IGN Functions List and Documentation

Igneous01

Contents

IGN_fnc_createEvent

Creates an event object, allowing registration of handlers and raising. Supports broadcasting to all machines in a multiplayer game. When broadcasting is set to true, the event will be created and the return variable broadcasted to all machines. The owner of the event is the machine that this command was run on. If broadcasting is enabled, this call ***must*** only be run on a single machine (such as a server) to prevent undefined broadcasting to all machines.

Parameters:

args (Array) (Optional) - Argument ***types*** the handlers must support
default value - [] (no arguments)

name (String) (Optional) - The name of the object (this will set the vehicle's variable name). It is strongly recommended that a name is supplied if broadcasting is enabled, otherwise remote machines will not be notified.
default value - "" (no name)

broadcast (Bool) (Optional) – determines if global event – if set to true, the event will support remote handlers from clients and will propagate across the network.
default value – false (no broadcasting)

Returns:

The created event object, with these getters in its namespace:
"name" - the name of the event (Argument passed)
"handlers" - an array containing all handlers currently registered to the event
"arg_types" - an array containing the argument types
"raised" - a bool value determining if the event is currently being raised
"broadcast" – a bool determining if this is a broadcasting event
"owner" – number, clientID that owns this event

Example:

```
// no args local event
myEvent = [[], "myEvent"] call IGN_fnc_createEvent;


// will send arguments of type object (_this select 0) and array (_this select 1)
myEvent = [[objNull, []], "myEvent"] call IGN_fnc_createEvent;


// global (broadcasting) event, is created only on the server, but clients can raise this event
// and addHandlers to it from their own machines
if (isServer) then {
	myGlobalEvent = [objNull, "myGlobalEvent", true] call IGN_fnc_createEvent;
};
```

IGN_fnc_deleteEvent

Deletes an existing event object. Broadcasts to all machines.

Parameters:
event (object) - the event to be deleted

Returns:
nothing

Example:

```
myEvent call IGN_fnc_deleteEvent;
[myEvent] call IGN_fnc_deleteEvent;
```

IGN_fnc_raiseEvent

Raises an existing event, calling all handlers in sequence with specified arguments
When an event object is raised, it will set its namespace attribute "raised" to true, until all handlers calls are finished
When all handlers have received the event, the attribute "raised" will be set to false.
A broadcasting event can be raised from any machine.
When a broadcasting event is raised from a machine, any handlers that are not owned by the current machine will be remote executed on their respective machines.

Parameters:
event (object) - the event to be raised
arguments (anything) (optional) - the arguments to be passed to each handler
default value - [] (no arguments will be passed)

Returns:
nothing

Example

```
[myEvent] call IGN_fnc_raiseEvent;  // raises myEvent, no arguments are passed to handlers

myEvent call IGN_fnc_raiseEvent;    // same as above, but without array brackets

[myEvent, [myString, myTank]] call IGN_fnc_raiseEvent;
// raises myEvent, passes myString as (_this select 0), myTank as (_this select 1) to all handlers

[myEvent, myTank] call IGN_fnc_raiseEvent;
// raises myEvent, passes myTank as _this to all handlers
```

ASYNCHRONOUS/NON-SCHEDULED/THREADED calls to an event

This is particularly useful if you have spawn'ed threads that may raise the event and want to synchronize calls

Example of Unsafe non-scheduled calls to an event:

```
// thread 1 only tracks player death
_thread_1 = spawn {
	waitUntil {!alive player};
	[eventUnitDead, player] call IGN_fnc_raiseEvent;
	// this can be called at the same time as the other one!
};

// thread 2 tracks any dead in enemyUnits
_thread_2 = spawn {
	private ["_grpAliveCount"];
	_grpAliveCount = count enemyUnits;
	while {_grpAliveCount > 0} do
	{
		{
			if (!alive _x) then
			{
```

```
                        _grpAliveCount = _grpAliveCount - 1;
                        [eventUnitDead, _x] call IGN_fnc_raiseEvent;
                        // this can be called at the same time as the other one!
                    };
                } foreach enemyUnits;
        };
};

// Potential for thread 1 and thread 2 to raise eventUnitDead at the same time! not safe!
```

Example of safer non-scheduled calls to an event

```
// thread 1 only tracks player death
_thread_1 = spawn {
        waitUntil {!alive player};
        waitUntil {!(eventUnitDead getVariable "raised")}; // block thread if event is busy
        [eventUnitDead, player] call IGN_fnc_raiseEvent;
};

// thread 2 tracks any dead in enemyUnits
_thread_2 = spawn {
        private ["_grpAliveCount"];
        _grpAliveCount = count enemyUnits;
        while {_grpAliveCount > 0} do
        {
                {
                        if (!alive _x) then
                        {
                                _grpAliveCount = _grpAliveCount – 1;
                                // block thread if event is busy
                                waitUntil {!(eventUnitDead getVariable "raised")};
                                [eventUnitDead, _x] call IGN_fnc_raiseEvent;
                        };
                } foreach enemyUnits;
        };
};
```

IGN_fnc_addEventHandler

Adds a handler to the specified event, handlerID is returned.
Appends the clientID of the machine that executed this command to the handlers queue. The client that adds a handler to an event becomes the owner of that event.
When this call is done on a broadcasting event, the handler is added to the event object, and the event object broadcasts it's changes to all clients.

Parameters:
event (object) - the event
handle (code) - event handler code

Returns:
handlerID (index of handler in this event)

Example:

```
handlerID = [myEvent, {hint format ["%1", _this];}] call IGN_fnc_addEventHandler;
// everytime myEvent is raised, a hint will display the argument passed to the handler (assuming myEvent sends 1 argument or array)

// Assuming myEvent was raised as such : [myEvent, ["weapon_class_tank_cannon", someTank]] call fnc_RaiseEvent;

handlerID = [myEvent,
{
	private ["_weaponName", "_tank"];
	_weaponName = _this select 0;		// raise of myEvent sends weapon string as arg 0
	_tank = _this select 1;			// tank object is sent as arg 1
	//...
}] call IGN_fnc_addEventHandler;
```

IGN_fnc_deleteEventHandler

Removes a handler from the specified event.
On a broadcasting event, the handler is removed and the change is broadcasted to all machines.

Parameters:
event (object) - the event
handleID (number) - event handler id

Returns:
nothing

Example:

```
[myEvent, handlerID] call IGN_fnc_deleteEventHandler;    // removes handlerID from myEvent
```

IGN_fnc_getEventHandlers

returns the array of all event handlers registered to the current event, including ones owned by different clients.

Parameters:
event (object) - the event

Returns:
array of code

Example:

```
allHandlers = myEvent call IGN_fnc_getEventHandlers;
```

IGN_fnc_getEventHandler

returns the specified handler at the specified index

Parameters:
event (object) - the event
id (number) - index of handler

Returns:
code

Example:

```
handler = [myEvent, 0] call IGN_fnc_getEventHandler;      // returns first registered handler
```

IGN_fnc_getEventHandlersClient

returns all event handlers that are owned by the provided clientID. If the client cannot be found, an empty array is returned.

Parameters:
event (object) - the event
clientID (number) - the clientID that has handlers registered to an event

Returns:
size (number)

Example:

```
numHandles = myEvent call IGN_fnc_getEventHandlerCount;
```

IGN_fnc_createFunctor

Creates a new functor object
arguments passed cannot be temporary values - they must be references to existing objects/tasks/groups/etc...

Parameters:
function (code) - the code the functor will call
arguments (*anything) - the arguments the functor will pass to the function (Must be References!!)

Returns:
Functor object, with these attributes
"**IGN_FUNCTOR_TYPE**" - type identifier
"**function**" - code
"**arguments**" - array of pointers to original arguments (if any)

Example:

```
myFunctor = [
{
	private ["_unit"];
	_unit = _this;
	_unit setUnitPos "MIDDLE";
}, _myUnit] call IGN_fnc_createFunctor;
// notice that _myUnit is local - this means you can call myFunctor outside of
// the script and it will still reference _myUnit
```

In the above example, myFunctor passes one argument (_myUnit) and will return nothing

Example:

```
myFunctor = [
{
	private ["_units"];
	_units = _this;
	{ _x setUnitPos "MIDDLE"; } foreach _units;
	true;
}, [_myUnit1, myUnit2]] call IGN_fnc_createFunctor;

// This time, myFunctor passes two arguments (local _myUnit and global myUnit2) and will return
// true upon completion
```

IGN_fnc_deleteFunctor

Deletes an existing functor, as well as all of its pointers

Parameters:
functor (object) - the functor to be deleted

Returns:
nothing

Example:

```
myFunctor call IGN_fnc_deleteFunctor;
[myFunctor] call IGN_fnc_deleteFunctor;
```

IGN_fnc_callFunctor

Calls the functors code, passing in the stored arguments
Arguments are dereferenced upon call

Parameters:
functor (object) - the Functor to call

Returns:
anything (function) - If the specified code returns a value(s), this will return that value(s)

Example:

```
myResult = myFunctor call IGN_fnc_callFunctor;
[myResult] = myFunctor call IGN_fnc_callFunctor;
```