The FHQ Task Tracker for ArmA II/Arma 3

Version 1.1

Introduction

Tracking tasks and briefings among different clients is tedious and error prone if you do it by hand, especially in multiplayer sessions, with possible clients joining in progress. The FHQ Task Tracker is designed to take over the management of these task, including handling of tracking tasks states, marking tasks as completed, and distributing the current state of tasks and the briefing to clients that join in progress.

History

- V1.0 – Initial release
- V1.1 – Added Arma 3 notification for task hints, using the default task notification resources

Usage

The FHQ Task Tracker is a single script that has to be executed on each client that wants to share in tracking tasks. The easiest way is to execute it in init.sqf. This is achieved by adding a single line to init.sqf like this:

```
call compile preprocessFileLineNumbers "fhqtt.sqf";
```

This will set up the system and pre-initialize a number of variables that contain code that can be called from your own code. All function variables have a prefix of `FHQ_TT_` to avoid clashes with other script packages.

Briefings are added by calling `FHQ_TT_addBriefing`. It is advisable to run this code on all clients, by creating the briefing from within init.sqf, or a script called from within init.sqf. Similarly, tasks are added by calling `FHQ_TT_addTasks` with the same constraints. Tasks and briefing entries can be added dynamically during the mission to adapt the mission to new circumstances.

Before going into the details of these functions, we need to discuss the concept of filters.

Concepts: Filters

Filters are used to select the units that will receive tasks or briefing entries. Functions that add briefing or task entries will be default assign entries to all playable/switchable units. A filter can be used to restrict entries to a certain subset. Once a new filter is encountered, the units matching the previous filter are removed from the pool and the new filter is applied to the remaining units. That way, you can work up from specific to general.

This sounds may sound confusing, so let us illustrate this with an example. Assume we have groups called "PlayerGroup1", "PlayerGroup2", and PlayerGroup3 which are of side "west". You want to add a briefing entries "Mission" and "Situation" for all of them, but PlayerGroup1 has to have different entries. Such a situation would look like this:

```
[
   PlayerGroup1,
     ["Mission", "Kill the enemy leader"],
     ["Situation", "The enemy leader has nuclear missile bomb"],
   west,
     ["Mission", "Assist Alpha in the advance"],
     ["Situation", "We ran out of beer"]
] call FHQ_TT_addBriefing;
```

When working through this list, the first filter encountered is PlayerGroup1. This is applied to all playable units, selecting all units that are in PlayerGroup1. These will receive the subsequent "Mission" and "Situation" entries. The next filter is "west". When encountered, all previous units (PlayerGroup1) will be removed from the available units pool, and the filter applied to the remaining units, so every remaining unit that is of side west will receive the following "Mission" and "Situation" entries (in our example, the remaining players of PlayerGroup2 and PlayerGroup3).

Filters can be one of the following:

- Single object: A single unit that will receive the entries
- Group: All playable units belonging to this group
- Side: All playable units belonging to the side
- Faction (string): All playable units of the given faction
- Code: A piece of code in curly braces. Return true if you want the unit to be included in the filter. The unit passed to the code as "_this".

Examples of valid filters:

- `{(side _this) != west}`: All units that are not BLUFOR
- `player`: The local player.
- `"BIS_BAF"`: All British soldiers

Defining Briefings

A briefing entry always consists of an array of two strings. Currently, all entries are added under the "Notes" menu item in game ("Diary" topic). Each entry consists of an array of two strings, the first being the header (which appears in the middle column right next to the "Notes"), and the second the actual text. This can be any structured text (including images and html tags).

Defining Tasks

Tasks are a little more complex than briefing entries. A task entry consists of an array with a variable number of elements. The elements are:

- The task name. This is either a string ("taskDestroy"), or an array of two strings (["taskDestroy1", "taskParent"]), the first string being the new task and the second one the parent task.
- Task long description. A single string (can be structured text) that will appear on the rightmost column describing the task in detail.
- Task short description. A single string that is used as a header in the center column of the briefing, and in task hints.
- Task waypoint text. A single string that is displayed on top of a waypoint in the main view. Should be short (like "DESTROY", or "MOVE", etc). This can also be an empty string, in which case nothing is displayed on the waypoint.
- Task waypoint (optional). This can be either an object, or a position (three element array), or it can be omitted. If it's an object, the task marker on the map and the waypoint in the main view will always point to the object. If a position, it will remain on that position. If omitted, no waypoint or map marker will be displayed.
- Initial state (optional). A string that denotes the initial state of the task. If omitted, the task is automatically set to "created". Note: It is good practice to set at least one task to the "assigned" state to give players a sense of a general approach.

A task can be one of five states, "created", "assigned", "succeeded", "failed", and "canceled". This corresponds to the same states that are passed to setTaskState.

An example of a simple task list is given here:

```
[
   PlayerGroup,
     ["taskDestroy", "Search and destroy",
      "Destroy enemy tanks", "", "assigned"],
     ["taskExfil", "Exfiltrate",
      "Exfiltrate", "", getMarkerPos "markExfil"]
] call FHQ_TT_addTasks;
```

This example defines two tasks and assigns them to the units in PlayerGroup. Note the first task has no position, but an initial state, while the second one has a position.

Tasks are added in the order they are given (FHQ Task Tracker automatically reverses the task to make them appear in the correct order in the briefing in ArmA II). In Arma 3, the engine automatically takes care of that. This also means that parent tasks must be defined *after* the child tasks in ArmA II, but *before* the child tasks in Arma 3.

ArmAII:

```
[
     PlayerGroup,
        [["taskTarget1", "taskDestroy"], ...],
        [["taskTarget2", "taskDestroy"], ...],
        ["taskDestroy", ...],
        ["taskExfil", ...]
] call FHQ_TT_addTasks;
```

Arma 3:

```
[
     PlayerGroup,
        ["taskDestroy", ...],
        [["taskTarget1", "taskDestroy"], ...],
        [["taskTarget2", "taskDestroy"], ...],
        ["taskExfil", ...]
] call FHQ_TT_addTasks;
```

(See taskDestroy's position).

Remaining functions

This chapter lists the remaining functions of the Task Tracker.

- `[_taskName, _state] call FHQ_TT_setTaskState;`

  Set task `_taskName` to state `_state`. This will pop up a task hint on every client controlling a unit that has this task.

- `_res = [_taskName] call FHQ_TT_getTaskState;`

  Query the current state of task `_taskName`. The result is a string representation of the state ("created", "assigned", "succeeded", "failed", or "canceled")

- `_res = [_taskName] call FHQ_TT_isTaskCompleted;`

  Check if the task `_taskName` is completed. Result is a bool. A task is considered completed if it's state is "succeeded", "failed", or "canceled".

- `_res = [_task1, _task2, ...] call FHQ_TT_areTasksCompleted;`

  Returns true if all tasks in the list are completed ("succeeded", "failed", or "canceled").

- `_res = [_taskName] call FHQ_TT_isTaskSuccessful;`

  Returns true if the task `_taskName` has state "succeeded".

- `_res = [_task1, task2, ...] call FHQ_TT_areTasksSuccessful;`

  Returns true if all tasks in the list are of state "succeeded".

- `_list = [_state] call FHQ_TT_getAllTasksWithState;`

  Returns an array of strings, each string corresponding to all tasks with state `_state`.

- `[_task1, _state1, task2, ...] call FHQ_TT_markTaskAndNext;`

  This function sets the state of _task1 to _state1, popping up a task hint. It then iterates through the remaining elements and checks each task name whether it's completed or not. If it finds a task that is not completed, it will set it's state to "assigned", in turn popping up a task hint. This function is useful for selecting a new task after finishing one.

The Demo Mission (Arma 3 only)

To use the demo mission, copy the fhq_tasktracker_demo.Stratis folder into your Arma 3 profile mission folder. This is usually in your Documents under Arma 3 Alpha/<name>/missions or Arma 3 Alpha - Other Profiles/<name>/missions. The init.sqf

defines the briefing and tasks. The mission file itself demonstrates how to do slightly more complex mission structures without much scripting, mainly using game logics, triggers and waypoints.

Terms of Use

You are free to use the task tracker in your own projects. I only ask that you do not modify the script without asking for permission first. Getting a mention in the credits is appreciated, although not a requirement.

Credits

FHQ Task Tracker was written by Varanon with ArmaDev by Alwarren. I can be reached by email at thomas@friedenhq.org or on the BIS forum under the username of Varanon.