IGN Events and Functions Library
Igneous01

Contents

Installation

General

Extract the IGN_EH file somewhere in your Arma 3 editing folder. This will create a new folder called 'IGN Events' with 3 different folders inside.

Each folder is a specific way of installing/using this library in your game. These are:

- **Scripted (Header Only)**
  - folder can be copy/pasted into your mission – only need to call the initialization from init.sqf
  - Supports debugging by enabling macro IGN_LIB_DEBUG

- **cfgFunctions**
  - folder can be copy/pasted into your mission – need to include IGN_EH_Definition.hpp into your description.ext
  - All Functions will be available through the in-game functions viewer
  - Supports debugging by enabling macro IGN_LIB_DEBUG

- **Addon (WIP)**

Install for Scripted (Header) version

- copy the 'IGN Events\Scripted\IGN_LIB' folder into your mission
- put this into your init.sqf file:

  call compile preprocessfilelinenumbers "IGN_EH\IGN_EH_INIT.sqf";

  note: the library is flexible with paths. If you require moving IGN_EH into another folder (ie. scripted\IGN_EH) then you can pass a string parameter that specifies the path where IGN_EH_INIT.sqf should look to build and compile all functions from.
  Example:

  "scripted\IGN_EH" call compile preprocessfilelinenumbers "scripted\IGN_EH\IGN_EH_INIT.sqf";

Install for cfgFunctions version

- copy the 'IGN Events and Functions\CfgFunctions\IGN_LIB' folder into your mission
- put this into your description.ext file:

  #include "IGN_EH\IGN_EH_Definition.hpp"

  Note that you may need to re-save your mission in-game for the functions to be compiled.

Install for Addon version

- Currently in the works

Enabling Debugging:

A Debug mode feature exists for this framework, which spits out important logging information when various calls are made. This is useful for when you are debugging a mission and need as much information as possible to identify a problem. Debug mode also validates event objects and their arguments, and will display an error if arguments types do not match for example.

```
 0:33:22 IGN_EH: IGN_fnc_raiseEvent: Event (SERVER_EVT_TASK_CHANGED) raised by client(3) with arguments ([PLAYER1,"TASK2","SUCCEEDED"])
 0:33:22 IGN_EH: IGN_fnc_raiseEvent: Calling handler for (SERVER_EVT_TASK_CHANGED) (handlerOwner(2), eventOwner(2), code({
private["_obj", "_task", "_status"];
_obj = _this select 0;
_task = _this select 1;
_status = _this select 2;



[
[_obj, _task, _status],
{
hint format ["player %1 updated task %2 to status %3",
(_this select 0), (_this select 1), (_this select 2)];
}
] remoteExecCall ["bis_fnc_call", 0];


if (_task == "TASK1") then
{
hvt addEventHandler ["killed", {[SERVER_EVT_TASK_CHANGED, [_this select 1, "TASK3", "SUCCEEDED"]] call IGN_fnc_raiseEvent;}];
}
})
 0:33:22 IGN_EH: IGN_fnc_raiseEvent: (SERVER_EVT_TASK_CHANGED) handlerOwner(2) == eventOwner(2) - calling handler normally
 0:33:22 IGN_EH: IGN_fnc_raiseEvent: Calling handler for (SERVER_EVT_TASK_CHANGED) (handlerOwner(3), eventOwner(2), code({
private["_obj", "_task", "_status"];
_obj = _this select 0;
_task = _this select 1;
_status = _this select 2;
[_task, _status, true] spawn BIS_fnc_taskSetState;


if (_task == "TASK1") then
{
[west, "TASK3", ["Kill the Officer", "Kill the Officer", "hvt"],  getpos hvt, true] spawn BIS_fnc_taskCreate;
}
})
 0:33:22 IGN_EH: IGN_fnc_raiseEvent: (SERVER_EVT_TASK_CHANGED) handlerOwner(3) != eventOwner(2) - remoteExecCall on event owner machine
```

Because Debug mode is very thorough, it also slows down performance. This is why the debugging is enabled through a macro, as the macro will add/remove the snippets of code on compilation. When debug is set to false, all debug code is not included in the final compiled functions, making them run at their optimal speed. It is recommended to turn of debug mode when releasing a mission to the public.

First, a limitation with macros in Arma – they cannot be included from a parent folder, therefore it is not possible to include a header file that exists one folder above from where the current file resides. Hence there are multiple copies of the same macro header file in the various subfolders.

These are as follows:
IGN_EH\IGN_EH_Macros.h
IGN_EH\event\IGN_EH_Macros.h
IGN_EH\functor\IGN_EH_Macros.h
IGN_EH\event\display\IGN_EH_Macros.h

The contents of all these files should match.

Here is what the header files looks like for debug mode enabled:

```
// uncomment next line to enable debugging and logging
#define IGN_LIB_DEBUG
```

If you wish to ***enable debug***, simple define IGN_LIB_DEBUG in all macro files
If you wish to ***disable debug***, remove the #define IGN_LIB_DEBUG in all macro files

Events

Events and Event Driven Programming

"*An event is a message sent by an object to signal the occurrence of an action. The action could be caused by user interaction, such as a mouse click, or it could be triggered by some other program logic. The object that raises the event is called the event sender. The object that captures the event and responds to it is called the event receiver.*" - MSDN - Events and Delegates for .NET

An Event is basically a kind of sender. It takes a message, and when its time, it will 'send' this message to anyone who is listening. That's all there is to it – ofcourse events can come in a variety of flavors that make them easier or more robust to work with. But that's all just overhead.

Any listeners that 'receive' the message will do something with it. This could be anything - ranging from complex tasks such as interpreting key presses and changing a windows state (GUI programming) to simple logging for debugging purposes. In short, the listener can do whatever he wants when the message has been received (including ignoring it).

I'm sure some of you are wondering "what does all this have to do with scripting?"

Well, we can use this idea (or paradigm) of messages and signals to help simplify our code.

Problem with handling mission flow

For example, suppose we are writing a multiplayer mission where we need the ability to notify all clients when something has happened. One method of accomplishing this is to use remote execution on all machines the moment something occurs.

This is great for one off items, but gets complex when both client and server need to remote executing code, as there is a large margin for error and bugs to appear that are difficult to fix or even identify. The chance that multiple clients may attempt to remote execute the same piece of code multiple times causes lag, unnecessary traffic, and odd behaviours. Using broadcasting events with remote handlers, we can organize our code into simple and understandable chunks that are easier to understand and maintain.

Events in Arma 3

Currently Arma 3 has some support for events via the BIS_fnc_addScriptedHandler and BIS_fnc_callScriptedHandler functions that were added into Arma 3. (CBA has it's own implementation of events). It covers some of the essentials, but does not support remote execution or multiplayer for that matter.

Events in Arma are pretty important for a number of reasons:

- You don't have to manage what order something should happen
- You're instantly notified when something happens

- Events check by frame with decent performance
- It abstracts the workload – you don't need to care about that stuff, the engine does it for you

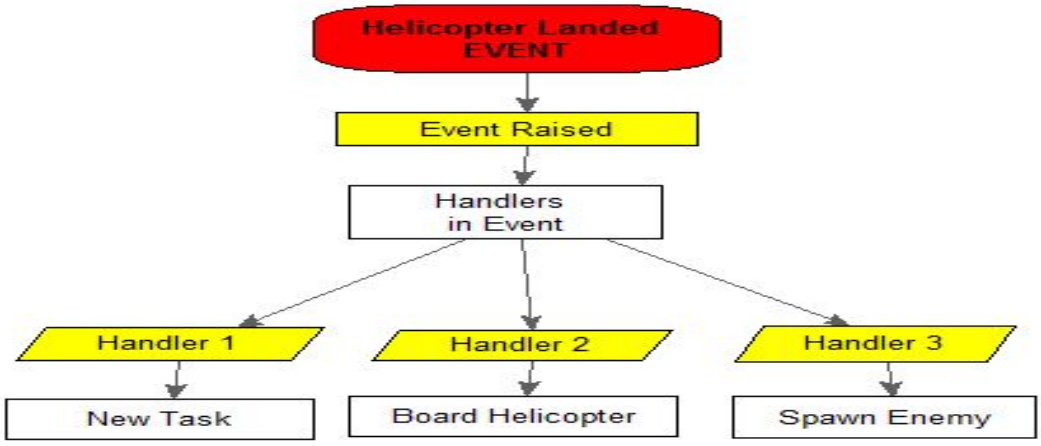
But also equally important – events make your life as a mission maker EASIER

Common Event Uses in Arma 3

You as a mission designer, want to know...

- when a helicopter lands
- when a player has connected
- when a task has been created
- when a unit/object has been destroyed
- when a group is dead
- if all surviving members are inside a vehicle
- when enemies have detected the player
- when a unit is out of ammo
- when the state of a task has changed
- when something is in range
- when the server does something that effects all clients

These are signals – you want the game to signal to you when one of these things happens, so you can respond to it. Hopefully with this guide you're life should be much easier, so you can focus on what you want from a mission, rather than managing when you want it.



Events and Multiplayer

Event handling has many uses in multiplayer, as it helps distinguish clear boundaries between individual clients and server, and helps organize code in an easy to read way. This section covers the design and architecture of broadcasting events.

When a broadcasting event is created, the machine that created the event is the owner of that event. When handlers are registered to the event, the machine that added the handler is the owner of that particular handler. Any machine can raise a broadcasting event / add a handler, but there can only be one owner for the event.

When a broadcasting event is raised, it is called from the machine that invoked IGN_fnc_raiseEvent. This means that machine runs the code necessary to notify all listeners that something has occurred.

As the handlers are called, any handlers that are local to the machine that raised the event are called normally. When a handler is encountered that is owned by a different machine, remote execution is used on the handler owners machine. This ensures data is propagated to all registered listeners.