AN2501 – IPhysical Entity Interfaces Explained

BSI Application Note

January 2025

Battlespace Simulations

8305 Catamaran Circle

Lakewood Ranch, FL 34202

If you have questions or comments, please contact us at support@bssim.com.

Overview

Those of you with some familiarity with the BSI API know that a common pattern is the need to cast an `IPhysicalEntity` instance to some other interface type in order to accomplish some tasks. For example, a common requirement would be the need to cast an `IPhysicalEntity` instance to an `IPhysicalEntityController` in order to assign a new heading, speed, or altitude to an entity, for example

```
IPhysicalEntity selectedEntity =
MissionInstance.Mission.Map.SelecteEntity;
IPhysicalEntityController ipec = selectedEntity as
IPhysicalEntityController;
if(ipec != null)
{
    ipec.AdjustAltitude_m(SelecteEntity.AltitudeMSL_m + 10000);
}
```

What are these additional `IPhysicalEntity` interfaces, why do they exist, and how can you use them?

What is an Interface?

A programming interface is like a blueprint or a contract in programming. Imagine you're designing different types of vehicles, like cars, bikes, and airplanes. While these vehicles are all different, they might share common features, like the ability to start, stop, or accelerate.

In the BSI API, an interface defines those shared features (methods, properties, or events) without providing any details about how they work. We expose the underlying property, method, or event by providing an *implementation* of the interface. The interface is our contract with you, the developer, on what we expose. Because it's a contract, we strive not to change it once it's in the wild, as changes to how things are implemented means your code could stop working.

Physical entities are quite complex objects, with different parameters controlling their movement, their camera, their aerodynamics, or their equipment. Rather than hang all of the elements a user might need on one massive object interface, we attempt to organize things by functionality. For example, if you want to control movement (heading, speed, altitude) programmatically, you will find the required properties and methods on the `IPhysicalEntityController` interface. Need to control the entity's sensor/camera? You will find what you need on `IPhysicalEntityCamera`.

For physical entities in MACE, the *implementation* of each of these interfaces happens on the same object type. That's why a cast will work – the same underlying object in the MACE runtime implements both the `IPhysicalEntity` and `IPhysicalEntityController` interfaces. Beginning in MACE 2024R1, however, we've begun exposing interfaces directly via properties on the `IPhysicalEntity` interface itself. Revisiting our previous example, you can now do the following:

```
IPhysicalEntity selectedEntity =
MissionInstance.Mission.Map.SelecteEntity;
selectedEntity.Controller.AdjustAltitude_m(SelecteEntity.AltitudeMSL_m +
10000);
```

The `IPhysicalEntity` Interfaces

Here's a short rundown on the `IPhysicalEntity` interfaces and what they do:

- **`IPhysicalEntity`** – the default interface for interacting w/ entities. Most of the commonly used properties, methods, and events will be on this interface. Note that `IPhysicalEntities` represent platforms, lifeforms, and weapons.
- **`IPhysicalEntityAero`** – for interacting with the aerodynamic model for the entity. It is exposed via `IPhysicalEntity.Aero`.
- **`IPhysicalEntityCamera`** – control over the entity's sensor/camera. It is exposed via `IPhysicalEntity.Camera`.
- **`IPhysicalEntityController`** – control over entity movement, such as heading, speed, and altitude assignments. It is exposed via `IPhysicalEntity.Controller`.
- **`IPhysicalEntityEquipment`** – an interface for gettings specialized lists of the entity's loaded equipment. An example would be the `ShootableEquipment` property, which would return not only weapons, but `IEquipment` instances that could be selected and "fired" from the entity control menu, like "Laser Range Finder". It is exposed via `IPhysicalEntity.Equipment`.
- **`IPhysicalEntityIFF`** – control over the entity's IFF properties, like the Mode3 squawk. It is exposed via `IPhysicalEntity.IFF`.
- **`IPhysicalEntityLighting`** – control over the entity's lighting. It is exposed via `IPhysicalEntity.Lighting`.
- **`IPhysicalEntityPresention`** – control over the entity's appearance bits, or control over the entity's icon on the MACE map.
- **`IPhysicalEntitySGE`** – access to properties otherwise exposed via Signal Generation Engine (SGE) interfaces. For instance, you can access the `BSI.SignalGeneration.Entity` instance via this interface, an entity object type used within the SGE.