Structure

TableSetup

An object that represents the arrangement of seats, equipment, and counters around the game table.

visionOS 2.0+

```
struct TableSetup
```

Overview

To create a `TableSetup` object, pass an object that conforms to the `Tabletop` or `EntityTabletop` protocol to the `init(tabletop:)` initializer. For example, implement a `Table` structure that conforms to the `EntityTabletop` protocol and pass an instance of it to the initializer.

```
let table = Table()
root = createRootEntity(table: table.entity)
var setup = TableSetup(tabletop: table)
```

Set the protocol properties, such as `shape`, `entity`, and `id` properties for the `EntityTabletop` protocol, in the initializer.

```
struct Table: EntityTabletop {
    var shape: TabletopShape
    var entity: Entity
    var id: EquipmentIdentifier

    init() {
        self.entity = try! Entity.load(named: "table/table", in: contentBundle)
```

```
        self.shape = .round(entity: entity)
        self.id = .table
    }
}
```

Then add seats, equipment, and counters to the `TableSetup` object.

To represent seats, create structures that conform to a seat protocol. To render seats using RealityKit, conform to the `EntityTableSeat` protocol and use the `add(seat:)` or a similar method to add seats. Otherwise, conform to the `TableSeat` protocol and use the `add(seat:)` or a similar method to add seats.

```
setup.add(seat: Seat(index: 0, position: .init(x: 0, z: -0.5)))
setup.add(seat: Seat(index: 1, position: .init(x: 0, z: +0.5)))
```

To represent equipment, create structures that conform to an equipment protocol. To render equipment using RealityKit, conform to the `EntityEquipment` protocol and use the `add(equipment:)` or a similar method to add equipment. Otherwise, conform to the `Equipment` protocol and use the `add(equipment:)` or a similar method to add equipment.

```
setup.add(equipment: Piece(position: .init(x: 0, z: 0.1)))
setup.add(equipment: Card(index: 0, faceUp: true, position: .init(x: -0.1, z: 0)))
setup.add(equipment: Card(index: 1, faceUp: true, position: .init(x: +0.1, z: 0)))
setup.add(equipment: Die(index: 0, position: .init(x: 0, z: 0.2)))
```

Some equipment can represent a group, such as a player's hand in a card game. To organize equipment hierarchically, set the `parentID` property of the `State` property during gameplay. In your equipment structure implementation, you can override the `layoutChildren(for:visualState:)` method to lay out the containing equipment.

Optionally, add one or more `ScoreCounter` objects to the `TableSetup` object to keep score of the game. Use either the `add(counter:)` or `add(counters:)` method to add score counters.

Finally, create the `TabletopGame` instance from the `TableSetup` object by passing it to the `init(tableSetup:version:)` initializer.

```
game = TabletopGame(tableSetup: setup)
```

Topics

Creating a setup object from a table

```
init(tabletop: some Tabletop)
```

```
init(tabletop: some EntityTabletop)
```

Adding seats to place players

```
func add(seat: some TableSeat)
```

Add the given seat to the table setup.

```
func add(seat: some EntityTableSeat)
```

```
func add(seats: some Sequence)
```

Add the given seats to the table setup.

```
func add(seats: some Sequence)
```

Adding equipment for gameplay

```
func add(equipment: some Equipment)
```

Add the given equipment to the table setup.

```
func add<E>(equipment: E)
```

```
func add(equipment: some EntityEquipment)
```

```
func add<E>(equipment: E)
```

Add the given equipment to the table setup.

```
func add(equipment: some Sequence)
```

Add the given equipment to the table setup.

```
func add<E>(equipment: some Sequence)
```

```
func add(equipment: some Sequence)
```

```
func add<E>(equipment: some Sequence)
```

Add the given equipment to the table setup.

Adding counters to keep score

```
func add(counter: ScoreCounter)
```

```
func add(counters: some Sequence<ScoreCounter>)
```

Registering an action

```
func register<Action>(action: Action.Type)
```

Register a custom action of given type. Each type of custom action needs to be registered before it can be used.

See Also

Essentials

{} Creating tabletop games

Develop a spatial board game where multiple players interact with pieces on a table.

{} Synchronizing group gameplay with TabletopKit

Maintain game state across multiple players in a race to capture all the coins.

`class TabletopGame`

An object that manages the setup and gameplay of a tabletop game.

`protocol Tabletop`

A protocol for the table surface in your game.

`protocol EntityTabletop`

A protocol for the table surface in your game when you render it using RealityKit.

`struct TabletopShape`

An object that represents the physical properties of the table.