, e.g.:

M:**GetMachine**("Machine"):**ChangeState**(TheState2, 0)

8.3.2 THE SCOPE OF VARIABLES

Every automaton has it own variable- and selectionset. This means that from now on the variables of the comprising ruleset can be reached by qualified names, that is:

- during mission: **M.nname**
- in command: **O.name**

8.3.3 EVENTS

If the **LocalEvent** is issued from an automaton, it will **search only in the actual STATE events**. So we can only call such subroutines with this that were written to the given state.

Every state of the automaton is defined by an everyday ruleset, with the usual RULEs. From that it emerges that we can create more sub-automatons within a STATE, so a given working mode can be disassembled to several sub phases. The containing ruleset can be called from now on with the

PEvent(name, param_setting)

command. (Naturally this goes through all of the ruleset's automatons, just like the normal events).

The supreme ruleset can be called with the

RootEvent(...)

command.

The **ExpireEvent** of dialogues are called as **local events**, so it is important that the when such a dialogue is used, the event has to be handled in the given state.

If the automaton steps out of the state, than all the automatons that are under that state, will get into neutral state (so the actual Out part of all of them is executed).

8.3.4 SYNTAX

Name <ruleset name>

CONST
<Const>
<value>
...
END //defining the constant (so it doesn't have to be written in the consts.ini)

[**MACHINE** "name"

STATE <name> //this is the definition of a state, it can be several states

[**Tick** <value>]

[**RULE event Tick**
…
END]

[**RULE event In**
…
END]

[**RULE event Out**
…
END]

[**RULE event** <event-name>
...
END] – traditional Rule, with all of its possibilities
...

END

END] **//**several separated automatons can be defined, with local states

```
RULE    event <name>
        condition <condition>
  :action
        Command(par1, par2, par3, …);
        …
  :end
END
```

8.4 COMMAND SCRIPT SYNTAX

8.4.1 PURPOSE

It is possible to bind such **rulesets** to **objects**, which scope extends only to that object. These are the commands. They can be useful, if the independent control of the object is needed, or a multitude of objects need a control that works in the same manner, but at the same time it is independent from the situation. For the first case the game's F1-F11 commands are good examples, because they ease the control for the player. The second case might be needed, when it would be too complicated to create an AI that is embedded in the script of many ships that work the same way, since the given ships always have to be selected, their circumstances have to be examined, and make a decision according to that. In case of identical circumstances they decide identically, but the handling of the differences would make the AI more complex. In this case the same command can be bound to the objects, and they only have to care about their own circumstances.

8.4.2 STRUCTURE

The commands form individual **rulesets**. The two important rules of this ruleset are that handles the the CommandInit and the ClearCommand events. The CommandInit event is called automatically, when an object gets the command, the ClearCommand does so when the command is taken away. The command, as well as any other traditional ruleset can contain a state-machine, and it can contain all of the commands that we introduced in the mission scripting.

It is important to mention that though the Out event of the state-machine's actual event will always be effected, when the given state becomes inactive, if we use state-machine in the command and the command stops, the out events will not be effected, only the ClearCommand event. Naturally the out events work the traditional way when there is a state change in the command.

8.4.3 THE SCOPE OF THE VARIABLES

The command is defined on the given object, all of its variables can be reached inside that. If we create another ruleset in the command, the **variables of the command** will be defined with an **O. prefix**. The O itself is the object, to which we attach a command, so the object itself will be this way the **O.this**, while the object's variables will be available in **O.variable** mode. Within the basic ruleset of the command, it is not necessary to use the O. prefix, so the object can be reached with the help of the **this** variable, and all the other variables to the object can be reached this way.

8.4.4 ADDING COMMANDS TO THE SYSTEM

We can substitute the existing individual and group commands in the game with our own. Giving these is done in the *universe/tactics/mod_commands.ini* file, following the syntax below:

Single index "command name" target types ;

.....

Group index "command name" target types ;

.....

The index will be one of the individual or group command button´s number, this is the command that will be overriden. Yopu need to create two separate icons for the command in the *textures/icons/commands* directory with the *command_commandname_on.tex*, and *..._off.tex* name, denoting the active/inactive status of the command on the commands bar.

Target types should be separated with SPACE; these decide the types of objects against which the command can be issued. These are the following:

#TRF_OWN	own ship
#TRF_FRIEND	friendly ship
#TRF_NEUTRAL	neutral ship
#TRF_ENEMY	enemy ship
#TRF_GATE	wormhole

If none of the above are listed, the system will note that the command does not require a target (For example, like the Hold Position command in the game).

8.4.5 APPLICATION WITHIN THE SCRIPT

In case we replace not one of the existent commands that can be reached with the GUI, but we order objectcontrol to it as the part of the mission, then we can bind in

SendCommand(obj, command_name, command_num, var_set);

mode command_name object to ’obj’ object. The command_number can be left out, but you can write 0 to its place too, var_set is the enumeration of the, to be delivered variables’ assignment.

8.4.6 SYNTAX

Name „command_name”

CONST
…
END //Const

```
[MACHINE  “name”
        …
END]

[RULE    event <event>
  :action
        …
  :end
END]

RULE     event CommandInit
  :action
        …
  :end
END

RULE     event ClearCommand
  :action
        …
  :end
END
```

8.5 SCRIPT STRUCTURE COMMANDS

8.5.1 STRUCTURE

> **Choose(weight1, command1, weight2, command2, …., weightN, commandN);**

It chooses randomly from N pieces of possibilities, according to the weight (’weight1’ .. ’weightN’). If every weight is 0, than it doesn’t do anything; implicitly, if only one weight is bigger than 0, than it will choose that line – with this you can virtually implement the SWITCH-CASE command that is known from C. ’Command’ here and everywhere else can contain several script commands that are separated by semicolons.

> **ChooseFirst(condition1, command1....conditionN, commandN);**

It will choose the first true line, where the condition is true.

> **Delay(_UID, time, command, variable_set);**

The ’command’ is carried out after ’time’ seconds. If possible, don’t use this for delay in state-machine, rather make a temporary state, in which the delayed operation is done by its tick part (the delay can be given in the Tick), and it changes state immediately.

> **If(condition, command1[, command1]);**

It evaluates the term, and executes a line according to the outcome. It is not obligatory to give the false line.

> **IsValid(obj);**

It tells, whether the submitted 'obj' identifier, identifies an existing object. E.g. if a variable contains a ship identifier, through that you can check, whether the ship is destroyed or not.

> **When(_UID, time, condition, command);**

The 'condition' term is evaluated in every 'time' seconds. If true, the 'command' is executed; if false, than the queuing-evaluating cycle restarts.

> **While(condition, command);**

It executes the command, until the 'condition' is true (=1). It is suitable only for framing immediately executing cycles. In other words, the 'condition' value can be changed _ONLY_ by the command; so this cycle is not suitable for delaying (when the condition is an event, a timer, or a user action)!!!

In the Delay and When functions (because of technical reasons) the unique identifier must be given ('_UID', arbitrary string) that can occur only once in a ruleset. If you tried to use it several times, than the reader will signal error. It is recommendable to start the '_UID' arbitrary string with a '_' character, because this will make it easier to read in the debug text. You can write 0 instead of UID, then the identifier will be generated automatically. You can only give an identifier this way, if there might be a chance that the event has to be stopped early (with ClearTimer), WHICH MEANS NEVER, or if you want to make the event easily findable, from debug point of view.

8.5.2 VARIABLE MANAGER

> **Ceil(a);**

It rounds up, it gives the next biggest integer.

> **Cos(x);**

Cosinus of 'x'.

> **Exp(x);**

The exponential power of 'x'.

> **Floor(a);**

It cuts the fractional part, that is it gives the next smallest integer.

> **Log(x);**

The e based logarithm of 'x'.

> **Max(a, b);**

It returns the bigger from two values.

> **Min(a, b);**

It returns the smaller from two values.

> **Not(expression);**

Negation. Where in case ’expression’ is 0, its value is 1, if the value of ’expression’ is not 0, than the returned value is 0.

> **Pow(x, y);**

’y’-th power of ’x’, ’y’ can also be a fraction.

> **Rnd();**

It returns a random number between 0 and 1.

> **Rnd(a, b);**

It generates a random number, on which ’a’ <= rnd < ’b’.

> **Round(a);**

It rounds the numbers.

> **Sin(x);**

The sinus of ’x’.

> **Sqrt(x);**

The square root of ’x’.

> **Col(red, green, blue);,**
Col(red, green, blue, alpha);

The creation of a color with the given components (0 - 1), the default value of ’alpha’ is 1.

> **Col(brightness);**

Grayscale with the given intensity (0 - 1)

> **CHeat(heat);**

Fake temperature color (with intense colors) 0 black – 0,25 red – 0,5 yellow – 0,75 white – 1 bluish-white – 1,25 violet.

> **CPlanck(heat);**

Real temperature color, the scale is the same.

> **CInv(col);**

Complementary color, alpha is unchanged.

> **CAdd(col1, col2);**

The sum of the two colors, alpha will be 1.

> **CAdda(col1, col2);**

The sum of the two colors, summarizing the alphas too.

> **CMul(num, col);**

The triple of the color, alpha is unchanged.

> **CMul(col1, col2);**

The product of the two colors, the alphas are multiplied by each other too.

> **CLerp(col1, col2, fact);**

Its a transition between the two colors, in proportion to 'fact' by fact=0 its col1, by fact=1 its col2.

> **CBrig(col);**

The brightness value of the color e.g. col(cbrig(c)) is the grayscale equivalent of color c.

> **CEqu(col1, col2);**

1, if the two colors are equivalent, otherwise 0.

> **GetVal(obj, variable);**

It returns the value of 'obj' object's certain named variable.

> **IsName(obj, name);**

1 if the name of 'obj' is the given string.

> **Rot(heading, pitch, bank);**

It creates a turning with the given components.

> **V2Rot(vec);**

It gives the turning of the give corresponding vector. Since a direction is fully described by a heading-pitch binate, the bank (which is the turning around the fore-axis) component remains a free parameter. This is set to 0 by the function. The vector doesn't need to be normalized.

> **R2Vec(rot);**

On the basis of the turning, it creates a normalized vector (naturally, the bank component does not influence the outcome)

> **Requ(rot1, rot2);**

1, if 'rot1' = 'rot2' 0, if 'rot1' != 'rot2'

> **RNull();**

Zero turning.

> **SLen(str);**

The length of 'str'.

> **SCat(str1, str2);**

The binding of two 'str'.

> **SSub(str, start, length);**

A piece of string, from the 'start' indexed character in 'length' length. The characters are indexed form 0!

> **SNum(num, prec);**

'num' framed into string, with 'prec' accuracy to the decimal.

> **SNumPad(num, width);**

Integer, completed by preliminary 0s, to 'width' width.

> **SCmp(str1, str2);**

-1, if 'str1' < 'str2' 0, if 'str1' = 'str2' 1, if 'str1' > 'str2'.

> **SiCmp(str1, str2);**

The same, but it is case insensitive.

> **Set(variable, value);**

It sets the value of the variable.

> **SetVal(obj, variable, value);**

It sets the value of 'obj's certain named variable.

> **variable := value**

You can give a value to a variable by the := operator too. (*081___demo.mission, line 145*)

> **XfIdent();**

Identity transformation (it doesn't do anything).

> **XfTran(dx, dy, dz);**

Displacement. Of course you can write vector term in the place of 'dx' in this case, the other two parameters will be ignored (but they have to be given).

> **XfRotX(angle);**

Turning around x axis in 'angle' angle.

> **XfRotY(angle);**

Turning around y axis in 'angle' angle.

> **XfRotZ(angle);**

Turning around z axis in 'angle' angle.

> **XfRot(heading, pitch, bank);**

Transformation corresponding to the usual three component turning (here in the place of 'heading' you can write rotation term too, in this case the other two parameters will be ignored, but they have to be given).

> **XfScale(sx, sy, sz);**

Extension along the three axes with the given multipliers.

> **XfInv(xf);**

The inverse of transformation.

> **XfTran(xf);**

The transposed of transformation (it can be used instead of xfinv by transformations that contain only turnings, it demands much less operations).

> **XfMul(xf1, xf2, ...);**

A transformation that is the correspondent of the consecutive application of the enumerated transformations'.

> **Vec(x, y, z);**

It creates a vector with the given co-ordinates.

> **VNull();**

Returns a zero vector.

> **VRandomOrth(vec);**

It gives a random vector that is perpendicular to the given vector.

> **VRandomDir();**

It returns a unit vector that points to random directions.

> **VLen(vec);**

The length of the vector numerically.

> **VSqLen(vec);**

The square of the vector's length numerically.

> **VNorm(vec);**

The normalized form of the vector.

> **VNeg(vec);**

The negate of the vector.

> **VAdd(vec1, vec2);**

The sum of two vectors.

> **VSub(vec1, vec2);**

The difference of two vectors.

> **VMul(num, vec);**

The product of the vector.

> **VDot(vec1, vec2);**

The scalar product of two vectors.

> **VCross(vec1, vec2);**

The vectorial product of two vectors.

> **VEqu(vec1, vec2);**

1, if ‘vec1’ = ‘vec2’ 0, if‘vec1’ != ‘vec2’

> **VXform(vec, xf);**

The transformation of the vector.

8.5.3 SELECTION

> **AddItem(list, item);**

Adds an object to the list. ’list’ is the identifier of the selection.

> **AddList(list1, list2);**

Puts the elements of ’list2’ into ’list1’.

> **AllNumOf(list);**

The full number of elements of the selection.

> **Copy(list1, list2);**

Copies ’list2’ into ’lista1’.

> **CopyIf(to-list, from-list, condition);**

It copies the elements that are corresponding to the ’condition’ from the ’from-list’ to the ’to-list’.

> **DeleteSelect(lista_id);**

It deletes a list entirely.

> **Deselect(list, condition);**

It takes out such elements from the 'list' that fulfill the 'condition'.

> **Dim(list, size);**

It changes the number of elements in the list to 'size' number. Those elements that have an index below 'size' remain, those that are above, will be deleted. If the list was shorter, than it will be filled up with 0-s. So if you want to create an array that contains only 0-s, than you have to empty first.

> **Empty(list);,**
ClearSelect(list);

Deletes the given list.

> **ExecList(list, condition);**

It evaluates the term for all of the list's elements. This usually means the execution of the commands, but it can mean something else too, e.g. summarizing. E.g.: ExecList(1, Explode_listobject()) all objects that are in the list will explode, sum=0; ExecList(1, set(sum, sum+S.energy)) it sums the energy of the list's objects into a variable called sum.

> **obj:Colony();**
obj:Nexus();
obj:Npc();
obj:Race();
obj:Station();
obj:System();
obj:UType();

These return 1, if 'obj' falls into the given category.

> **FindUnder(list, root, condition);**

In the universe, it copies those elements that fulfill the condition, from 'root' to 'list'.

> **FirstNotScanned(list, race);**

In the given list, which is the first ship that the 'race' race did not scan yet. If there is no such ship, than the value is 0.

> **GetFreeSel();**

It returns a list ID that is empty, and it hasn't been used so far. You can use this, and the DeleteSelect() typically in subroutines to create and delete our lists.

> **GetN(list, index);**

It returns the element with the given index. It differs from PickN, because it returns the number in that case too, when it is not a valid object identifier (the PickM returns 0 in such occasions), so it cannot be used to handle arrays.

> **IndexOf(list, value);**

It gives the index of the first occurrence of 'value' in the list. If it did not occur, it gives -1.

> **InSelection(list, item);**

Is there 'item' in the list?

> **MaxList(list, n, condition);**

It leaves the biggest 'n' element in the 'list', according to the 'condition'.

> **NumOf(list);**

Number of elements in 'list', and only those elements that still exist.

> **Pick(list);**

It returns an element from the list, which was chosen by weights. If all the elements' weight is the same (this is the default), than it is random. If there are no elements, than its 0.

> **PickFirst(list);**

It picks the first element of the list, if there are no elements, than its 0.

> **PickMax(list, condition);**

It returns the max. elements according to the given condition.

> **PickN(list, index);**

It returns the given list's 'index'th element. The indexing begins from 0. E.g. to send out a ship's 2. bomber squadron: Select(0, S.weapon & S.owner=ship & S.devType=60); WeaponMode(PickN(0, 1), 1, target); it is suitable to make the selection between the same type of weapons on one ship.

> **RemoveFirst(list);**

It removes the first element of the list from the selection.

> **RemoveItem(list, obj);**

Removes 'obj' from 'list'.

> **Select(list, condition);**

Adds elements to the list from the scene; those elements, that fulfill the condition's term value (or it equals to 1). You can refer to the just examined object's variables with the S. prefix. E.g.: select(1,S.NPC) – it adds all of the NPCs to certain lists. Its return value is the number of elements.

> **SelectBoats(list, condition);**

It searches only between the small boats and rockets (S.boat & condition). Its return value is the number of elements.

> **SelectDevices(list, ship, condition);**

It searches only between the given devices of the 'ship' (S.device & S.owner=ship & condition). Its return value is the number of elements.

> **SelectEx(condition, command);**

It executes the 'command' on those objects that fulfill the 'condition'.

> **SelectShips(list, condition);**

It searches only between the big ships (S.ship & condition). It disregards the wreckages by the selection. Its return value is the number of elements.

> **SelWeightMul(list, mul, condition);**

Where the 'condition' returns with 1, it multiplies their weight by 'mul'.

> **SetN(list, index, value);**

It replaces the given indexed element with 'value', which cannot be only object identifier, but it can be an arbitrary number. This can be used to handle arrays.

> **SubList(list1, list2);**

Removes the elements of 'list2' from 'list1'.

8.5.4 EVENT HANDLING

> **Call(event, var_set);**

It gives event for the tactical subroutines. Since all the tactical subroutines are considered as one ruleset, the Timer can be used here too for delay, real time, and if you want to call a subroutine from another subroutine, than any the Call and LocalEvent commands is suitable for that. It can be given in 'var_set' E.variable:=value mode, in case of several values you have to separate them with semicolons.

> **ClearTimer(event_name);**

It deletes the local timer with the given name that is the event will not be activated. It is local, so the timer that was started by StartTimer can only be loaded from the mission script, and the timer that was started by UniverseTimer can only be loaded from the universe script by it.

> **Disable(rule_name);**

Disables the given event, with the given name. The use of quotation marks is obligatory.

> **Enable(rule_name);**

Enables the given event, with the given name. The use of quotation marks is obligatory.

> **EventEx(event, var_set);**

It is more than the simple event, because it will try to execute the mission's events, and if it does not handle them, then comes the script of the unit command. The event handling means that you set the E.Off variable to 1.0.

> **LocalEvent(event);**

It gives an 'event' event within a ruleset that does not have figurative variables. A mission, a unit command, an episode, a state-machine, and all the subroutines are considered as rulesets.

> **LocalEvent(event, var_set);**

It gives an 'event' event within a ruleset. Var_set are the event's variables, they can be separated by semicolons.

> **MEvent(event, var_set);**

It generates an event that can only be caught by the mission's ruleset.

> **PEvent(event, var_set);**

It generates an event that can only be caught by the parent machine's ruleset.

> **Return(value);**

It sets the return value of the event.

> **RootEvent(event, var_set);**

It generates an event that can be caught by all the rulesets down from the top machine.

> **Timer(event, time, var_set);**

It is the same as LocalEvent, but the event happens with a delay. 'time' is given in seconds. The variables of the event are set at the moment the command is issued, not when the timer lapses!

> **StartTimer(event, time, var_set);**

It is the same as MEvent, but the event happens with a delay. 'time' is given in seconds, and naturally it works only on the tactical screen. The variables of the event are set at the moment the command is issued, not when the timer lapses!

> **UCall(event, var_set);**

It gives an event to the strategical subroutines. Since all the strategical subroutines count as one subroutine, the Timer and the Delay can also be used here to set a delay, but they work in universe time.

> **UEvent(event, var_set);**

It gives an event to the script of the universe, so you can catch it in the rules.ini.

> **UTimer(event, time_in_years, var_set);**

The same, with the difference that the universe's rules will get the end-event. Here the 'time_in_year' time has to be given in years of the game. If you needed real time timing, than use the basic Timer command.

> **EnableGUIEvents(enable);**

Enables/disables the sending of events from GUI-operations, e.g. selection. It is disabled by default.

> **RequestFiredEvent(weapon, request);**

If request is not 0, than the weapon sends out a Fire event every time it fires, where E.location is the weapon itself.

The following call event only once, after that they have to be reactivated!

> **obj:tRequestCriticalDamageEvent(airace, damage);**

The 'obj' sends a CO_CriticalDamage event, if it is damaged to a certain percent. The mission and the command of the 'obj' will get the event.

> **obj:tRequestDetectionEvent();**

Actually this is a request, if you issue the command, then the next time a shot is aimed towards 'obj' ship, a CO_Detected event will be generated. This could be important by stealth ships!

> **obj:tRequestEnergyOnMax(system);**

A request if the energy system reaches the maximum (150%), it generates an event.

> **obj:tRequestNoShipsInRangeEvent(range, mode);**

Event request: if everything gets to 'range' range from 'obj', than a 'mode' defined ship, CO_NoShipsInRange event is generated. E.shipInRange gives the ship. The event is generated only once, for another observation you have to give the command again. range: 2 - combat, 3 - artillery, 4 – bomber; mode: 1 – own ship, 2 – hostile ship, 3 – hostile, or unidentified ship.

> **obj:tRequestOverforceEvent(limit);**

Request, if there is superior force against the given 'obj' ship within artillery range, than a CO_OverForce event is generated.

> **obj:tRequestShipInRangeEvent(range, mode);**

Event request: if a 'mode' defined ship gets within 'range' range to 'obj', than a CO_ShipInRange event is generated. E.shipInRange gives the ship. The event is generated only once, for another observation you have to give the command again. range: 2 - combat, 3 - artillery, 4 – bomber; mode: 1 – own ship, 2 – hostile ship, 3 – hostile, or unidentified ship.

8.5.5 STATEMACHINE

> **ChangeState(state, var_set);**

Changing of state in 'obj' state-machine. You can submit parameters in the usual way with the 'var_set' commands, that can be used by the new state's „In" rule.

(081___demo.mission, line 348)

> **ChangeTick(tick);**

It changes tick value of the automaton's current state.

> **GetMachine(„name");**

You get the automaton's identifier with this function, and this has to be used for the commands.

(081___demo.mission, line 348)

> **IsInState(state);**

is it in given 0/1 state?

> **LeaveState();**

It sets the automaton into 'neutral' (there won't be an actual state). Be careful with it!

8.5.6 MULTIPLAY

> **MpGameOver();**

Close the game. If the time runs out, it happens automatically.

> **MpGetPlayer(index);**

The identifier of given serial number player, 1 <= index <= mpGetPlayerCount(). These two functions can be used in one event for the examination of all of the players. Between the events, the Count and the players' index can change. But the identifier of a player will not change, until he is connected.

> **MpGetPlayerCount();**

The number of connected players.

> **MpGetPlayerKills(player);**

The number of ships that were destroyed/disabled by the player.

> **MpGetPlayerGain(player);**

The points that were gained by the player.

> **MpGetTeamKills(team);**

The number of ships that were destroyed/disabled by the team.

> **MpGetTeamGain(team);**

The number of points that were gained by the team. This cannot be used in individual type games, there you have to use the request by players.

> **MpGetTeamRace(team);**

It returns the race of the 'team'.

> **MpGetWinningCondition();**

Its return values:

0 - Kills, param: how many ships you have to shoot out

1 - Points, param: how many points you have to gain

2 - Last Man Standing, param: -

> **MpGetWinningConditionParam();**

It returns with the parameter of the winning condition.

> **MpHoldPrepare(hold);**

If 'hold' is not 0, than it suspends the countdown of PrepareTime. It carries on to 1. You can use it to bind the game's beginning to certain events instead of a time span, or you can combine the two.

> **MpIsIndividual();**

Whether the game is individual mode or not.

> **MpIsPlayerValid(player);**

Whether the given identifier identifies an existing player or not. You can find out this way, whether that certain player did disconnect.

> **MpEnableJumpIn(team, enable);**

It enables/disables the entering of new ships for the team.

> **MpEnableJumpOut(team, enable);**

It enables/disables the exiting of new ships for the team.

> **MpSetDead(player);**

You can declare a player dead from script. If this command issued, if the player had ships inside, those will jump out, and the 'player' won't be able to bring in new ships anymore. His voice connection will be disabled towards his teammates, and enabled towards the rest of the dead players (but he cannot speak to those, who did not choose teams YET, they form another voice group).

> **MpSetTeamGain(team, gain);**

> **MpSetPlayerGain(player, gain);**

The setting of team/player points.

> **MpSetWinnerTeam(team);**

> **MpSetWinnerPlayer(player);**

The indication if the winner team/player. You can set both of them simultaneously. If they are not called even once during the game, then there won't be a winner indicated.

> **MpSpawnPoint(team, cx, cy, cz, r1, r2, dx, dy, dz);**

Setting the spawn zone of the team, they jump into a random place within this zone. 'team' is the team, 'cx', 'cy', 'cz' give the center of the zone, 'r1', 'r2' are the radius of the sphere, 'dx', 'dy', 'dz' give the direction of the jump, and if this is 0,0,0 than they jump towards the center of the sphere.

> **SetSceneRadius(radius);**

Gives the maximum radius of the game, no ship can go beyond that.

8.6 MISSION SCRIPT COMMANDS

8.6.1 SCENE

> **AddMechaLegion(count);**

The scene is surrounded by a threatening mecha cloud, that counts 'count' members. They rustle right at the size limit of the antimecha shield. Rationally it should be given from the SceneInit.

> **AddNavigPoint(name, x,y,z, r,g,b); ,**
> **AddNavigPoint(name, x,y,z, r,g,b, race);**

It creates a navigation point, and returns the identifier of the created object. The object is ship type, but without any functions. Its shipclass is 0 (Navigation Point). But you can see its position, it appears in the shiplist (with the given color), you can give closing on it. It can be deleted by the DeleteShip command. IF 'race' is given, only the 'race' can see the navigation point. 'name' can be localized too.

> **Area(index);**

Returns the given indexed area.

> **AreaExplosion(posx, posy, posz, efxid, efxsize, substance, radius, damageshield, penetrateshield, damagehull, damagedevice);**

It creates an area effect explosion with the given parameters. It is practical to choose the 'efxid' from the existing bomb effects (they can be found in the End column of the area type weapons in the WEAPONS chart), its relative size can be set with 'efxsize' (1 is the original size). 'radius' is the range of destruction (see area weapons)

> **AssignSecret(obj, secret);**

It assigns a secret to an object, it can be deleted with 'secret'=0. If by the scanning the target ship has a secret assigned, it can be scanned normally. There should be only one secret on a ship at once.

> **celestial:uPlanetSurfBattle(enable);**

It turns on/off the effects on the planet that indicate that there is a fight on the surface.

> **ClearArea(index);**

It deletes the index number areaselection.

> **CloseGate(gate, delay);**

Closing of the gate with a delay.

> **EnableAI(enable);**

You can turn off the AI with this. 'enable' 1/0.

> **EndTraffic(traffic);**

It stops the traffic, and deletes the traffic object.

> **GameOver();**

It ends your agony.

> **GenAsteroidField(maxdb, cx,cy,cz, rx,ry,rz, classfrom,classto,weight, range);**

It generates a local asteroid field. 'maxdb': is the maximum amount of them, 'cx','cy','cz': the center of the field, 'rx','ry','rz': the size of the field, 'classfrom','classto': from which shipclass should it choose the asteroids from, 'weight': in which way should the selection rather fall, 'range': minimum distance between two asteroids.

> **GetHidden(name);**

It returns the 'name' named Dynamic object's reference.

> **GetSceneObj(name);**

It returns the named object's reference.

> **GetStation();**

If there is a station in the scene, it gives it, otherwise its 0.

> **GuiEnable(item, enable);**

It renders the enabling/disabling of the tactical GUI elements. 'item' can be:
#GUI_ManualPanel
#GUI_RepairPanel
#GUI_Objectives

#GUI_EnemyDevicesPanel
#GUI_EnergyButtons

> **GuiSelect(interface);**

Changing of GUI interface: 'interface': 0 nothing, 3 tactical, 4 only scene view

> **MissionObjective(objective);**

It returns the mission objective's state.

> **MissionObjective(objective, status);**

It sets the state of the mission objective.

> **MoveNPC(npc, ship);**

It relocates the NPC to another ship.

> **npc:uStepRank()**

It gives the NPC the next highest rank.

> **npc:uAddMedal(medal)**

It gives a medal to the NPC. 'medal' is at the time an identifier that goes from 1-10, it still needs constants, and names.

> **npc:uHasMedal(medal)**

Does the NPC possess the given medal?

> **OpenGate(gate);**

It opens the gate.

> **PlayEfx(id, size, positionx, positiony, positionz);**

It plays the given id effect (efx.xls, efx.ini) on the given place.

> **PlayMovie(name, endevent);**

It plays the movies\name.msq engine movie sequence. While it is played, the mission's time is stopped, so the timer events are not affected, and the duration of the movie does not count into their time. At the end of the movie, the so called 'endevent' event is effected.

> **PlayMusic(music);**

It fades out the actual music, and starts the one with the given serial number.

#music_Arrival_EmptySpace
#music_Arrival_AstField
#music_Arrival_Fog
#music_Danger_General
#music_Danger_Gorg
#music_Danger_Unknown
#music_Contact_General
#music_Contact_Gorg
#music_Contact_Raptor
#music_Contact_Ghost
#music_Contact_Vardrag

> **PreCacheNPC(npcid);**

This command can be mainly used in the Scene's init, it preloads the given NPC's animation, so that when it first sounds the program won't lag. Max. 10 NPCs can be loaded simultaneously. After that, at the loading of another one the most rarely used NPC will be deleted.

> **UQuitMission(success);**

It ends the mission, and exits to the solar system screen. 'success'=1, 0;

> **ScanEnable(enable);**

It turns on/off the global scouting. 'enable': 0,1

> **SelectShip(ship);,**
SelectGate(gate);

These make the ship/gate selected in the GUI.

> **SelectSquadron(weapon);**

It makes the given squadron weapon selected in the GUI, and deselects everything else. Only your own squadron can be given.

> **SetArea(index, centeritem, radius);**

The 'centeritem' is an object, and from that a 'radius' radius circle gets the 'index' given area. This can go from 0-9, but it is not the same as the selection's indexes.

> **SetAreaAbs(index, centerX, centerY, centerZ, radius);**

Just like the previous one, but here the 'centerX', 'centerY','centerZ' give the center.

> **SetFogColor(r,g,b);**

'r','g','b': 0-1, this color is modified by the color of the system's suns.

> **SetFogDensity(density);**

The density of the fog 0-100.

> **SetListOrder(index, type);**

It sets the shiplists order types.

'index': 0 – own ships, 1 – different ships; 'type': 0 - size, 1 - range, 2 - damage, 3 - priority

> **SetListPriority(ship, priority);**

It sets the ship's priority. The higher it is, the higher it will be on the list, default is 0.

> **SetMotionFactor(factor);**

The speeding up/slowing down of animationspeed. 1 is the normal speed.

> **SetObjectivePrimary(objective, primary);**

If 'primary' is not 0, than the given indexed objective gets to be primary, otherwise secondary.

> **SetOwnRace(race);**

Changing of the GUI controlled race (not identical to PlayersRace!!!).

> **SetPlayersRace(race);**

The player gets the 'race' race.

> **SetScreenFilter(enable);**

It puts a red filter on the picture. 'enable': 0,1

> **SetXPCategories(skill, cat1, cat2, cat3, cat4, cat5);**

It redefines the XP points of the 'skill' in the given mission (the default values are in the tacticsbase.ini)

> **ShowMissionStatus(page);**

0 - close, 1 - event log, 2 - objectives, 3 - briefings

> **SoundEnable(enable);**

It turns on/off the sound effects and the dialogue sounds. It only affects the current mission.

> **StartCountdown(from, endevent);**

A timer appears on the tactical screen that counts back from 'from' seconds, and it gives the 'endevent' at 0, if it is not 0.

> **StartTraffic(list, density);**

It can be a little help to make the atmosphere of space-stations, and group of ships living. The given select list has to contain a group of ships. Between/around the ships will start the traffic smaller ships. Density gives how many small ships can be in the air at the time.

> **StopCountdown();**

It stops the countdown.

> **Transmission(fromship, toship, duration);**

An effect for Angel's back and forth loading. I suggest the time span should be 5 seconds.

8.6.2 JUMP

> **AppearFleet(dynamicfleet, list);**

It brings into the mission all the ships of one FLEET that is in the DYNAMIC section. It enumerates the loaded ships in the given list. The fleet object can be requested by the GetHidden, but for that you have to give a name to the fleet.

> **JumpInShips(list, fleetentry);**

It jumps all the ships of a given list, according to the fleetentry's defined way. It can only be used for those ships that are already in the scene, so in practice, they have to be applied in a way that first you have to bring in the ships that were given in the DYNAMIC with appear, and you collect them in the list.

> **SetFleetEntry(index, gateposx, gateposy, gateposz, gatedirx, gatediry, gatedirz);**

It makes it possible to set the FLEETENTRY's gateposition during the mission.

> **StartJump(listid, leadership, gate, express);**

This is recommended for starting jump formations. Here, 'gate' is the targeted gate, if it is 0, then there will be a local jump, within the system. If express is not 0, than the formation will not come together, but the ships will be sucked immediately into the gate. Otherwise the leader automatically approaches its position before the gate, and after all the ships have taken their places, the jump starts. It returns a formation identifier.

8.6.3 OPERATIONS OF RACES

> **ForbidFirePurp(race, targetunit, purpose, forbid);**

If 'forbid' is not 0, then the ships of the race can use weapons by FREE fire against 'targetunit' that are not suitable for 'purpose'. You can disable several purposes simultaneously. It can be removed by 'forbid'=0. Disabling is not valid on tFireTo's target.

> **GetNavigPoint(ship, race);**

The navigation point of a ship that was created for the given race. (You can give several navpoints to one ship too, one for every race, from their point of view, it is currently undetected.) If the ship is detected towards the race, than it gives 0.

> **GetRelation(airace, race);**

It returns the relation's state.

> **Race2civilization(race);**

The civilization of the given race.

> **SetRelation(race1, race2, relation);**

Setting of relation from'race1' towards 'race2'. This is not automatically back and forth state. 'relation':

UREL_OWN	0
UREL_FRIEND	1
UREL_ENEMY	2
UREL_NEUTRAL	3
UREL_FRIEND2	4
UREL_FRIEND3	5
UREL_NEUTRAL2	6
UREL_NEUTRAL3	7

8.6.4 MESSAGES AND POSTINGS

> **Dialog(dialog, location, pri);**

It starts a dialogue. The higher priority message appears automatically. If the 'location' is 0, than the message comes from the place of the NPC that was given in the dialogue.

> **DeletePendingDialogs(actual);**

It deletes all the pending messages. If 'actual' is not 0, than it will close the currently played message too.

> **HideDialog(dialog);**

It hides the dialogue.

> **NoTitle();**

It hides the title texts.

> **NoTutorial();**

You can hide the tutorial messages with this command.

> **Title(date, system, planet, text);**

Title, ‘date’: whether it posts the date, ‘system’: the system, ‘planet’: the planet. It gets localized subtext from the text.ini.

> **TitleNoTime(date, system, planet, text);**

Title, it does not disappear by itself. If one of the first three parameters is true, it posts the fitting data too.

> **Tutorial("title", "text", position, "pointto", event1, event2);**

The position can go from 1-5, it meant the screen’s corners, and center. pointto is the name of a GUI object, where the arrow has to point. In case of an empty string (’’) there is no arrow. event1 and event2 is effected, if the player presses the 1. and 2. key on the tutorial screen. If either of them is 0, then the key itself wont appear either.

PointTo:

```
// Own Ship List
OSL
OSL.GroupBtnX            // X = 0..3
OSL.SortBtn
OSL.SortDirBtn
OSL.List.ShipName        // ShipName = ship IDName
OSL.List.ShipName/Idx    // Idx = Squadron Index

// Own Ship Panel
OSP
OSP.Head.SwitchBtn
OSP.Head.NPCBtn
NPCPanelX                // X = 1..4
OSP.Energy.XXX                  // XXX = Support, Engine, Shield, Weapon
OSP.Energy.XXX.BtnY    // Y = 0, 1, 2
OSP.Fire           // the part where the number of torpedoes are
OSP.Fire.TorpedoBtn
OSP.Fire.MissileBtn
OSP.Behaviour
OSP.Behaviour.BtnXXX   // XXX = Aggressive, Defensive, Cautious, NoFire

// Own Ship Buttons
OSB
OSB.InfoBtn
OSB.ManualBtn
OSB.RepairBtn

// Manual Panel
MP
```

```
MP.Shield
MP.Shield.ShieldBtn
MP.Weapons
MP.Weapons.WeaponIDName
MP.Supports
MP.Supports.SupportIDName
MP.Engines
MP.Engines.EngineIDName
MP.MovePanel
MP.MovePanel.CombatBtn
MP.MovePanel.ArtilleryBtn
MP.MovePanel.BomberBtn
MP.MovePanel.Slider      // the slide on the move panel

// Repair panel
RP
RP.Supports
RP.Supports.DeviceIDName
RP.Engines
RP.Engines.DeviceIDName
RP.Generators
RP.Generators.DeviceIDName
RP.Weapons
RP.Weapons.DeviceIDName

// Enemy Ship List
ESL
ESL.DevicesBtn
ESL.InfoBtn
ESL.SortBtn
ESL.SortDirBtn
ESL.List.ShipIDName
ESL.ShipIDName/SqdIdx
ESL.GateIDName

// Enemy Ship Panel
ESP
ESP.Ship.shipname          // 'shipname' the name of a ship

// Enemy Devices Panel
EDP
EDP.Shield
EDP.Shield.DeviceIDName
EDP.Engines
EDP.Engines.DeviceIDName
EDP.Supports
EDP.Supports.DeviceIDName
EDP.Weapons

// command bar
CmdBar
CmdBar.BtnX      // X = 0..13
```

8.6.5 CAMERA HANDLING

> **CamBoatView(boat);**

Rocket/bomber/commando view. If boat=0, than it changes back to normal view.

> CamGoRound(round);

It controls the evading feature of the camera, whether it should avoid the ships. During the game it is a good idea to avoid the ships, but in some special views, it is worth to turn it off, because this way you can get closer to the ships. 0 – no avoiding 1 – it only avoids the ship you are looking at (it is defined only by traditional camera commands, in case of camLocate, there is no ship that you could look at) 2 – it avoids every ship. Don't forget to set it back to 2 after the movie.

> CamLinear();

The camera's movement is accelerating/decelerating by default. This is not favorable in some views, because it wouldn't look too dynamic, if the camera started from a static point of view. The effect of this command is that the camera movement that is followed by it, will create a linear speed movement. In case of CamLocate, this also means linear movement, while in the other commands, the turning around the looked point will have a linear speed. The effect comprises to one command, so if you want to create a complex linear movement, than you have to issue it before every command.

> CamLocate(fromobj, mode, x,y,z, atobj, mode, x,y,z, roll, time);

In 'time' time it moves the camera to 'from', while it moves the looked point to 'at', and it brings the camera's turning around the axis into 'roll'. This command can be mixed arbitrarily with the aforementioned camera movement commands.

If obj=0, than x,y,z define absolute co-ordinates. If obj is a ship, than depending on the mode: 1 - x,y,z are relative co-ordinates for the ship's center; 2 – the position is defined by the ship's own coordinate system, that is the point increases the turning of the ship too (Z is the fore-axis of the ship, Y is towards the top of the ship, X is towards the right side of the ship); 3 – the ship's bounding box coordinate system is valid. This is a special system that was stretched to the ship's size, where the (-1,-1,-1) - (1,1,1) cube is suitable for the ship'S bounding box. This way, you can easily create views that are independent of the size of the ship. The axes are the same as before. The ship can be a boat too, you can create bomber views that are different from the others. If object is not a ship, then x,y,z are relative co-ordinates for the object's position. If you give an object, than the point will follow the object's movement from than on, according to the given relative position.

In case mode is a negative number, than its absolute value will be used for the mode, but in that case x,y,z means moving compared to the previous position. If you give different obj, or different mode than the previous one, than it keeps the position's absolute place in space (that is its former place will transform into the new co-ordinate system), and this is how the moving is added (the moving is always understood in the new system).

> CamLockPos(lock);

If lock is not zero, it binds the camera to its current position, but it leaves its turning free, so you can create look after views for the ships. lock=0 releases the binding.

> CamLookAt(ship, time);

Changes the watched ship. If its 0, than its disengagement.

> **CamMoveAndTurn(lookedship, distance, turntoitem, roll, time);**

The looked ship will be lookedship, from which the camera is in 'distance' distance, and it turns in a way that the turntoitem will be visible.

> **CamMoveTo(positionx, positiony, positionz, lookatx, lookaty, lookatz, roll, time);**

The 'position' is the intended position of the camera, while 'lookat' is the looked point.

> **CamMoveToShip(positionx, positiony, positionz, lookedship, roll, time);**

Here you can define a ship as a looked point, which will turn into the looked ship in the game too. It follows from this that the position gives only a starting position of the camera, since when the ship moves, it follows (it will keep the relative distance).

> **CamOrient(ship, heading, pitch, distance, time);**

It sets the camera to a certain position according to a ship, in given time, the turnings are to be understood relative to the ship's fore-axis. After the operation, the ship will be the looked ship. It doesn't set roll=0, but it leaves the actual roll.

> **CamOrientAbs(ship, heading, pitch, roll, distance, time);**

It is similar to the CamOrientRoll, but the turnings are not given according to the ship's fore-axis, but in the absolute co-ordinate system. Distance is to be understood according to the ship.

> **CamOrientRoll(ship, heading, pitch, roll, distance, time);**

It is the same as the CamOrient, but here you can set the end position's roll too. (the CamOrient is roll=0).

> **CamRoll(roll, time);**

The relative changing of the roll.

> **CamShift(deltaheading, deltapitch, deltadistance, time);**

The moving of a state according to the looked point. heading, pitch: turning horizontally and vertically, in degrees. distance: distance, exponential factor.

> **CamTargetView(item);**

The camera always turns in a way that item can be seen too. (see Ins)

> **CamTripleMode(mode);**

In case of triple Viewport, the setting of the view. mode: 0 – panorama, 1- triple view

> **CamTurnTo(item, time);**

It turns the camera in a way that it does not change the looked ship, and that the item can be seen (similarly to CamTargetView, but after this the camera won't follow the object).

> **CamWeaponView(weapon);**

If weapon=0, than normal view. Naturally the CamBoatView can be used henceforward to single boats. The two commands supervise each other.

> **ChangeLocation(location_id);**

It brings the camera to a new place.

> **FadeScreen(level, time, endevent);**

It fades in/out the screen. 'level'=1 fade out, 'level'=0 fade in, in 'time' time. If 'endevent' is not 0, than at the end of the fade out this event will be effected in the mission script. The fade out happens UNDER the GUI layer, so it is better if the GUI is hidden.

8.7 SHIP REALTED COMMANDS

8.7.1 GENERAL

> **Add2Fleet(ship);**

'ship' ship is added to the fleet.

> **AddBreachPoint(ship, innerx, innery, innerz, outerx, outery, outerz);**

It adds an entrance place to the ship. I created this, so when you give certain navigation points a breach, you can send out commando like vehicles to them, which after that 'sit' on that point. 'inner#' is the place of the entrance line's first point,'outer#' is the last point's place of entrance line, and it is not expressed in the ship's co-ordinate system.

> **AnimateShip(ship, animate);**

It starts/stops the ship's default animations (turning parts, etc.), and the blinkers. The change is gradual, at the moment its accelerating/decelerating time is 10 seconds.

> **Appear(ship);**

It makes the ship appear, and thus it gets into the game.

> **BlinkShip(ship);**

It flashes the ship's icon.

> **ClearLampSignal(ship);**

The lamps return to their original functions.

> **CreateShip(shiptype, race, name);**

It creates a ship with these parameters, and it places it into the origin.

> **DeleteShip(ship);**

It makes the ship disappear, without any fuss.

> **Disappear(ship);**

It takes the ship out of the scene, and it gives back an identifier, with which, you can make it appear again (the original identifier cannot be used any longer). The ship cannot be found after that neither with get, nor with select commands, similarly as if it were in the DYNAMIC selection. The only difference is that not even the GetHidden command finds it, so its identifier has to be stored, if you want to see that ship ever again. E.g.: angelhidden:=disappear(angel); angel:=appear(angelhidden);

> **DockOut(ship, fromship);**

'ship' docks out from 'fromship''s free dock.

> **Evacuate(ship, targetship);**

If 'targetship'=0, than it divides the people on your own ships.

> **Explode(missile);**

It makes the rocket to explode.

> **GetDockPoint(ship, index, outer);**

It gives the given indexed dock's outer/inner endpoint, from 1 (outer is not 0 – outer)

> **SetLampSignal(ship, "signal");**

When this command is issued, all of the ship's lamps will signal Morse-like string. The string repeats endlessly. The string is given by the signal, in which all the characters are understood in the following way: 1 – short signal, 2 – long signal, everything else – pause E.g. SOS: "1112221110"

> **NumBreachPoints(ship, freeonly, forrace);**

It gives the number of commando breaches on the ship. If 'freeonly' is not 0, than it only gives the still free places. This time, you have to give in the 'forrace', for which race's ship are we looking for a place. (Here, because of simplification reasons, the commandos of only one race can enter the ship, so if there is already a commando that belongs to another race, than the function will give 0.)

> **NumDockPoints(ship);**

It gives the number of docking places on the ship, including the breach points too. If not 0, you can dock.

> **PlayShipAnim (ship, anim_name, backward);**

It plays the shipmodel defined, given named animation. If 'backward' is not 0, than it plays it backward.

> **RenameShip(ship, name);**

It renames the ship.

> **SendTransport(ship, targetship, transportshipclass);**

The given class transport ship is sent from 'ship' to 'targetship'. If 'transportshipclass' = 0, than the default transport ship of the race will start (see races.ini). It is just for the sight, it does not carry anything.

> **obj:tSetWarning(warning);**

It sets the given serial number warning on 'obj' ship.

> **SetUndamageable(ship, undam);**

You can set the invulnerability of the ship here. 'undam': 1/0

> **SplitShip(ship, numparts, partclass, list);**

The ship divides to 'numparts' pieces of 'partclass' class parts. The parent ship is deleted, but before that, it sends a Splitted event of itself. If 'list' >= 0, than it gives back the created part ships in the given number list. If a list isn't needed, than you have to give -1. The part ships inherit the parent ship's scouting state, and lower lever movement commands. On AI level nothing is inherited right no, this question has not been cleared up.

> **Terminate(ship);**

The destruction of a ship with an explosion.

> **UniteShips(listid, destclass);**

It unites the ships in the list into one destclass class ship. The original ships will be deleted, but before that they send a United event about themselves. The united ship inherits the states of the lists first element.

8.7.2 TACTICS

> **NumShooters(ship);**

The number of those who shot at the ship in the last 3 seconds.

> **IsShot(ship, attackership);**

Did 'attackership' shoot on 'ship' in the last 3 seconds?

> **EverShot(ship, attackership);**

Did 'attackership' shoot on 'ship' anytime during the battle?

> **EnableEvacBoats(ship, enab);**

It enables/disables the little ships to fly out at the time of evacuation. Caution! If it is disabled, then the ship's squadrons won't try to escape, in other words, they will be lost.

> **EnableUnit(unit, enable);**

It turns on/off the AI of the 'unit'. In off state, it does not give any kind of control to the ship, so it can be controlled from the script.

> **GetDefense(ship, substance);**

It gives the current defense value of the ship's shield against the current bullet type.

> **SendCommand(ship, command_name, command_number, var_set);**

They are in the universe/commands/*.command file. Right at the providing, it gets a 'commandInit' event with set, changing config. These commands can catch the CO's events, and they can see the ship's variables by default. See in greater detail by the descriptions of **Commands**.

> **SetFlakTargets(ship, flags);**

It defines on what type of boats can the ship's flaks shoot. The following flags are combinable:

```
 #FLAK_BOMBERS
#FLAK_COMMANDOES
#FLAK_MISSILES
#FLAK_TRANSPORTS
#FLAK_NOTRANSP = #FLAK_BOMBERS+#FLAK_COMMANDOES+#FLAK_MISSILES
#FLAK_ALL = #FLAK_BOMBERS+#FLAK_COMMANDOES+#FLAK_MISSILES+
                #FLAK_TRANSPORTS
```

The default setting by every ship is #FLAK_NOTRANSP.

> **obj:tAwayFrom(targetship, range);**

The target tries to stay out of the given range. You can give it to max. 3 ships simultaneously.

> **obj:tAwayFromIntensity(intensity);**

The tAwayFrom motivations' intensities tactics by default are: AwayFromIntensity. You can supervise the value with this command. If it is 0, than it sets back to the default value. The intensity refers globally to all of the AwayForms (to those too that are already valid).

> **obj:tClearAway(targetship);**

It deletes the obligatory minimum distance regarding to the given ship.

> **obj:tClearAwayAll();**

It deletes all the minimum distances.

> **obj:tClearCommand();**

It clears the command that was issued to the 'obj' ship. You can issue a command with the SendCommand command.

> **obj:tClearFire();**

Clears the special target.

> **obj:tClearForbid(flags);**

It deletes the given inhibition(s), and leaves the rest unchanged. The delete is hierarchic, that is, e.g. tClearForbid(#FORBID_FIRE) clears all of the FIRE inhibitions, or e.g. the FORBID_MOTION_MOTIVATION clears the CLOSE and AWAY inhibitions too. If the flags is 0, it deletes all the inhibitions.

> **obj:tClearGuard();**

You can delete the Guard command with this command. It has to be applied on the protecting ship, not on the protégé!

> **obj:tClearTerritory();**

Id clears the territory command.

> **obj:tDamage(targetunit, range, purpose);**

How much damage can the unit cause in a second, in the given firing mode. Purpose can be only 1, 2, and 3.

> obj:tDamageRecalc();

If the purpose is changed (tFireTo), it recounts the damage that was done by the CO.

> obj:tDanger(enemyunit, range);

The danger level of the hostile unit on the given 'range', from 0-200.

> obj:tDangerBase(enemyunit, range, purpose);

The damage value that is the basis by the calculation of the danger level. 'purpose' can only be 2, 3.

2 (hull): the tactics that are used by the enemy: during DangerPeriod the percentage of hull damage that can be done from 'range' to the ship's current HP.

3 (tact): the tactics that are used by the enemy: during DangerPeriod the percentage of device damage that can be done from 'range' according to the average HP of the ship's device.

> obj:tAutoEnergyMode(system, auto);

A power subsystem can work two ways:
1-fixed: the consumption is limited by the set consumption mode (100,150,200%), and in case of engine, the speed factor is set automatically to the allowed maximum that is defined by the consumption mode.
2-automatic: the reverse one of the afore mentioned, the consumption mode is automatically set according to the consumption demand.

The command sets the given subsystem between these two modes. By clicking on any of the energy lines on the GUI, the given system's mode will become fixed. By clicking on the speed control line, the engine's system mode will automatically change. 'system': 1-weapon, 2-shield, 3-engine, 4-support. Level: 0-green, 1-yellow, 2-red.

> obj:tEnergyMode(system, level);

It sets the separate power systems' consumption quantity, it changes to fixed. 'system': 1-weapon, 2-shield, 3-engine, 4-support. Level: 0-green, 1-yellow, 2-red.

> obj:tEngineMode(mode);

It influences the engine selection, 'mode':
0: normal – automatic engine selection, according to speed or dodge values, in accordance with the situation
1: speed – speed optimization from command, it selects the fastest engine
2: dodge – dodge optimization from command, it selects the engine with the best target value
3: hide – hide optimization from command, it selects the engine with the best hiding value

> obj:tFireTo(target, purpose);

It selects the target. It has an absolute priority, every weapon that is capable for the purpose, will fire on the given target, if it can cause damage with them, and if there is enough power for their operation, it fires with the rest of the weapons independently on

other (not inhibited) targets 'purpose': 1 – shield demolition, 2 – hull damage, 3 - device damage

> **obj:tGuard(protégé, decayradius, intesity);**

You can order the ship to protect another one.

> **obj:tPlayerControl(pc);**

If 'pc' is not zero, it clears all the forbids, otherwise it disables everything.

> **obj:tRequestFirstHitEvent();**

After this command has been issued, when the ship gets hit for the first time, a FirstHit event is generated. The other hits won't create an event.

> **obj:tSetForbid(flags);**

It sets the given forbid(s), but leaves the rest unchanged. The 'flags' is the sum of one, or more forbids. Constants are defined for the certain forbids:
#FORBID_MOTION
#FORBID_MOTION_MOTIVATION
#FORBID_MOTION_MOTIVATION_CLOSE
#FORBID_MOTION_MOTIVATION_AWAY
#FORBID_MOTION_TERRITORY
#FORBID_FIRE
#FORBID_FIRE_NOTSHOTBY
#FORBID_FIRE_COUNTED
#FORBID_FIRE_MISSILE
#FORBID_FIRE_BOMBER
#FORBID_FIRE_FREE
#FORBID_SHIELD
#FORBID_ENGINE
#FORBID_ENGINE_NORMAL

8.7.3 MOVEMENT AND TERRITORY

> **ClearOrient(ship);**

It clears the forceOrient's effect.

> **ClearLongRangeDir();**

It clears the set direction of the jump, the jump will go to a random direction.

> **CloseAbs(ship, destX, destY, destZ);**

It approaches the absolute co-ordinates.

> **CloseOff();**

It suspends the Close… commands.

> **CloseShip(ship, targetship, range);**

Approaches the ship for 'range' distance.

> **DeactivateEngine(ship);**

It turns off the ship's engine. At that time, the ship doesn't even do evasive movement when someone fires on it.

> **Distance(item1, item2);**

The distance of the two items (e.g.: ship).

> **EnableDodge(ship, enable);**

It enables/disables the evasive movement of the ship. If according to the shipclass the ship is not capable of dodging, then it is ineffective.

> **EnableFortPass(ship, enable);**

It enables/disables that the ship can go through on own/friendly fort shields.

> **ForceOrient(ship, heading, pitch, bank);**

If the ship comes to a stand, than it will turn into the given direction.

> **ForceOrient(ship, targetobj, bank);**

In case the ship is in standing position, it turns the ship towards the given targetobject, similarly to the other version of the command. The setting can be deleted by the ClearOrient. Since the targetobject determines only the direction, apart from this you can also give one rotation position ('bank').

> **InArea(ship, areaindex);**

Is the ship in the area at the time?

> **LongRangeArrive(ship);**

The ship arrives with longrange effects. The arriving position is the same, where the ship stood before the command, but it will point to the direction according to the arrival. The arriving direction can be recorded here too with the SetLongRangeDir command (you have to give the direction from WHERE the ship should arrive), otherwise it is random. During the time span of the arriving effect the ship is in a disappeared state, so it cannot be reached.

Suggested application: place the ship on the corresponding arrival point, and give a Disappear command on it in the SceneInit. This way, it will remain invisible until the arrival command. At that time, you have to give the ORIGNAL identifier to the LongRangeArrive command, not that one that was given by the Disappear (no matter how much this is inconsistent with what was written in the Disappear command.)

> PutShip(ship, posx, posy, posz, heading, pitch, bank);

It puts the ship immediately to the given position.

> PutShipRel(ship, targetship, relx, rely, relz, orient);

It puts the ship to a position that is compared to another ship. The position has to be given in the targetship's co-ordinate system (Z is the fore-axis of the ship, Y is the perpendicular axis). 'orient' is the direction of the ship: 0 – it is corresponding with the targetship, 1 – it looks towards the targetship, 2 – it stands with its back to the targetship.

> Range(item1, item2);

The two objects are in 'range' distance from each other.

> SetAbsSpeed(ship, speed);

It tries to set the ship's speed to the given absolute value. This is done by the setting of the speed factor, and it takes the shipclass and the active engine's values into consideration. If there is no active engine than it turns on one. It changes the engine's power system mode to automatic. It can be used to move several ships together.

> SetGlobalSpeedFactor(factor);

All ships' speed that are in the scene is multiplied by the given percentile factor.

> SetLongRangeDir(ship, directionx, directiony, directionz);

The longrange jump of the ship will follow the given direction. The command has to be issued BEFORE you would turn on the longrange.

> SetMicroJumpArea(ship, centerx, centery, centerz, radius1, radius2);

After this command, all the jumps of the ship that it makes with the Jump type engine, will be on a random point of the sphere. If 'radius2' = 0, than the default behaviour is re-established.

> SetSpeedFactor(ship, factor);

It sets the ship's speed factor (0-150). The engine's power system mode changes to automatic.

> SetTerrArea(unit, areaindex, intensity);

The 'areaindex' sets a territory that is identical with the area. It only works, if the area is active.

> **SetTerritory(unit, centeritem, innerradius, outerradius, intensity);**

It creates an area, where the 'unit' can get only with a given 'intensity'. But it cannot get closer to the center than 'inerradius'.

> **ShipGoRound(ship, enable);**

It turns on/off, whether the ship should avoid the objects. It can be used to special movie scenes, where you want to put ship closer to each other than usual.

> **obj:tClearMove();**

It clears the movement command.

> **obj:tGetMaxVelocity();**

The speed that can be reached by the fastest engine.

> **obj:tGetVelocity();**

The current speed of the ship.

> **obj:tMoveTo(target, range, stealth);**

It will move to the 'target' unit, to given 'range' distance (2-combat/ 3-artillery/ 4-bomber/ 5-out of range). If 'stealth'=1, it moves in stealth (towards the asteroids and the radiating fields). It has absolute priority over all other movements.

> **obj:tMoveTo(tx,ty,tz, stealth);**

It moves on an absolute co-ordinate in the scene. If 'stealth'=1, it moves in stealth (towards the asteroids and the radiating fields). It has absolute priority over all other movements.

> **obj:tMoveToIntensity(intensity);**

The MoveTo motivation's intensity is by default tacticsbase:MoveToIntensity. You can change the value with this command. If 'intensity'=0, than the default value is restored.

> **obj:tSetTerritory(areaindex, intensity, command);**
> **obj:tSetTerritory(center, minradius, maxradius, decayradius, intensity, command);**

'center': An object that is in the center. 'minradius': Inner radius. 'maxradius': Outer radius. (maxradius>=minradius) 'decayradius': Between 'maxradius' and 'decayradius' the intensity of other movement motivations' decreases linear to zero. (decayradius>=maxradius) That is, the further the target is in which direction the ship could move, the less its effect will show up. It is certain that it won't move towards targets that are beyond the 'decayradius'. 'intensity': 0-200: The intensity of the default behaviour. See hereafter. 'command': This is to determine the default behaviour on the area of the territory.

0 - normal: There is no default behaviour, it goes till the edge of the territory from outside, it stops there, and if there is no other inclinations (independent from the territory), than it does not do anything.

1: guard: It wants to move to the center of the territory, with the given intensity. This means that if it has any other targets, (or it is afraid of something), than it will only move to a limited extent from its position, and if the other effect ends, than it goes back.
2: patrol: It does a guarding move within the boundaries of the territory, with the given intensity.
4: hide: It hides to the best place within the territory.

8.7.4 DEVICES AND WEAPONS

> **ActivateDevice(device, active);**

Activates the Device.

> **ActivateDeviceType(ship, devicetype, active);**

Activates the Devicetype.

> **CanPenetrate (weapon_type, ship);**

Whether a given_type_of_weapon can brake through the ship's (actual, proper type) shield (0/1)

> **DeleteDevice(device);**

It deletes the given device from the ship. It can be used for squadrons too.

> **DeviceFocused(device, race);**

It returns whether the given 'device' is aimed for 'race'.

> **EnableDevice(device, enabled);**

If 'enabled'=0, than the device deactivates, and cannot be activated hereupon anyhow, until it is not enabled again. Automatic devices (e.g. flak) can only be set with this method.

> **EnableActivateEvents(device, enable);**

It enables/disables the generation of an event when the device is turned on/off. It is disabled by default. Enable it only for the indispensable devices.

> **EnableMechaEmit(ship, enable);**

It enables/disables the sending out of a mecha from the ship. If the player wants to send out a mecha despite the prohibition, than it creates an event.

> **FocusDevice(device, race, on);**

If ’on’ is not 0, all of the device damaging weapons of the given race that fires at the ship that has the ’device’, will target the ’device’. If ’on’ = 0, then the selection is ends. Several devices can be selected simultaneously on one ship, at that time the hits are divided between these. The accuracy of the shots is smaller in this mode. It works only on the devices that were scanned by the ’race’.

> **GetDevices(ship, list);**

It collects all the devices of the ’ship’ into a ’list’.

> **GetDevScanned(device, race);**

It returns the device’s scanning level by given race.

> **HasDevType(ship, devicetype);**

Whether the ship possesses the given type of WORKING device.

> **InSet(device, set);**

Whether the device is a member of the given ’set’.

> **InstallDevice(ship, devtype);**

The creation and installation of a new device on the ship.

> **LoadWeapon(weapon, count);**

It increases the weapons’ charge by ’count’. It increases the number by rockets and bombs, and the number of people by commandos. You cannot reach a higher number than the default value by bombs and commandos. ’count’ can be negative too, so it is also good to empty something.

> **Purpose(weapon, purpose);**

Whether the weapon is used for its given purpose. ’purpose’: 1 – shield demolition, 2 – hull damage, 3 - device damage, 4 – artillery

> **SetDevScanned(device, race, level);**

It sets the device’s scanning level by given race. ’level’: 0 - unscanned, -1 – half scanned (only its category is known), 1 – fully scanned.

> **SetPrecision(weapon, precision);**

Every shot of the weapon will have given accuracy. If ’precision’ < 0, than it returns to the originally calculated accuracy.

> SetRepairFactor(race, factor);

The device repairing speed of the given race will be the 'factor' percent of the default value.

> SetScanned(ship, race, bool);

It sets all the devices' scanning level of the ship by given race ('bool' can be only 0 or 1).

> TargetWeapon(weapon, targetship);

It aims the weapon to 'targetship', and activates it. If 'targetship'=0, than it deactivates the weapon. It only works if the AI is disabled.

> TransferDevice(device, ship);

It transfers a device to another ship. It should be primarily used for bomber squadrons' rehosting.

> TransferSquadron(squad, ship);

It transfers the squadron to another ship. If the target ship cannot hold the squadron, then it does not dock, but it hovers around the ship in CLOSE mode. If there is enough space, than the squadron can be called in manually. But this time the other squadron will stay outside. Summoning home of the squadrons that remained outside also changes automatically to CLOSE.

> WeaponDamage(weapon, targetunit, range, purpose);

How much damage can the weapon, in average, cause in a second, in the given firing mode.

> WeaponMode(squad, mode, targetship);

Where 'squad' is a bomber, or commando type weapon.

'mode'	'target'	active weapon
0 - home	0	-
1 - evade	squad	flak
2 - close	ship v. squad	-
3 - guard	ship v. squad	flak
4 - patrol	0	flak
5 - attack	ship	prim.weapon + flak
	squad	flak
6 - sec.attk	ship	sec.weapon + flak
	squad	flak

'squad' everywhere means the weapon system that sends out the bombers.

The primary weapon is the first non flak weapon given in the DefDevices in the SHIPCLASS chart. If there is only one given, than it is also the secondary. The targetship only has to be given by the attack, otherwise its 0.

8.7.5 DETECTION

> **DetectEnable(ship, race, enable);**

Enables/disables the detectability of the ship towards the given race. It does not affect the already detected ships. In case of disabling, the undetection does not happen either, so the detection state freezes towards the race.

> **FreezeReconState(ship, freeze);**

> **FreezeReconState(ship, race, freeze);**

If freeze in not 0, it „freezes" the ship's current recon level for the given race, that is they won't know anything new about it from that moment on, but at the same time it cannot disappear either e.g. behind an asteroid. It can be released with freeze=0. If you give the command without race, than its going to be valid for all races.

> **GetDetected(ship, race);**

Whether the ship is detected by the race.

> **GetIdentified(ship, race);**

The identification level of the ship on behalf of the race. 0 – unidentified, -1 – incorrectly identified, 1 – identified.

> **GetScanned(ship, race);**

Whether all of the ship's devices are scanned by the race.

> **HideShip(ship);**

The ship gets undetected, and it becomes undetectable for all races. It literally disappears, and it doesn't appear ever again.

> **IdentEnable(ship, race, enable);**

It enables/disables the identifabiltiy of the ship towards the given race. It does not affect the already identified ships. There is no unidentifing after the disabling, so the identification state freezes towards the race.

> **MakeFullyKnown(ship);**

> **MakeFullyKnown(ship, race);**

It makes the ship detected, identified, and scanned from the race's point of view. If you issue the command without race, than it will be like this for all of the ships of all races.

> **ScanEnable(ship, race, enable);**

It enables/disables the analyzability towards the given race. It does not affect the already analyzed ships. In case of disabling the analyzed tools will not disappear, so the analyzability state freezes towards the given race.

> **SetDetected(ship, race, detected);**

detected: 0,1

> **SetDetectedFaded(ship, race, detected);**

It is the same as setDetected, but the appearing/disappearing is not immediate, but gradual.

> **SetDevScanned(device, race, scanned);**

It makes the device scanned for a race.

> **SetIdentified(ship, race, identlevel);**

identlevel: 0 unidentified, -1 incorrectly identified, 1 identified.

> **SetMisidentified(ship, misident);**

If misident is not 0, than the ship can only be identified as an asteroid (if it wasn't identified so far). It can be released with misident=0.

> **UnhideShip(ship, detect);**

The ship gets detectable. If 'detect' is not 0, than it gets detected towards everybody.

8.7.6 FORMATIONS

> **EndFormation(formation);**

It ends the whole formation.

> **PutFormationWings(formation);**

It immediately puts the formation's wing ships to points according to the leader's current position.

> **StartFormation(listid, leadership, devtype);**

It brings all the ships from the given list to a formation, and it returns the formation's identifier. 'leadership' is the leader of the formation, if its 0, than the leader is chosen automatically. 'devtype' is needed if you have to create a formation that is bound to a special device. At that time you have to give the 'leadership', and the given type of device has to be on this ship. The only suitable device is so far the Siege Laser.

> obj:tClearFollow();

If used on Wingman, it takes it out from the formation. The formations can so far stand together nicely in motion, because they don't have the ability yet to align themselves to each other's orientation (while in motion, this happens because of the common target). It is not worth to keep up big formations during a battle, because it is not necessarily beautiful. But it is cool at a march up. In battle rather those formations are good that contain 2-3 ships, and the Guard command. You can achieve through this that the ships won't go on their ways, and that they move very nicely.

> obj:tClearFormation();

If used on the leader ship, than it clears the formation.

> obj:tFollow(leader, decayradius, intensity);

This command has to be applied on the other ships (wingmans) of the formation. You can fit them into formations with this command. The positions of the formation are distributed by size, so the arrangement of new ships can cause the rearranging of the formation. 'leader': leader ship (it also defines the formation) 'decayradius': it is the radius that is defined in relation to the wingman's position, where intensity of the movement motivation that come from outside decreases to 0. In other words, the wingman can move from its place of the formation, but only inside a defined distance.' intensity': the intensity of the command (it is influenced by separateness!)

> obj:tSetFormation(id, type);

You can begin the building of a formation with this command. It has to be used on the leader ship. In case of a formation only the leader ship functions normally, the other wingmen move according to the formation, in relation to the leader ship. 'id': the identifier of the formation. At the time the 3-defense ring and the 2-arrack wedge formations are created. These can later change ship by ship, you can build new ones in the INI. 'type': 0: normal: moves only with the others, 1: offensive: fires with greater intensity to that target that is the target of the leader ship, 2: defensive: fires with greater intensity to those ships that attack the leader ship.

9 VARIABLES, PARAMETERS

> **this – the reference of the item. In case of list, you have to give this value to the commander, where item, ship, gate, etc. is.**

9.2 UNIVERSE

> **universe – the universe itself (as an entity);**

> **year**

> **month**

> **day**

> **globalTime – the time elapsed since the beginning of the game (in days);**

9.3 SHIP

> **ship (r) – it gives 1 only, if the ship is not a wreck (I hope that this won't turn everything upside down, at most the commands will not choose wrecked targets automatically. It doesn't give back 1 than either, if the ship has already started the jump.**

> **class (r)**

> **race (rW)**

> **name (rW) – name of the ship**

> **color (rW) – gui color**

> **team (r) – multiplay team, or 0**

> **player (rW) – in multiplay the identifier of the player who controls the ship, otherwise 0**

> **wreck (r) – gives 1 only on wrecked ships**

> **disabled (r) – the ship remains disabled hereinafter, which means that**
- all engines are fully damaged AND
- all weapons OR all weapon generators are fully damaged (the techscanner does not count as a weapon here!)
This state is considered as kill in the Multiplayer mode.

- **disabledNoSquad (r) – besides all the afore mentioned, it doesn't even have a single squadron. In Multiplayer mode the ship even explodes in this case.**

- **captain (r) – the NPC of the ship's captain, 0, if there is no captain. The first NPC on the ship.**

- **behaviour (rW) – the behaviour of the ship, so if e.g. a ship can be wrecked without giving the damage (2), or it can be brought back from wreck to normal state (0).**

- **commandBehaviour (rW) – it returns the actual behaviour (1-aggressive, 2-defensive, 3-stealth, 4-focus);**

- **damage (rW) 0-100, percentile value**

- **groupLeader (r) – in case the ship executes a group command, than it will be the leader of that group that got the command.**

- **isGroupLeader (r) – the ship is the leader of a group.**

- **formLeader (r) – if the ship is in given formation, it tells who the leader of that formation is. If it is not in a formation, it gives 0. It applies to _Low_level_formations.**

- **isFormLeader (r) – am I a formation leader? It applies to _Low_level_formations.**

- **formedInto (r) – the formation in which it is, or 0. It applies to _High_level_formations.**

- **boat (r)**

- **docking (r) – docking state 0 – flies freely (no docking), 1 – it is approaching the docking point (it still flies freely, but it already reserved the dock), 2 – it sits on the end of the dock, 3 – it is coming inside in the dock, 4 – finished docking / penetrated, 5 – it is going outside in the dock**

- **docktarget (r) – the targetship of the dock**

- **dockpoint (r) – the index of the reserved dock**

- **missile (r) – it gives 1, if the ship is a rocket.**

- **bomber (r) – it gives 1, if the ship is a fighter, a gunboat, or a bomber.**

- **commando (r) – it gives 1, if the ship is a commando**

- **transport (r) – it gives 1, if the ship is a transport**

- **transportRace (rW) – if this is >= 0, than the race if the transport boats that were sent out at the evacuation, or any other means, will be set to this. If it is -1, than the current automatic raceselection is valid (they take the race of that ship that sent them out).**

- **probe (r) – it gives 1, when the ship is a probe**

- **station (r) – if the ship is a station, than it gives the station's (that belongs to the strategical system) identifier, otherwise 0.**

- **fleet (r) – if the ship belongs to a fleet, than it is the identifier of the fleet, otherwise 0 (on contrary to the station's home fleet name, it is not a fleet, but I'm sure that everyone knows that already) launcher (r) – the ship that sent it out (it cannot be used by the transport!!!);**

- **launcherWeapon (r) – the device that started it, it can be used by squadrons' and commandos' ships.**

- **longrange (r) – it returns the longrange device that is mounted on the ship, if you activate this it starts the evacuation from the script.**

- **longrangeDir (r) – in case the ship's longrange device is activated, than it gives that vector in which way the jump will go. Otherwise it gives a zero vector.**
  This is a directionvector, so you have to use the v2rot function by the setting of the camera orientations!

- **target (rW) – the targetship (it can also be 0);**

- **breachedTo (r) – the ship, in which it has breached (commando), 0, if not**

- **hasCommand (r) – is there a command script called to the unit (SendCommand);**

- **hasDirection (r) – is there a special tactical command issued on the unit (tMoveTo, tFireTo, tGuard);**

- **hasMotion (r) – does the unit have movement motivation (it works in case of unit too, in that case, you have to understand the movement command under it, that was issued by the player);**

- **hasFire (r) – is there a weapon aimed towards the enemy ship (by AI = is there a fire motivation);**

- **closing (r) – what kind of approach does the ship use at the moment 1 – relative, 2- absolute, 0 – none, ONLY PLAYER SHIP**

- **closedShip (r) – by relative closing it is the targetship, otherwise 0, ONLY PLAYER SHIP**

- **closedRange (r) – by relative closing, the aimed range is (2,3,4);, otherwise 0, ONLY PLAYER SHIP**

- **isShielded (r) – did the ship raise its shield**

- **fort (r) – in case the ship is currently under a fort shield, it gives the SHIP that generated the shield, otherwise it gives 0.**
  It works in case of an enemy fort too. If it is used on the ship that generated the shield, it gives itself. shieldEnergy (r) – the shield-accumulator's current charge in energy. If the shield is currently not raised than 0.

- **fullyUnderFort (r) – whether the ship is fully under the fort shield**

- **shieldIntegrity (rW) – the shield-accumulator's current charge in energy. If the shield is not raised, than it is 0. This is not percentage, it is an energy unit! It can be overcharged, the surplus will be lost.**

- **shieldLevel (r) – the shield-accumulator's current charge in percentage. If the shield is currently not raised, than it is 0.**

- **position**
  positionX (r)
  positionY (r)
  positionZ (r)

- **heading (r) – direction**

- **pitch (r)**

- **bank (r)**

- **orientation (r) – it gives the ship's position in rotation form**

- **radius (r) – the radius of the ship (the biggest size, measured from the center);**

- **availSuppForDevices (r) – the power that is available in every tick to supply for the at the moment support devices. (after allowing for the sustainment requirements)**

- **reserve (rW) – the reserve power at the moment**

- **reserveLevel (rW) – the same in percentage (compared to the mounted reserve stores. If there isn't mounted anything like this, it gives 0);**

- **mobile (r) – whether the ship can still move**

- **moving (r) – the identifier of the manoeuvre**

- **movTarget (r) – the aim of the manoeuvre**

- **armor (r) – from the entered (weakened by the shield) hull damage this value will be subtracted**

- **devArmor (r) – the same on device damage**

- **selected (rW) – whether the ship is selected on the GUI. There can be several ships selected from your own.**

- **hpCur (rW) – actual hp**

- **hpMax (r) – max hp**

- **hpRegen (r) – the ship's HP regenerates this much in every tick. At the moment this is the privilege of the Angelwing.**

- **security (rW) – the ship's default defense value against the commandos. Its default value is set in the SHIPCLASS chart, but you can change it with this variable in special cases.**

- **theory (r) – for which theory can the ship give a point. 0 if it cannot give any points to either of them.**

- **availTheoryPoints (r) – If all the circumstances are taken into consideration (the state of the ship, category max., theory max.); how many points can be maximally scanned out of the ship.**

- **secret (r) – which secret does the ship hide. 0 if there is no secret.**

- **missingSecretPoints (r) – how many points have to be scanned out of the ship to get the secret.**

9.4 DEVICE

- **device (r) – 1, if device**
- **working (r) – is the device working?**
- **destroyed (r) – it is absolutely damaged**
- **noTarget (r) – it cannot be targeted (e.g. the docks…);**
- **engine (r) – 1, if its engine**
- **support (r) – 1, if its support device**
- **devType (r) – the serial number of the type**
- **ecm (r) - (on the basis of SET);**
- **eccm (r) - (on the basis of SET);**
- **devGroup (r) – in which category it falls into? For the categories see the constants.ini.**
- **owner (r) – the ship that is equipped**
- **damage (rW) – in percentile value**
- **active (rW) – whether the device is active or not**
- **energyIn (r) – it is the same as the chart's EnergyIn parameter**
- **repairPriority (rW) – the priority of the repair of the device, the default is 1.**
  1 – low
  2 – medium
  3 – high

9.5 SHIELD

- **shield (r) – 1, if shield**
- **energy (r) – it does not protect from energy (on the basis of SET);**
- **particle (r) – it does not protect from particle weapons (on the basis of SET);**

9.6 WEAPON

- **weapon (r) – 1, if it is weapon**
- **contFire (rW) – if 0, than the weapon is deactivated after every firing, if its not 0, than it fires constantly. By default all the weapons fire constantly, except the ones that cause power crisis, and the blinding weapons.**
- **mode (rW) – by bombers and commandos, see WeaponMode**
- **count (r) – by the counted weapon types (rockets, bombers, commandos); it gives the actual quantity, by other weapons it gives 1.**
- **target (rW) – target**
- **docktype (r) – it gives 1, if the weapon is bomber, or commando type**
- **groupFire – with how many boats does the commando leave (let this remain 1)**
- **countDec – how many people does a boat carry**
- **batValue – the fighting value of the commando**
- **hullValue (r) – it is able to damage hull (0/1); (on the basis of Purpose);**
- **deviceValue (r) – it is able to damage device (0/1); (on the basis of Purpose);**
- **shieldValue (r) – it is able to damage shield (0/1); (on the basis of Purpose);**
- **tactical (r) – tactical weapon (on the basis of SETS);**

> **heavy (r) – heavy weapon (on the basis of SETS);**

> **artillery (r) – artillery weapon (on the basis of SETS);**

9.7 ENGINE

> **main (r) – main engine (on the basis of SETS);**

> **secondary (r) – secondary engine (on the basis of SETS);**

> **combat (r) – combat engine (on the basis of SETS);**

> **stealth (r) – stealth engine (on the basis of SETS);**

9.8 FORMATION

> **formation (r) – 1, if the formation is an identifier**

> **race (r) – which race's formation**

> **maneuver (r) – the number of manoeuvre**

> **leader (r) – the leader ship**

9.9 GROUP

> **group (r) – 1, if group**

> **leader (r) – the biggest ship in the group**

9.10 GATE

> **gate (r) – 1, if gate**

- **locked (rW) – whether the gate is opened or closed.**
- **locked – Parameter, is immediately closed at the starting of the gate.**
- **lockableBy <race> <ticks> - In case a ship of the given race aims a data scanner to the gate, than after executing ticks scanning steps, the gate closes again. At that time the normal scanning functions are not working.**
- **unlockableBy <race> <ticks> - the same to open the gate**
- **leadsToNexus – the gate brings you right into the nexus. At that time it has a different effect in the scene**

9.11 FLEETENTRY

- **gate <x><y><z> <dirx><diry><dirz> - these are used exclusively at the jump ins of ships, it gives the place and direction of the gate.**
- **direction <dir> - the possible values of dir: 1 – the formation faces a random direction, 2 – the formation faces the closest planet, 3 – faces towards the closest planet's random horizontal point, 4 – the fleet faces toward a strategical level direction (so in the traveling or orbiting direction)**
  If you write these with negative signs, you will get the opposite directions.
- **shipName <name> <x><y><z> <heading><pitch><bank> - The positions and orientations of concrete, named ships' of the fleet can be given this way.**
- **leader <name> <x><y><z> <heading><pitch><bank> - the position of the leader ship.**
- **shipType <type> <x><y><z> <heading><pitch><bank> - the position of given type ships.**
- **shipClass <name> <x><y><z> <heading><pitch><bank> - the position of given type ships.**

9.12 NPC

- **npc (r)**

- **age (r)**
- **name <name> – the name of the npc**
- **face <num> – which picture belongs to the npc**
- **rank (r) – another feature of the NPC'S, to means the military rank**
- **crewlevel (rW)**
- **xp (rW) – the number of XP's collected since the last time you passed a level**
- **skill <skill> <level> - it is to set the three skills individually. 'skill': #SKL_MILITARY, #SKL_ENGINEER, #SKL_SCIENCE; 'level': 1-10**

9.13 EFFECT

An area that can have different features. It can be given in the ENTITIES section.

- **effect (r)**
- **category <category> - 2: radioactive, 3: hiding**
- **center <x> <y> <z> - center**
- **radius <r> – end of the scene**
- **decayType <type> – 0:constant, 1: linear, 2: inverse squared**
- **decayRad <r> - the radius where the effect stops**
- **strength <strength> – from 0-100**

9.14 OBSTACLE

A sphere into which the given ships cannot enter. It can be given in the ENTITIES section.

> **obstacle (r)**

> **center <x> <y> <z>**

> **radius <r> (rW)**

> **targetTypes <t1> <t2> ... ; - The given shiptypes cannot go through.**

> **race <r1> <r2> .... ; - None of the ships of the given races can go through.**

9.15 PIPE

A gate that gets the ship to another gate, if it approaches it. It can be given in the ENTITIES section.

> **entry <x> <y> <z> <r> - the ships that are inside the ('x','y','z'); centered, 'r' radius sphere, will be sucked in ruthlessly.**

> **exit <x> <y> <z> <r1> <r2> - the sucked in ships come out randomly between the('x','y','z'); centered, 'r1' and 'r2' radius spheres.**

> **chargeTime <time> - the gate lets through only one ship at once, and than comes a 'time' reload time in seconds, before another ship could enter.**

9.16 MULTIPLAY

The * in front of the parameters makes the given parameter changeable (e.g.: *duration 1800)

9.16.1 GAME

- **gameDescr <description>**
- **sceneRadius <r>**
- **music <song_num>**
- **winningCondition <type> – condition of victory, 'type': 0:kills, 1:points, 2:LMS**
- **winningConditionParam <param> - the limit of the condition of victory**
- **duration <mp> - game time**
- **prepareTime <mp> - preparation time**
- **jumpInTime <mp> - after that time no one can enter the game**

9.16.2 TEAM

- **name <name of the team>**
- **teamed – the players form groups, and that is a team**
- **individual – everybody is an independent team**
- **Copy <team> - copies the team's parameters there, you only have to define the difference**
- **teamChangeEnabled <t> - the players are enabled to change teams in the game, but they have to wait t seconds after every change, before they can enter again. They can assort their fleet during that time.**
- **maxPlayers <max> - max players**
- **maxShips <maxship> - the team can control maximum that number of ships simultaneously**
- **maxShipsPerPlayer <maxship> - the player can control maximum that number of ships simultaneously**

- **availShipValue <num> - max available value**

- **minShipValue <num> - the value cannot drop below this. Those ship that were bought when the point was below this value, don't give back points when they are withdrawn, or jumped out. The devices that were taken from these ships never give back points, and if the team reached the minimum point, it cannot even be equipped with another device.**

- **destroyedShipValueRet <percent> - the returned shipvalue when the ship is destroyed.**

- **jumpedShipValueRet <percent> - the returned shipvalue when the ship jumped out**

- **jumpOutEnabled – enables the jump**

- **simulatorPanel – it enables the simulator panel**

- **commandPanel – it enables the commandpanel**

- **civilization <civ> – through this command you can give such shiptypes and device types to the team that belong to the same civilization, or they don't have a given civilization.**

- **shipTypes – shiptype from which you can choose.**

- **devTypeStock – Device stock. You can list device types here with serial numbers. The end has to be closed with a ; .**
  e.g.: DevTypeStock #weap_lilaser/10 #weap_plasma/4 ;

- **spawnPoint <x> <y> <z> <minr> <maxr> <pitch> <blank> - Spawn point**

9.17 VARIABLES OF THE SCRIPT LANGUAGE

9.17.1 ROTATION

- **rotation (r)**

- **heading (rW) – the components**

- **pitch (rW)**

> **bank (rW)**

9.17.2 STRING

> **string (r)**

9.17.3 SPACE VECTORS

> **vector (r)**

> **x (rW) – the co-ordinates of the vector**

> **y (rW)**

> **z (rW)**

10 EVENTS

> **AreaEntered**

A ship has entered into the area. E.ship: the ship, E.area: the index of the area

> **AreaLeft**

A ship has left the area. E.ship: the ship, E.area: the index of the area

> **Arriving**

Docking starts (2 --> 3 state). E.ship – the boat, E.location - targetship

> **Beh_Changed**

A ship's behaviour has changed, E.Ship: the ship, E.CommandBehaviour: the behaviour it has changed to (1-aggressive, 2-defensive, 3-cautious)

> **Breached**

An enemy boat has breached into the ship (commando). E.location: the ship, E.ship: the boat

> **CO_BomberAlert**

Enemy bomber within range, E.ship – the ship

> **CO_BombersDestroyed**

All your bombers are destroyed, E.ship – the ship

> **CO_ChargingWeapons**

The given ship charges its weapons (hostility), E.ship – the ship

> **CO_CriticalDamage**

If the ship's damage reaches a certain percent, it sends a tRequestCriticalDamageEvent event. The event is received by the mission and ONLY those tactics and commands that belong to the airace.

> **CO_Detected**

If the ship is hit, this activates, E.ship – the ship (it has to be requested in particular!)

> **CO_DeviceDisabled**

A device is damaged and it is deactivated, E.ship – the ship, E.device – the object, E.devtype – the type of the device

> **CO_DeviceDestroyed**

It is sent out if a device is fully damaged E.device - the device E.devtype - the type

> **CO_EnergyDrain**

Energy that is decreasing very rapidly, E.system: power system, E.ship – the ship

> **CO_EnergyLow**

Low power value, E.system: power system, E.ship – the ship

> **CO_EnergyOnMax**

A given power system is fully charged, E.ship – the ship

> **CO_EnginesDisabled**

All the engines are damaged and disabled, E.ship – the ship

> **CO_FlakActivated**

Given ship activated the flak system, E.ship – the ship

> **CO_HullLow**

Critical damage, E.ship – the ship

> **CO_MissileAlert**

Enemy rocket within range, E.ship – the ship

> CO_NoShipsInRange

E.shipinrange gives the ship. But the event is generated only once, for another watch you have to issue the command again. range: 2 - combat, 3 - artillery, 4 – bomber mode: 1 – own ship, 2 – hostile ship, 3 – hostile or unidentified ship.

> CO_OverForce

It activated in case of superior force, E.ship – the ship (it has to be requested in particular!)

> CO_PurposeDisabled

All the weapons of a unit - that is in FireMode - that were suitable for the given purpose are disabled.

> CO_ShieldsDisabled

All the shields are damaged and deactivated, E.ship – the ship

> CO_ShipInRange

E.shipinrange gives the ship. But the event is generated only once, for another watch you have to issue the command again. range: 2 - combat, 3 - artillery, 4 – bomber mode: 1 – own ship, 2 – hostile ship, 3 – hostile or unidentified ship.

> CO_TargetReached

Given ship reached its target that was given in the command (in case of MoveTo command), E.ship – the ship

> CO_WeaponRunOut

Weapon run out of load, E.ship – the ship

> CO_WeaponsDisabled

All the weapons are damaged and deactivated, E.ship – the ship

> Collapsed

It is sent out if a mechanoid dies, E.ship – the annihilated ship, E.result – the wreck

> Deleted

The shipobject is deleted, E.ship – the ship

> Deselected

The ship has been deselected on the GUI. E.location, E.ship: the ship

> Detected

The ship is detected by a race, E.ship – the ship, E.race – the race, E.navpoint – the appearing and deleted navpoint.

> DeviceActivated

The device is activated, E.device – the device

> DeviceDeactivated

The device is deactivated, E.device – the device

> Docked

A ship docked on the ship, E.location: the ship, E.ship: the boat

> Exploding

The ship reached the full damage, and it is about to explode, E.ship: the ship

> Exploded

In case a rocket reaches its target and explodes, then this event is called. E.ship – the rocket

> FatalDamaged

The ship reached the fatal damage (90%), E.ship: the ship

> FirstHit

After the issued request, it is the first time that the ship is hit by another ship, E.ship: the victim, E.shooter: the attacker. In case the shot comes from a bomber, than the identifier of the bomber's MOTHER SHIP appears here. The event appear even if the shield deflects the shot entirely. In case of fort shield at the time of passing, an event will be sent from both ship that creates the fort shield, and from the attacked shield (in case the two are different). The event arrives BEFORE the damage that was caused by the shot would be registered, so the damage requests here will still give the previous value.

> FirstShot

It is the first time that the ship was shot at by another ship, E.ship: the victim, E.shooter: the attacker

> ForbiddenMechaEmitAttempted

The player tried to send out the AngelMecha when it was disabled. Only the mission script receives the event. E.ship – the ship, E.squadron – the mecha weapon

> Formed

It arrives when all of the members of a formation are on their places (and connected with an energy beam in case of siege). E.formation – the formation, E.ship – the leader ship

> GateLocked

The Gate is locked, E.ship – the scanning ship, E.gate – the gate

> GateUnlocked

The gate is open, E.ship – the scanning ship, E.gate – the gate

> Identified

The ship was identified by a race, E.ship: the ship, E.race: the race, E.identlevel: -1 incorrectly identified, 1 identified

> JumpedIn

The ship finished the jump in, E.ship – the ship

> JumpedOut

The ship finished the jump out, E.ship – the ship

> Launched

Something docked out from the ship, E.location – the ship, E.ship: the something

> Launching

Something is docking out from the ship, E.location – the ship, E.ship: the something

> LongRangeActivated

The ship's longrange engine is turned on, and the turning has been started. E.ship – the ship

> LongRangeArrived

It is generated at the moment the ship arrives, E.ship – the ship

> MicroJumping

It is called right before the jump. Here you have the possibility to set the targetarea. The jump is carried out only if the return value of the event is 0 (that is, you don't have to give anything to the event unless you want to disable the jump, in which case you need a return (1)). E.ship – the ship

> MissionInit

At the beginning of the mission, BEFORE the Tscene is generated (most of the itemtypes don't exist yet)

> MPGameOver

It is called at the end of multiplay, when the time runs out, and when the MpGameOver() is called.

> MPShipAdded

A player has selected a ship in multiplay. E.ship – the ship, E.team – the team. Caution! The ship is not yet in the scene, the only sensible operation is the jump in.

> MPShipJumpingOut

A player has deleted/jumped out a ship in multiplay. Before it was really jumped out. E.ship – the ship, E.team – the team.

> MPShipRemoved

A player has deleted/jumped out a ship in multiplay. E.ship – the ship, E.team – the team

> Occupied

The ship is occupied by an enemy commando. E.ship: the ship, E.race: the race of the commando

> OverridePlanner

The suitable eventcontroller is called in the center of the planner. Quite much amount of data is already calculated here, but the commands have not yet been issued, this is an ideal place to supervise the planner's decisions.

> RelationChanged

The relation of AI race towards another race is changed. E.airace – the AI race, E.race – the other race, E.prevrelation – the relation up until now, E.relation – the new relation.

> RunAway

The ship left with longrange, E.ship – the ship

> Scanned

ALL of the ship's devices were successfully scanned by a race. E.ship – the ship, E.race – the race

> SceneEnd

At the end of the mission, when the TScene still exists

> SceneInit

After the TScene was created

> ScannerOn

The scientific scanner is on (it refers to the real working, so it does not necessarily mean the turn on of the scanner device.) E.scanner – the scanner device (which is in reality a weapon).

> ScannerOff

The scientific scanner is off (it refers to the real working, so it does not necessarily mean the turn on of the scanner device.) E.scanner – the scanner device (which is in reality a weapon).

> SecretRevealed

We revealed the secret that was ordered to the ship (scanning at start), E.ship – the scanning ship, or the commando boat, if the secret was revealed by a commando, E.target – the scanned ship.

> SecretScanFinished

The secret that was ordered to the ship has been fully revealed. E.ship – the scanning ship, or commando boat if the scanning was done by a commando, E.target – the scanned targetship.

> Selected

A ship has been selected/deselected on the GUI. E.location, E.ship: the ship

> ShieldDown

Shield is down, E.ship – the ship, E.shield – the shield device

> ShieldUp

Shield is up, E.ship – the ship, E.shield – the shield device

> Splitted

If the mechanoid is splitted, it sends this

> TheoryScanFinished

All possible theorypoints were scanned from the target, E.ship – the scanning ship, or commando boat if the scanning was done by a commando, E.target – the scanned targetship.

> Undetected

The ship became invisible towards a race, E.ship – the ship, E.race – the race, E.navpoint – the appearing, and deleted navpoint.

> Unformed

When a formation disintegrates, either because of disbanding, or because of the displacement/destruction of a ship. If it happened because of displacement, than after the rearranging an unknown Formed event is generated. E.formation – the formation, E.ship – the leader ship.

> WeaponEmpty

The weapon is empty. E.ship – the ship, E.weapon – the weapon

11 DATA TABLES

11.1 THE MAIN MENU

The title displayed in the main menu as well as menu items should be defined in the *universe/main.ini* file, in the following syntax.

Title "TITLE TEXT"

MAINMENU

"Menu item name" command

.....

END

A maximum of 4 menu items can be given, and the first one will be displayed on the top. Usable commands:

In a Singleplayer MOD:

startmodmission – starts the selected mission. If this is the first menu item than the list of missions will appear in the main menu, and the selected one will be launched.

loadgame – opens the load game screen.

In a Multiplayer MOD:

mpclient – opens the Join multiplyer game screen.

mpserver – opens the Start server screen.

mpoptions – opens the multiplayer options screen.

11.2 CONSTANTS

In mission scipts and .ini files, predefined symbolical names can be used instead of numbers. These are called constants. The references always begin with a # sign, followed by the constant´s name.

11.2.1 TYPE IDENTIFIERS

Ship classes, device types, ship types and race names are all automatically placed with the constants, assuming the type indentifier´s value.. Example: We can refer to the Cruiser ship class by the **#cls_cruiser** name or refer to the gorg race with the **#race_gorg** name.

11.2.2 BUILT-IN CONSTANTS

The game contains a lot of other constants, these are given in the according scrpit commands or table´s. Example: Blending modes and Play modes of animation sequences.

11.2.3 OWN CONSTANTS

In addition, we can expand the constant set in the MOD indefinitely in the *universe/ mod_consts.ini* file. One constant can be defined on each row by giving its name (without the "#"), followed by its numeric value.

11.3 TACTICAL TABLE

11.3.1 PURPOSE

The datachart contains the occuring ships, devices and the shiptypes comprising of these, as well as the chart that contains the accuracy of the weapons in the game. This can be created and edited in the *__work\tactics.xls*. The **XLS** datachart is built in with a Visual Basic script, and it generates the *universe/tactics/tacticstypes.ini* file from that, which is then read by the game.

The attached *mod_tools/docs/Nexus tactict.xls* file contains the data that are used by the single player.

11.3.2 INI GENERATION

The script that was written in Visual Basic generates the **tacticstypes.ini file** into the given folder of the Mod **universe\tactics** if you press the shorcut key.

The script reads the data from the beginning of the 3. line of all worksheets' til it reaches a line that starts emptily. That is why those lines that contain a command have to be started by a *. It takes the keywords from the 2. line.

11.3.3 SHIPCLASSES

You can define the in game hulls and their data within this page.

> **#**

The serial number of the shipclass. It has to be unique within the shipclasses.

> **Name**

The name of the shipclass. With this name you can refer to the class in **#name** format, as a constant in the script or in the subsequent parts of the chart.

> **Mesh**

The name and route of the ship, you only have to give the route within the **meshes**.

> **Serial**

It is the string on the model that has to be written on the place of the ship, which can embody the name too.

> **GUIIcon**

The access path to the shipicon that has to be shown on the gui. You only have to give the route within the **textures** folder.

> **CannonMesh**

In case all the weapons that are mounted on the given ship has to look differently, than it was defined by the weapon, you can give the mesh's route here. You only have to give the route within the **meshes** folder.

> **Efx,**
ImpactEfx,
FinalEfx,
ChannelEfx

It defines the effectgroup that is effected on the ship.

> **FinalTime**

A parameter that is not used anymore.

> **Civilization**

Which race's shipclass is the ship.

> **Velocity**

The acceleration value, the ship's speed is calculated (the product of them) from this value and the engine's value.

> **RotVelFact**

The rotation speed, it is modified by the used engine.

> **RangeBehav 2**

It determines the behaviour within the Combat range: 1: keeps a given distance, 2: it is orbiting around the target ship.

> **Behaviour**

The type of the shipclass: 0: ship, 2: wreck, 4: asteroid, 5: navigation point.

> **TargetType**

It determines in which shiptype does the ship belong when the hit-chances are calculated. See in the **Basehitchance** part.

> **SupportOutput**

The performance of the basic support-generator that belongs to the ship.

> **SupportIn**

The support-power demand of the ship's sustainment system.

> **Armor**

The armor of the hull, this value is subtracted from the direct damage.

> **DevArmor**

The armor of the devices, this value is substracted from the direct damage.

> **WeapCntFact**

The size of the warehouse, in case of piece weapons, it can hold **WeapCntFact** times the default number of the weapon.

> **Hpmax**

The maximum Health Point of the Hull.

> Hpregen

In case this is not 0, it regenerates this amount of HP of the ship in every tick, that is, in about every tenth of a second.

> DeviceRepairHP

This is the total repair capacity of the ship.

> DetectBase

The base of the ship's perceptability, the lower it is, the easier the ship can be detected.

> Security

How much force does it have against the commandos that breached in, that is, how long can the commando stay on the deck.

> Special

Special parameter, the possibilities:
NoCrew: it doesn't have a crew, nobody evacuates when it trurns to a wreck
Shadow: no GUI appearance
Ufo 33, Ufo 100, Ufo 300: different sized mechanoid light effects

> Special

Special parameter, the possibilities:
Organic <a> <b> <c>: it is an organic ship, it does not explode, but rather splutters, and it does not get sooty, but bleeds with the give color, it does not evacuate when it turns to a wreck.
CollapseTo <num>: when it is destroyed it is exchanged to this shipclass.
Module <num>: it is the 'num'-th modul of the modular base.

> Special

Special parameter, the possibilities:
Absorb <num>: It charges the ship's reserve pool with the 'num' percent of the shots that hit the ship.

> TechCat

In case of scanning the given category and given quantity of points can be obtained from the ship.

> Weapon Slots

The types of weapons that can be mounted on the ship, according to the SET. If more weapons can be mounted to one place, than you have to separate them with the / mark, and you have to finish the enumeration with a ; .

> **Shield Slots**

The types of shields that can be mounted on the ship, according to the SET. If several types of devices can be mounted to one place, than you have to separate them with the / mark, and you have to finish the enumeration with a ; .

> **Engine Slots**

The types of engines that can be mounted on the ship, according to the SET. If several types of devices can be mounted to one place, than you have to separate them with the / mark, and you have to finish the enumeration with a ; .

> **Support Slots**

The types of supplementary devices that can be mounted on the ship, according to the SET. If several types of devices can be mounted to one place, than you have to separate them with the / mark, and you have to finish the enumeration with a ; .

> **Weapon Gen. Slots**

The types of weapongenerators that can be mounted on the ship, according to the SET. If several types of devices can be mounted to one place, than you have to separate them with the / mark, and you have to finish the enumeration with a ; .

> **Carrier**

How much point of squadron can it hold.

> **MaxCarried**

The maximum size of a squadron that can dock to the ship.

> **DefDevices**

These are the devices that are on the ship by default. Here you can only give the number of devices, the constants that were made of their names cannot be used, since they will be read only after this.

> **Available**

It doesn't have any function anymore.

11.3.4 WEAPONS

> #

The serial number of the weapon. It has to be unique among the devices. (Weapon, Shield, Engine, Support)

> Name

The name of the weapon. With this name you can refer to the weapon in **#name** format, as a constant in the script or in the subsequent parts of the chart.

> Mesh

The name and route of the weapon's model, you only have to give the route within the **meshes**.

> Category

Always 1.

> Civilization

Which civilization's device?

> Sets

Into which SETs does the device belong to.

> Special

The special features of the device, there can be only one in one square:
NoTarget: the device cannot be damaged by attacking
Automatic: the weapon selects a target by itself, and fires
Crisis <num>: it causes a crisis in the power system, every power systems will work slower for a while
CrisisOut <num>: in case of hit, the causes a crisis in the target's power system, and all of its power systems will work slower for a while.
ReserveCons <num>: it is needed for the functioning, and it is an immediately consumed reserve power.
ReserveConsOut <num>: in case of a hit, the target's reserve pool will decrease with this amount.

> Purpose

The useage possibities of the weapon:
1: shield breaker
2: hull demolisher
3: device damager
4: it affects an area
5: anti-aircraft defense
6: scanner

> Carried

If the weapon is a squadron, this is the size of it.

> HitChanceCat

The hit accuracy of the weapon, see the BaseHitChance chart.

> DeviceHitChance

The chance of the weapon, the hit a certain device.

> AutoDistance

In case of an automatic weapon, this is that distance in which it fires.

> MaxCount

In case of a piecemeal weapon, this is the default number of piecesmeal.

> CountDec

In case of firing, this is the amount with which the piecemeal decreases.

> GroupFire

In case of firing this is number of effects start, at that time the damage is evaluated so many times. In case of squadron, this is the number of pieces of the boats within the squadron.

> BatValue

In case of commando, this is the value that engages the target ship's security value.

> Substance

The material of the shot, it matters because of the shields' components:
1: energy
2: solid
3: bomber

> DamageSet

Not used.

> Charge

The weapon generator can load this much energy into the weapon in every tick, that is, approximately in every tenth of a second.

> EnergyIn

The amount of energy that is needed to fire the weapon. The weapon's rate of fire can be calculated from this and the charge value.

> **PenetrateShield**

The weapon's shield penetration value. PenetrateShield/Defense(Shield) gives how many percent of the damage can reach the ship. In case this is lower than a treshold (tacticsbase.ini), than the shot won't get through the shield.

> **DamageShield**

It damages the shield's integrity with this value by default.

> **DamageHull**

It damages the hull with this value by default.

> **DamageDevice**

It damages the hit device with this value by default.

> **DamageEnergy**

It decreases the charging, but not raised shield's energy with this value by default.

> **IonizeDevice**

It puts the weapon out of service for a certain time.

> **MaxHullDam**

The weapon can damage the hull with maximum this value.

> **Detected**

In case the device is functioning, it modifies the ship's perceptability with Detect percent.

> **Scanned**

In case the device is functioning, the chance of getting identified is increased by this percent.

> **Hpmax**

The weapon's Health Point.

> **RepairHP**

The given weapon can recover maximum this amount in every tick, that is, in every tenth of a second.

> Start

The launching parameters of the shot:
<param1>: 1: it starts docked out
2: the weapon fires
<param2>: it is not used, it is always 0.
<param3>: the number of the launching effect
<param4>: the scaling of the Eeffect.

> Travel

The travelling parameters of the shot:
<param1>: 1: the radius from the firing to the target
2: the travelling shot
3: the shot that has a shipclass
4: the shot that aims a breaching point, and it does not have a shipclass
<param2>: ‘param1’=1: the maximum length of the radius
‘param1’=2: the speed of the shot
<param3>: the number of the travel effect
<param4>: the scaling of the Eeffect

> End

The impact parameters of the shot:
<param1>: 1: impact
3: it affects an area
3: the assault drive of the fighters
4: breaching
<param2>: the maximum area of the effect, outside this the effect decreases quadratically.
<param3>: the number of the impact effect
<param4>: the scaling of the Eeffekt

> CannonMaxFireAng

The firing angle of the weapon.

> ShipMaxFireAng

Not used.

> Maneuver

The number of maneouver that is needed to fire the weapon.

> Upgrade

Not used.

> Available

Not used.

> **InstallRP**

Not used.

11.3.5 SHIELDS

> #

The serial number of the device. It has to be unique among the devices (Weapon, Shield, Engine, Support).

> **Name**

The name of the device. With this name you can refer to the weapon in **#name** format, as a constant in the script or in the subsequent parts of the chart.

> **Category**

Shield, always 2.

> **Civilization**

Which race's deviece.

> **Sets**

In which SETs does the device belong to.

> **Components**

Enumeration, which components does the shield contain, there can be maximum 3 of the following:
1: protection against energy
2: protection against solid material
3: protection against bomber weapons
4: the masking of devices, it makes them difficult to indentify
5: integrity protection

> **EnergyOutput**

With how much is the energy charged in every tick.

> **MaxIntegrity**

The maximum integrity of the shield, this can be broken by shiel breaking weapons.

> **Defense**

The defense value of the shield, the percentage of the damage that was caused by the weapon can be calculated from this.

> **Mask**

The modifying value of the devices.

> **IntegrityProtect**

This gets substracted from the shield breaking weapon's damage.

> **CM 2**

A value that jams the aiming of weapons that are in Combat distance.

> **CM 3**

A value that jams the aiming of weapons that are in Aritllery distance.

> **Detector**

In case the device is functioning, it can modify the ship's perception ability in percentile.

> **Detected**

In case the device is functioning, it modifies the ship's perceptability with Detected percent.

> **Scanner**

In case the device is functioning, it modifies the ship's device-identifing ability in percentile.

> **Scanned**

In case the device is functioning, the chance of for the identifing of the device is increased in percentile with this value.

> **Hpmax**

The Health Point of the device.

> **RepairHP**

The given device can recover maximum this amount in every tick, that is, in approximately every tenth of a second.

> **OnEfx**

The number and size of the build up effect.

> **OffEfx**

The number and size of the degradation effect.

> **Maneuver**

It is not used.

> **Special**

It is a special shield feature:
Fort <param1> <param2>: Fort shield. ‘param1’: the radius of the shieldsphere, if it is -1, that it covers the whole scene. ‘param2’: those who are under the shield get this amount of support energy in every tick. This is not effected if ‘param’ = -1.

11.3.6 ENGINES

> **#**

The serial number of the device. It has to be unique among the devices (Weapon, Shield, Engine, Support).

> **Name**

The name of the device. With this name you can refer to the weapon in **#name** format, as a constant in the script or in the subsequent parts of the chart.

> **Category**

Engine, always 3.

> **Civilization**

Which civilization’s device.

> **Sets**

Into which SETs does the device belong to.

> **Shortcut**

The keycombination that helps to turn on the engine.

> **EnergyOutput**

The engine provides this much energy in every tick.

> **EnergyIn**

This much energy is needed for the 100% functioning.

> **VelocityFactor**

The engine is capable of this speed. The maximum speed can be calculated from this and that value that was given by the shipclass.

> **SideVelFact**

The speed of sideward movement.

> **RotVelFact**

The speed of the rotation.

> **Dodge**

The speed of the dodging speed, this can be seen, when someone is shooting to the ship. Short, quick sideward movement.

> **DodgeRot**

The speed of the dodgeing rotation, it is difficult to lock the devices.

> **Detected**

In case the device is functioning, it modifies the ship's perceptability with Detected percent.

> **Scanned**

In case the device is functioning, the chance of for the identifing of the device is increased in percentile with this value.

> **CM 2**

A value that jams the aiming of weapons that are in Combat distance.

> **CM 3**

A value that jams the aiming of weapons that are in Artillery distance.

> **Hpmax**

The Health Point of the device.

> **RepairHP**

The given device can recover with maximum this amount in every tick, that is, in aprroximately every tenth of a second.

> **Efx**

The number and size of the engine's effect.

> **SoundEfx**

The number and size of the engine's soundeffect.

> **Special**

A special engine feature, only one can be written into one column:
Automatic: Automatic, always active.
Gravity: the pull of gravity of the black hole does not affect it.
Crisis <num>: if used, it causes a crisis, the power systems service will decrease temporarily.
Jump: jumpengine, if the normal engine has a target, when it is activated, it teleports the ship into the destination zone. Its functioning can be set to be random, and it can be only activated, if the required amount of reserve energy available.

11.3.7 SUPPORTS

> **#**

The serial number of the device. It has to be unique among the devices (Weapon, Shield, Engine, Support).

> **Name**

The name of the device. With this name you can refer to the weapon in **#name** format, as a constant in the script or in the subsequent parts of the chart.

> **Category**

Support, always 4.

> **Civilization**

Which civilization's device.

> **Sets**

Into which SETs does the device belong to.

> Special

Special support feature:
Automatic: Automatic, always active.

> EnergyOutput

The device provides this much support energy in every tick.

> EnergyIn

The amount of energy needed in every tick for the functioning of the device.

> MaxReserve

The device can hold this much reserve power.

> Detector

In case the device is functioning, it can modify the ship's perception ability in percentile.

> Detected

In case the device is functioning, it modifies the ship's perceptability with Detected percent.

> Scanner

In case the device is functioning, it modifies the ship's device-identifing ability in percentile.

> Scanned

In case the device is functioning, the chance of for the identifing of the device is increased in percentile with this value.

> Ionizer 2

It ionizes the ship's devices with this power within the Combat range

> Ionizer 3

It ionizes the ship's devices with this power within the Artillery range

> CM 2

A value that jams the aiming of weapons that are in Combat distance.

> **CM 3**

A value that jams the aiming of weapons that are in Artillery distance.

> **CCM**

The enumeration of device and jamming decreasing values, the given device's CM value is decreased by the given value.

> **Hpmax**

The Health Point of the device.

> **RepairHP**

The given device can recover with maximum this amount in every tick, that is, in aprroximately every tenth of a second.

> **OnEfx**

The number and size of the turing on effect.

> **OffEfx**

The number and size of the turing off effect.

11.3.8 SHIPTYPES

> **#**

The serial number of the shiptype. It has to be unique within the shiptypes.

> **Name**

The name of the shiptype. With this name you can refer to the class in **#name** format, as a constant in the script or in the subsequent parts of the chart.

> **Class**

The constant or number of the default shipclass.

> **Weapon Devices**

The constant or number, or the piecemeal of the device which is on the type and cannot be dismounted. Enumeration has to be ended with a ; . Only those devices can be mounted that are the size of the slot, and that fit into the slot.

> **Weapon DefDevices**

The constant or number, or the piecemeal of the device which is on the type and can be refitted. Enumeration has to be ended with a ; . Only those devices can be mounted that are the size of the slot, and that fit into the slot.

> **Shield Devices**

The constant or number, or the piecemeal of the device which is on the type and cannot be dismounted. Enumeration has to be ended with a ; . Only those devices can be mounted that are the size of the slot, and that fit into the slot.

> **Shield DefDevices**

The constant or number, or the piecemeal of the device which is on the type and can be refitted. Enumeration has to be ended with a ; . Only those devices can be mounted that are the size of the slot, and that fit into the slot.

> **Engine Devices**

The constant or number, or the piecemeal of the device which is on the type and cannot be dismounted. Enumeration has to be ended with a ; . Only those devices can be mounted that are the size of the slot, and that fit into the slot.

> **Engine DefDevices**

The constant or number, or the piecemeal of the device which is on the type and can be refitted. Enumeration has to be ended with a ; . Only those devices can be mounted that are the size of the slot, and that fit into the slot.

> **Support Devices**

The constant or number, or the piecemeal of the device which is on the type and cannot be dismounted. Enumeration has to be ended with a ; . Only those devices can be mounted that are the size of the slot, and that fit into the slot.

> **Support DefDevices**

The constant or number, or the piecemeal of the device which is on the type and can be refitted. Enumeration has to be ended with a ; . Only those devices can be mounted that are the size of the slot, and that fit into the slot.

> **DisabDevices**

The enumeration of devices that cannot be mounted on the type.

11.3.9 BASEHITCHANCE

> **#**

The serial number of the hit accuracy.

> **<targettype> <range>**

'targettype': this can be the TargetType of a shipclass
'range': 2: combat; 3: artillery, 4:bomber
The hit chances of the weapon under normal circumstances for the given distance. If it does not have hit chances against the certain shipclass on certain range, than the weapon will not fire. All weapons have 100% hit chances against ships that's engine is turned off, or don't have an engine at all.

11.3.10 MANEUVER

> #

The number of the manoeuvre.

> **Name**

The name of the manoeuvre, it can be used for the reference of a constant.

> **EngineEnergy**

It sets the engines' consuming energy.

> **FormShape**

The number of the formation.

> **FormType**

The type of the formation (in detail at the formation commands).

> **FormDecayRad**

The movement radius of the formation (in detail at the formation commands).

> **FormIntensity**

The intensity of the formation (in detail at the formation commands).

11.4 TACTICSBASE

This file is located at *universe/tactics/tacticsbase.ini.*

11.4.1 CATEGORIES

> **Range <range> <distance> <par1> <par2> <name>**

The settings of the borders of the Ranges

11.4.2 ENERGYSYSTEM

> **Period <mp>**

The setting of the tick.

> **CrisisDecay <regen>**

In case of an energy crisis this amount is recovered from the production in every tick.

> **MaxSpeedFactor <max>**

The maximum speed, the ’max’ percent of the basic speed.

> **ShieldFactor <comp_count> <eff>**

The shield components simultaneous functioning decreases the effectivness of the defense. The real defense will be **’Defense’ * ’eff’ / 100.**

> **ShieldLimit <percent>**

The damage value does not get through the shield under ’percent’ percent.

> **ShieldCollapse <percent>**

The shield collapses if it falls to ’percent’ percent of charge.

> **ShieldBuild <percent>**

The shield builds up, if it reaches ’percent’ percent charge.

> **RadiationSuppReq <energy>**

It eleminates the damage of radioactivity.

> **RadiationDamage <dam>**

The default value of the radioactivity damage.

> **SupportLowReactivateLimit <time>**

It switches back on the AI device only, if there is enough energy to provide it for 'time' seconds.

> **RepairDisabPeriod <time>**

After 'time' seconds of the hit the repairing can begin.

> **DevDisableLimit <percent>**

If the device is damaged to 'percent' percent, than it will becomes out of service.

> **DevReactivateLimit <percent>**

If the device is recovered to 'percent' percent, it can be reactivated.

11.4.3 DEVICESLOTS

> **slotx..sloty – set1 set2 set3 … ;**

Ordering sets to slots that are located on the model.

11.4.4 SCANNING

> **DetectBlind <time>**
> **IdentBlind <time>**
> **ScanBlind <time>**

The paralising time for different detections.

> **BlindDecay <regen>**

The regeneration of Blind in every tick.

> **SurpriseTime <time>**

The surprise time after the detection.

> **SurpriseBonus <percent>**

The multiplier of the events during the surprise.

> **SurpriseDamage <percent>**

The multiplier of the shots that were fired during the surprise.

> **TechScanShielded <percent>**
> **TechScanAttacking <percent>**
> **TechScanAttacked <percent>**
> **TechScanTargetAttacked <percent>**
> **TechScanECM <percent>**

The multipliers that influence the scanning.

11.4.5 CREW

> **EvacDamageLimit <crew_level> <percent>**

Evacuation levels.

> **SkillBonus <percent>**

It increases the related calculations' result with this percent for every skill point.

> **CommandoBattlePeriod <time>**

The calculation period of the commando's loss.

> **CommandoScanPeriod <time>**

The period of the commando's scanning.

11.5 TACTICSAI AND FORMATIONS

This file is located at *universe/tactics/tacticsai.ini.*

11.5.1 PURPOSE

The values that are required for the functioning of the AI can be given within the tacticsai.ini.

11.5.2 AISYSTEM

> **ShieldOnDistance <distance>**

The shield starts to charge from distance 'distance', if it is controlled by the automation.

> **ShieldHoldTime <time>**

If there is no detected enemy, than the automation removes the shield after 'time' seconds.

11.5.3 FORMATION SYNTAX

FORMATION <formation_number>
  Position <x> <y> <z>
  Position <x> <y> <z>
FORMATION

11.6 RACES

This file is located at *universe/tactics/races.ini.*

11.6.1 SYNTAX

CIVILIZATION <civ_number>
  Name "civ_name"
  EffectColor <r> <g> <b> <a> //it colors the default effects, if it is used by the civilization
  EffectColorB <r> <g> <b> <a>
CIVILIZATION

RACE <race_num>
  Name "race_name" //player
  Civilization <civ_num>
  GUIColor <r> <g> <b> <a>
RACE

RACETACTICS <race_num>
  TransportClasses <shipclass_num> ;
                    //evacuating shipclass, constant cannot be used!
RACETACTICS

RACEAI <race_num>
RACEAI

12 DEBUGGING

12.1 START DEBUGGING

The Debug function only available after the mission started from the Mission scene editor. In this case a consol window is opened next to the mission's window, and the debug commands are activated too.

12.2 CONSOLE WINDOW

12.2.1 PURPOSE

The console window serves observation purposes on the one hand, it can be scrolled, but it preserves only the last 1000 lines. On the other hand it provides the possibility **to issue commands, and to set the variables** during the mission. You can regulate the amount of automatically posted informations with script commands, it is more detailed there.

12.2.2 EDITING

To write in a command, you have to click on the lower **editorline**. All the general editing and standard copy-paste functions work in the editorline.

You can bring back the earlier issued commands in the console window, with the **Ctrl-Up** key combination. This does not work if the script stopped because of an error, and that message window is still there.

The lines that were typed in the console window can be **saved** with the **Ctrl-F1…F8** key combinations, and can be reloaded with the **F1…F8** keys. The saves will be in the save subfolder.

12.2.3 SCOPE

Within the console window, default scope is the mission script's scope, the only exception is in that case, if the running stops because of a script error, because in that case the scope of the error's place can be reached directly.

12.3 WATCH WINDOW

12.3.1 PURPOSE

In the watch window you can steadily observe the variables', and expressions' value during the running of the mission. The watch window can be opened with the **Ctrl-W** key combination.

12.3.2 EDITING

You can paste new variable observations into the window with the **Ins** key. You can edit the existing ones if you press **Enter** or with a **double click** after the selection. You can delete the selected line with the **Del** key.

12.4 CHEAT EVENTS

You can put such Rules into the mission script's ruleset that handle events that were called by key combinations. The **Ctrl-1…9** key combinations call **cheat0…9** events from the **numeric pad**.

12.5 DEBUG SCRIPT COMMANDS

> **Assert(condition);**

This is a debug-intent tool, with which you can check the conditions that are needed for the proper running, before the execution of a script piece. If the condition is not fulfilled, than it stops running with an error message (this works absolutely the same as in the previous letter).

It's substantial that in the Release version these checks are executed, they can only be used for debugging.

> **Break();**

Debug command, don't use it (it gives assert).

> **Debug(str);**

It writes the ’str’ on the debug console.

> **Debug(condition);**

It writes the value of the ‘condition’, not the text.

> **Debug(str, condition);**

It writes the text into the debug console, and than the value of the ‘condition’.

> **DebugOff();**

After this command no messages will get to the console, or into the log file. It is suitable to use by cycles to avoid those postings that slow down the running.

> **DebugOn();**

The opposite of the previous one. After you issue this command, the postings will appear again.

> **Dump(list);**

It writes the elements of the list into the debug window (the names of the elements).

> **EnableLog(condition);**

You can turn on and off the logging into the log file. The ’condition’ will turn it off if the value is 0, and on in any other cases.

> **EventOn();**

It activates the postings of events in the log file.

> **EventOff();**

It deactivates the postings of events in the log file.

> **MsgBox(string);**

When this command is entered, a windows message box appears.

> **ParamCheckOff();**

It turns off the checking of parameters by the calling of script commands, if a parameter has an error, it returns with a 0.

> ParamCheckOn();

It turns back on the checking of the parameters.

> PPos(ship);

It writes the name of the 'ship' and its current position into the consol window. By this, you can even list with it, e.g. Select(0, S.ship&S.race=0); Execlist(0, PPos(S.this);

> TraceOff();

It finishes the tracing.

> TraceOn();

From this command the script can be executed in steps. The following things can be considered as one step:

- a command separated by a ; including the separated parts in the terms too, so e.g. in the if, while commands, or in the submitting of parameters by the event calls.

- every examined object by the selects (this is not obligatory, see below)

Before the steps, the executable command is posted on the console, and the running stops. Then, you have three choices:

- F10: executes the step, namely in a way that it counts the event calls and the selects as steps too

- F11: it executes the step, on case of event call you can also follow the called part step-by-step, and in case of select you will see the evaluations of the conditions as objects.

- F9: Finish tracing, runs in normal mode. This occurs due to the script embedded TraceOff() command, or if there is nothing like this, than it will happen when you return to the command prompt.

When the steps are executed, the debug posting mode is activated. This also developed since the since the foregoing, now you can see the outcomes of the evaluations of every part term, so you can follow exactly what is happening.

In stopped state, you can give any arbitrary script command from the console. This can be very useful on one hand for fast variable query (so you don't have to put such things into watch, that you are not always interested in), and on the other hand you can change the variables' values from test purposes while it is running, if needed. Of course you have to be very careful with this possibility.

> Write(str);

It writes out debug strings.

13 LOCALISATION

13.1 SIMPLE TEXTS

When the system expects a text in .ini files and mission scripts, you can give a text identifier instead of the actual text, to which you can define text in several languages.

You can create any number of *ini* files in the t*exts/texts* subdirectory, in which you can define any amount of text definitions. The syntax of these is:

TEXT "identifier"

language1 "text1"

language2 "text2"

.....

END

The languagues can be teh following (the number after the language denotes which Windows codepage to use for denoting the language):

0	english	1252
1	german	1252
2	hungarian	1250
3	french	1252
4	spanish	1252
5	italian	1252
6	russian	1251
7	chinese	936

13.2 TYPE AND OBJECT NAMES

The names of ships and device types, races and all objects created in the solar systems and missions are localised automatically if we create a text definition for them with the id of the object´s name. Example:

```
TEXT "cls_cruiser"
        0 "cruiser"
        1 "kreuzer"
        2 "cirkáló"
        ....
  END
```

13.3 DIALOGUES

These are the speeches and answers given by NPC´s. You can create any number of *ini* files in the t*exts/texts* subdirectory, in which you can define any amount of dialogue definitions. The syntax of these is:

The dialogues can be portrayed by their identifiers with the **dialog** script command, see the script chapter.

If a dialogue WAV file of the correct language and with the same name as the dialogue is found in the (*sound/dialogs/xx*) directory, then the WAV will play during the dialogue.

13.3.1 MESSAGE-DIALOGUE

```
DIALOG
  Name name
  [link]
  layout n
  NPC npc_identifier
  text
  {
        language1 "text1"
        language2 "text2"
```

```
        .........
  }
END
```

Displays/plays the given text with the given NPC´s face image than disappears.

The layout gives the position of the dialogue window:

Own ship NPC-s:

0	bottom left
1	bottom centre
2	bottom right

alien ship NPC-s:

10	upper left
11	upper centre
12	upper right

If alink is given, then it will automatically link to the previously defined dialogue. Dialogues linked this way will play in sequence when the first dialogue in the chain is activated.

13.3.2 ANSWERING DIALOGUE

```
DIALOG
  name name
  Layout n
  [ChooseTime t]
  [ExpireEvent event]
  NPC npc_identifier
  text
  {
        language1 "text1"
        language2 "text2"
        .........
  }
  options
        {
```

```
        language1 "text1"
        language 2 „text 2”
        .........
    }
    event1
    {
        language1 "text1"
        language2 "text2"

        .........
    }
    event1
    .............
END
```

Displays/speaks the text and displays the answer alternatives given in the options. When the player chooses an alternative , the corresponding event given in teh script will occur.

If ChooseTime is given, then this is the amount of time in seconds the player has to give answer. When the time expires the event scripted under ExpireEvent will occur.