Physical entities can be created via calls to the `CreatePhysicalEntity` method of the physical world. It can create entities of the following types:

- **PE_STATIC**: Immovable entity. It can still be moved manually by setting position from outside but in order to ensure proper interactions with simulated objects, it is better to use PE_RIGID entity with infinite mass.
- **PE_RIGID**: A single rigid body. Can have infinite mass (specified by setting mass to 0) in which case it will not be simulated but will interact properly with other simulated objects;
- **PE_ARTICULATED**: An articulated structure, consisting of several rigid bodies connected with joints (most commonly used for ragdolls). It is also possible to manually connect several PE_RIGID entities with joints but in this case they will not know that they comprise a single object, and thus some useful optimizations will not be used.
- **PE_WHEELEDVEHICLE**: A wheeled vehicle. Internally it is built on top of a rigid body, with an extra vehicle-specific functionality (wheels, suspensions, engine, brakes). PE_RIGID, PE_ARTICULATED and PE_WHEELEDVEHICLE are so-called 'purely physical' entities that comprise the core of the simulation engine. The other entities are processed independently;
- **PE_LIVING**: A special entity type to represent player characters that can move through the physical world and interact with it.
- **PE_WALKINGRIGID**: A "modern" replacement for PE_LIVING. It can replicate and extend PE_LIVING's functionality, but unlike PE_LIVING it's derived from PE_RIGID, so it can interact with other physicalized objects better.
- **PE_PARTICLE**: A simple entity that represents a small lightweight rigid body. It is simulated as a point with some thickness and supports flying, sliding and rolling modes. Recommended usage: rockets, grenades and small debris.
- **PE_ROPE**: A rope object. It can either hang freely or connect two purely physical entities.
- **PE_SOFT**: A system of non-rigidly connected vertices that can interact with the environment. It has two simulation modes - constraint-based and spring-based. The former is used for cloth and similar objects, and the latter for volumetric jelly-like substances.

Entities can be created in either permanent or on-demand mode, which is specified by the parameter *lifeTime* (0 for permanent entities). In on-demand mode the placeholders of entities should be created first via `CreatePhysicalPlaceholder` and then the physics will call the outer system back to create the full entity whenever some interaction is required in the bounding box of the placeholder.

If a non-permanent entity is not involved in any interactions (which also includes interactions with rays shot from outside) for the specified lifetime, it will be destroyed, with the placeholder remaining. The advantage is that placeholders require much less memory than full entities (ca. 70 vs. ca. 260 bytes). It is even possible for an outer system to support hierarchical placeholders, i.e. meta-placeholders that create other placeholders upon request.

Additionally, there is a sector-based on-demand physicalization (currently used for vegetation and optionally for brushes). It is activated after `RegisterBBoxInPODGrid` is called. Entities are created and destroyed on a sector basis. The sector size is specified in `SetupEntityGrid`.

In order to maintain associations with outside engine objects, physical entities store an additional void pointer and two 16-bit integers (*pForeingData*, *iForeignData*, and *iForeignFlags*). The entities themselves never modify these parameters, they are only set from outside. The intended usage is to store a pointer to the outside engine reference entity in *pForeignData* and to store the entity type, if applicable, in *iForeignData*. However, there are no restrictions that enforce this.

All physical entities have unique ids. It is possible to not specify an id during creation and let the physics engine generate one automatically, however, during serialization entities use these ids to save dependency information, so there should be a guarantee that entities will have the same ids when reading the saved state. It is also possible to set a new id later. Currently the physics engine uses a simple array to map ids to entity pointers, so using large id numbers will result in allocation of an equally large array.

`DestroyPhysicalEntity` destroys a physical entity or suspends/restores it (if mode parameter is equal to 0, 1, and 2 correspondingly). Suspending an entity causes it to clear all its connections with other entities (this includes constraints) but without the actual deletion. Restoring this entity afterwards will not automatically restore all lost connections. Deleted entities are not destroyed immediately, they are put into a "recycle bin" instead (some entities that they had one-way connections with might need to remove references to them). Deleted entities with reference count <=0 are deleted from the recycle bin at the end of each TimeStep (this can also be forced by calling `PurgeDeletedEntities`).

The physical world can have one special static terrain object (set up with `SetHeightfieldData`) but it is also possible to create terrain geometry manually and add it to an entity in which case the number of terrains is unlimited.

For each material index, the physical world stores the friction coefficient, a bounciness (restitution) coefficient and some flags. Whenever two surfaces contact, the contact's friction and bounciness are computed as an average of these values of both surfaces. The flags only affect raytracing (see below).

Internally, physical entities are grouped by their simulation class. It is important to know this because some interface functions, such as ray tracing and querying entities in an area, allow filtering entities by this type. The types are:

- 0 (bitmask **ent_static**): Static entities
- 1 (bitmask **ent_sleeping_rigid**): Deactivated, purely physical objects (currently rigid bodies, articulated bodies and wheeled vehicles).
- 2 (bitmask **ent_rigid**): Active, purely physical objects.
- 3 (bitmask **ent_living**): Living entities.
- 4 (bitmask **ent_inpependent**): Physical entities that are simulated independently from the others (e.g. particles, ropes and soft objects). As a rule, entities that have a lower simulation type are not aware of entities with higher simulation types (types 1 and 2 are considered as one for this purpose), so players (type 3) and particles (type 4) check collisions against purely physical entities (types 1 and 2) but purely physical entities do not know anything about them. Similarly, ropes (type 4) can check collisions against players but not vice versa. However, entities of higher types can still affect entities with lower types using impulses and sometimes constraints.
- 6 (bitmask **ent_triggers**): Entities (or placeholders) that are not simulated and used solely to generate events when other entities enter their bounding box.
- 7 (bitmask **ent_deleted**, should not be used directly): Objects in the recycle bin.
  Terrain does not have a special simulation mode (it is considered as static) but can be filtered independently with the **ent_terrain** bitmask. Most entities expect a particular simulation mode (and will automatically be set to the proper value). However, there are exceptions, such as articulated entities which can be simulated in modes 1 and 2 as fully physicalized dead bodies or in mode 4 as skeletons that play impact animations without affecting the environment and being affected by it.