DIFFERENCES IN SCRIPTING BETWEEN OPERATION FLASHPOINT 1.96 AND COLD WAR ASSAULT 1.99

Differentiating between OFP and CWA

Before we start using new features we need to have a method of checking whether our script is running on OFP or CWA. Unfortunately the game does not feature a clean method to check its version but there are workarounds:

1. **Check against a new command**

```
if (Format ["%1",getworld]!="scalar bool array string 0xfcffffef")
then {hint "CWA"} else {hint "OFP"}
```

Works until someone would use the name `getworld` as a variable (which I find unlikely).

2. **Fwatch information command**

```
if (call loadFile ":info version extended" select 1)
then {hint "CWA"} else {hint "OFP"}

if (call loadFile ":info version extended" select 4 == 1.99)
then {hint "CWA"} else {hint "OFP"}
```

This is the most reliable way but it requires Fwatch 1.13 or newer.

New Commands

`find`

This command searches for the specified item in the given array and returns index of the first occurrence or `-1` if item was not found. Case sensitive.

```
_index = [1,2,3,4] find 4
; _index will be 3
```

How to do it in OFP: use a custom function like FindInArray.

Remarks: advantage of this command is that it can safely (without generating an error) search array that holds various data types while custom function requires a workaround. Downside is that it returns error when searching for a non-existent item (like an undeclared variable) and it doesn't feature option for case insensitivity (custom function could deal with both of those issues).

`isServer`

Returns `true` if a machine on which this command is executed is acting as a game server (dedicated or player-hosted). Returns `false` on clients and in single player.

```
if (isServer) then {}
```

How to do it in OFP: insert Game Logic in the Mission Editor, name it and then use `local` command for detection. For example:

```
if (local server) then {}
```

Remarks: useful when writing scripts with multiplayer compatibility. CWA makes a bit more convenient. For example when you're writing/using an external script (e.g. from an addon) you don't have to worry whether missionmaker placed Game Logic in the mission.

getWorld

Returns extension name of the island on which the current mission is taking place.

```
hint getWorld
; e.g. game will display: eden
```

How to do it in OFP: use Fwatch command `:mem getworld`.

```
hint loadFile ":mem getworld"
```

Remarks: useful when writing general scripts that have island dependency. For example displaying mini-map in the vehicle cockpit. Perhaps writing a script that has pre-defined information about islands (e.g. locations of all the of the forests). Also finding mission folder name for logging purposes.

getPosASL

Returns position of the given object with absolute height (over the sea level).

```
_pos = getPosASL player
; e.g. _pos will be [9475.12, 4057.52, 29.8336]
; getPos would give [9475.12, 4057.52, -0.00136375]
```

How to do it in OFP: use a custom function like getPosABS.

Remarks: see `setPosASL`.

setPosASL

Places given object at the specified coordinates with absolute height.

```
player setPosASL [9475.12, 4057.52, 29.8336]
```

How to do it in OFP: use a custom function like setPosABS.

Remarks: useful for keeping an object at a constant altitute regardless of terrain irregularity and other objects below.

You may already know that `setPos` command has trouble with placing static objects. There is an offset between given coordinates and the actual position of the object. This issue does not seem to occur with `setPosASL`.

New Functionality In Existing Commands

addEventHandler "AnimChanged"

Will execute given code every time unit changes animation. Array `_this` contains:

```
0 – object
1 – current animation name
```

```
player addEventHandler ["AnimChanged","hint (_this select 1)"]
; e.g. game will display: combatrunf
```

How to do it in OFP: Fwatch features command `:mem getplayeranim` which returns current animation but only for the player. Also it returns a number and not a string (there is a way to convert it but it's not reliable, see Fwatch practical examples __animChanged.Intro).

Remarks: for programming player movement I prefer using the Fwatch command as it returns current state. This new event handler could prove useful in programming AI units movement or checking stance. Other application could be to support `playMove` command, that is to the use the latter only when unit is idle (`playMove` does not interrupt current animation but adds a new one to the queue which can create animation lag when using it multiple times).

disableAI "ANIM"

Will prevent the specified AI unit from changing animations rendering it immoblized.

```
soldier disableAI "ANIM"
```

Remarks: this is a stronger version of the `disableAI "MOVE"` as the unit won't even watch target, shoot, rearm, get up/down etc. Could prove useful when writing totally new AI routines for units that aren't traditional soldiers (e.g. something like the wolf addon). I conisder it useless in enhancing existing AI behaviour as the effects of this command cannot be reverted. Other application could be to block AI units when testing missions.

Updated Commands

missionName

Returns mission filename. In CWA in multiplayer it returns briefing name instead (it's set in the Mission Editor in the Intel section).

```
hint missionName
; e.g. OFP will show: 1-10_T_TeamFlagFight
; and CWA will show: Team Flag Fight
```

How to do it in OFP: use Fwatch command :mem missioninfo which returns much more information.

```
info = call loadFile ":mem missioninfo";
hint (if (info select 0 in ["__cur_sp","__cur_mp"]) then {(info select
2)+"."+(info select 1)+".pbo"} else {(info select 0)+"."+(info select 1)})
```

Remarks: It could be used to write a game log (which missons were played). Perhaps to write scripts dedicated for specific missions.

missionStart

Returns array with date indicating when the mission has started (after the briefing screen). In OFP this command is bugged – it works only in multiplayer on a player-hosted machine and on a client. In single player it would return `[0,0,0,0,0,0]` and on a dedicated server – `[1977,1,1,1,0,0]`. In CWA this command is fixed.

```
hint Format ["%1", missionStart]
; e.g. CWA will show: [2014,8,13,2,58,47]
```

How to do it in OFP: use Fwatch command :info date in the `init.sqs` file, at the beginning, after a short time interval.

```
~0.001
hint Format ["%1", call loadFile ":info date"]
```

Remarks: It could be used to write a game log (mission playing times).