Save load system.

Always saves/loads **entire** level.

Saveable object representation and interaction with the system.

**IMySaveable** interface must be implemented by each UObject that is managed by the system. Now **only UObjects** are **supported** as saveable.

**IMySavebleHandle** instances are created for each UObject that is to be managed by the system.

The handles are created using the corresponding save load system interface call **in the constructor** of the owning UObject (as the implementation uses the CreateDefaultSubobject, which must be called in the constructor only). So, we can configure properties of the handle in the editor.

After the handle is created, all object-level actions are performed through this handle, and not using the system interface.

Relationship is one-to-one – for each call a valid saveable handle instance is created always. This allows you to make the calls on the handle always without checking for null on the client’s level – the system will check whether this object is really saveable.

When object handle is destroyed, it notifies the system automatically about its destruction through its native UObject::BeginDestroy overload.

Configuring Saveable object.

Per-object properties are located inside the savable object handle. To make them configurable in the editor, the variable must be marked as with Edit\*.

Implementing IMySaveable.

Each IMySaveable virtual function has default implementation available (Prefixed with Default\_), located inside the MySaveableUtils header.

IMySaveable interface implements its virtual functions with these default implementations by default.

Each saveable UObject MUST implement the GetWorld() appropriately (attention – by default it’s not implemented) because whether the object is in the editor or game world is determined by it;