[R3F] LOGISTICS

VERSION 3.1

FOR ARMA III

Documentation

[R3F] LOGISTICS - Documentation

SUMMARY :

I. Software presentation

The "[R3F] Logistics" is a collection of scripts to be installed in a mission.

The software is delivered as a multiplayer mission containing :

- the logistics scripts centralized in the folder "R3F_LOG"
- a multiplayer **tutorial mission** to learn about the features and usage

Its main features are :

- Movable objects : the player can carry an object and walk with it. Some controls allow placing the object at the exact willed position.
- Transportable objects : objects can be loaded into vehicles or containers to transport them.
- Helicopter lift : light vehicles, containers, ammunition crates, etc. can be lifted.
- Towing : some objects, vehicles, and boats, can be towed to a vehicle.
- Creation factory : one or more creation factories can be used as a spawning system. The credits, content and side restrictions can be set.

Each logistics feature can be added or removed on any object or vehicle. It is highly flexible and configurable. Read chapters III and IV to learn more about it.

No additional addon is needed. But objects and vehicles provided by any addons can benefit of the logistics features.

It is MP and SP compatible. It can be hosted on a dedicated or non-dedicated server.

The [R3F] Logistics was developed keeping always in mind to be very intuitive to the player, and to be easy to install and configure for the mission maker.

II. Basic installation

1. Download and try the tutorial mission

The software is delivered as a multiplayer tutorial mission. To try it, you have the choice to install the mission in three ways :

- Subscribe to the Steam Workshop to automatically install the mission.
  Steam Workshop : http://steamcommunity.com/sharedfiles/filedetails/?id=332279165

- Download the mission on the R3F's website, and install it manually as explained below.
  Web download : http://forum.team-r3f.org/index.php?action=downloads;sa=downfile&id=76

  - Copy the PBO file named "R3F_Logistics_3_1_tutorial.Altis.pbo" in the "MPMissions" present in your Arma installation directory. The path to the folder should be :
    C:\Program Files (x86)\Steam\SteamApps\common\Arma 3\MPMissions\

  - If you can't install the PBO file, you can install the mission folder named "R3F_Logistics_3_1_tutorial.Altis" in your Arma profile directory.

    Its path should be one of these :
    C:\Users\<windows login>\Documents\Arma 3\MPMissions\
    C:\Users\<windows login>\Documents\Arma 3 - Other Profiles\<Arma profile>\MPMissions\

    If the "MPMissions" folder does not exist, you can create it.

Then launch Arma, and host a multiplayer game by clicking on "NEW" in the multiplayer servers display. Select the world "Altis", you should see the tutorial mission.

2. Quick installation guide

If you are used to install script systems in a mission, you can follow this quick guide. There is also a detailed guide in the section 3 with more explanation.

- Copy the folder "**R3F_LOG**" from the demo mission to your mission.

- Add this line in your "**init.sqf**" :

```
execVM "R3F_LOG\init.sqf";
```

The line must **NOT** be copied in a conditional block, like "if (isServer)". It must be placed outside of all sub-sections of code or condition.

- Add this line in your "**description.ext**", outside of all braces block "{ ... }" :

```
#include "R3F_LOG\desc_include.h"
```

- The basic installation is done. To learn more about advanced usage and configuration, read the chapters III and IV.

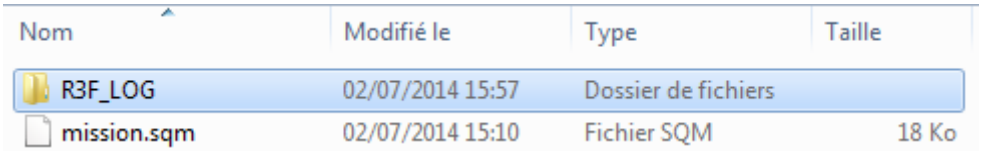
3. Detailed installation procedure

STEP 1. Copying the logistics folder

If you read this document, you should have downloaded the official [R3F] Logistics release, which includes a demo mission folder named "R3F_Logistics_3_1_tutorial.Altis". Open this mission directory and copy the folder named "R3F_LOG".

Then, open your own mission directory. Its path should look like :
C:\Users\<*User*>\Documents\Arma 3 - Other Profiles\<*Arma profile*>\MPMissions\<*Your mission*>\

Paste the "R3F_LOG" folder in this directory, at the same place as the existing file "mission.sqm" :

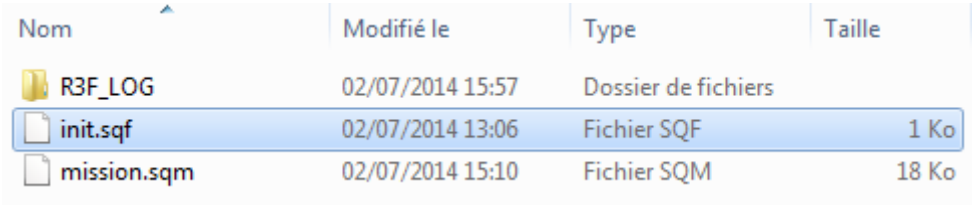


STEP 2. Editing "init.sqf"

We need to initialize the logistics system thanks to this code line :

```
execVM "R3F_LOG\init.sqf";
```

The best place to write it, is the file "init.sqf" in the mission's root folder :

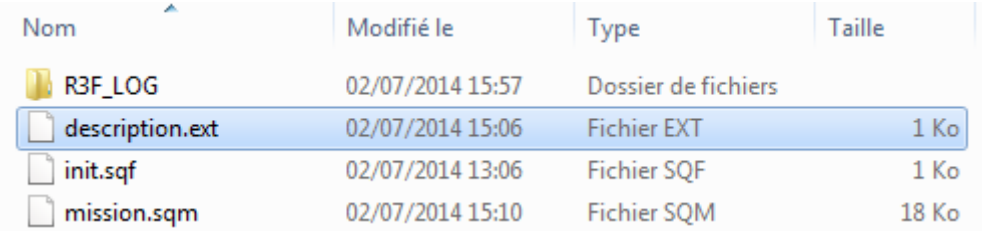


Might this file already exists. If so, open it. Otherwise, you need to create an empty file named "init.sqf". Be sure to set the file extension to ".sqf" and not to ".sqf.txt".

When you opened the file, copy the code line above in it. The safer place to insert it, is the very first line. Be sure to NOT paste the line in a conditional block like "if (isDedicated)" or "if (!isServer)".

STEP 3. Editing "description.ext"

Now we have to edit a file named "description.ext". If this file does not already exist, create an empty file. Once again, be sure to set the file extension to ".ext" and not to ".ext.txt".



We need to include a file defining the logistics dialogs, by copying this line in the "description.ext" :

```
#include "R3F_LOG\desc_include.h"
```

Be sure to NOT paste the line inside a block delimited by braces ("{" and "}").

The basic installation is done. To learn more about advanced usage and configuration, read chapters III and IV.

4. Installing an update

This section explains how to install a new version of the [R3F] Logistics system, when an older version is already installed in the mission.

The procedure is applicable only for an update from the version 3.0 and later. If you are using an older release (not officially ported to Arma III), you have to delete it, then follow the normal installation procedure previously described.

To make the update, you won't need to edit the files "init.sqf" and "description.ext".

If you edited the default configuration files, keep a copy of them to reproduce the modification in the updated folder The configuration files are :

- config.sqf
- config_creation_factory.sqf
- all the logistics configuration in the folder /addons_config/
- all the language files named "XX_strings_lang.sqf"

Now you just have to completely delete the folder "R3F_LOG", then copy the new one at the same place as the deleted one.

If needed, you have to restore your previously saved customized configuration files. It is highly recommended to use a tool like WinMerge to check if the configuration files' content has changed with the update.

A. Specific instructions to update from 3.0 to 3.1

config.sqf :

New configuration variables appear :

- R3F_LOG_CFG_lock_objects_mode
- R3F_LOG_CFG_unlock_objects_timer

If one of these variables is not defined, a default value will be used.

The "#include" path "addons_config\A3_vanilla_1.22.sqf" has been changed to "addons_config\A3_vanilla.sqf". Take care, if you want to keep your old addons configuration folder.

config_creation_factory.sqf

New configuration variables appear :

- R3F_LOG_CFG_CF_sell_back_bargain_rate
- R3F_LOG_CFG_CF_creation_cost_factor

If one of these variables is not defined, a default value will be used.

\addons config\

- Updated : A3 vanilla config
  - folder renamed from "A3_vanilla_**1.22**" to "A3_vanilla"
  - more towing and towed vehicles
  - added a file taking into account new objects : "delta_A3_1.32_to_1.35.sqf"

- Added : All in Arma config (auto-activated only is the addon is in use)

Language files "XX_strings_lang.sqf"

New localization variables to be translated if you added your own translation :

```
STR_R3F_LOG_action_heliport_attente = "Hooking... (%1)";
STR_R3F_LOG_action_heliport_echec_attente = "Lift aborted ! Stay hover during the hooking.";

STR_R3F_LOG_action_revendre_usine_direct = "Send back ""%1"" to the factory";
STR_R3F_LOG_action_revendre_usine_deplace = "Send back to the factory";
STR_R3F_LOG_action_revendre_usine_selection = "... send back to the factory";
STR_R3F_LOG_action_revendre_en_cours = "Sending back to the factory...";
STR_R3F_LOG_action_revendre_fait = "The object ""%1"" has been sent back to the factory.";
STR_R3F_LOG_action_revendre_decharger_avant = "You can't sent it back while its cargo
content is not empty !";

STR_R3F_LOG_deverrouillage_en_cours = "Unlocking... (%1)";
STR_R3F_LOG_deverrouillage_echec_attente = "Unlock canceled ! Hold the aiming of the object
during the countdown.";
STR_R3F_LOG_deverrouillage_succes_attente = "Object unlocked.";
STR_R3F_LOG_action_deverrouiller = "Unlock ""%1""";
STR_R3F_LOG_action_deverrouiller_impossible = "Object locked";
```

Edited localization variables to update if you have your own translation :

```
STR_R3F_LOG_action_remorquer_direct = "Tow ""%1""";
STR_R3F_LOG_action_contenu_vehicule = "View the vehicle's content";
STR_R3F_LOG_action_decharger_deplacable_exceptionnel = "Once released, this object will no
more be movable manually.<br/>Do you confirm the action ?";
STR_R3F_LOG_dlg_CV_titre = "Vehicle's content";
STR_R3F_LOG_nom_fonctionnalite_actif_transporte_capacite = "max load %1";
```

Deleted localization variable :

```
STR_R3F_LOG_action_remorquer_selection = "... tow to ""%1""";
```

\USER_FUNCT\

"watch_objects_to_not_lose.sqf" is replaced by "do_not_lose_it.sqf " :

This new script responds to the same need, more efficiently and in a smarter way.

If you are using the script "USER_FUNCT\watch_objects_to_not_lose.sqf", you should update your mission to use the new script "USER_FUNCT\ do_not_lose_it.sqf".

Read the updated section "III.3.A Protect objects to not lose" to know how to use this new script.

III. Advanced usage

1. Logistics

A. Enable/disable logistics on specific objects

At any time, you can enable or disable all the logistics features of an object or a vehicle. There is a script variable associated to each object to individually enable or disable it.

It is especially useful to make some non-movable nor transportable essential objects on the base.

To **disable** all the logistics features on an object, use this command :

```
<the_object> setVariable ["R3F_LOG_disabled", true];
```

The key-word "*<the_object>*" must be the object/vehicle variable name in a general script context. In the specific case of the initialization line of the mission editor, you can use the variable "this" :

```
this setVariable ["R3F_LOG_disabled", true];
```

To enable all the logistics features, use the same script line, but replace true by **false**.

Take care of the execution locality of the command (i.e. on server or on all clients). If you put it in the initialization line (i.e. executed on all clients), the line above will works. If you put it in a script executed only on the server side, you will have to broadcast the value, by adding a third parameter (true) to the setVariable command, in order to have a global effect :

```
<the_object> setVariable ["R3F_LOG_disabled", true, true]; // Broadcast
```

Any object/vehicle can be disabled and enabled, several times, at any time during the mission.

Note that the configuration variable "**R3F_LOG_CFG_disabled_by_default**" (config.sqf) permits to choose if all the objects/vehicles are enabled or disabled when they are created (mission start or dynamic spawn). If set to true, all the objects will be non-logistics, except those which are individually activated with the setVariable script command. If set to false, all the objects will be enabled, except those which are individually deactivated with the setVariable script command.

B. Load automatically a vehicle or cargo

An easy-to-use script allows loading automatically and instantly objects in a transport vehicle or cargo. It can be used at mission start or any time during the mission. This function is able to load both existing objects and class names to spawn before loading it.

The script will check if the objects to load are set as "transportable" in the logistics configuration, and if the vehicle has enough free space to load it.

The script to use is "**R3F_LOG\USER_FUNCT\auto_load_in_vehicle.sqf**". You can open it to read the description of the two required parameters.

The **first parameter** corresponds to the vehicle or cargo in which to load the objects. It can be the variable **this** in the initialization line of the vehicle, or the object name set in the editor, or a script variable.

The **second parameter** is a very flexible **array** which can contain a mix of existing objects, class names to spawn before loading it, and class names associated to the quantity to spawn then load :

- An existing object can be defined thanks to the object's name set in the editor, or a script variable.
- A class name to spawn before loading must be delimited by quotes like a string (e.g. "Box_IND_Ammo_F").
- A class name associated to the quantity to spawn and load must be written as an array of two elements : the class name as a string, and the quantity (e.g. ["Box_IND_Ammo_F", 3]).

Each element must be separated by a comma. Below are some usage examples. The key-word "<my_vehicle>" can be **this** in the initialization line of the editor.

The following command loads the existing objects "<my_object1>" and "<my_object2>" into "<my_vehicle>" :

```
nul = [<my_vehicle>, [<my_object1>, <my_object2>] ] execVM
"R3F_LOG\USER_FUNCT\auto_load_in_vehicle.sqf";
```

The following command spawns then loads one "Box_IND_Ammo_F" and one "Box_IND_Grenades_F" into "<my_vehicle>" :

```
nul = [<my_vehicle>, ["Box_IND_Ammo_F", "Box_IND_Grenades_F"] ] execVM
"R3F_LOG\USER_FUNCT\auto_load_in_vehicle.sqf";
```

The following command spawns then loads **two** "Box_IND_Ammo_F" and one "Box_IND_Grenades_F" into "<my_vehicle>" :

```
nul = [<my_vehicle>, [ ["Box_IND_Ammo_F", 2], "Box_IND_Grenades_F" ] ]
execVM "R3F_LOG\USER_FUNCT\auto_load_in_vehicle.sqf";
```

The following command loads "<my_object1>", two "Box_IND_Ammo_F" and one "Box_IND_Grenades_F" into "<my_vehicle>" :

```
nul = [
      <my_vehicle>,
      [
            <my_object1>,
            ["Box_IND_Ammo_F", 2],
            "Box_IND_Grenades_F"
      ]
] execVM "R3F_LOG\USER_FUNCT\auto_load_in_vehicle.sqf";
```

The following command spawns then loads a large set of objects into the vehicle associated to the initialization line (**this**) :

```
nul = [
      this,
      [
            ["B_HMG_01_high_F", 2],
            "B_Mortar_01_F",
            "Flag_NATO_F",
            "MapBoard_altis_F",
            "Land_CampingTable_F",
            ["Land_HBarrier_3_F", 8],
            ["Land_HBarrier_5_F", 15],
            ["Land_Obstacle_Ramp_F", 5],
            ["Land_CampingChair_V2_F", 2],
            ["Land_HBarrierTower_F", 2],
            "Land_BagBunker_Small_F",
            ["Land_BagFence_Long_F", 6],
            ["Land_BagFence_Round_F", 2]
      ]
] execVM "R3F_LOG\USER_FUNCT\auto_load_in_vehicle.sqf";
```

C. Access permission management

You have the ability to allow or deny the access to all the logistics features, to specific players, at any time.

To do that, there is a configuration variable in "config.sqf" named :

```
R3F_LOG_CFG_string_condition_allow_logistics_on_this_client
```

This variable is a **string** to be evaluated as a **SQF condition** during the game. It permits to allow the logistics to only some specific players or according to any scriptable condition. Read the description of this variable in the file "config.sqf". There are some examples based on the **player's name or game ID**.

2. Creation factory

A. Creation factory initialization

To add one or more creation factories, you have to initialize the object representing the factory. Any object or vehicle can be use as a creation factory. For example, you can use a container, a flag pole, a ground vehicle or anything you want.

A creation factory can have a credits (kind of money) limitations which decreases according to the cost of the created objects. The cost of an object doesn't pretend to correspond to a real price or anything else. It is computed thanks to the property "cost" of the object (set by the object's author in the config file). You can have interactions on credits with your mission (e.g. when an objective is done, money is sent to the factory as a recompense). It is explained in the part B.

A factory can be configured to be accessible to any side, or only to a specific side. Its accessible content can also be configured.

To initialize a factory, you need to call the following script with some parameters :
"**R3F_LOG\USER_FUNCT\init_creation_factory.sqf**".

Open it and read the description to understand how to define the parameters.

Only the first parameter is required : the object representing the factory. The three others parameters are optional and permits to set the credits/side/content restrictions. Below are some initialization example :

Initialize a creation factory, with unlimited credits, no side restriction and all content available :

```
nul = [<my_factory>] execVM
"R3F_LOG\USER_FUNCT\init_creation_factory.sqf";
```

The key-word "<my_factory>" can be **this**, if you are in the initialization line of the factory.

Factory with 50 000 credits, accessible only to the **independent** side, with all content available :

```
nul = [<my_factory>, 50000, independent] execVM
"R3F_LOG\USER_FUNCT\init_creation_factory.sqf";
```

Factory with unlimited credits, accessible only to the **west** side, with **"MEDIUM"** white list content :

```
nul = [<my_factory>, -1, west, "MEDIUM"] execVM
"R3F_LOG\USER_FUNCT\init_creation_factory.sqf";
```

There are three predefined contents : "LIGHT", "MEDIUM" and "FULL". You can also define your own allowed content.

For more information about configuring available content with a white list, read these files :

- R3F_LOG\USER_FUNCT\init_creation_factory.sqf
- R3F_LOG\ config_creation_factory.sqf
- And the section IV.3. of this document

B. Creation credits management

You can interact on the credits of a specific creation factory. It can make an interaction with your mission progress (e.g. when an objective is done, money is sent to the factory as a recompense).

The amount of credits is associated to a specific factory and is global to every players.

The following script to be executed on the **server side** (or only one client) adds 15 000 credits to the creation factory named "my_factory" :

```
private ["_credits"];

// Get the current credits of my_factory
_credits = my_factory getVariable "R3F_LOG_CF_credits";

// Add 15 000 to the value
_credits = _credits + 15000;

// Set the new credits
my_factory setVariable ["R3F_LOG_CF_credits", _credits, true];
```

C. Access permission management

You have the ability to allow or deny the access to all the creation factories, to specific players, at any time.

To do that, there is a configuration variable in "config.sqf" named :

```
R3F_LOG_CFG_string_condition_allow_creation_factory_on_this_client
```

This variable is a **string** to be evaluated as a **SQF condition** during the game. It permits to allow the access to only some specific players or according to any scriptable condition. Read the description of this variable in the file "config.sqf". There are some examples based on the **player's identity, or the server admin**.

In addition to this configuration variable, which allow/deny all the creation factory at the same time, there is a variable associated to each creation factory in order to enable/disable it **individually**. This variable can be set to **true** (i.e. factory disabled) with this command :

```
<the_object> setVariable ["R3F_LOG_CF_disabled", true];
```

It works exactly the same way as the variable described in part III.1.A. Except there is no configuration variable like "**R3F_LOG_CFG_disabled_by_default**" to define if the factory is enabled or disabled when initialized.

3. Advanced script interaction

A. Protect objects to not lose

When a transporter vehicle or cargo is destroyed, the objects loaded into it are lost without possibility to unload them, because in reality, they are supposed to be destroyed during the vehicle burning.

In some case, your mission needs to never lose some objects, like a flag pole giving access to some features or a main ammo crate.

The script "**R3F_LOG\USER_FUNCT\do_not_lose_it.sqf**" does :

- It makes the object indestructible (from any kind of damage).
- It denies the ability to send / sell back the object to any creation factory (if applicable).
- If the object is loaded in a vehicle or cargo which has been destroyed, it will be unloaded / teleported to a given position.
- If the object was lifted to a crashed helicopter, it will be detached / teleported to a given position.
- If the object was towed to a destroyed vehicle, it will be untowed / teleported to a given position.

There is only one required parameter to call the script : the reference to the object to not lose. If called in the mission editor's initialization line, this reference is the keyword **this** :

```
nul = [this] execVM "R3F_LOG\USER_FUNCT\do_not_lose_it.sqf";
```

There is a **second optional parameter** to choose where to teleport the object when needed. Read its description in the script to know how to use it.

B. How to know the logistics state of an object or vehicle

The [R3F] Logistics uses few object's variable to store their logistics state. To satisfy some specific needs, you may have to read their values by using the command **getVariable**.

Note that, except if you know exactly what you are doing, you should NOT modify their values with the command setVariable.

These variables can give you the answer to questions like :

- Is my object currently transported in a vehicle or cargo ?
- Is my object carried and moved by a player ?
- What is the content of my vehicle/cargo ?
- Which vehicle/cargo is transporting my object ?
- Etc.

Below is the listing of useful variables. The value "objNull" or " [ ] " corresponds to the default value if the variable is not applicable or not yet defined.

- my_object getVariable ["**R3F_LOG_est_deplace_par**", objNull]

  "est_deplace_par" means "**is (this object) carried by (a player)**".
  This variable is applicable to an object which is manually movable by a player.
  If the object is currently carried, the value corresponds to the player carrying it.
  If it is not currently carrying or non-carry-able, the value is objNull (test with *isNull*).

- my_object getVariable ["**R3F_LOG_est_transporte_par**", objNull]

  " est_transporte_par" means "**is (this object) transported by (a vehicle)**".
  This variable is applicable to an object which can be transported / lifted / towed.
  If the object is currently transported (cargo, lifted or towed), the value corresponds to the vehicle which is transporting it.
  If it is not currently transported or not applicable, the value is objNull (test with *isNull*).

- my_object getVariable ["**R3F_LOG_objets_charges**", **[ ]**]

  "objets_charges" means "**(list of) loaded objects (into this vehicle/cargo)**".
  This variable is applicable to vehicle/objects able to transport objects.
  If the vehicle/cargo is currently transporting some objects, the value corresponds to the **array** of the transported objects.
  If it is not currently transporting objects or not applicable, the value is an empty array.

- my_object getVariable ["**R3F_LOG_remorque**", objNull]

  "remorque" means "**is (this vehicle) towing (something)**".
  This variable is applicable to a vehicle able to tow something.
  If the vehicle is currently towing, the value corresponds to the towed object.
  If it is not currently towing or not applicable, the value is objNull (test with *isNull*).

- my_object getVariable ["**R3F_LOG_heliporte**", objNull]

  "heliporte" means "**is (this helicopter) lifting (something)**".
  This variable is applicable to a helicopter able to lift something.
  If the helicopter is currently lifting, the value corresponds to the lifted object.
  If it is not currently lifting or not applicable, the value is objNull (test with *isNull*).

- my_object getVariable ["**R3F_LOG_proprietaire_verrou**", <mixed>]

  "proprietaire_verrou" means "**which (side/faction/player) owns the lock of (something)**".
  The type of this variable can be a side, a faction or a player according to the settings of the configuration variable R3F_LOG_CFG_lock_objects_mode.
  If the object is not owned, the value is *nil* (test with *isNil*).

  To know if an object is locked for the player, according to the lock mode, use this condition :

```
if ([_object, player] call R3F_LOG_FNCT_objet_est_verrouille) then {/* LOCKED */} else {/* NOT LOCKED */};
```

  To modify the lock's owner, according to the lock mode, use this function :

```
[_object, player] call R3F_LOG_FNCT_definir_proprietaire_verrou;
```

- **R3F_LOG_joueur_deplace_objet**

  "joueur_deplace_objet" means "**player is carrying and moving (something)**".
  This variable is NOT to be read with getVariable. It is a simple script variable, local to each client.
  If the local player is currently carrying and manually moving an object, the value corresponds to the carried object.
  If the player is not carrying an object, the value is objNull (test with *isNull*).

C. Quick hard global deactivation without uninstalling

It is possible to quickly deactivate (and reactivate) the whole logistics (and creation factory) system without uninstalling it.

This deactivation method is quick and easy because :

- You don't have to remove the line in "init.sqf" and "description.ext"
- You don't have to remove the calls to the scripts from "R3F_LOG\USER_FUNCT\" like :
  - auto_load_in_vehicle.sqf
  - init_creation_factory.sqf
  - watch_objects_to_not_lose.sqf

This method is a "hard" deactivation because there will be zero CPU or memory usage due to the logistics during the mission. There will be no interference from the logistics to the mission.

To deactivate it, open the file "**R3F_LOG_ENABLE.h**" in the folder "R3F_LOG". If you add a **double-slash** at the begin of the line 11, the whole logistics system will be deactivated :

```
//#define R3F_LOG_enable
```

If you keep this line without adding a double-slash, the system will be active :

```
#define R3F_LOG_enable
```

Note that this method is useful for a temporary deactivation. You must **NOT delete the folder** "R3F_LOG". If you want to delete the folder "R3F_LOG" to save space in your mission, you have to delete the initialization lines in "init.sqf" and "description.ext", and to remove all the calls to the scripts from "R3F_LOG\USER_FUNCT\" (if you used them).

IV. Configuration

The [R3F] Logistics is quite configurable, especially concerning the decision to give or not some logistics features to a type of object/vehicle.

1. General configuration variables

The main configuration file "**config.sqf**" defines some general configuration variables listed below. They are all well documented in the configuration file, so read it for more explanation.

- R3F_LOG_CFG_**disabled**_by_default
  Chose if the logistics features are enabled or disabled when the objects are created.
  Individual enable/disable can be done to overwrite this default value.
  Read the part III.1.A. to learn more about it.

- R3F_LOG_CFG_**lock**_objects_mode
  Permit to choose if the logistics features on an object are locked to the owning side, faction or player.
  The lock feature can be disabled by setting this variable to "none".
  See also the following variable to configure the unlock feature.

- R3F_LOG_CFG_**unlock**_objects_timer
  Define how many seconds are needed to unlock the logistics features of an object which is owned by another side/faction/unit than the player.
  To deny the unlock feature, set this variable to -1.

- R3F_LOG_CFG_**no_gravity_objects**_can_be_set_in_height_**over_ground**
  Define if movable objects with no gravity simulation can be set in height over the ground (i.e. no ground contact), or if they can't be elevated higher than the ground/floor.

- R3F_LOG_CFG_**language**
  Used to add new translations.
  Read the chapter 4. to know how to.

- R3F_LOG_CFG_string_condition_**allow_logistics**_on_this_client
  SQF condition to allow or deny all the logistics features.
  The condition is evaluated on-the-fly on each client. It permits to allow it **only on some specific clients**.
  Read the part III.1.C. to know how to use it.

- R3F_LOG_CFG_string_condition_**allow_creation_factory**_on_this_client
  SQF condition to allow or deny the access to all the creation factories.
  The condition is evaluated on-the-fly on each client. It permits to allow it **only on some specific clients**.
  Read the part III.2.0. to know how to use it.

2. Logistics features configuration

A. Explanation about generic/parent class, inheritance and CfgVehicles

For each logistics feature, there is an array in the configuration file which defines the list of objects/vehicles which must have this feature. Each array contains the class names of the willed objects. Each class name corresponds to an entry mentioned in the "BIS' config file" named CfgVehicles.

To get a class name, you can aim the object in-game and type "**typeOf cursorTarget**" in a "watch" line of the BIS' debug console. You can also view the list of all the existing class names in the CfgVehicles by using the "config" button in the BIS' debug console :

All the objects present in the CfgVehicles are organized according a tree view (hierarchy), which permits to have generic classes, also called base class or parent class. The "BIS' config viewer" shows this hierarchy in the line "Parents". But is a not very practical to use. You can see a clearer view of this hierarchy here (example for A3 1.32) :
http://madbull.arma.free.fr/A3_1.32_CfgVehicles_tree.html

Here is a sample of this hierarchy :

- Truck_F
  - Truck_01_base_F
    - B_Truck_01_transport_F
      - B_Truck_01_covered_F
      - B_Truck_01_medical_F
      - B_Truck_01_mover_F
        - B_Truck_01_Repair_F
        - B_Truck_01_ammo_F
        - B_Truck_01_box_F
        - B_Truck_01_fuel_F
  - Truck_02_base_F
    - I_Truck_02_ammo_F
    - I_Truck_02_box_F
      - I_Truck_02_medical_F

As we can see, the class name "Truck_F" is the parent classes of "Truck_01_base_F" and "Truck_02_base_F". These two classes have also their own child and grandchild classes, etc.

If the class "Truck_F" is added to the array configuration variable associated to the towing ability, then this class, and all child (and grandchild) classes will have this feature.

But if we put only "Truck_01_base_F" in the array, only the trucks derived from it will have the towing feature. All the trucks deriving from "Truck_02_base_F" won't have the feature.

B. Features related configuration variables

An array of class names is defined for each logistics feature. As explained in the previous part A., the system uses the CfgVehicles **inheritance principle**. Here are the available features' arrays :

- R3F_LOG_CFG_can_**tow**
- R3F_LOG_CFG_can_**be_towed**

- R3F_LOG_CFG_can_**lift**
- R3F_LOG_CFG_can_**be_lifted**

- R3F_LOG_CFG_can_**transport**_cargo
- R3F_LOG_CFG_can_**be_transported**_cargo

- R3F_LOG_CFG_can_**be_moved_by_player**

Read carefully the explanation in the file "config.sqf". Also, take care of the features "can transport" and "can be transported" which have a numeric quantification of the **capacity and cost of transport** (kind of volume/weight unit).

As explained in "config.sqf", there are **two ways** to add new objects in the logistics system :

- The first one is quick and appropriate to add/fix **only some various class names**. It consists to add these classes names directly in the (initially empty) arrays of the file "config.sqf".

- The second one is cleaner and is especially appropriate to take into account an **additional addon**. It consists to create a new file in the folder "/addons_config/".

  The new file must be a copy of the template file "**/addons_config/TEMPLATE.sqf**".
  Then add the addon's class names in the arrays according their logistics capability.
  To take this new config file into account, you must add a "**#include**" line to your own file, in the file "config.sqf", just below the already existing "#include" lines.

Syntax example to add some objects which can be towed :

```
R3F_LOG_CFG_can_be_towed = R3F_LOG_CFG_can_be_towed +
[
     "MyTowableObjectClassName1",
     "MyTowableObjectClassName2",
     "MyTowableObjectClassName3"
];
```

Syntax example to add some objects which can transport, and their associated load capacity :

```
R3F_LOG_CFG_can_transport_cargo = R3F_LOG_CFG_can_transport_cargo +
[
     ["MyTransporterClassName1", 150],
     ["MyTransporterClassName2", 200],
     ["MyTransporterClassName3", 100]
];
```

C. Helping tool to edit the logistics configuration

To manage the huge list of the Arma III vanilla class names, in order to create their logistics configuration, we made a little tool. This tool is delivered with the [R3F] Logistics, because it may be helpful to configure an additional addon. But it is an unofficial tool for **expert only**.

Thanks to this tool, you will **browse in-game** your own list of objects/vehicles to configure. You will be able to use some keyboard controls to **set on-the-fly** their associated logistics features.

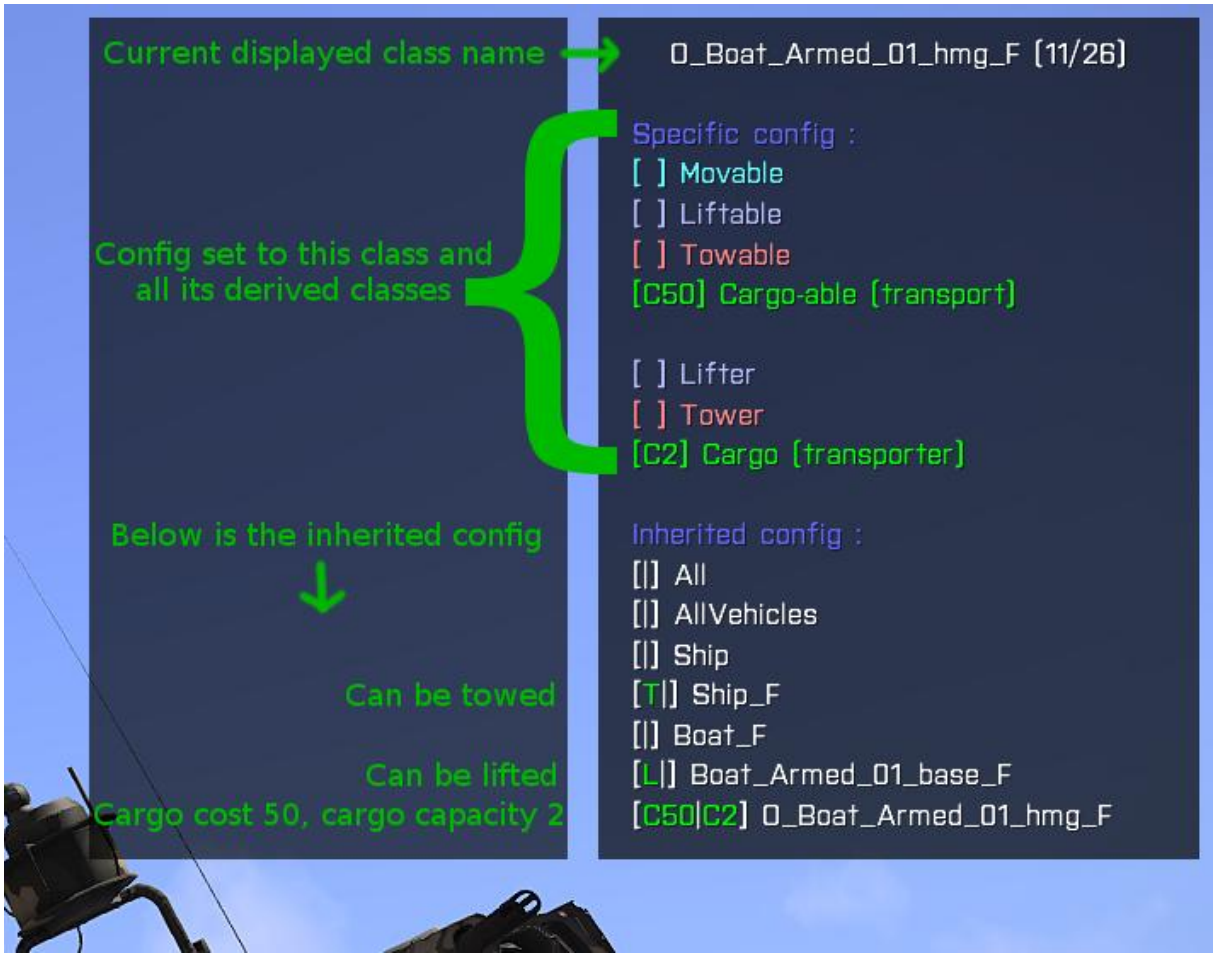
When you finished, you can use the action "**Dump config to RPT**" in your menu to print all the logistics configuration arrays in the Arma RPT file. You will probably need to use a software like **WinMerge** to view the modifications in order to manually update the configuration files, as explained in the previous part B.

When you are on the tool's interface (shown on the next page), the following **keyboard controls** will be available to edit the configuration :

- Right/left arrows : switch the next/previous class name
- In lower-case are the "-able" features :
    - m : Movable, the object can be carried and moved by the player
    - l : Liftable, the object can be lifted
    - t : Towable, the object can be towed
    - c **+ number** : Cargo-able (transportable) with associated **cost capacity**, entered with the numpad before validating with "enter".
- In upper-case are the "-er" features :
    - L : Lifter, the object can lift
    - T : Tower, the object can tow
    - C **+ number** : Cargo (transporter) with associated **load capacity**, entered with the numpad before validating with "enter".
- Enter key : add the logistics feature which has just been typed. For example :
    - "m" then "enter" add the feature "Movable"
    - "T" then "enter" add the feature "Tower"
    - "C" then "120" then "enter" add the "Cargo" features with a load capacity of 120
- Delete key : remove the logistics feature which has just been typed. For example :
    - "m" then "delete" remove the feature "Movable"
    - "T" then "delete" remove the feature "Tower"
    - "C" then "delete" remove the feature "Cargo" (and the associated capacity)

To use this tool, go to the folder "R3F_LOG\addons_config\logistics_config_maker_tool".
Othe file "**list_of_objects_to_config.sqf**" and list the class names you want to configure.
Launch the mission and execute this script (from the debug console or the player's init line) :

```
execVM "R3F_LOG\addons_config\logistics_config_maker_tool\launch_config_tool.sqf";
```



The interface has two parts :

- The "specific config" : it shows the logistics features given to the displayed class name. This class and its child (and grandchild) will have these features.

  If the brackets of a feature are empty, the feature is not set (but could be inherited).
  If there is a letter in the brackets, the associated feature is set.
  For the "cargo-able" and "cargo", there is also an associated cost or capacity quantification.

- The "inherited config" : it shows all the parent and grandparent classes.
  For each of them, we can see the features that the displayed class name inherits.
  The brackets are divided in two features' categories : first the "-able" features, then the "-er" features. The same codification as the keyboard controls is used :

  For example [LTC50 |] means the object can be lifted (L), towed (T), and transported in cargo with a cost of 50 units (C50).
  Other example : [| LC120] means the object is a lifter (L) and can transport in cargo a capacity of 120 units (C120).

For the shown example, the effective logistics configuration is :

- It can be towed thanks to the features inherited from the parent class "Ship_F".
- It can be lifted thanks to the inheritance from "Boat_Armed_01_base_F".
- Thanks to its own configuration, the boat can be loaded in a vehicle/cargo with a cost of 50 units.
- And the boat can itself be a transporter vehicle with a load capacity of 2 units.

3. Creation factory configuration

There is a specific configuration file named "**config_creation_factory.sqf**" to configure the predefined lists of available content according to the "LIGHT" / "MEDIUM" / "FULL" white lists and also to the black list.

This file defines also a variable named "R3F_LOG_CFG_CF_sell_back_bargain_rate" to allow/deny to send back objects to the factory, and to define the percentage of reimbursement.

The creation factory's content is split in several object's categories, which are exactly the same as displayed in the mission editor. In a technical point of view, each category corresponds to an entry in the "BIS' config file" named CfgVehicleClasses.

The white lists and the black list are using the class names of these categories listed in the CfgVehicleClasses.

You can get the list of existing categories by using a script which writes to the RPT file the list of categories' class names. You can call it with the BIS' debug console by executing this line :

```
nul = [true, true] execVM
"R3F_LOG\USER_FUNCT\dump_creation_factory_categories.sqf";
```

The content available in a factory is defined thanks to the parameters passed to the factory initialization script "R3F_LOG\USER_FUNCT\ init_creation_factory.sqf" (described in part III.2.A.).

If the third parameter is set to "LIGHT" / "MEDIUM" / "FULL", the corresponding white list variable "R3F_LOG_CFG_CF_whitelist_XXX_categories" defined in "config_creation_factory.sqf" will be used. If this parameter is not defined, all the game content will be allowed, except the categories mentioned in the black list. This parameter can also be a custom array of categories class names.

The variable "R3F_LOG_CFG_CF_creation_cost_factor" permits to modify the default prices for some categories.

Read also the explanation in the files :

- config_creation_factory.sqf
- R3F_LOG\USER_FUNCT\ init_creation_factory.sqf
- And the part III.2.A. of this document

4. Adding a translation

The [R3F] Logistics implements a multi-language support system. It is NOT based on the BIS' stringtable files (xml or csv) which are not enough easy to use for the mission maker.

This alternative multi-language system uses SQF files named "**XX_strings_lang.sqf**" in the folder "R3F_LOG", where XX is the language code (e.g. "en" for the English language).

The official delivery of the logistics supports two language :

- **en**_strings_lang.sqf (**English** translation)
- **fr**_strings_lang.sqf (**French** translation)

You can create an additional translation by duplicating and editing an existing language file. If you want to translate the logistics in **Polish**, create a file named "**pl**_strings_lang.sqf".

Then, you have to register this new language in the file "**config.sqf**", in the variable "**R3F_LOG_CFG_language**" :

```
R3F_LOG_CFG_language = switch (language) do
{
	case "English":{"en"};
	case "French":{"fr"};

	case "Polish":{"pl"};

	default {"en"}; // If language is not supported, use English
};
```

V. Contact and credits

For any request related to the [R3F] Logistics software, visit the official thread on the BIS' forum :
http://forums.bistudio.com/showthread.php?180049-R3F-Logistics

The R3F is a French team playing on the Arma series since its first opus.
Our philosophy : realism, team play and discipline.

Credits :

~R3F~ madbull
~R3F~ Juw
~R3F~ Bidasse
~R3F~ Leto
~R3F~ Nash
~R3F~ John
~R3F~ Chepadbol
~R3F~ Mexmarsouin

All the ~R3F~ members for their tests, support and suggestions.

VI. License