



# Entity Events

A way to hook game logic to the physical world.

## 1 | Setting Up Events

While it is possible to scan over the results of the engine every frame and collect information, it can be convenient to get notifications of specific occurrences involving specific entities. BEPUphysics provides for this using a variety of event types.

Every entity has its own `EntityEventManager`, accessible in the entity's `EventManager` property. To set up an event:

1. Create a method with a signature matching one of the entity event types (found in `BEPUpHysics.Events`)
2. Add it to the corresponding event in the `entity.EventManager`.

### 1.A | Event Types

Each of the following types can be found in an entity's `EventManager`. Methods matching the event handler's required signature can be used as event handlers.

- **CollisionPairCreated:** Triggered when two collidable objects' bounding boxes begin to overlap.
- **CollisionPairRemoved:** Triggered when two collidable objects' bounding boxes cease to overlap.
- **ContactCreated:** Triggered when a `CollisionPair`'s contact list is updated with an additional contact.
- **InitialCollisionDetected:** Triggered when a `CollisionPair`'s contact list is updated with an additional contact when there were zero contacts previously.
- **ContactRemoved:** Triggered when a `CollisionPair`'s contact list is updated by the removal of a contact point.
- **CollisionEnded:** Triggered when a `CollisionPair`'s contact list is updated by the removal of a contact point and there are no more contact points in the `CollisionPair`'s contact list.
- **CollisionPairUpdated:** Triggered when a `CollisionPair`'s contact list update method runs.
- **CollisionPairTouched:** Triggered when a `CollisionPair`'s contact list update method runs and the `CollisionPair`'s contact list is not empty.

- **EntityUpdated:** Triggered when an Entity's update method is called.

## 2 | Immediate versus Deferred Events

---

All of the events listed above are called at the end of the update in which the event occurred. They are known as “deferred” events for this reason. Their end-of-update positioning allows them to change settings of a wide scope without interfering with the engine's execution. They execute sequentially, preventing a situation where a deferred event handler could interfere with other concurrently executing deferred event handlers.

For every event listed above, there also exists another event with a present-tense name (“ing” instead of “ed”). These 'immediate' events are called directly by the actions that trigger the events in the first place. Essentially, they allow arbitrary code to be inserted directly into the execution of the engine.

There are safety considerations to take into account with both immediate and deferred events. These are addressed in section 3.

### 2.A | Immediate Events in the Pipeline

---

Since immediate events can be used to modify information while the engine is executing, it helps to know exactly where they trigger from in the process. In the following list, “broad phase” refers to the process of collecting of collision pairs between potentially colliding objects. Its followup, the “narrow phase,” is implemented in the CollisionPair's UpdateContactManifold method which tests and collects a list of contact points between objects of a collision pair.

- **CreatingCollisionPair:** Triggered when the broad phase adds a new CollisionPair to the space due to overlapping bounding boxes. Initialization is performed before the event is triggered so the event handler can override and change information in the CollisionPair before it is used in simulation.
- **RemovingCollisionPair:** Triggered when two collidable objects' bounding boxes cease to overlap as determined by the broad phase.
- **CreatingContact:** Triggered when a CollisionPair's contact list is updated with an additional contact during the CollisionPair's UpdateContactManifold method. Both the CollisionPair and Contact settings can be changed within this event.
- **DetectingInitialCollision:** Triggered when a CollisionPair's contact list is updated with an additional contact during the CollisionPair's UpdateContactManifold method when there were zero contacts previously. Both the CollisionPair and Contact settings can be changed within this event.
- **RemovingContact:** Triggered when a CollisionPair's contact list is updated by the removal of a contact point in the CollisionPair's UpdateContactManifold method. The CollisionPair's settings can be changed in this event.

- **CollisionEnding:** Triggered when a CollisionPair's contact list is updated by the removal of a contact point in the CollisionPair's UpdateContactManifold method and there are no more contact points in the CollisionPair's contact list. The CollisionPair's settings can be changed in this event.
- **CollisionPairUpdating:** Triggered at the end of CollisionPair's UpdateContactManifold method. The CollisionPair's settings and the settings of any Contacts within the CollisionPair can be changed from this method.
- **CollisionPairTouching:** Triggered at the end of CollisionPair's updateContactManifold method when the CollisionPair's contact list is not empty. The CollisionPair's settings and the settings of any Contacts within the CollisionPair can be changed from this method.
- **EntityUpdating:** Triggered when an Entity's update method is called during the position update phase. The entity's data can be read, but other operations must be done carefully.

## 3 | Safe Operations Inside Event Handlers

---

Depending on the context from which an event handler is called, some operations are unsafe. Unsafe operations are those which either negatively interfere with the functioning of the engine in some way or can cause unintended behavior and errors.

### 3.A | Deferred Event Safety

---

For all event types with past-tense names, the events handlers are called at the end of a space update. In general, this is a fairly safe operation. Their deferred nature prevents them from directly interfering with the engine's execution.

However, it is still possible for some events to cause an issue. In particular, be careful when performing any action which could trigger other entity events. It is possible to create a feedback loop which continually triggers events. This would not create an application-halting infinite loop since the deferred event handler's spawned events defer to the next frame, but it can lead to strange results and bad performance.

If a deferred event handler's body triggers the same type of event as the event handler itself on the same entity, the engine will throw a protective exception. This prevents a possible feedback loop from forming and offers notification that something strange is happening. For example, in the following situation, the exception would be thrown:

1. Entity A has a deferred event handler which triggers when a collision pair is created. When triggered, another entity is spawned at the position of Entity A.
2. Entity A collides with some other entity B in the space, enqueueing the deferred collision pair event to entity A's deferred event queue.
3. Entity A handles the event and creates a new entity C on top of itself.

4. When the entity C is inserted into the space, a new collision pair is immediately formed and the deferred collision pair event is enqueued to entity A's deferred event queue.
5. Entity A completes its event handler and finds that its deferred event collection has been improperly modified by the previous step.

CollisionPair and Contact object references passed into a deferred event handler are not always guaranteed to be valid, as CollisionPairs and Contacts can quickly return to the resource pool. Be careful when collecting information from these object instances. References to these objects should not be kept outside of the event handler.

### 3.B | Immediate Event Safety

---

Immediate events are called from within the execution of the engine and the scope of safe operations is much smaller than the deferred versions. However, due to their in-engine execution, they can intercept and modify interactions. In general, operations should be performed only on the CollisionPair or Contact passed into the event handler.

If the engine is utilizing multiple threads, the event handlers will be called from the context of worker threads. Any operations performed outside of the CollisionPair or Contact passed into the event handler must be handled very carefully. Even reading data outside of the CollisionPair or Contact may be unreliable depending on the event.

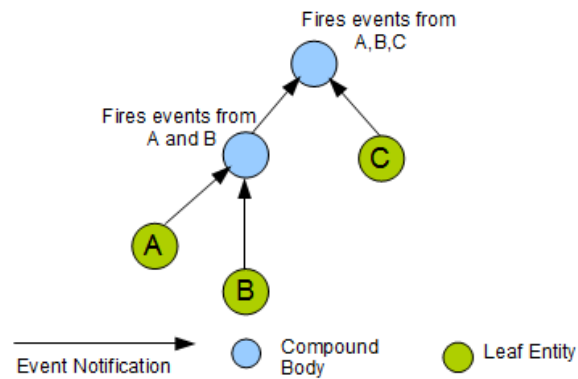
CollisionPair and Contact object references passed into an immediate event handler are guaranteed to be valid for the duration of the event handler, but should not be kept outside of the event handler.

## 4 | Events with Compound Bodies

---

Every CompoundBody has a tree of entities. Compound bodies can have entity children which are also compound bodies, forming a multi-level tree. However, a CompoundBody has no physical geometry of its own; its non-compound children are its geometry. Therefore, all collision-related events originate in a non-compound body child in the tree.

In a CompoundBody, it may be inconvenient to attach an event handler to every non-compound child entity. To avoid having to do this, compound bodies collect the events of their children. Attaching an event to a CompoundBody will trigger on the events of its children.



Any CompoundBody in the tree above a event-causing child will receive events from that child. This allows the root body to fire all events and the sub-compound bodies to fire events only from particular entities in the tree.

Any entity that is a child of a CompoundBody will not trigger an EntityUpdated event, since only the root CompoundBody has its update method called.