

# New EventSystem Engine Documentation

## (By: RazoRapid)

EventSystem Engine has been introduced since **Release Candidate 2.5 Beta** version of the patch.

You can use it to bind certain events with your scripts, which will be executed when such events will take place.

In short, EventSystem Engine allows scripters to have scripted events.

Right now, EventSystem Engine supports 6 events:

- connected
- disconnected
- spawn
- damage
- kill
- keypress
- intermission
- servercommand

Scripter can write a script that will get executed by EventSystem Engine when one of them occurs.

Such a script is called **event callback handler** and it have to be registered in the system before it can be used by it.

To register event callback handlers, EventSystem Engine provides scripters with two commands:

- `registerev` - registers event callback handler
- `unregisterev` - unregisters event callback handler

To see their full documentation please see **New Script Commands documentation**

When scripter registers event callback handler it will get bound with certain event types listed earlier.

EventSystem Engine will then execute his script (event callback handler) whenever events of given type will occur.

Now, we will describe each event and show an example of usage.

---

EVENT: connected

*Code:*

```
connected ( Entity entity )
```

DESCRIPTION:

*Code:*

```
Event that is generated when player has entered the server.
```

REGISTERING EXAMPLE:

*Code:*

```
local.result = registerev "connected" global/example.scr::connected  
  
or  
  
local.result = registerev "connected" global/connectedhandler.scr
```

EVENT CALLBACK HANDLER:

*Code:*

```
connected local.player:  
    // your code here  
end  
  
or  
  
main local.player:  
    // your code here  
end
```

VARIABLES:

*Code:*

```
local.player - player that has connected to server
```

---

EVENT: disconnected

*Code:*

```
disconnected ( Entity player )
```

DESCRIPTION:

*Code:*

```
Event that is generated when player has disconnected from the server.  
(it's executed just right before real disconnection)
```

REGISTERING EXAMPLE:

*Code:*

```
local.result = registerev "disconnected" global/example.scr::disconnected  
  
or  
  
local.result = registerev "disconnected" global/disconnectedhandler.scr
```

## EVENT CALLBACK HANDLER:

### Code:

```
disconnected local.player:
    // your code here
end

or

main local.player:
    // your code here
end
```

## VARIABLES:

### Code:

```
local.player - player that has disconnected from server.
```

---

## EVENT: spawn

### Code:

```
spawn ( Entity player )
```

## DESCRIPTION:

### Code:

```
Event that is generated when player has spawned for the first time after entering the server, and each time he respawns. (It won't be generated for the first time when player is already spawned and map restart will occur. To force this event to be called after map restart, you have to move player to spectator before map restart.)
```

## REGISTERING EXAMPLE:

### Code:

```
local.result = registerev "spawn" global/example.scr::spawn

or

local.result = registerev "spawn" global/spawnhandler.scr
```

## EVENT CALLBACK HANDLER:

### Code:

```
spawn local.player:
    // your code here
end

or

main local.player:
    // your code here
end
```

## VARIABLES:

### Code:

```
local.player - player that has spawned or respawned.
```

---

## EVENT: damage

### Code:

```
damage ( Entity target, Entity inflictor, Float damage, Vector position,  
Vector direction, Vector normal, Integer knockback, Integer damageflags,  
Integer meansofdeath, Integer location, Entity entity)
```

## DESCRIPTION:

### Code:

```
Event that is generated when entity (not only player) gets damaged.
```

## REGISTERING EXAMPLE:

### Code:

```
local.result = registerev "damage" global/example.scr::damage  
  
or  
  
local.result = registerev "damage" global/damagehandler.scr
```

## EVENT CALLBACK HANDLER:

### Code:

```
damage local.target local.inflictor local.damage local.position  
local.direction local.normal local.knockback local.damageflags  
local.meansofdeath local.location local.entity:  
    // your code here  
end  
  
or  
  
main local.target local.inflictor local.damage local.position  
local.direction local.normal local.knockback local.damageflags  
local.meansofdeath local.location local.entity:  
    // your code here  
end
```

## VARIABLES:

### Code:

```
local.target - target entity isn't always player or actor entity. It can  
be a weapon entity or world entity  
local.inflictor - inflictor entity, entity that deals damage  
local.damage - float damage, damage amount  
local.position - vector position  
local.direction - vector direction  
local.normal - vector normal  
local.knockback - int knockback value  
local.damageflags - int damageflags  
local.meansofdeath - int meansofdeath  
local.location - int location id  
local.entity - entity that get's damage, often a player but can be any  
oder damageable entity
```

---

## EVENT: kill

### Code:

```
kill ( Entity attacker, Float damage, Entity inflictor, Vector position,  
Vector direction, Vector normal, Integer knockback, Integer damageflags,  
Integer meansofdeath, Integer location, Entity player)
```

## DESCRIPTION:

### Code:

```
Event that is generated when player gets killed or when player kills  
somebody.
```

## REGISTERING EXAMPLE:

### Code:

```
local.result = registerev "kill" global/example.scr::kill  
  
or  
  
local.result = registerev "kill" global/killhandler.scr
```

## EVENT CALLBACK HANDLER:

### Code:

```
kill local.attacker local.damage local.inflictor local.position  
local.direction local.normal local.knockback local.damageflags  
local.meansofdeath local.location local.player:  
    // your code here  
end  
  
or  
  
main local.attacker local.damage local.inflictor local.position  
local.direction local.normal local.knockback local.damageflags  
local.meansofdeath local.location local.player:  
    // your code here  
end
```

## VARIABLES:

### Code:

```
local.attacker - attacker entity (player) that killed  
local.damage - float damage, damage amount  
local.inflictor - inflictor entity, in most cases it isn't a player  
entity, it can be a weapon entity or world entity  
local.position - vector position  
local.direction - vector direction  
local.normal - vector normal  
local.knockback - int knockback value  
local.damageflags - int damageflags  
local.meansofdeath - int meansofdeath  
local.location - int location id  
local.player - player entity that got killed
```

---

EVENT: keypress

*Code:*

```
keypress ( Entity player, Integer keynum )
```

DESCRIPTION:

*Code:*

```
Event that is generated when player has sent special key press command to server.
```

REGISTERING EXAMPLE:

*Code:*

```
local.result = registerev "keypress" global/example.scr::keypress  
  
or  
  
local.result = registerev "keypress" global/keypresshandler.scr
```

EVENT CALLBACK HANDLER:

*Code:*

```
keypress local.player local.keynum:  
    // your code here  
end  
  
or  
  
main local.player local.keynum:  
    // your code here  
end
```

VARIABLES:

*Code:*

```
local.player - player that has sent special keypress command to server  
local.keynum - KeyID
```

To use this special event, scripter needs to bind certain player keys with command:

*Code:*

```
keyp #id
```

where #id is a number.

Example:

*Code:*

```
bind NumPad1 "keyp 1"
```

When error occurs (for example player instead of number has sent command like: **keyp "randomtext"**) a default KeyID will be returned, which is 0.

---

EVENT: servercommand

*Code:*

```
servercommand ( Entity player, String command, String args )
```

DESCRIPTION:

*Code:*

```
Event that is generated when player has sent special server command to server.
```

REGISTERING EXAMPLE:

*Code:*

```
local.result = registerev "servercommand"  
global/example.scr::servercommand  
  
or  
  
local.result = registerev "servercommand" global/servercommandhandler.scr
```

EVENT CALLBACK HANDLER:

*Code:*

```
servercommand local.player local.command local.args:  
    // your code here  
end  
  
or  
  
main local.player local.command local.args:  
    // your code here  
end
```

VARIABLES:

*Code:*

```
local.player - player that has sent special server command to server  
local.command - command  
local.args - command arguments as single not splitted string
```

To use this special event, scripter needs to bind certain player keys with command:

*Code:*

```
scmd command arg1 arg2 arg3 ...
```

where *command* is a command, and *arg1...* are command arguments.

Example:

*Code:*

```
bind NumPad1 "scmd getplayerpos UnnamedSoldier"
```

---

EVENT: intermission

*Code:*

```
intermission ( Integer type )
```

DESCRIPTION:

*Code:*

```
Event that is generated when server enters intermission state, which  
happens during map changes, map restarts and player intermission screen.
```

REGISTERING EXAMPLE:

*Code:*

```
local.result = registerev "intermission" global/example.scr::intermission  
  
or  
  
local.result = registerev "intermission" global/intermissionhandler.scr
```

EVENT CALLBACK HANDLER:

*Code:*

```
intermission local.type:  
    // your code here  
end  
  
or  
  
main local.type:  
    // your code here  
end
```

VARIABLES:

*Code:*

```
local.type - type of server intermission  
  
0 = Player intermission screen  
1 = Map change (happens after using commands: map, gamemap , but also  
right after player intermission screen)  
2 = Map restart (happens after restart command)
```

---



## MISC:

Kill and damage locations:

*Code:*

```
-1 General
0 Pelvis
1 Lower Torso
2 Mid Torso
3 Upper Torso
4 Neck
5 Head
6 RUpperArm
7 RForearm
8 RHand
9 LUpperArm
10 LForearm
11 LHand
12 RThigh
13 RCalf
14 RFoot
15 LThigh
16 LCalf
17 LFoot
```

---

## Important:

EventSystem Engine allows only 1 to 1 binds, which means that you can't register one event type with more than one event callback handler.

When scripter registers event callback handler, it won't be overwritten by next `registerev` commands, which means that EventSystem Engine will execute only those event callback handlers that were registered before any other callback handlers (in short: only firstly registered callback handlers).

After map change, all events will be unregistered, so there's a need to register them again each map change, which shouldn't be a problem.

**Future EventSystem Engine may get improved or its architecture may get changed a bit.**

*RC2.5 Gamma Release introduces few changes:*

- *registerev and unregisterev calling conventions, now they return a value*
- *new event callback handler - intermission*

*Read above post to follow them.*