Namespace PWS.DungeonBlueprint.Construction

Structs

ConstructionGraph

Main data structure for the Construction phase of the transformation pipeline.
This is the first phase where each room and door have their position assigned.

ConstructionGraph.ReadOnly

Read-only wrapper around a construction graph instance.
This is only a view over the existing memory and not a deep copy.

Struct ConstructionGraph

Namespace: PWS.DungeonBlueprint.Construction

Main data structure for the Construction phase of the transformation pipeline.
This is the first phase where each room and door have their position assigned.

```
[BurstCompile]
public struct ConstructionGraph : IDisposable
```

Implements
IDisposable

Inherited Members
ValueType.Equals(object) , ValueType.GetHashCode() , ValueType.ToString() , object.Equals(object, object) , object.GetType() , object.ReferenceEquals(object, object)

Fields

AdjacencyOccupiedConnectionIdx

Connection index of the node's room data an adjacency occupies in the dungeon.
Required to know which connections are used by doors and which need to spawn block doors.

```
public NativeArray<int> AdjacencyOccupiedConnectionIdx
```

Field Value

NativeArray<int>

Remarks

This object owns this native memory and frees it in Dispose().
This array is in parallel with NodeAdjacencies and AdjacencySlotToEdge meaning the arrays contain information about the same adjacency at a given index.

AdjacencySlotToEdge

Mapping from adjacency to edge.
Every valid adjacency belongs to an edge. Since adjacencies are always bidirectional and thus take up two slots in NodeAdjacencies, an edge combines two adjacency slots into a single edge, helping to reduce data duplication.
Every index in the list is either an index for an edge collection or InvalidEdge.

```
public NativeArray<int> AdjacencySlotToEdge
```

Field Value

NativeArray<int>

Remarks

This object owns this native memory and frees it in Dispose().
This array is in parallel with NodeAdjacencies and AdjacencyOccupiedConnectionIdx meaning the arrays contain information about the same adjacency at a given index.

EdgeDoorDataIdx

Door identifier of each node.
Identifier stems from the DBPRegistry.

```
public NativeArray<int> EdgeDoorDataIdx
```

Field Value

NativeArray<int>

Remarks

This array is in parallel with EdgeNames, NodePositions and NodeRotations meaning the arrays contain information about the same node at a given index.
This object owns this native memory and frees it in Dispose().

See Also
NodeNames, NodePositions, NodeRotations

EdgeDoorDirections

Direction of each edge.
Determines where and with what rotation each door will be spawned in the final dungeon.

```
public NativeArray<GridDirection> EdgeDoorDirections
```

Field Value

NativeArray<GridDirection>

Remarks

This array is in parallel with EdgeNames, EdgeDoorDataIdx and EdgeDoorPositions meaning the arrays contain information about the same edge at a given index.
This object owns this native memory and frees it in Dispose().

See Also
EdgeNames, EdgeDoorDataIdx, EdgeDoorPositions

EdgeDoorPositions

Position of each edge.
This is the global grid position, not a position local to a room.
Determines where each door will be spawned in the final dungeon.

```
public NativeArray<GridPosition> EdgeDoorPositions
```

Field Value

NativeArray<GridPosition>

Remarks

This array is in parallel with EdgeNames, EdgeDoorDataIdx and EdgeDoorDirections meaning the arrays contain information about the same edge at a given index.
This object owns this native memory and frees it in Dispose().

See Also
EdgeNames, EdgeDoorDataIdx, EdgeDoorDirections

EdgeNames

Name of each edge.
The name is carried all the way into the spawned dungeon and will then be queryable with GetDoorByName(FixedString64Bytes, int).

```
public NativeArray<FixedString64Bytes> EdgeNames
```

Field Value

NativeArray<FixedString64Bytes>

Remarks

This array is in parallel with EdgeDoorDataIdx, EdgeDoorPositions and EdgeDoorDirections meaning the arrays contain information about the same edge at a given index.
This object owns this native memory and frees it in Dispose().

See Also
EdgeDoorDataIdx, EdgeDoorPositions, EdgeDoorDirections

NodeAdjacencies

Adjacencies between nodes.
Adjacencies of a given node `n`, are in the range `[n*NrAdjacenciesPerNode, n*NrAdjacenciesPerNode+NrAdjacenciesPerNode)`.
Total length of this collection is the `number of nodes * NrAdjacenciesPerNode`.
Every index is either a node or InvalidNode.

```
public NativeArray<int> NodeAdjacencies
```

Field Value

NativeArray<int>

Examples

If `NrAdjacenciesPerNode=3`, the adjacencies for node `0` are at the indices 0, 1 and 2 (the range is `[0*3=0, 0*3+3=3)`).

If `NrAdjacenciesPerNode=2`, the adjacencies for node `4` are at the indices 8 and 9 (the range is `[42=8, 42+2=10)`).

Remarks

Adjacencies are always bidirectional, so if a node has an adjacency to another node, the second node must also have an adjacency to the first node.
This object owns this native memory and frees it in Dispose().
This array is in parallel with AdjacencySlotToEdge and AdjacencyOccupiedConnectionIdx, meaning the arrays contain information about the same adjacency at a given index.

See Also
NrAdjacenciesPerNode

NodeEntryPortalNames

Additional data about some nodes.
This data only exists for nodes that were connected to a portal node with portal type Entrance in the mission graph before the topology conversion.
Portals do not exist past the mission graph stage, but we keep track of the names so we can query them later.

```
public NativeHashMap<int, FixedString64Bytes> NodeEntryPortalNames
```

Field Value

NativeHashMap<int, FixedString64Bytes>

Remarks

This object owns this native memory and frees it in Dispose().

See Also
NodeExitPortalNames

NodeExitPortalNames

Additional data about some nodes.
This data only exists for nodes that were connected to a portal node with portal type Exit in the mission graph before the topology conversion.

Portals do not exist past the mission graph stage, but we keep track of the names so we can query them later.

```
public NativeHashMap<int, FixedString64Bytes> NodeExitPortalNames
```

Field Value

NativeHashMap<int, FixedString64Bytes>

Remarks

This object owns this native memory and frees it in Dispose().

See Also
NodeEntryPortalNames

NodeNames

Name of each node.
The name is carried all the way into the spawned dungeon and will then be queryable with GetRoomByName(FixedString64Bytes, int).

```
public NativeArray<FixedString64Bytes> NodeNames
```

Field Value

NativeArray<FixedString64Bytes>

Remarks

This array is in parallel with NodeRoomDataIdx, NodePositions and NodeRotations meaning the arrays contain information about the same node at a given index.
This object owns this native memory and frees it in Dispose().

See Also
NodeRoomDataIdx, NodePositions, NodeRotations

NodePositions

Grid position of each node.
Determines where each room will be spawned in the final dungeon.

```
public NativeArray<GridPosition> NodePositions
```

Field Value

NativeArray<GridPosition>

Remarks

This array is in parallel with NodeNames, NodeRoomDataIdx and NodeRotations meaning the arrays contain information about the same node at a given index.
This object owns this native memory and frees it in Dispose().

See Also
NodeNames, NodeRoomDataIdx, NodeRotations

NodeRoomDataIdx

Room identifier of each node.
Identifier stems from the DBPRegistry.

```
public NativeArray<int> NodeRoomDataIdx
```

Field Value

NativeArray<int>

Remarks

This array is in parallel with NodeNames, NodePositions and NodeRotations meaning the arrays contain information about the same node at a given index.
This object owns this native memory and frees it in Dispose().

See Also
NodeNames, NodePositions, NodeRotations

NodeRotations

Grid rotation of each node.
Determines how each room will be spawned in the final dungeon.

```
public NativeArray<GridRotation> NodeRotations
```

Field Value

NativeArray<GridRotation>

Remarks

This array is in parallel with NodeNames, NodeRoomDataIdx and NodePositions meaning the arrays contain information about the same node at a given index.
This object owns this native memory and frees it in Dispose().

See Also
NodeNames, NodeRoomDataIdx, NodePositions

NrAdjacenciesPerNode

Maximum number of adjacencies any node in the graph has.
Used to determine how many slots are reserved for each node in the adjacency collection.

```
public int NrAdjacenciesPerNode
```

Field Value

int

Properties

IsCreated

Checks whether this instance currently holds the necessary memory to be valid.
This is a necessary condition for a valid instance, but not a sufficient one.
This means that if this is false, it's definitely not a valid instance to run logic on, but if this is true, that is not a guarantee that it is a valid instance.

```
public readonly bool IsCreated { get; }
```

Property Value

bool

Methods

AsReadOnly()

Creates a new read-only wrapper and returns it.

```
public ConstructionGraph.ReadOnly AsReadOnly()
```

Returns

ConstructionGraph.ReadOnly

Read-only view of this instance.

See Also
ConstructionGraph.ReadOnly

CreateAlloc(int, int, int, out ConstructionGraph)

Creates a new construction graph instance and allocates all native collections for the given number of nodes, adjacencies and edges.

```
[BurstCompile]
public static void CreateAlloc(int nrNodes, int nrAdjacenciesPerNode, int nrEdges,
out ConstructionGraph constructionGraph)
```

Parameters

`nrNodes` int

Number of nodes the construction graph will have.

`nrAdjacenciesPerNode` int

Number of adjacencies per node the construction graph will have.

`nrEdges` int

Number of edges the construction graph will have.

`constructionGraph` ConstructionGraph

Output parameter that will contain the created ConstructionGraph instance.

Remarks

This allocates native memory with Unity.Collections.Allocator.Persistent. You must call Dispose() to free it.

Dispose()

Frees native memory. You must not continue using this instance after calling this.

```
public void Dispose()
```

Struct ConstructionGraph.ReadOnly

Namespace: PWS.DungeonBlueprint.Construction

Read-only wrapper around a construction graph instance.
This is only a view over the existing memory and not a deep copy.

```
public readonly struct ConstructionGraph.ReadOnly
```

Inherited Members
ValueType.Equals(object) , ValueType.GetHashCode() , ValueType.ToString() , object.Equals(object, object) , object.GetType() , object.ReferenceEquals(object, object)

Constructors

ReadOnly(ref ConstructionGraph)

Creates a new read-only wrapper instance around the given construction graph.

```
public ReadOnly(ref ConstructionGraph original)
```

Parameters

`original` ConstructionGraph

Construction graph to wrap.

Properties

AdjacencyOccupiedConnectionIdx

Read-only view on AdjacencyOccupiedConnectionIdx.

```
public NativeArray<int>.ReadOnly AdjacencyOccupiedConnectionIdx { get; }
```

Property Value

NativeArray<int>.ReadOnly

AdjacencySlotToEdge

Read-only view on AdjacencySlotToEdge.

```
public NativeArray<int>.ReadOnly AdjacencySlotToEdge { get; }
```

Property Value

NativeArray<int>.ReadOnly

EdgeDoorDataIdx

Read-only view on EdgeDoorDataIdx.

```
public NativeArray<int>.ReadOnly EdgeDoorDataIdx { get; }
```

Property Value

NativeArray<int>.ReadOnly

EdgeDoorDirections

Read-only view on EdgeDoorDirections.

```
public NativeArray<GridDirection>.ReadOnly EdgeDoorDirections { get; }
```

Property Value

NativeArray<GridDirection>.ReadOnly

EdgeDoorPositions

Read-only view on EdgeDoorPositions.

```
public NativeArray<GridPosition>.ReadOnly EdgeDoorPositions { get; }
```

Property Value

NativeArray<GridPosition>.ReadOnly

EdgeNames

Read-only view on EdgeNames.

```
public NativeArray<FixedString64Bytes>.ReadOnly EdgeNames { get; }
```

Property Value

NativeArray<FixedString64Bytes>.ReadOnly

IsCreated

Mirrors IsCreated of the instance the readonly instance was created with.

```
public bool IsCreated { get; }
```

Property Value

bool

NodeAdjacencies

Read-only view on NodeAdjacencies.

```
public NativeArray<int>.ReadOnly NodeAdjacencies { get; }
```

Property Value

NativeArray<int>.ReadOnly

NodeEntryPortalNames

Read-only view on NodeEntryPortalNames.

```
public NativeHashMap<int, FixedString64Bytes>.ReadOnly NodeEntryPortalNames { get; }
```

Property Value

NativeHashMap<int, FixedString64Bytes>.ReadOnly

NodeExitPortalNames

Read-only view on NodeExitPortalNames.

```
public NativeHashMap<int, FixedString64Bytes>.ReadOnly NodeExitPortalNames { get; }
```

Property Value

NativeHashMap<int, FixedString64Bytes>.ReadOnly

NodeNames

Read-only view on NodeNames.

```
public NativeArray<FixedString64Bytes>.ReadOnly NodeNames { get; }
```

Property Value

NativeArray<FixedString64Bytes>.ReadOnly

NodePositions

Read-only view on NodePositions.

```
public NativeArray<GridPosition>.ReadOnly NodePositions { get; }
```

Property Value

NativeArray<GridPosition>.ReadOnly

NodeRoomDataIdx

Read-only view on NodeRoomDataIdx.

```
public NativeArray<int>.ReadOnly NodeRoomDataIdx { get; }
```

Property Value

NativeArray<int>.ReadOnly

NodeRotations

Read-only view on NodeRotations.

```
public NativeArray<GridRotation>.ReadOnly NodeRotations { get; }
```

Property Value

NativeArray<GridRotation>.ReadOnly

NrAdjacenciesPerNode

Read-only view on NrAdjacenciesPerNode.

```
public int NrAdjacenciesPerNode { get; }
```

Property Value

int

Namespace PWS.DungeonBlueprint.Core

Classes

DBPConstants

A few central constants used across Dungeon Blueprint. Not all constants in the project are collected here, only those that are relevant across large parts of the project and possibly for user code.

DBPPipelineSettingsSO

A scriptable object wrapper for DBPPipelineSettings.
Useful for storing your settings as assets.

DBPRegistry

A central data registry. Dungeon Blueprint's data is registered here to be used across its algorithms.

DBPWorldSettingsSO

A scriptable object wrapper for DBPWorldSettings.
Useful for storing your settings as assets.

Structs

DBPPipelineSettings

A settings object for configuring DBPPipeline.

DBPWorldSettings

A settings object for configuring DBPWorld.

Class DBPConstants

Namespace: PWS.DungeonBlueprint.Core

A few central constants used across Dungeon Blueprint. Not all constants in the project are collected here, only those that are relevant across large parts of the project and possibly for user code.

```
public static class DBPConstants
```

Inheritance

object ← DBPConstants

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Fields

InvalidConnection

Represents an invalid/non-existent connection in the graph.

```
public const int InvalidConnection = -2147483648
```

Field Value

int

InvalidDoor

Represents an invalid/non-existent door in the graph.

```
public const int InvalidDoor = -2147483648
```

Field Value

int

InvalidEdge

Represents an invalid/non-existent edge in the graph.

```
public const int InvalidEdge = -2147483648
```

Field Value

int

InvalidNode

Represents an invalid/non-existent node in the graph.

```
public const int InvalidNode = -2147483648
```

Field Value

int

InvalidRoom

Represents an invalid/non-existent room in the graph.

```
public const int InvalidRoom = -2147483648
```

Field Value

int

Struct DBPPipelineSettings

Namespace: PWS.DungeonBlueprint.Core

A settings object for configuring DBPPipeline.

```
[Serializable]
public struct DBPPipelineSettings
```

Inherited Members
ValueType.Equals(object) , ValueType.GetHashCode() , ValueType.ToString() , object.Equals(object, object) , object.GetType() , object.ReferenceEquals(object, object)

Constructors

DBPPipelineSettings(uint, uint, uint, uint, GridSize, ulong)

Creates a new instance of DBPPipelineSettings.

```
public DBPPipelineSettings(uint maxMissionGraphResolutionTicks, uint
maxTopologyConversionTicks, uint maxConstructionConversionTicks, uint
maxRuntimeConversionTicks, GridSize gridSize, ulong constructionMaxLoopIterations)
```

Parameters

`maxMissionGraphResolutionTicks` uint

See MaxMissionGraphResolutionTicks.

`maxTopologyConversionTicks` uint

See MaxTopologyConversionTicks.

`maxConstructionConversionTicks` uint

See MaxConstructionConversionTicks.

`maxRuntimeConversionTicks` uint

See MaxRuntimeConversionTicks.

`gridSize` GridSize

See GridSize.

`constructionMaxLoopIterations` ulong

See ConstructionMaxLoopIterations.

Properties

ConstructionMaxLoopIterations

The maximum number of loop iterations the construction conversion may take before it aborts and fails.
This is an important mechanism to control the time dungeon generation can take. Some seeds may run multiple orders of magnitudes slower than others. Therefore, it may be faster to abort a slow seed early and rerun the generation with a different seed than waiting for the slow seed to complete.

```
public readonly ulong ConstructionMaxLoopIterations { get; }
```

Property Value

ulong

Remarks

The best value for this is dependent on the power of your target hardware and the complexity of your mission.
The higher this number, the higher your success rate when generating.
The lower this number, the lower the maximum time a single generation may take.

GridSize

The size of the grid.
Each room is placed on the grid so the grid size directly affects the bounds of the dungeon.

```
public readonly GridSize GridSize { get; }
```

Property Value

GridSize

MaxConstructionConversionTicks

Maximum number of ticks construction conversion is allowed to take. On the tick that reaches this tick number construction conversion completes synchronously.

```
public readonly uint MaxConstructionConversionTicks { get; }
```

Property Value

uint

Remarks

Tick rate is determined by user code. See Tick().

MaxMissionGraphResolutionTicks

Maximum number of ticks mission graph resolution is allowed to take. On the tick that reaches this tick number mission graph resolution completes synchronously.

```
public readonly uint MaxMissionGraphResolutionTicks { get; }
```

Property Value

uint

Remarks

Tick rate is determined by user code. See Tick().

MaxRuntimeConversionTicks

Maximum number of ticks runtime conversion is allowed to take. On the tick that reaches this tick number runtime conversion completes synchronously.

```
public readonly uint MaxRuntimeConversionTicks { get; }
```

Property Value

uint

Remarks

Tick rate is determined by user code. See Tick().

MaxTopologyConversionTicks

Maximum number of ticks topology conversion is allowed to take. On the tick that reaches this tick number topology conversion completes synchronously.

```
public readonly uint MaxTopologyConversionTicks { get; }
```

Property Value

uint

Remarks

Tick rate is determined by user code. See Tick().

Methods

CheckForErrors(out string)

Checks the configured settings for errors.

```
public readonly bool CheckForErrors(out string errorMessage)
```

Parameters

`errorMessage` string

A human-readable error message. `null` if return value is true.

Returns

bool

True if the settings are valid, false otherwise.

See Also

Setup(MissionGraph, uint, EntryRoomInfo, DBPPipelineSettings, object)

Class DBPPipelineSettingsSO

Namespace: PWS.DungeonBlueprint.Core

A scriptable object wrapper for DBPPipelineSettings.
Useful for storing your settings as assets.

```
[CreateAssetMenu(fileName = "DBPPipelineSettingsSO", menuName = "Dungeon
Blueprint/Pipeline Settings", order = 1000)]
public class DBPPipelineSettingsSO : ScriptableObject
```

Inheritance

object ← Object ← ScriptableObject ← DBPPipelineSettingsSO

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Properties

Settings

The settings this object wrap.

```
public DBPPipelineSettings Settings { get; }
```

Property Value

DBPPipelineSettings

Class DBPRegistry

Namespace: PWS.DungeonBlueprint.Core

A central data registry. Dungeon Blueprint's data is registered here to be used across its algorithms.

```
public class DBPRegistry
```

Inheritance

object ← DBPRegistry

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Remarks

The registry assigns unique IDs to each data object.
Assigned IDs must stay consistent across your Dungeon Blueprint usage.
Therefore, you should usually only have a single registry object.

Constructors

DBPRegistry()

Creates a new instance of DBPRegistry without any registered objects.

```
public DBPRegistry()
```

Properties

NrRegisteredDoors

Number of doors currently registered with the registry.

```
public int NrRegisteredDoors { get; }
```

Property Value

int

NrRegisteredRooms

Number of rooms currently registered with the registry.

```
public int NrRegisteredRooms { get; }
```

Property Value

int

Methods

GetDoor(int)

Finds a previously registered door data by its assigned ID.

```
public DoorDataSO GetDoor(int id)
```

Parameters

`id` int

ID the door data was registered with. The ID is returned when calling RegisterDoor(DoorDataSO).

Returns

DoorDataSO

Door data the registry has under the given ID.

Exceptions

ArgumentOutOfRangeException

If the registry has no door data collection registered with the given ID.

GetDoorCollection(int)

Finds a previously registered door data collection by its assigned ID.

```
public DoorDataCollectionSO GetDoorCollection(int id)
```

Parameters

`id` int

ID the door data collection was registered with. The ID is returned when calling Register DoorCollection(DoorDataCollectionSO).

Returns

DoorDataCollectionSO

Door data collection the registry has under the given ID.

Exceptions

ArgumentOutOfRangeException

If the registry has no door data collection registered with the given ID.

GetMissionGraphAsset(int)

Finds a previously registered mission graph asset by its assigned ID.

```
public MissionGraphAsset GetMissionGraphAsset(int id)
```

Parameters

`id` int

ID the mission graph asset was registered with. The ID is returned when calling Register MissionGraphAsset(MissionGraphAsset).

Returns

MissionGraphAsset

Mission graph asset the registry has under the given ID.

Remarks

Used for generating and importing mission graph assets. If you only use the node based editor for generating Missions, you do not need this.

Exceptions

ArgumentOutOfRangeException

If the registry has no mission graph asset registered with the given ID.

GetRoom(int)

Finds a previously registered room data by its assigned ID.

```
public RoomDataSO GetRoom(int id)
```

Parameters

`id` int

ID the room data was registered with. The ID is returned when calling Register Room(RoomDataSO).

Returns

RoomDataSO

Room data the registry has under the given ID.

Exceptions

ArgumentOutOfRangeException

If the registry has no room data collection registered with the given ID.

GetRoomCollection(int)

Finds a previously registered room data collection by its assigned ID.

```
public RoomDataCollectionSO GetRoomCollection(int id)
```

Parameters

`id` int

ID the room data collection was registered with. The ID is returned when calling Register RoomCollection(RoomDataCollectionSO).

Returns

RoomDataCollectionSO

Room data collection the registry has under the given ID.

Exceptions

ArgumentOutOfRangeException

If the registry has no room data collection registered with the given ID.

GetUnmanagedDoorCollectionsAlloc()

Turns all currently registered door data collections into their unmanaged counter-part (see UnmanagedDoorDataCollection) and returns them in a newly allocated native array.
This is used to transfer the door data collections over into Burst compatible code.

```
public NativeArray<UnmanagedDoorDataCollection> GetUnmanagedDoorCollectionsAlloc()
```

Returns

NativeArray<UnmanagedDoorDataCollection>

A newly allocated native array containing the unmanaged counter-part to each currently registered door data collection.

Remarks

This allocates O(NrRegisteredDoors) memory with Unity.Collections.Allocator.Persistent. You must call Unity.Collections.NativeArray<T>.Dispose() or Unity.Collections.NativeArray<T>.Dispose(Unity.Jobs.JobHandle) to free it.

GetUnmanagedRoomCollectionsAlloc()

Turns all currently registered room data collections into their unmanaged counter-part (see UnmanagedRoomDataCollection) and returns them in a newly allocated native array. This is used to transfer the room data collections over into Burst compatible code.

```
public NativeArray<UnmanagedRoomDataCollection> GetUnmanagedRoomCollectionsAlloc()
```

Returns

NativeArray<UnmanagedRoomDataCollection>

A newly allocated native array containing the unmanaged counter-part to each currently registered room data collection.

Remarks

This allocates O(NrRegisteredRooms) memory with Unity.Collections.Allocator.Persistent. You must call Unity.Collections.NativeArray<T>.Dispose() or Unity.Collections.NativeArray<T>.Dispose(Unity.Jobs.JobHandle) to free it.

RegisterDoor(DoorDataSO)

Registers a door data with the registry. If you call this with the same door multiple times, the registry does not register the door multiple times.

```
public int RegisterDoor(DoorDataSO door)
```

Parameters

`door` DoorDataSO

Door data to register.

Returns

int

ID the given door data was registered with. If you call this method with the same door multiple times, the return value will always be the same.

Exceptions

ArgumentNullException

If `door` is null.

RegisterDoorCollection(DoorDataCollectionSO)

Registers a door data collection with the registry. If you call this with the same door collection multiple times, the registry does not register the collection multiple times.

```
public int RegisterDoorCollection(DoorDataCollectionSO doorCollection)
```

Parameters

`doorCollection` DoorDataCollectionSO

Door data collection to register.

Returns

int

ID the given door data collection was registered with. If you call this method with the same door collection multiple times, the return value will always be the same.

Remarks

The first time a collection is registered, RegisterDoor(DoorDataSO) is called on every door in the collection.

Exceptions

ArgumentNullException

If `doorCollection` is null.

RegisterMissionGraphAsset(MissionGraphAsset)

Registers a mission graph asset with the registry. If you call this with the same mission graph asset multiple times, the registry does not register the asset multiple times.

```
public int RegisterMissionGraphAsset(MissionGraphAsset missionGraphAsset)
```

Parameters

`missionGraphAsset` MissionGraphAsset

Mission graph asset to register.

Returns

int

ID the given mission graph asset was registered with. If you call this method with the same mission graph asset multiple times, the return value will always be the same.

Remarks

Used for generating and importing mission graph assets. If you only use the node based editor for generating Missions, you do not need this.

Exceptions

ArgumentNullException

If `missionGraphAsset` is null.

RegisterRoom(RoomDataSO)

Registers a room data with the registry. If you call this with the same room multiple times, the registry does not register the room multiple times.

```
public int RegisterRoom(RoomDataSO room)
```

Parameters

`room` RoomDataSO

Room data to register.

Returns

int

ID the given room data was registered with. If you call this method with the same room multiple times, the return value will always be the same.

Exceptions

ArgumentNullException

If `room` is null.

RegisterRoomCollection(RoomDataCollectionSO)

Registers a room data collection with the registry. If you call this with the same room collection multiple times, the registry does not register the collection multiple times.

```
public int RegisterRoomCollection(RoomDataCollectionSO roomCollection)
```

Parameters

`roomCollection` RoomDataCollectionSO

Room data collection to register.

Returns

int

ID the given room data collection was registered with. If you call this method with the same room collection multiple times, the return value will always be the same.

Remarks

The first time a collection is registered, RegisterRoom(RoomDataSO) is called on every room in the collection.

Exceptions

ArgumentNullException

If `roomCollection` is null.

Struct DBPWorldSettings

Namespace: PWS.DungeonBlueprint.Core

A settings object for configuring DBPWorld.

```
[Serializable]
public struct DBPWorldSettings
```

Inherited Members
ValueType.Equals(object) , ValueType.GetHashCode() , ValueType.ToString() , object.Equals(object, object) , object.GetType() , object.ReferenceEquals(object, object)

Constructors

DBPWorldSettings(uint, Vector3, Vector3)

Creates a new instance of DBPWorldSettings.
Used to configure DBPWorld.

```
public DBPWorldSettings(uint maxSpawningTicks, Vector3 dungeonOriginWorldPosition,
Vector3 roomGridToWorld)
```

Parameters

`maxSpawningTicks` uint

See MaxSpawningTicks.

`dungeonOriginWorldPosition` Vector3

See DungeonOriginWorldPosition.

`roomGridToWorld` Vector3

See RoomGridToWorld.

Properties

DungeonOriginWorldPosition

World origin of the dungeon grid when its rooms are spawned.
Pivot of the grid is the front lower left corner (as with rooms).

```
public readonly Vector3 DungeonOriginWorldPosition { get; }
```

Property Value

Vector3

Remarks

If you need the entry room of the dungeon to align with other objects in the world, you must calculate the dungeon origin position using the entry room's grid position and the room to grid world scale.

See Also
Position, RoomGridToWorld

MaxSpawningTicks

Maximum number of ticks spawning is allowed to take. On the tick that reaches this tick number spawning completes synchronously.

```
public readonly uint MaxSpawningTicks { get; }
```

Property Value

uint

Remarks

Tick rate is determined by user code. See Tick().

RoomGridToWorld

World scale of rooms. A 1x1x1 room is expected to have this size in world space. If you pass this value to the world, but your rooms have a different size, the spawned dungeon may be invalid.

```
public readonly Vector3 RoomGridToWorld { get; }
```

Property Value

Vector3

Methods

CreateWithEntryRoomAtWorldPosition(uint, Vector3, GridPosition, Vector3)

Creates a new instance of DBPWorldSettings which will place the entry room at a given world position.

```
public static DBPWorldSettings CreateWithEntryRoomAtWorldPosition(uint
maxSpawningTicks, Vector3 roomGridToWorld, GridPosition entryRoomGridPosition,
Vector3 entryRoomWorldPosition)
```

Parameters

`maxSpawningTicks` uint

See MaxSpawningTicks.

`roomGridToWorld` Vector3

See RoomGridToWorld.

`entryRoomGridPosition` GridPosition

Entry room grid position to align the origin to.

`entryRoomWorldPosition` Vector3

Goal position of the entry room.

Returns

DBPWorldSettings

Newly created DBPWorldSettings with the origin calculated to spawn the entry room at `entryRoomWorldPosition`.

See Also

Setup(ReadOnly, DBPWorldSettings)

Class DBPWorldSettingsSO

Namespace: PWS.DungeonBlueprint.Core

A scriptable object wrapper for DBPWorldSettings.
Useful for storing your settings as assets.

```
[CreateAssetMenu(fileName = "DBPWorldSettingsSO", menuName = "Dungeon
Blueprint/World Settings", order = 1001)]
public class DBPWorldSettingsSO : ScriptableObject
```

Inheritance

object ← Object ← ScriptableObject ← DBPWorldSettingsSO

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Properties

Settings

The settings this object wrap.

```
public DBPWorldSettings Settings { get; }
```

Property Value

DBPWorldSettings

Namespace PWS.DungeonBlueprint.Data

Classes

DoorDataCollectionSO

Collections of DoorDataSO which can be assigned to nodes in the MissionGraph. The generation algorithm will then pick a door data of the assigned collection when placing the door.

DoorDataSO

Data about a door that can be placed in a dungeon. Door data can be collected in a DoorDataCollectionSO and then assigned to a node in the MissionGraph. One of the doors in the assigned collection will then be used for the node during generation.

RoomDataCollectionSO

Collections of RoomDataSO which can be assigned to nodes in the MissionGraph. The generation algorithm will then pick a room data of the assigned collection when placing the room.

RoomDataSO

Data about a room that can be placed in a dungeon. Room data can be collected in a RoomDataCollectionSO and then assigned to a node in the MissionGraph. One of the rooms in the assigned collection will then be used for the node during generation.

Structs

RoomDoorConnection

Data about how a room can connect to doors (and thus other rooms). When defining your room data, you should add one room door connection for every spot where there is space for a door.
Metaphorically, you can consider these to be doorways.

UnmanagedDoorData

Unmanaged counter-part of DoorDataSO. Used for crossing the door data over into Burst compatible code.

UnmanagedDoorDataCollection

The unmanaged counter-part of DoorDataCollectionSO. This is used for crossing the door data collection over into Burst compatible code.

UnmanagedRoomData

Unmanaged counter-part of RoomDataSO. Used for crossing the room data over into Burst compatible code.

UnmanagedRoomDataCollection

Unmanaged counter-part of RoomDataCollectionSO. Used for crossing the room data collection over into Burst compatible code.

UnmanagedRoomDoorConnection

Unmanaged counter-part of RoomDoorConnection. Used for crossing the room door connections over into Burst compatible code.

Class DoorDataCollectionSO

Namespace: PWS.DungeonBlueprint.Data

Collections of DoorDataSO which can be assigned to nodes in the MissionGraph. The generation algorithm will then pick a door data of the assigned collection when placing the door.

```
[CreateAssetMenu(fileName = "New DoorDataCollection", menuName = "Dungeon
Blueprint/Door Data Collection", order = 104)]
public class DoorDataCollectionSO : ScriptableObject
```

Inheritance

object ← Object ← ScriptableObject ← DoorDataCollectionSO

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Properties

Doors

All door data in this collection.

```
public IReadOnlyList<DoorDataSO> Doors { get; }
```

Property Value

IReadOnlyList<DoorDataSO>

See Also

DoorDataSO

Name

Name of the door collection.

```
public string Name { get; }
```

Property Value

string

Remarks

This is used for organizational purposes only. It has no effect on dungeon generation.

See Also

UnmanagedDoorDataCollection
DoorDataSO

Class DoorDataSO

Namespace: PWS.DungeonBlueprint.Data

Data about a door that can be placed in a dungeon. Door data can be collected in a DoorDataCollectionSO and then assigned to a node in the MissionGraph. One of the doors in the assigned collection will then be used for the node during generation.

```
[CreateAssetMenu(fileName = "New DoorDataSO", menuName = "Dungeon Blueprint/Door
Data", order = 103)]
public class DoorDataSO : ScriptableObject
```

Inheritance

object ← Object ← ScriptableObject ← DoorDataSO

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Properties

Name

Name of the door.

```
public string Name { get; }
```

Property Value

string

Remarks

This is used for organizational purposes only. It has no effect on dungeon generation.

Prefab

Prefab that will be spawned when this door is placed in a dungeon.

```
public GameObject Prefab { get; }
```

Property Value

GameObject

Size

Size of this door. Used as a compatibility check with RoomDoorConnection by comparing this to DoorSize.

```
public GridSize Size { get; }
```

Property Value

GridSize

Remarks

The size itself is not used for dungeon generation or spawning, only for compatibility checks.
Therefore, the exact value you use is not important, and it is possible to categorize doors using other factors than their size.

See Also

UnmanagedDoorData
DoorDataCollectionSO

Class RoomDataCollectionSO

Namespace: PWS.DungeonBlueprint.Data

Collections of RoomDataSO which can be assigned to nodes in the MissionGraph. The generation algorithm will then pick a room data of the assigned collection when placing the room.

```
[CreateAssetMenu(fileName = "New RoomDataCollection", menuName = "Dungeon
Blueprint/Room Data Collection", order = 102)]
public class RoomDataCollectionSO : ScriptableObject
```

Inheritance

object ← Object ← ScriptableObject ← RoomDataCollectionSO

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Properties

Name

Name of the room collection.

```
public string Name { get; }
```

Property Value

string

Remarks

This is used for organizational purposes only. It has no effect on dungeon generation.

Rooms

All room data in this collection.

```
public IReadOnlyList<RoomDataSO> Rooms { get; }
```

Property Value

IReadOnlyList<RoomDataSO>

See Also
RoomDataSO

See Also

UnmanagedRoomDataCollection
RoomDataSO

Class RoomDataSO

Namespace: PWS.DungeonBlueprint.Data

Data about a room that can be placed in a dungeon. Room data can be collected in a RoomDataCollectionSO and then assigned to a node in the MissionGraph. One of the rooms in the assigned collection will then be used for the node during generation.

```
[CreateAssetMenu(fileName = "New RoomDataSO", menuName = "Dungeon Blueprint/Room
Data", order = 101)]
public class RoomDataSO : ScriptableObject
```

Inheritance

object ← Object ← ScriptableObject ← RoomDataSO

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Properties

MaximumDoorCount

Number of RoomDoorConnections defined for this room. Metaphorically, how many doorways this room has.

```
public int MaximumDoorCount { get; }
```

Property Value

int

Name

Name of the room.

```
public string Name { get; }
```

Property Value

string

Remarks

This is used for organizational purposes only. It has no effect on dungeon generation.

NrAllowedRotations

Number of allowed rotations for this room. This is either PWS.DungeonBlueprint.Grid.GridRotationExtensions.NrAllRotations or PWS.DungeonBlueprint.Grid.GridRotationExtensions.NrNoneRotations, depending on whether this room can be rotated or not.

```
public int NrAllowedRotations { get; }
```

Property Value

int

Prefab

Prefab that will be spawned when this room is placed in a dungeon.

```
public GameObject Prefab { get; }
```

Property Value

GameObject

Methods

GetGridOffsetForRotation(GridRotation)

Calculates the offset needed for placing this room at a specific position with the given rotation.
This offset is required to know which tiles of the grid the placed room occupies, since this is influenced by the room's rotation.

```
public GridPosition GetGridOffsetForRotation(GridRotation rotation)
```

Parameters

`rotation` GridRotation

Rotation to calculate the offset.

Returns

GridPosition

Offset needed to place this room at a position the given rotation.

Remarks

Pivot of a room is the lower left backward corner of the lower left backward tile. Therefore, there is a rotation offset even for 1x1x1 rooms.
If `rotation` is Cw0, the returned offset is Zero.

Exceptions

InvalidOperationException

If `rotation` is a rotation that is not allowed for this room.

GetRotatedConnection(int, GridRotation)

Gets a specific door connection for a rotated room.

```
public RoomDoorConnection GetRotatedConnection(int idx, GridRotation rotation)
```

Parameters

`idx` int

Door connection index to get. Check MaximumDoorCount to find out how many connections there are.

`rotation` GridRotation

Room's rotation.

Returns

RoomDoorConnection

Door connection specified by `idx` when the room is rotated by `rotation`.

Remarks

The room's rotation changes the door connections because the connection's local position and direction must rotate with the room.

Exceptions

ArgumentOutOfRangeException

If `idx` is out of range.

InvalidOperationException

If `rotation` is a rotation that is not allowed for this room.

See Also
RoomDoorConnection

GetRotatedSize(GridRotation)

Calculates the size of the room when rotated by the given rotation.

```
public GridSize GetRotatedSize(GridRotation rotation)
```

Parameters

`rotation` GridRotation

Rotation to use for calculating the size.

Returns

GridSize

Size of the room when it is rotated by the given rotation.

Remarks

If `rotation` is Cw0 or Cw180, this returns the size you originally set for this room.
If `rotation` is Cw90 or Cw270, this returns the size you originally set for this room with width and depth switched.

See Also

UnmanagedRoomData
RoomDataCollectionSO

Struct RoomDoorConnection

Namespace: PWS.DungeonBlueprint.Data

Data about how a room can connect to doors (and thus other rooms). When defining your room data, you should add one room door connection for every spot where there is space for a door.
Metaphorically, you can consider these to be doorways.

```
[Serializable]
public struct RoomDoorConnection
```

Inherited Members
ValueType.Equals(object) , ValueType.GetHashCode() , ValueType.ToString() , object.Equals(object, object) , object.GetType() , object.ReferenceEquals(object, object)

Remarks

All connections of a room must be on the edges of the room's bounds.

Properties

BlockDoor

Reference to a door data that will be used if this connection is not used but the room is placed.
Metaphorically, you can consider this to be something to plug the hole in the wall an empty doorway would otherwise leave.

```
public readonly DoorDataSO BlockDoor { get; }
```

Property Value

DoorDataSO

Direction

Direction this door points on its grid position. Effectively, which edge of the tile the door would be on.

```
public readonly GridDirection Direction { get; }
```

Property Value

GridDirection

Remarks

Remember door connections must always be on an outside ege of the room.
This means on a 2x1x1 room and a (0, 0, 0) grid position, the direction HorizontalRight would be invalid.

DoorSize

Size of door this connection can fit.
Used as a compatibility check with other connections and DoorDataSO by comparing this to Size.

```
public readonly GridSize DoorSize { get; }
```

Property Value

GridSize

Remarks

The size itself is not used for dungeon generation or spawning, only for compatibility checks.
Therefore, the exact value you use is not important, and it is possible to categorize connections using other factors than their size.

LocalGridPosition

Grid position of this door connection local to the room. Origin (0, 0, 0) is always the lower left tile of the room. Increasing X moves right in the grid, increasing Y moves up and increasing Z moves forward.

```
public readonly GridPosition LocalGridPosition { get; }
```

Property Value

GridPosition

Examples

On a 1x1x1 room, this would always be (0, 0, 0).
On a 2x1x1 room, this may be (0, 0, 0) or (1, 0, 0).

Remarks

Remember door connections must always be on an outside ege of the room.
This means on a 3x1x3 room, a position of (1, 0, 1) would be invalid.

See Also

UnmanagedRoomDoorConnection
RoomDataSO

Struct UnmanagedDoorData

Namespace: PWS.DungeonBlueprint.Data

Unmanaged counter-part of DoorDataSO. Used for crossing the door data over into Burst compatible code.

```
public struct UnmanagedDoorData
```

Inherited Members
ValueType.Equals(object) , ValueType.GetHashCode() , ValueType.ToString() , object.Equals(object, object) , object.GetType() , object.ReferenceEquals(object, object)

Fields

RegistryId

ID the corresponding RoomDataSO is registered with in the DBPRegistry used to create this instance.

```
public int RegistryId
```

Field Value

int

See Also
Create(DoorDataSO, DBPRegistry), DBPRegistry

Size

See Size.

```
public GridSize Size
```

Field Value

GridSize

Struct UnmanagedDoorDataCollection

Namespace: PWS.DungeonBlueprint.Data

The unmanaged counter-part of DoorDataCollectionSO. This is used for crossing the door data collection over into Burst compatible code.

```
public struct UnmanagedDoorDataCollection : IDisposable
```

Implements
IDisposable

Inherited Members
ValueType.Equals(object) , ValueType.GetHashCode() , ValueType.ToString() , object.Equals(object, object) , object.GetType() , object.ReferenceEquals(object, object)

Fields

Doors

See Doors.

```
public UnsafeList<UnmanagedDoorData> Doors
```

Field Value

UnsafeList<UnmanagedDoorData>

Remarks

This list is allocated by CreateAlloc(DoorDataCollectionSO, DBPRegistry) using Unity.Collections.Allocator.Persistent. Call Dispose() to free it.

RegistryId

ID the corresponding DoorDataCollectionSO is registered with in the DBPRegistry used to create this instance.

```
public int RegistryId
```

Field Value

int

See Also
CreateAlloc(DoorDataCollectionSO, DBPRegistry), DBPRegistry

Methods

Dispose()

Frees native memory. You must not continue using this instance after calling this.

```
public void Dispose()
```

Struct UnmanagedRoomData

Namespace: PWS.DungeonBlueprint.Data

Unmanaged counter-part of RoomDataSO. Used for crossing the room data over into Burst compatible code.

```
public struct UnmanagedRoomData : IDisposable
```

Implements
IDisposable

Inherited Members
ValueType.Equals(object) , ValueType.GetHashCode() , ValueType.ToString() , object.Equals(object, object) , object.GetType() , object.ReferenceEquals(object, object)

Fields

DoorConnectionsGrouped

All door connections of this room, cached for all possible rotations and grouped by the direction the connection points to for fast compatibility checking. To access the door connections pointing in a specific direction on a room with a specific rotation, use `DoorConnectionsGrouped[(int)rotation * GridDirectionExtensions.NrHorizontalDirections + (int)direction - (int)GridDirection.HorizontalForward]`. The inner list is then just a simple list containing the door connections.

```
public UnsafeList<UnsafeList<UnmanagedRoomDoorConnection>> DoorConnectionsGrouped
```

Field Value

UnsafeList<UnsafeList<UnmanagedRoomDoorConnection>>

Remarks

This list is allocated by CreateAlloc(RoomDataSO, DBPRegistry) using Unity.Collections. Allocator.Persistent. Call Dispose() to free it. You should not free this list itself, because it

leaves the room data unusable, but does not free the other unmanaged memory allocated. **You will not be able to free all memory manually because some of it isn't accessible. Call Dispose() instead.**

NrAllowedRotations

See NrAllowedRotations.

```
public int NrAllowedRotations
```

Field Value

int

NrDoorConnections

See MaximumDoorCount.

```
public int NrDoorConnections
```

Field Value

int

RegistryId

ID the corresponding RoomDataSO is registered with in the DBPRegistry used to create this instance.

```
public int RegistryId
```

Field Value

int

See Also
CreateAlloc(RoomDataSO, DBPRegistry), DBPRegistry

SizeByRotation

All rotations allowed for this room cached in a list.

```
public UnsafeList<GridSize> SizeByRotation
```

Field Value

UnsafeList<GridSize>

Remarks

This list is allocated by CreateAlloc(RoomDataSO, DBPRegistry) using Unity.Collections.Allocator.Persistent. Call Dispose() to free it. You should not free this list itself, because it leaves the room data unusable, but does not free the other unmanaged memory allocated. **You will not be able to free all memory manually because some of it isn't accessible. Call Dispose() instead.**

See Also
GetRotatedSize(GridRotation)

Methods

Dispose()

Frees native memory. You must not continue using this instance after calling this.

```
public void Dispose()
```

Struct UnmanagedRoomDataCollection

Namespace: PWS.DungeonBlueprint.Data

Unmanaged counter-part of RoomDataCollectionSO. Used for crossing the room data collection over into Burst compatible code.

```
public struct UnmanagedRoomDataCollection : IDisposable
```

Implements
IDisposable

Inherited Members
ValueType.Equals(object) , ValueType.GetHashCode() , ValueType.ToString() , object.Equals(object, object) , object.GetType() , object.ReferenceEquals(object, object)

Fields

RegistryId

ID the corresponding RoomDataCollectionSO is registered with in the DBPRegistry used to create this instance.

```
public int RegistryId
```

Field Value

int

See Also
CreateAlloc(RoomDataCollectionSO, DBPRegistry), DBPRegistry

Rooms

See Rooms.

```
public UnsafeList<UnmanagedRoomData> Rooms
```

Field Value

UnsafeList<UnmanagedRoomData>

Remarks

This list is allocated by CreateAlloc(RoomDataCollectionSO, DBPRegistry) using Unity.Collections.Allocator.Persistent. Call Dispose() to free it.

Methods

Dispose()

Frees native memory. You must not continue using this instance after calling this.

```
public void Dispose()
```

Struct UnmanagedRoomDoorConnection

Namespace: PWS.DungeonBlueprint.Data

Unmanaged counter-part of RoomDoorConnection. Used for crossing the room door connections over into Burst compatible code.

```
public struct UnmanagedRoomDoorConnection
```

Inherited Members
ValueType.Equals(object) , ValueType.GetHashCode() , ValueType.ToString() , object.Equals(object, object) , object.GetType() , object.ReferenceEquals(object, object)

Fields

ConnectionIdx

Index of this connection in the room data the connection belongs to.

```
public int ConnectionIdx
```

Field Value

int

Direction

See Direction.

```
public GridDirection Direction
```

Field Value

GridDirection

DoorSize

See DoorSize.

```
public GridSize DoorSize
```

Field Value

GridSize

LocalGridPosition

See LocalGridPosition.

```
public GridPosition LocalGridPosition
```

Field Value

GridPosition

Methods

Create(RoomDoorConnection, int)

Creates a new instance of UnmanagedRoomDoorConnection by copying all data of the given RoomDoorConnection.

```
public static UnmanagedRoomDoorConnection Create(RoomDoorConnection
roomDoorConnection, int connectionIdx)
```

Parameters

`roomDoorConnection` RoomDoorConnection

Managed door connection to create the unmanaged door connection for.

`connectionIdx` int

Index of the managed door connection in its room.

Returns

UnmanagedRoomDoorConnection

Unmanaged door connection for the given managed door connection.

Namespace PWS.DungeonBlueprint.Grid

Classes

GridDirectionExtensions
: Extension methods for GridDirection enum.

GridRotationExtensions
: Extension methods for GridRotation enum.

Structs

GridPosition
: Represents a position in the dungeon grid. Sometimes also used to represent a position offset.
Similar to UnityEngine.Vector3Int.

GridSize
: Represents a size in the dungeon. The unit depends on the context and is not necessarily number of tiles in the dungeon grid.
Similar to GridPosition. The main difference is the semantics they are used with.

Enums

GridDirection
: Represents a direction in the dungeon grid.

GridRotation
: Represents a rotation in the dungeon grid.
Rotations are always around the y-axis.

Enum GridDirection

Namespace: PWS.DungeonBlueprint.Grid

Represents a direction in the dungeon grid.

```
public enum GridDirection : byte
```

Extension Methods
GridDirectionExtensions.IsAlongXAxis(GridDirection) ,
GridDirectionExtensions.IsAlongYAxis(GridDirection) ,
GridDirectionExtensions.IsAlongZAxis(GridDirection) ,
GridDirectionExtensions.IsHorizontal(GridDirection) ,
GridDirectionExtensions.IsNegative(GridDirection) ,
GridDirectionExtensions.IsPositive(GridDirection) ,
GridDirectionExtensions.IsVertical(GridDirection) ,
GridDirectionExtensions.Opposite(GridDirection) ,
GridDirectionExtensions.Rotate(GridDirection, GridRotation) ,
GridDirectionExtensions.RotationToMakeOpposite(GridDirection, GridDirection) ,
GridDirectionExtensions.ToDirectionVector(GridDirection, out GridPosition) ,
GridDirectionExtensions.ToRotation(GridDirection)

Fields

`None = 0`

No direction. Typically considered an invalid or null direction.

`HorizontalForward = 1`

Positive direction along z-axis (similar to UnityEngine.Vector3Int.forward).

`HorizontalRight = 2`

Positive direction along x-axis (similar to UnityEngine.Vector3Int.right).

`HorizontalBackward = 3`

Negative direction along z-axis (similar to UnityEngine.Vector3Int.back).

`HorizontalLeft = 4`

Negative direction along x-axis (similar to UnityEngine.Vector3Int.left).

`VerticalUp = 5`

Positive direction along y-axis (similar to UnityEngine.Vector3Int.up).

`VerticalDown = 6`

Negative direction along y-axis (similar to UnityEngine.Vector3Int.down).

Remarks

A direction can be turned to a direction vector using ToDirectionVector(GridDirection, out GridPosition).
A position and direction vector result in a new position.

See Also

GridDirectionExtensions

Class GridDirectionExtensions

Namespace: PWS.DungeonBlueprint.Grid

Extension methods for GridDirection enum.

```
public static class GridDirectionExtensions
```

Inheritance

object ← GridDirectionExtensions

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Methods

IsAlongXAxis(GridDirection)

Checks whether the given direction points along the x-axis (either positive or negative).

```
public static bool IsAlongXAxis(this GridDirection direction)
```

Parameters

`direction` GridDirection

Direction to check.

Returns

bool

Whether the direction points along the x-axis.

Remarks

This is equivalent to checking whether the direction is HorizontalRight or HorizontalLeft.

See Also
IsAlongYAxis(GridDirection), IsAlongZAxis(GridDirection)

IsAlongYAxis(GridDirection)

Checks whether the given direction points along the y-axis (either positive or negative).

```
public static bool IsAlongYAxis(this GridDirection direction)
```

Parameters

`direction` GridDirection

Direction to check.

Returns

bool

Whether the direction points along the y-axis.

Remarks

This is equivalent to checking whether the direction is VerticalUp or VerticalDown.
This is also equivalent to IsVertical(GridDirection).

See Also
IsAlongXAxis(GridDirection), IsAlongZAxis(GridDirection), IsVertical(GridDirection)

IsAlongZAxis(GridDirection)

Checks whether the given direction points along the z-axis (either positive or negative).

```
public static bool IsAlongZAxis(this GridDirection direction)
```

Parameters

`direction` GridDirection

Direction to check.

Returns

bool

Whether the direction points along the z-axis.

Remarks

This is equivalent to checking whether the direction is HorizontalForward or HorizontalBackward.

See Also
IsAlongXAxis(GridDirection), IsAlongYAxis(GridDirection)

IsHorizontal(GridDirection)

Checks whether the given direction is along the x-z-plane.

```
public static bool IsHorizontal(this GridDirection direction)
```

Parameters

`direction` GridDirection

Direction to check.

Returns

bool

Whether the direction is along the x-z-plane.

Remarks

This is equivalent to checking whether the direction is HorizontalForward or HorizontalRight or HorizontalBackward or HorizontalLeft.
This is also equivalent to checking whether the direction is IsAlongXAxis(GridDirection) or IsAlongZAxis(GridDirection).

See Also

IsAlongXAxis(GridDirection), IsAlongZAxis(GridDirection), IsVertical(GridDirection)

IsNegative(GridDirection)

Checks whether the given direction points in the negative direction of its axis.

```
public static bool IsNegative(this GridDirection direction)
```

Parameters

`direction` GridDirection

Direction to check.

Returns

bool

Whether the direction points in the negative direction of its axis.

Remarks

This is equivalent to checking whether the direction is **not** HorizontalForward or HorizontalRight or VerticalUp or None.

See Also
IsPositive(GridDirection)

IsPositive(GridDirection)

Checks whether the given direction points in the positive direction of its axis.

```
public static bool IsPositive(this GridDirection direction)
```

Parameters

`direction` GridDirection

Direction to check.

Returns

bool

Whether the direction points in the positive direction of its axis.

Remarks

This is equivalent to checking whether the direction is HorizontalForward or HorizontalRight or VerticalUp.

See Also
IsNegative(GridDirection)

IsVertical(GridDirection)

Checks whether the given direction points along the y-axis (either positive or negative).

```
public static bool IsVertical(this GridDirection direction)
```

Parameters

`direction` GridDirection

Direction to check.

Returns

bool

Whether the direction points along the y-axis.

Remarks

This is equivalent to checking whether the direction is VerticalUp or VerticalDown.
This is also equivalent to IsAlongYAxis(GridDirection).

See Also
IsAlongYAxis(GridDirection), IsHorizontal(GridDirection)

Opposite(GridDirection)

Calculates the direction pointing in the opposite direction of the given direction.

```
public static GridDirection Opposite(this GridDirection direction)
```

Parameters

`direction` GridDirection

Direction to calculate the opposite direction for.

Returns

GridDirection

Opposite direction.

Remarks

Opposite direction is always on the same axis as the given direction. Positive directions are flipped to negative and negative directions are flipped to positive.
Opposite of None is None.

Rotate(GridDirection, GridRotation)

Rotates the given direction by the given GridRotation. We always rotate clockwise on the x-z plane.
Do not call this function with a vertical direction or None. This will throw an exception if ENABLE_UNITY_COLLECTIONS_CHECKS is defined.

```
public static GridDirection Rotate(this GridDirection direction,
GridRotation byRotation)
```

Parameters

`direction` GridDirection

Direction to rotate.

`byRotation` GridRotation

Rotation to rotate the direction by.

Returns

GridDirection

Rotated direction.

Exceptions

InvalidOperationException

When called with a vertical direction or None and ENABLE_UNITY_COLLECTIONS_CHECKS is defined.

See Also
RotationToMakeOpposite(GridDirection, GridDirection), IsVertical(GridDirection)

RotationToMakeOpposite(GridDirection, GridDirection)

Calculates the rotation the first given direction would have to be rotated with to become the opposite of the second given direction.
Do not call this function with a vertical direction or None. This will throw an exception if ENABLE_UNITY_COLLECTIONS_CHECKS is defined.

```
public static GridRotation RotationToMakeOpposite(this GridDirection current,
GridDirection oppositeToBe)
```

Parameters

`current` GridDirection

Direction that will become the opposite of the other direction if rotated by the return value.

`oppositeToBe` GridDirection

Direction that will be opposite of the other direction if it is rotated by the return value.

Returns

GridRotation

Rotation the first direction needs to be rotated with to become the opposite of the second direction.

Exceptions

InvalidOperationException

When called with a vertical direction or None and ENABLE_UNITY_COLLECTIONS_CHECKS is defined.

See Also
Rotate(GridDirection, GridRotation), Opposite(GridDirection)

ToDirectionVector(GridDirection, out GridPosition)

Turns the given direction into a direction vector that acts as a 1 tile offset in the dungeon grid along the given direction.

```
public static void ToDirectionVector(this GridDirection direction, out
GridPosition directionVector)
```

Parameters

`direction` GridDirection

Direction to calculate the direction vector for.

`directionVector` GridPosition

Output: gets assigned to the direction vector. Adding this to a GridPosition results in neighboring position in direction of the given direction.

Remarks

This uses the respective constants defined in GridPosition, i.e. HorizontalForward turns into Forward.

See Also
GridPosition

ToRotation(GridDirection)

Turns the given direction to a GridRotation. HorizontalForward is considered the base (unrotated) direction.
Do not call this function with a vertical direction or None. This will throw an exception if ENABLE_UNITY_COLLECTIONS_CHECKS is defined.

```
public static GridRotation ToRotation(this GridDirection direction)
```

Parameters

`direction` GridDirection

Direction to turn into a rotation.

Returns

GridRotation

Rotation fitting the given direction.

Remarks

This is a simple mapping of the two enums.
HorizontalForward -> Cw0 HorizontalRight -> Cw90 HorizontalBackward -> Cw180 HorizontalLeft -> Cw270

Exceptions

InvalidOperationException

When called with a vertical direction or None and ENABLE_UNITY_COLLECTIONS_CHECKS is defined.

See Also

GridDirection

Struct GridPosition

Namespace: PWS.DungeonBlueprint.Grid

Represents a position in the dungeon grid. Sometimes also used to represent a position offset.
Similar to UnityEngine.Vector3Int.

```
[Serializable]
public struct GridPosition : IEquatable<GridPosition>
```

Implements
IEquatable<GridPosition>

Inherited Members
object.Equals(object, object) , object.GetType() , object.ReferenceEquals(object, object)

Constructors

GridPosition(int, int, int)

Creates a new instance of GridPosition with the given coordinates.

```
public GridPosition(int x, int y, int z)
```

Parameters

x int

X-coordinate of the new instance.

y int

Y-coordinate of the new instance.

z int

Z-coordinate of the new instance.

Fields

Backward

Position (0, 0, -1).

```
public static readonly GridPosition Backward
```

Field Value

GridPosition

Down

Position (0, -1, 0).

```
public static readonly GridPosition Down
```

Field Value

GridPosition

Forward

Position (0, 0, 1).

```
public static readonly GridPosition Forward
```

Field Value

GridPosition

Invalid

Position (int.MinValue, int.MinValue, int.MinValue).
Used to represent invalid positions, since default(GridPosition) is (0, 0, 0) but that is a valid

position.

```
public static readonly GridPosition Invalid
```

Field Value

GridPosition

Left

Position (-1, 0, 0).

```
public static readonly GridPosition Left
```

Field Value

GridPosition

Right

Position (1, 0, 0).

```
public static readonly GridPosition Right
```

Field Value

GridPosition

Up

Position (0, 1, 0).

```
public static readonly GridPosition Up
```

Field Value

GridPosition

Zero

Position (0, 0, 0).

```
public static readonly GridPosition Zero
```

Field Value

GridPosition

Properties

X

X-coordinate of the position.

```
public readonly int X { get; }
```

Property Value

int

Y

Y-coordinate of the position.

```
public readonly int Y { get; }
```

Property Value

int

Z

Z-coordinate of the position.

```
public readonly int Z { get; }
```

Property Value

int

Methods

Equals(GridPosition)

Checks if the given position has the same x, y and z-coordinates as this.

```
public bool Equals(GridPosition other)
```

Parameters

`other` GridPosition

Position to check for equality.

Returns

bool

`true` if `other` has equal x, y and z-coordinates as this, otherwise `false`.

Equals(object)

Checks `obj` is a GridPosition and then calls Equals(GridPosition) for equality check.

```
public override bool Equals(object obj)
```

Parameters

`obj` object

Object to check for equality with this position.

Returns

bool

Whether `obj` is a GridPosition with the same coordinates as this.

GetHashCode()

Returns the hash code for this instance.

```
public override int GetHashCode()
```

Returns

int

A 32-bit signed integer that is the hash code for this instance.

ManhattanDistance(GridPosition, GridPosition)

Calculates the manhattan distance between two positions.

```
public static int ManhattanDistance(GridPosition a, GridPosition b)
```

Parameters

a GridPosition

First position.

b GridPosition

Second position.

Returns

int

Manhattan Distance between the two positions.

ToString()

Turns the position into a human-readable string.

```
public override string ToString()
```

Returns

string

String with the format (X, Y, Z).

Operators

operator ==(GridPosition, GridPosition)

```
public static bool operator ==(GridPosition left, GridPosition right)
```

Parameters

`left` GridPosition

`right` GridPosition

Returns

bool

explicit operator Vector3Int(GridPosition)

```
public static explicit operator Vector3Int(GridPosition position)
```

Parameters

`position` GridPosition

Returns

Vector3Int

explicit operator GridPosition(Vector3Int)

```
public static explicit operator GridPosition(Vector3Int vector)
```

Parameters

`vector` Vector3Int

Returns

GridPosition

operator !=(GridPosition, GridPosition)

```
public static bool operator !=(GridPosition left, GridPosition right)
```

Parameters

`left` GridPosition

`right` GridPosition

Returns

bool

Enum GridRotation

Namespace: PWS.DungeonBlueprint.Grid

Represents a rotation in the dungeon grid.
Rotations are always around the y-axis.

```
public enum GridRotation : byte
```

Extension Methods
GridRotationExtensions.ToQuaternion(GridRotation)

Fields

`Cw0 = 0`

No rotation.

`Cw90 = 1`

90-degree clockwise rotation.

`Cw180 = 2`

180-degree clockwise rotation.

`Cw270 = 3`

270-degree clockwise rotation.

See Also

GridRotationExtensions

Class GridRotationExtensions

Namespace: PWS.DungeonBlueprint.Grid

Extension methods for GridRotation enum.

```
public static class GridRotationExtensions
```

Inheritance

object ← GridRotationExtensions

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Methods

ToQuaternion(GridRotation)

Turns the given rotation to a Quaternion rotated on the y-axis.
Amount of rotation on the y-axis directly maps to the degrees of the given rotation.

```
public static Quaternion ToQuaternion(this GridRotation rotation)
```

Parameters

`rotation` GridRotation

Rotation to turn into a quaternion.

Returns

Quaternion

Quaternion with a rotation on the y-axis according to the given rotation.

See Also

GridRotation

Struct GridSize

Namespace: PWS.DungeonBlueprint.Grid

Represents a size in the dungeon. The unit depends on the context and is not necessarily number of tiles in the dungeon grid.
Similar to GridPosition. The main difference is the semantics they are used with.

```
[Serializable]
public struct GridSize : IEquatable<GridSize>
```

Implements
IEquatable<GridSize>

Inherited Members
object.Equals(object, object) , object.GetType() , object.ReferenceEquals(object, object)

Constructors

GridSize(int, int, int)

Creates a new instance GridSize with the given dimensions.

```
public GridSize(int width, int height, int depth)
```

Parameters

`width` int

`height` int

`depth` int

Fields

One

Size of (1, 1, 1).

```
public static readonly GridSize One
```

Field Value

GridSize

Zero

Size of (0, 0, 0).

```
public static readonly GridSize Zero
```

Field Value

GridSize

Properties

Depth

Depth of the size.
Depth is the z-axis dimension.

```
public readonly int Depth { get; }
```

Property Value

int

Height

Height of the size.
Height is the y-axis dimension.

```
public readonly int Height { get; }
```

Property Value

int

Width

Width of the size.
Width is the x-axis dimension.

```
public readonly int Width { get; }
```

Property Value

int

Methods

Equals(GridSize)

Indicates whether the current object is equal to another object of the same type.

```
public bool Equals(GridSize other)
```

Parameters

`other` GridSize

An object to compare with this object.

Returns

bool

true if the current object is equal to the `other` parameter; otherwise, false.

Equals(object)

Indicates whether this instance and a specified object are equal.

```
public override bool Equals(object obj)
```

Parameters

`obj` object

The object to compare with the current instance.

Returns

bool

true if `obj` and this instance are the same type and represent the same value; otherwise, false.

GetHashCode()

Returns the hash code for this instance.

```
public override int GetHashCode()
```

Returns

int

A 32-bit signed integer that is the hash code for this instance.

ToString()

Turns the position into a human-readable string.

```
public override string ToString()
```

Returns

string

String with the format (Width, Height, Depth).

Operators

operator ==(GridSize, GridSize)

```
public static bool operator ==(GridSize left, GridSize right)
```

Parameters

`left` GridSize

`right` GridSize

Returns

bool

explicit operator Vector3Int(GridSize)

```
public static explicit operator Vector3Int(GridSize size)
```

Parameters

`size` GridSize

Returns

Vector3Int

explicit operator GridSize(Vector3Int)

```
public static explicit operator GridSize(Vector3Int vector)
```

Parameters

`vector` Vector3Int

Returns

GridSize

operator !=(GridSize, GridSize)

```
public static bool operator !=(GridSize left, GridSize right)
```

Parameters

`left` GridSize

`right` GridSize

Returns

bool

Namespace PWS.DungeonBlueprint.Mission

Classes

MissionGraphAsset

Represents a serialized Mission asset. Allows serializing an instance of MissionGraph to a ScriptableObject asset and deserializing back to a functionally equivalent MissionGraph instance. Serialization is editor-only, while deserialization is available both in editor and runtime.

Structs

MissionGraph

Main data structure for Mission phase of the transformation pipeline.
This is effectively the input into the pipeline, which is built by the user, either through editor tooling or by code.
This data structure bundles the main graph along with all its subgraphs each as Partial MissionGraph.

MissionGraph.ReadOnly

Read-only wrapper around a mission graph instance.
This is only a view over the existing memory and not a deep copy.

MissionGroupMetaData

Container for packing multiple data points about mission group nodes into a single 32-bit int.

PartialMissionGraph

Part of a mission graph. This is where the actual nodes of the mission are.
Every mission graph has at least one partial mission graph. Additionally, it has a partial mission graph for every group node in the main partial graph.

PartialMissionGraph.ReadOnly

Read-only wrapper around a partial mission graph instance.
This is only a view over the existing memory and not a deep copy.

ResolvedMissionGraph

Main data structure for the ResolveMission phase of the transformation pipeline.
This is the first phase where the group nodes of the mission graph have been fully inserted or omitted.

ResolvedMissionGraph.ReadOnly

Read-only wrapper around a resolved mission graph instance.
This is only a view over the existing memory and not a deep copy.

Enums

MissionNodeType

Type of nodes in a mission. Every node has exactly one type.

MissionPortalType

Type of portals in a mission. Every portal has exactly one type.

Struct MissionGraph

Namespace: PWS.DungeonBlueprint.Mission

Main data structure for Mission phase of the transformation pipeline.
This is effectively the input into the pipeline, which is built by the user, either through editor tooling or by code.
This data structure bundles the main graph along with all its subgraphs each as PartialMissionGraph.

```
public struct MissionGraph : IDisposable
```

Implements
IDisposable

Inherited Members
ValueType.Equals(object) , ValueType.GetHashCode() , ValueType.ToString() , object.Equals(object, object) , object.GetType() , object.ReferenceEquals(object, object)

Fields

Root

Main graph of this mission graph.
A valid mission graph always has a valid root partial mission graph.

```
public PartialMissionGraph Root
```

Field Value

PartialMissionGraph

Subgraphs

All subgraphs used in the main graph.
The key is the subgraph identifier in the group node in the main graph and also the

identifier the mission graph asset of the subgraph is registered with in the DBPRegistry.

```
public UnsafeHashMap<int, PartialMissionGraph> Subgraphs
```

Field Value

UnsafeHashMap<int, PartialMissionGraph>

Properties

IsCreated

Checks whether this instance currently holds the necessary memory to be valid.
This is a necessary condition for a valid instance, but not a sufficient one.
This means that if this is false, it's definitely not a valid instance to run logic on, but if this is true, that is not a guarantee that it is a valid instance.

```
public readonly bool IsCreated { get; }
```

Property Value

bool

Remarks

This is identical to checking IsCreated on Root. The member Subgraphs is not checked, since mission graphs that have no subgraphs may not allocate it.

Methods

AsReadOnly()

Creates a new read-only wrapper and returns it.

```
public MissionGraph.ReadOnly AsReadOnly()
```

Returns

MissionGraph.ReadOnly

Read-only view of this instance.

See Also
MissionGraph.ReadOnly

Dispose()

Frees native memory. You must not continue using this instance after calling this.

```
public void Dispose()
```

See Also

MissionGraphAsset

Struct MissionGraph.ReadOnly

Namespace: PWS.DungeonBlueprint.Mission

Read-only wrapper around a mission graph instance.
This is only a view over the existing memory and not a deep copy.

```
public readonly struct MissionGraph.ReadOnly
```

Inherited Members
ValueType.Equals(object) , ValueType.GetHashCode() , ValueType.ToString() , object.Equals(object, object) , object.GetType() , object.ReferenceEquals(object, object)

Constructors

ReadOnly(ref MissionGraph)

Creates a new read-only wrapper instance around the given mission graph.

```
public ReadOnly(ref MissionGraph original)
```

Parameters

`original` MissionGraph

Mission graph to wrap.

Properties

IsCreated

Mirrors IsCreated of the instance the readonly instance was created with.

```
public bool IsCreated { get; }
```

Property Value

bool

Root

Read-only view on Root.

```
public PartialMissionGraph.ReadOnly Root { get; }
```

Property Value

PartialMissionGraph.ReadOnly

Methods

TryGetSubgraph(int, out ReadOnly)

Read-only view on Subgraphs accessible only by per key.

```
public bool TryGetSubgraph(int subgraphId, out PartialMissionGraph.ReadOnly
subgraph)
```

Parameters

`subgraphId` int

Subgraph identifier you want to get.

`subgraph` PartialMissionGraph.ReadOnly

Output parameter for the returned subgraph

Returns

bool

True if there was a subgraph with the given identifier, otherwise false.

Class MissionGraphAsset

Namespace: PWS.DungeonBlueprint.Mission

Represents a serialized Mission asset. Allows serializing an instance of MissionGraph to a ScriptableObject asset and deserializing back to a functionally equivalent MissionGraph instance. Serialization is editor-only, while deserialization is available both in editor and runtime.

```
public class MissionGraphAsset : ScriptableObject
```

Inheritance

object ← Object ← ScriptableObject ← MissionGraphAsset

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Fields

CurrentVersion

Current version of the serialization logic.
Required to identify assets that were serialized with older serialization logic than the current.

```
public const int CurrentVersion = 1
```

Field Value

int

Properties

DoorDataCollections

All DoorDataCollectionSO instances used in this mission graph.

```
public IReadOnlyList<DoorDataCollectionSO> DoorDataCollections { get; }
```

Property Value

IReadOnlyList<DoorDataCollectionSO>

Remarks

This does not include any instances used in any subgraphs.

Name

Name of the mission graph.

```
public string Name { get; }
```

Property Value

string

Remarks

This is used for organizational purposes only. It has no effect on dungeon generation.

RoomDataCollections

All RoomDataCollectionSO instances used in this mission graph.

```
public IReadOnlyList<RoomDataCollectionSO> RoomDataCollections { get; }
```

Property Value

IReadOnlyList<RoomDataCollectionSO>

Remarks

This does not include any instances used in any subgraphs.

SubgraphAssets

All MissionGraphAsset instances used as subgraphs in this mission graph asset.

```
public IReadOnlyList<MissionGraphAsset> SubgraphAssets { get; }
```

Property Value

IReadOnlyList<MissionGraphAsset>

Methods

DeserializeAlloc(DBPRegistry, out MissionGraph)

Deserializes this asset into a newly allocated MissionGraph instance.

```
public void DeserializeAlloc(DBPRegistry registry, out MissionGraph missionGraph)
```

Parameters

`registry` DBPRegistry

Registry that all RoomDataCollectionSO, DoorDataCollectionSO and MissionGraphAsset used in the serialized mission graph will be registered in when the method returns successfully.

`missionGraph` MissionGraph

Output parameter that will contain the deserialized MissionGraph when the method returns successfully.

Remarks

This allocates native memory with Unity.Collections.Allocator.Persistent. You must call Dispose() on the returned instance to free it.

Exceptions

FormatException

If the asset was serialized with a different version than CurrentVersion.

DeserializeRootPartialGraphAlloc(DBPRegistry, out PartialMissionGraph)

Deserializes a partial mission graph. This may be the main graph of a MissionGraph or one of its subgraphs.
This is called from within DeserializeAlloc(DBPRegistry, out MissionGraph) and is otherwise only needed during import of newly created mission graphs.
It is unlikely you need to call this yourself.

```
public void DeserializeRootPartialGraphAlloc(DBPRegistry registry, out
PartialMissionGraph partialGraph)
```

Parameters

`registry` DBPRegistry

Registry that all RoomDataCollectionSO, DoorDataCollectionSO and MissionGraphAsset used in the serialized mission graph will be registered in when the method returns successfully.

`partialGraph` PartialMissionGraph

Output parameter that will contain the deserialized PartialMissionGraph when the method returns successfully.

Remarks

This allocates native memory with Unity.Collections.Allocator.Persistent. You must call Dispose() on the returned instance to free it.

Exceptions

FormatException

If the asset was serialized with a different version than CurrentVersion.

ArgumentOutOfRangeException

If `partialGraph` is malformed.

Struct MissionGroupMetaData

Namespace: PWS.DungeonBlueprint.Mission

Container for packing multiple data points about mission group nodes into a single 32-bit int.

```
public struct MissionGroupMetaData
```

Inherited Members
ValueType.Equals(object) , ValueType.GetHashCode() , ValueType.ToString() , object.Equals(object, object) , object.GetType() , object.ReferenceEquals(object, object)

Constructors

MissionGroupMetaData(int)

Creates a new instance of MissionGroupMetaData from an encoding of the data.
The data typically comes from NodeData.

```
public MissionGroupMetaData(int packedData)
```

Parameters

`packedData` int

Encoding of the metadata.

Remarks

IsOptional: first bit
SubgraphId: next 10 bits
RepeatsMin: next 10 bits
RepeatsMax: next 11 bits

Fields

IsOptional

Whether the subgraph is optional in the main graph.
During mission resolution, a 50/50 roll decides whether to insert an optional subgraph.

```
public readonly bool IsOptional
```

Field Value

bool

RepeatsMax

Maximum number of repetitions of this subgraph if it is inserted.

```
public readonly int RepeatsMax
```

Field Value

int

Remarks

This must be equal to or larger than RepeatsMin.

RepeatsMin

Minimum number of repetitions of this subgraph if it is inserted.

```
public readonly int RepeatsMin
```

Field Value

int

Remarks

This must be equal to or smaller than RepeatsMax.

SubgraphId

The identifier the subgraph is registered with in the DBPRegistry.

```
public readonly int SubgraphId
```

Field Value

int

See Also
DBPRegistry

Methods

Pack(bool, int, int, int)

Packs the given parameters into a single 32-bit int according to the metadata format.

```
public static int Pack(bool isOptional, int subgraphId, int repeatsMin,
int repeatsMax)
```

Parameters

`isOptional` bool

See IsOptional.

`subgraphId` int

See SubgraphId.

`repeatsMin` int

See RepeatsMin.

`repeatsMax` int

See RepeatsMax.

Returns

int

A single 32-bit int encoding the given parameters

Remarks

Result can be passed into MissionGroupMetaData(int) to revert the encoding back into the original parameters.

See Also
MissionGroupMetaData(int)

See Also

NodeData

Enum MissionNodeType

Namespace: PWS.DungeonBlueprint.Mission

Type of nodes in a mission. Every node has exactly one type.

```
public enum MissionNodeType : byte
```

Fields

`Portal = 0`

Node type for portals.

`Room = 1`

Node type for rooms.

`Door = 2`

Node type for doors.

`Group = 3`

Node type for groups.

Enum MissionPortalType

Namespace: PWS.DungeonBlueprint.Mission

Type of portals in a mission. Every portal has exactly one type.

```
public enum MissionPortalType : byte
```

Fields

`Entrance = 0`

Used to represent an entry portal.

`Exit = 1`

Used to represent an exit portal.

Struct PartialMissionGraph

Namespace: PWS.DungeonBlueprint.Mission

Part of a mission graph. This is where the actual nodes of the mission are.
Every mission graph has at least one partial mission graph. Additionally, it has a partial mission graph for every group node in the main partial graph.

```
public struct PartialMissionGraph : IDisposable
```

Implements
IDisposable

Inherited Members
ValueType.Equals(object) , ValueType.GetHashCode() , ValueType.ToString() , object.Equals(object, object) , object.GetType() , object.ReferenceEquals(object, object)

Fields

AdjacencyPortalName

Additional data about some adjacencies.
This data only exist for two kinds of nodes:

- nodes that connect to group nodes to identify which portal node inside the group the node will connect to during mission resolution.
- portals inside subgraphs that connect to another portal to create loops if the group will be inserted multiple times during mission resolution.
  Key is the adjacency index, value is the name of the portal.

```
public NativeHashMap<int, FixedString64Bytes> AdjacencyPortalName
```

Field Value

NativeHashMap<int, FixedString64Bytes>

Remarks

This object owns this native memory and frees it in Dispose().

See Also
NodeAdjacencies

NodeAdjacencies

Adjacencies between nodes.
Adjacencies of a given node `n`, are in the range `[n*NrAdjacenciesPerNode, n*NrAdjacenciesPerNode+NrAdjacenciesPerNode)`.
Total length of this collection is the `number of nodes * NrAdjacenciesPerNode`.
Every index is either a node or InvalidNode.

```
public NativeArray<int> NodeAdjacencies
```

Field Value

NativeArray<int>

Examples

If `NrAdjacenciesPerNode=3`, the adjacencies for node `0` are at the indices 0, 1 and 2 (the range is `[0*3=0, 0*3+3=3)`).
If `NrAdjacenciesPerNode=2`, the adjacencies for node `4` are at the indices 8 and 9 (the range is `[4*2=8, 4*2+2=10)`).

Remarks

Adjacencies are always bidirectional, so if a node has an adjacency to another node, the second node must also have an adjacency to the first node.
This object owns this native memory and frees it in Dispose().

See Also
NrAdjacenciesPerNode

NodeData

Data of each node.
Meaning of this data changes depending on the type of the node.

- portals: data is the MissionNodeType.
- rooms: data is the identifier the node's RoomDataCollectionSO is registered with in the DBPRegistry.
- doors: data is the identifier the node's DoorDataCollectionSO is registered with in the DBPRegistry.
- groups: data is an encoding of optionality, subgraph identifier and repetition range. See MissionGroupMetaData.

```
public NativeArray<int> NodeData
```

Field Value

NativeArray<int>

Remarks

This array is in parallel with NodeTypes and NodeNames, meaning the arrays contain information about the same node at a given index.
This object owns this native memory and frees it in Dispose().

See Also
NodeTypes, NodeNames, MissionNodeType, RoomDataCollectionSO, DoorDataCollectionSO, MissionGroupMetaData, DBPRegistry

NodeNames

Name of each node.
The name is carried all the way into the spawned dungeon and will then be queryable with GetRoomByName(FixedString64Bytes, int) and GetDoorByName(FixedString64Bytes, int).

```
public NativeArray<FixedString64Bytes> NodeNames
```

Field Value

NativeArray<FixedString64Bytes>

Remarks

This array is in parallel with NodeTypes and NodeData, meaning the arrays contain information about the same node at a given index.

This object owns this native memory and frees it in Dispose().

See Also

NodeTypes, NodeData

NodeTypes

Type of each node.

```
public NativeArray<MissionNodeType> NodeTypes
```

Field Value

NativeArray<MissionNodeType>

Remarks

This array is in parallel with NodeNames and NodeData, meaning the arrays contain information about the same node at a given index.
This object owns this native memory and frees it in Dispose().

See Also

NodeNames, NodeData

NrAdjacenciesPerNode

Maximum number of adjacencies any node in the graph has.
This is used to determine how many slots are reserved for each node in the adjacency collection.

```
public int NrAdjacenciesPerNode
```

Field Value

int

Properties

IsCreated

Checks whether this instance currently holds the necessary memory to be valid.
This is a necessary condition for a valid instance, but not a sufficient one.
This means that if this is false, it's definitely not a valid instance to run logic on, but if this is true, that is not a guarantee that it is a valid instance.

```
public readonly bool IsCreated { get; }
```

Property Value

bool

Remarks

The member AdjacencyPortalName is not checked, since partial mission graphs that aren't subgraphs and have no subgraphs may not allocate it.

Methods

AsReadOnly()

Creates a new read-only wrapper and returns it.

```
public PartialMissionGraph.ReadOnly AsReadOnly()
```

Returns

PartialMissionGraph.ReadOnly

Read-only view of this instance.

See Also
PartialMissionGraph.ReadOnly

Dispose()

Frees native memory. You must not continue using this instance after calling this.

```
public void Dispose()
```

See Also

MissionGraph

Struct PartialMissionGraph.ReadOnly

Namespace: PWS.DungeonBlueprint.Mission

Read-only wrapper around a partial mission graph instance.
This is only a view over the existing memory and not a deep copy.

```
public readonly struct PartialMissionGraph.ReadOnly
```

Inherited Members
ValueType.Equals(object) , ValueType.GetHashCode() , ValueType.ToString() , object.Equals(object, object) , object.GetType() , object.ReferenceEquals(object, object)

Constructors

ReadOnly(ref PartialMissionGraph)

Creates a new read-only wrapper instance around the given partial mission graph.

```
public ReadOnly(ref PartialMissionGraph original)
```

Parameters

`original` PartialMissionGraph

Partial mission graph to wrap.

Properties

AdjacencyPortalName

Read-only view on AdjacencyPortalName.

```
public NativeHashMap<int, FixedString64Bytes>.ReadOnly AdjacencyPortalName { get; }
```

Property Value

NativeHashMap<int, FixedString64Bytes>.ReadOnly

IsCreated

Mirrors IsCreated of the instance the readonly instance was created with.

```
public bool IsCreated { get; }
```

Property Value

bool

NodeAdjacencies

Read-only view on NodeAdjacencies.

```
public NativeArray<int>.ReadOnly NodeAdjacencies { get; }
```

Property Value

NativeArray<int>.ReadOnly

NodeData

Read-only view on NodeData.

```
public NativeArray<int>.ReadOnly NodeData { get; }
```

Property Value

NativeArray<int>.ReadOnly

NodeNames

Read-only view on NodeNames.

```
public NativeArray<FixedString64Bytes>.ReadOnly NodeNames { get; }
```

Property Value

NativeArray<FixedString64Bytes>.ReadOnly

NodeTypes

Read-only view on NodeTypes.

```
public NativeArray<MissionNodeType>.ReadOnly NodeTypes { get; }
```

Property Value

NativeArray<MissionNodeType>.ReadOnly

NrAdjacenciesPerNode

Read-only view on NrAdjacenciesPerNode.

```
public int NrAdjacenciesPerNode { get; }
```

Property Value

int

Struct ResolvedMissionGraph

Namespace: PWS.DungeonBlueprint.Mission

Main data structure for the ResolveMission phase of the transformation pipeline.
This is the first phase where the group nodes of the mission graph have been fully inserted or omitted.

```
[BurstCompile]
public struct ResolvedMissionGraph : IDisposable
```

Implements
IDisposable

Inherited Members
ValueType.Equals(object) , ValueType.GetHashCode() , ValueType.ToString() , object.Equals(object, object) , object.GetType() , object.ReferenceEquals(object, object)

Fields

NodeAdjacencies

Adjacencies between nodes.
Adjacencies of a given node `n`, are in the range `[n*NrAdjacenciesPerNode, n*NrAdjacenciesPerNode+NrAdjacenciesPerNode)`.
Total length of this collection is the `number of nodes * NrAdjacenciesPerNode`.
Every index is either a node or InvalidNode.

```
public NativeList<int> NodeAdjacencies
```

Field Value

NativeList<int>

Examples

If `NrAdjacenciesPerNode=3`, the adjacencies for node `0` are at the indices 0, 1 and 2 (the range is `[0*3=0, 0*3+3=3)`).

If `NrAdjacenciesPerNode=2`, the adjacencies for node `4` are at the indices 8 and 9 (the range is `[42=8, 42+2=10)`).

Remarks

Adjacencies are always bidirectional, so if a node has an adjacency to another node, the second node must also have an adjacency to the first node.
This object owns this native memory and frees it in Dispose().

See Also
NrAdjacenciesPerNode

NodeData

Data of each node.
The meaning of this data changes depending on the type of the node.

- portals: data is the MissionNodeType.
- rooms: data is the identifier the node's RoomDataCollectionSO is registered with in the DBPRegistry.
- doors: data is the identifier the node's DoorDataCollectionSO is registered with in the DBPRegistry.

```
public NativeList<int> NodeData
```

Field Value

NativeList<int>

Remarks

This array is in parallel with NodeTypes and NodeNames, meaning the arrays contain information about the same node at a given index.
This object owns this native memory and frees it in Dispose().

See Also
NodeTypes, NodeNames, MissionNodeType, RoomDataCollectionSO, DoorDataCollectionSO, DBPRegistry

NodeNames

Name of each node.
The name is carried all the way into the spawned dungeon and will then be queryable with GetRoomByName(FixedString64Bytes, int) and GetDoorByName(FixedString64Bytes, int).

```
public NativeList<FixedString64Bytes> NodeNames
```

Field Value

NativeList<FixedString64Bytes>

Remarks

This array is in parallel with NodeTypes and NodeData, meaning the arrays contain information about the same node at a given index.
This object owns this native memory and frees it in Dispose().

See Also
NodeTypes, NodeData

NodeTypes

Type of each node.

```
public NativeList<MissionNodeType> NodeTypes
```

Field Value

NativeList<MissionNodeType>

Remarks

This array is in parallel with NodeNames and NodeData, meaning the arrays contain information about the same node at a given index.
This object owns this native memory and frees it in Dispose().

See Also
NodeNames, NodeData

NrAdjacenciesPerNode

Maximum number of adjacencies any node in the graph has.
Used to determine how many slots are reserved for each node in the adjacency collection.

```
public int NrAdjacenciesPerNode
```

Field Value

int

Properties

IsCreated

Checks whether this instance currently holds the necessary memory to be valid.
This is a necessary condition for a valid instance, but not a sufficient one.
This means that if this is false, it's definitely not a valid instance to run logic on, but if this is true, that is not a guarantee that it is a valid instance.

```
public readonly bool IsCreated { get; }
```

Property Value

bool

Methods

AsReadOnly()

Creates a new read-only wrapper and returns it.

```
public ResolvedMissionGraph.ReadOnly AsReadOnly()
```

Returns

ResolvedMissionGraph.ReadOnly

Read-only view of this instance.

See Also
ResolvedMissionGraph.ReadOnly

CreateAlloc(int, int, ref ResolvedMissionGraph)

Creates a new resolved mission graph instance and allocates all native collections for the given number of nodes and adjacencies.

```
[BurstCompile]
public static void CreateAlloc(int estimatedNrNodes, int
estimatedNrAdjacenciesPerNode, ref ResolvedMissionGraph resolvedMissionGraph)
```

Parameters

`estimatedNrNodes` int

Number of nodes the resolved mission graph is estimated to have.

`estimatedNrAdjacenciesPerNode` int

Number of adjacencies per node the resolved mission graph is estimated to have.

`resolvedMissionGraph` ResolvedMissionGraph

Output parameter that will contain the created ResolvedMissionGraph instance.

Remarks

This allocates native memory with Unity.Collections.Allocator.Persistent. You must call Dispose() to free it.

Dispose()

Frees native memory. You must not continue using this instance after calling this.

```
public void Dispose()
```

Struct ResolvedMissionGraph.ReadOnly

Namespace: PWS.DungeonBlueprint.Mission

Read-only wrapper around a resolved mission graph instance.
This is only a view over the existing memory and not a deep copy.

```
public readonly struct ResolvedMissionGraph.ReadOnly
```

Inherited Members
ValueType.Equals(object) , ValueType.GetHashCode() , ValueType.ToString() ,
object.Equals(object, object) , object.GetType() ,
object.ReferenceEquals(object, object)

Constructors

ReadOnly(ref ResolvedMissionGraph)

Creates a new read-only wrapper instance around the given resolved mission graph.

```
public ReadOnly(ref ResolvedMissionGraph original)
```

Parameters

`original` ResolvedMissionGraph

Resolve mission graph to wrap.

Properties

IsCreated

Mirrors IsCreated of the instance the readonly instance was created with.

```
public bool IsCreated { get; }
```

Property Value

bool

NodeAdjacencies

Read-only view on NodeAdjacencies.

```
public NativeArray<int>.ReadOnly NodeAdjacencies { get; }
```

Property Value

NativeArray<int>.ReadOnly

NodeData

Read-only view on NodeData.

```
public NativeArray<int>.ReadOnly NodeData { get; }
```

Property Value

NativeArray<int>.ReadOnly

NodeNames

Read-only view on NodeNames.

```
public NativeArray<FixedString64Bytes>.ReadOnly NodeNames { get; }
```

Property Value

NativeArray<FixedString64Bytes>.ReadOnly

NodeTypes

Read-only view on NodeTypes.

```
public NativeArray<MissionNodeType>.ReadOnly NodeTypes { get; }
```

Property Value

NativeArray<MissionNodeType>.ReadOnly

NrAdjacenciesPerNode

Read-only view on NrAdjacenciesPerNode.

```
public int NrAdjacenciesPerNode { get; }
```

Property Value

int

Namespace PWS.DungeonBlueprint.Pipeline

Classes

DBPPipeline

Dungeon generation pipeline. This is one of the main components of Dungeon Blueprint. Pipeline takes a MissionGraph and a few additional settings and runs through various built-in transformation steps resulting in a RuntimeGraph, which can then be spawned. All work done in the pipeline is independent of the Unity scene, and you can run the entire pipeline without affecting the scene.
The interaction with the scene is handled by DBPWorld.

Structs

EntryRoomInfo

Container for all information the DBPPipeline needs regarding the entry room.

PipelineData

Container for each graph structure used and created during in a DBPPipeline run.

PipelineState

Container for various state used in a DBPPipeline run.

Interfaces

IDBPPipelineStep

Interface defining a step of a run of DBPPipeline. All steps the pipeline runs must implement this interface.

Enums

DBPPipelineStage

Stages of the pipeline. The pipeline will go through each stage during a run in the order defined in the enum.
A stage may have multiple steps registered in the pipeline.
There is a built-in step for each stage which builds the graph data structures. Therefore, the stage of the pipeline implies which data structures already exist.

Class DBPPipeline

Namespace: PWS.DungeonBlueprint.Pipeline

Dungeon generation pipeline. This is one of the main components of Dungeon Blueprint.
Pipeline takes a MissionGraph and a few additional settings and runs through various built-in transformation steps resulting in a RuntimeGraph, which can then be spawned.
All work done in the pipeline is independent of the Unity scene, and you can run the entire pipeline without affecting the scene.
The interaction with the scene is handled by DBPWorld.

```
public class DBPPipeline : IDisposable
```

Inheritance

object ← DBPPipeline

Implements

IDisposable

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Examples

Generate a dungeon at runtime

```
var pipeline = new DBPPipeline(registry);
pipeline.Setup(mission, seed, entryInfo, settings);
pipeline.WantsToRun = true;

while(pipeline.Tick()) { /* wait */ }

if(pipeline.IsValid) { /* pipeline.Runtime has a valid runtime graph instance. */ }
```

Remarks

Additionally, you may register custom steps with the pipeline to modify the graph transformation.

Constructors

DBPPipeline(DBPRegistry)

Creates a new instance of DBPPipeline.

```
public DBPPipeline(DBPRegistry registry)
```

Parameters

`registry` DBPRegistry

Registry your rooms and doors are registered with.

See Also
DBPRegistry

Properties

Construction

Construction created during the pipeline run in stage Construction.

```
public ConstructionGraph.ReadOnly Construction { get; }
```

Property Value

ConstructionGraph.ReadOnly

Remarks

Instance is disposed when the pipeline is disposed, reset or set up again. The result is that you can not use this instance and rerun the pipeline in the meantime.

ConstructionUsedLoopIterations

How many loop iterations the construction conversion took.
Not a valid value until the built-in construction step was completed.

```
public ulong ConstructionUsedLoopIterations { get; }
```

Property Value

ulong

See Also
ConstructionMaxLoopIterations

IsFinished

Whether the pipeline has finished successfully.
This is the case when the pipeline was initialized and all steps have executed successfully.

```
public bool IsFinished { get; }
```

Property Value

bool

IsValid

Whether the pipeline run has been successful so far.
This is true if the pipeline has been initialized and all steps executed so far returned true in Complete(ref PipelineData, ref PipelineState).

```
public bool IsValid { get; }
```

Property Value

bool

Mission

Mission this pipeline is running on. This is passed to the pipeline in Setup(MissionGraph, uint, EntryRoomInfo, DBPPipelineSettings, object).

```
public MissionGraph.ReadOnly Mission { get; }
```

Property Value

MissionGraph.ReadOnly

ResolvedMission

Resolved mission created during the pipeline run in stage ResolveMission.

```
public ResolvedMissionGraph.ReadOnly ResolvedMission { get; }
```

Property Value

ResolvedMissionGraph.ReadOnly

Remarks

Instance is disposed when the pipeline is disposed, reset or set up again. The result is that you can not use this instance and rerun the pipeline in the meantime.

Runtime

Runtime created during the pipeline run in stage Runtime.

```
public RuntimeGraph.ReadOnly Runtime { get; }
```

Property Value

RuntimeGraph.ReadOnly

Remarks

Instance is disposed when the pipeline is disposed, reset or set up again. The result is that you can not use this instance and rerun the pipeline in the meantime.
Since the runtime graph is required for use in DBPWorld, there is particular

caution needed.
If you want to keep the world usable while rerunning the pipeline, you may want to create a full copy of the memory using DeepCopyAlloc(out RuntimeGraph).

Stage

Current stage of the pipeline.
None when the pipeline is not initialized.
Finished when the pipeline has run through all its steps.
Otherwise, Stage of the current step.

```
public DBPPipelineStage Stage { get; }
```

Property Value

DBPPipelineStage

Topology

Topology created during the pipeline run in stage Topology.

```
public TopologyGraph.ReadOnly Topology { get; }
```

Property Value

TopologyGraph.ReadOnly

Remarks

Instance is disposed when the pipeline is disposed, reset or set up again. The result is that you can not use this instance and rerun the pipeline in the meantime.

WantsToRun

Whether the pipeline wants to run. This is set to true only by user code.
The pipeline may set this to false when a step fails or the pipeline finished all steps.

Changing this from true to false invokes OnStopRunning.

```
public bool WantsToRun { get; set; }
```

Property Value

bool

Methods

Dispose()

Frees native memory. You must not continue using this instance after calling this.

```
public void Dispose()
```

RegisterStep(IDBPPipelineStep)

Adds a new step to the pipeline. The order in the pipeline is dependent on the implementation of the pipeline step interface.
You must not add pipeline steps while the pipeline is running. If you want to add steps to a pipeline that is running, either let the pipeline finish first or call Reset() on it.

```
public void RegisterStep(IDBPPipelineStep step)
```

Parameters

`step` IDBPPipelineStep

Step to add to the pipeline.

Exceptions

ArgumentNullException

If `step` is null.

InvalidOperationException

If the stage of `step` is None or Finished.

InvalidOperationException

If the pipeline is in the middle of a run.

See Also
IDBPPipelineStep

Reset()

Resets the pipeline to its original state.
Pipeline can not run after it has been reset until you call Setup(MissionGraph, uint, EntryRoomInfo, DBPPipelineSettings, object) again.

```
public void Reset()
```

RestartWithSeed(uint)

Restarts the pipeline with a different seed for random number generation.

```
public void RestartWithSeed(uint seed)
```

Parameters

`seed` uint

New seed for random number generation.

Remarks

Calling this is equivalent to calling Setup(MissionGraph, uint, EntryRoomInfo, DBPPipelineSettings, object) with the same parameters as before except for the different seed.

Setup(MissionGraph, uint, EntryRoomInfo, DBPPipelineSettings, object)

Sets up the pipeline for a run. You must call this before the pipeline can run.
Calling this resets the pipeline so you can alternate between setting up the pipeline and running it without having to call Reset() manually.

```
public void Setup(MissionGraph missionGraph, uint seed, EntryRoomInfo entryRoomInfo,
DBPPipelineSettings settings, object customStateData = null)
```

Parameters

`missionGraph` MissionGraph

Mission the pipeline will base its run on. See Mission.

`seed` uint

Seed used by the random number generator for dungeon generation.

`entryRoomInfo` EntryRoomInfo

Information about the entry room.

`settings` DBPPipelineSettings

Additional settings used to set up a pipeline run.

`customStateData` object

Additional state data passed to pipeline steps. See UserData.

Remarks

Once the pipeline has been set up, you may call RestartWithSeed(uint) instead of this. However, you must call this for the first run of the pipeline.

Exceptions

ArgumentException

If `missionGraph` is invalid.

InvalidOperationException

If `entryRoomInfo` or `settings` are invalid.

Tick()

Runs a single tick of the pipeline.
If WantsToRun is false, this is a noop.
Also checks itself whether the pipeline is initialized and not yet finished.
A tick consists of either scheduling or completing the current pipeline step or counting the current step's tick if the asynchronous work is not yet finished and the configured maximum number of ticks is not reached yet.

```
public bool Tick()
```

Returns

bool

True if the pipeline has more work to do (and so you should continue calling this method to finish it), false otherwise. Therefore, false could mean the pipeline has failed or that it has finished successfully.

Remarks

Tick rate is determined by your code and may be both fixed or variable.

UnregisterStep(IDBPPipelineStep)

Removes a step from the pipeline. You must not remove pipeline steps while the pipeline is running. If you want to remove steps from a pipeline that is running, either let the pipeline finish first or call Reset() on it.

```
public void UnregisterStep(IDBPPipelineStep step)
```

Parameters

`step` IDBPPipelineStep

Step to remove from the pipeline.

Exceptions

ArgumentNullException

If `step` is null.

InvalidOperationException

If the pipeline is in the middle of a run.

InvalidOperationException

If `step` is not in the pipeline.

Events

OnStopRunning

Event invoked when the pipeline stops its run.
There are multiple reasons why the pipeline might stop:

- a step failed
- all steps finished
- WantsToRun was set to false by user code
  The parameter of the delegate is the pipeline object which stopped its run.

```
public event Action<DBPPipeline> OnStopRunning
```

Event Type

Action<DBPPipeline>

Remarks

The event is not cleared when resetting this pipeline. It is reset when disposing it.

See Also

IDBPPipelineStep

Enum DBPPipelineStage

Namespace: PWS.DungeonBlueprint.Pipeline

Stages of the pipeline. The pipeline will go through each stage during a run in the order defined in the enum.
A stage may have multiple steps registered in the pipeline.
There is a built-in step for each stage which builds the graph data structures. Therefore, the stage of the pipeline implies which data structures already exist.

```
public enum DBPPipelineStage
```

Fields

`None = 0`

Default state. This is the stage of the pipeline when it is not initialized. The pipeline can not run in this stage.

`ResolveMission = 1`

Stage for mission graph resolution.

`Topology = 2`

Stage for topology conversion.

`Construction = 3`

Stage for construction conversion.

`Runtime = 4`

Stage for runtime conversion.

`Finished = 5`

End state. This is the stage of the pipeline when it has finished its run. The pipeline can not run further in this stage. Call Reset() or Setup(MissionGraph, uint, EntryRoomInfo, DBPPipelineSettings, object) to reuse it.

Remarks

Stages None and Finished represent the start and end stages of the pipeline. No steps can be registered for those stages.

Struct EntryRoomInfo

Namespace: PWS.DungeonBlueprint.Pipeline

Container for all information the DBPPipeline needs regarding the entry room.

```
[Serializable]
public struct EntryRoomInfo
```

Inherited Members
ValueType.Equals(object) , ValueType.GetHashCode() , ValueType.ToString() , object.Equals(object, object) , object.GetType() , object.ReferenceEquals(object, object)

Constructors

EntryRoomInfo(string, GridPosition, GridRotation)

Creates a new instance of EntryRoomInfo.

```
public EntryRoomInfo(string entryPortalName, GridPosition position,
GridRotation rotation)
```

Parameters

`entryPortalName` string

EntryPortalName

`position` GridPosition

Position

`rotation` GridRotation

Rotation

Properties

EntryPortalName

Name of the entry portal the entry room is connected to the in the mission graph. Required to identify the room you want to be the entry room.

```
public readonly string EntryPortalName { get; }
```

Property Value

string

Position

Position in the grid the entry room will be placed at.

```
public readonly GridPosition Position { get; }
```

Property Value

GridPosition

Rotation

Rotation the entry room will be placed with.

```
public readonly GridRotation Rotation { get; }
```

Property Value

GridRotation

Methods

CheckForErrors(GridSize, out string)

Checks the info for errors.

```
public bool CheckForErrors(GridSize gridSize, out string errorMessage)
```

Parameters

`gridSize` GridSize

Size of the grid you want to generate the dungeon with. This is needed to check if Position is inside the grid.

`errorMessage` string

A human-readable error message. `null` if return value is true.

Returns

bool

True if the info is valid, false otherwise.

Interface IDBPPipelineStep

Namespace: PWS.DungeonBlueprint.Pipeline

Interface defining a step of a run of DBPPipeline. All steps the pipeline runs must implement this interface.

```
public interface IDBPPipelineStep
```

Remarks

Implement this interface to create a custom pipeline step and register it with the pipeline using RegisterStep(IDBPPipelineStep).

Properties

IsCompleted

Whether this step has been completed.
Typically initialized to false, set to true when Complete(ref PipelineData, ref PipelineState) is called and set back to false when Reset() is called.

```
bool IsCompleted { get; }
```

Property Value

bool

Remarks

This is used to check whether the next action to take is completing the step or continue to the next step of the pipeline. You have control over setting this so you can delay completion. The pipeline will not continue to the next step until this is set to true.

IsCurrentJobCompleted

Whether the async operation inside this step is completed.
This is typically connected to Unity.Jobs.JobHandle.IsCompleted using the job handle

created during scheduling of the step, but it can represent the completion state of any asynchronous operation you are running.

```
bool IsCurrentJobCompleted { get; }
```

Property Value

bool

Remarks

If IsScheduled and this return true, the pipeline will call Complete(ref PipelineData, ref PipelineState) on the step in its tick.
This allows the pipeline to finish the step before it has run the configured maximum number of ticks for this stage.
Once the maximum number of ticks has been reached, the pipeline will call Complete(ref PipelineData, ref PipelineState) even if this property is false.
If your step runs synchronously, you probably want this to always return true.

IsScheduled

Whether this step has been scheduled.
Typically initialized to false, set to true when Schedule(ref PipelineData, ref PipelineState) is called and set back to false when Reset() is called.

```
bool IsScheduled { get; }
```

Property Value

bool

Remarks

This is used to check whether the next action to take on this step is scheduling or completing. You have control over setting this so you can delay scheduling. The step will not complete until this is set to true. This also means that resources you may have allocated in a call to Schedule(ref PipelineData, ref PipelineState) and want to free in Complete(ref PipelineData, ref PipelineState) will only actually be freed if this returns true.

Order

Ordering of this pipeline step. This determines the order of the steps within the same stage. Lower order number comes first. If two steps have the same order number, whichever is added to the pipeline first will be executed first.

```
uint Order { get; }
```

Property Value

uint

Examples

If you have 2 custom pipeline steps with the stage Topology, one with order 2 and one with order 11, the pipeline stage topology will have the steps ordered as 1. built-in PWS. DungeonBlueprint.Topology.TopologyStep, 2. custom step with order 2, 3. custom step with order 11.

Remarks

All built-in steps have order number 0 and are added to the pipeline immediately on object creation.
Since this is an uint, you can not have a custom pipeline step with an order number below 0, so you can not have a pipeline step execute before the built-in step of a stage.
If you want a custom pipeline step to be executed right before a built-in step, insert it to the prior stage with a high order number (possibly MaxValue)

Stage

Stage of this pipeline step. This determines when the pipeline executes this step.
Order is the same as in DBPPipelineStage.
None and Finished are not valid stages for pipeline steps.

```
DBPPipelineStage Stage { get; }
```

Property Value

DBPPipelineStage

Examples

A custom pipeline step with the stage Topology will execute after the built-in PWS.Dungeon Blueprint.Topology.TopologyStep and will therefore have access to the TopologyGraph in Schedule(ref PipelineData, ref PipelineState) and Complete(ref PipelineData, ref Pipeline State).
While the ResolvedMissionGraph will also be available, you should likely not modify it, since changes will not be carried over into the TopologyGraph at this point.

Remarks

A custom pipeline step is always executed after the built-in step that corresponds to the pipeline stage.
Since the built-in steps build the graph data structures of the pipeline, the stage of a custom pipeline step determine which data structures it will have access to.

Methods

Complete(ref PipelineData, ref PipelineState)

Completes the work for this step.
If the step runs asynchronously, this should block until it is completed. If the step runs synchronously, this should do the bulk of the work.

```
bool Complete(ref PipelineData data, ref PipelineState state)
```

Parameters

`data` PipelineData

Graph data structures of the pipeline containing the results of the pipeline stages. You have read and write access.

`state` PipelineState

Transient state of the pipeline run. You have read and write access.

Returns

bool

Whether the step was successful.
If this is true, the pipeline will continue to the next step.
If this is false, the pipeline will stop running.

Remarks

If the step was fully completed, this should typically make IsCompleted return true.

Reset()

Reset the pipeline step.
This should reset all state of the pipeline step and return it to a state that allows scheduling the step again, with different data, even if it was completed previously.

```
void Reset()
```

Schedule(ref PipelineData, ref PipelineState)

Schedule the work for this step.
If the step runs asynchronously, this should do everything needed to start the asynchronous work. If the step runs synchronously, you may do any preparation you need here, but do the bulk of the work in Complete(ref PipelineData, ref PipelineState).

```
void Schedule(ref PipelineData data, ref PipelineState state)
```

Parameters

`data` PipelineData

Graph data structures of the pipeline containing the results of the pipeline stages. You have read and write access.

`state` PipelineState

Transient state of the pipeline run. You have read and write access.

Remarks

When a step is scheduled, it is guaranteed that Complete(ref PipelineData, ref PipelineState) will be called. However, the pipeline will check IsScheduled to see if a step was scheduled, and that property is in your control. This means that if you need Complete(ref PipelineData, ref PipelineState) to be called (for example if you allocated native memory and need to free it), you must make IsScheduled return true.

Struct PipelineData

Namespace: PWS.DungeonBlueprint.Pipeline

Container for each graph structure used and created during in a DBPPipeline run.

```
public struct PipelineData : IDisposable
```

Implements
IDisposable

Inherited Members
ValueType.Equals(object) , ValueType.GetHashCode() , ValueType.ToString() , object.Equals(object, object) , object.GetType() , object.ReferenceEquals(object, object)

Remarks

This object is created and disposed of inside the pipeline and passed to pipeline steps with read and write access.
If you are not implementing a custom IDBPPipelineStep, you likely do not need to care about this object.

Fields

Construction

Created in PWS.DungeonBlueprint.Construction.ConstructionStep.

```
public ConstructionGraph Construction
```

Field Value

ConstructionGraph

Remarks

Data object owns this native memory, and it is freed when Dispose() is called.

Mission

Original mission graph the pipeline run is based on.

```
public MissionGraph Mission
```

Field Value

MissionGraph

Remarks

Data object does **not** own the native memory of the mission graph, and it is not disposed when Dispose() is called.

ResolvedMission

Created in PWS.DungeonBlueprint.Mission.ResolveMissionStep.

```
public ResolvedMissionGraph ResolvedMission
```

Field Value

ResolvedMissionGraph

Remarks

Data object owns this native memory, and it is freed when Dispose() is called.

Runtime

Created in PWS.DungeonBlueprint.Runtime.RuntimeStep.

```
public RuntimeGraph Runtime
```

Field Value

RuntimeGraph

Remarks

Data object owns this native memory, and it is freed when Dispose() is called.

Topology

Created in PWS.DungeonBlueprint.Topology.TopologyStep.

```
public TopologyGraph Topology
```

Field Value

TopologyGraph

Remarks

Data object owns this native memory, and it is freed when Dispose() is called.

Methods

Dispose()

Frees native memory. You must not continue using this instance after calling this.

```
public void Dispose()
```

Struct PipelineState

Namespace: PWS.DungeonBlueprint.Pipeline

Container for various state used in a DBPPipeline run.

```
public struct PipelineState : IDisposable
```

Implements
IDisposable

Inherited Members
ValueType.Equals(object) , ValueType.GetHashCode() , ValueType.ToString() , object.Equals(object, object) , object.GetType() , object.ReferenceEquals(object, object)

Remarks

This object is created and disposed of inside the pipeline and passed to pipeline steps with read and write access.
If you are not implementing a custom IDBPPipelineStep, you likely do not need to care about this object.

Fields

IsValidRef

Native reference to the overall validity state of the pipeline.
Can be read and set from jobs which allows aborting already running jobs.

```
public NativeReference<bool> IsValidRef
```

Field Value

NativeReference<bool>

Remarks

State object owns this native memory, and it is freed when Dispose() is called.

Random

Random number generator used for dungeon generation.

```
public Random Random
```

Field Value

Random

UserData

Custom data object to store additional data in the pipeline state.
You may use this to pass data to your custom pipeline steps.
This object is not used by Dungeon Blueprint.

```
public object UserData
```

Field Value

object

Methods

Dispose()

Frees native memory. You must not continue using this instance after calling this.

```
public void Dispose()
```

Namespace PWS.DungeonBlueprint.Runtime

Classes

BakedRuntimeGraphsAsset

Represents a variable number of serialized runtime graphs all generated from the same mission graph asset.
Serialization is editor-only, while deserialization is available both in editor and runtime.

Structs

RuntimeGraph

Main data structure for the Runtime phase of the transformation pipeline.
This is effectively the output of the pipeline, which the final dungeon can be constructed from.
This instance can also be serialized into an asset.

RuntimeGraph.ReadOnly

Read-only wrapper around a runtime graph instance.
This is only a view over the existing memory and not a deep copy.

Class BakedRuntimeGraphsAsset

Namespace: PWS.DungeonBlueprint.Runtime

Represents a variable number of serialized runtime graphs all generated from the same mission graph asset.
Serialization is editor-only, while deserialization is available both in editor and runtime.

```
[CreateAssetMenu(fileName = "New Baked Runtime Graphs", menuName = "Dungeon
Blueprint/Baked Runtime Graphs", order = 2)]
public class BakedRuntimeGraphsAsset : ScriptableObject
```

Inheritance

object ← Object ← ScriptableObject ← BakedRuntimeGraphsAsset

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Properties

Mission

Mission graph that all serialized runtime graphs in this asset are based on.

```
public MissionGraphAsset Mission { get; }
```

Property Value

MissionGraphAsset

Name

Name of the baked runtime graphs.

```
public string Name { get; }
```

Property Value

string

Remarks

This is used for organizational purposes only. It has no effect on dungeon generation.

NrSerializedGraphs

Number of serialized runtime graphs in this asset.

```
public int NrSerializedGraphs { get; }
```

Property Value

int

SeedList

List of all seeds of the serialized runtime graphs in this asset.
Used in combination with GetRuntimeGraphBySeedAlloc(uint, DBPRegistry, out RuntimeGraph).

```
public IReadOnlyList<uint> SeedList { get; }
```

Property Value

IReadOnlyList<uint>

Remarks

First call to this method may be slower than subsequent calls because of a lazily calculated seed-to-graph mapping.
C# allows casting this to List<T> because of how IReadOnlyList<T> works. However, you should never modify this list yourself.

Methods

GetRandomRuntimeGraphAlloc(DBPRegistry, out RuntimeGraph)

Gets any of the runtime graphs that are serialized in this asset.

```
public void GetRandomRuntimeGraphAlloc(DBPRegistry registry, out
RuntimeGraph runtimeGraph)
```

Parameters

`registry` DBPRegistry

Registry that all rooms and doors of the runtime graph will be registered with when this method returns.

`runtimeGraph` RuntimeGraph

Output parameter that will contain the deserialized runtime graph.

Remarks

This allocates native memory with Unity.Collections.Allocator.Persistent. You must call Dispose() to free it.

Exceptions

InvalidOperationException

If no graphs are serialized in this asset.

GetRuntimeGraphByIndexAlloc(int, DBPRegistry, out RuntimeGraph)

Gets the runtime graph serialized in this asset with the given index.
Use NrSerializedGraphs to check how many indices are valid.

```
public void GetRuntimeGraphByIndexAlloc(int index, DBPRegistry registry, out
RuntimeGraph runtimeGraph)
```

Parameters

`index` int

Index of the runtime graph.

`registry` DBPRegistry

Registry that all rooms and doors of the runtime graph will be registered with when this method returns.

`runtimeGraph` RuntimeGraph

Output parameter that will contain the deserialized runtime graph.

Remarks

This allocates native memory with Unity.Collections.Allocator.Persistent. You must call Dispose() to free it.

Exceptions

IndexOutOfRangeException

If `index` is not a valid index for a runtime graph in this asset.

GetRuntimeGraphBySeedAlloc(uint, DBPRegistry, out RuntimeGraph)

Gets the runtime graph serialized in this asset with the given seed.
Use SeedList to check which seeds are valid.

```
public void GetRuntimeGraphBySeedAlloc(uint seed, DBPRegistry registry, out
RuntimeGraph runtimeGraph)
```

Parameters

`seed` uint

Seed of the runtime graph.

`registry` DBPRegistry

Registry that all rooms and doors of the runtime graph will be registered with when this method returns.

`runtimeGraph` RuntimeGraph

Output parameter that will contain the deserialized runtime graph.

Remarks

The first call to this method may be slower than subsequent calls because of a lazily calculated seed-to-graph mapping.
This allocates native memory with Unity.Collections.Allocator.Persistent. You must call Dispose() to free it.

Exceptions

KeyNotFoundException

If `seed` is not a valid seed for a runtime graph in this asset.

Struct RuntimeGraph

Namespace: PWS.DungeonBlueprint.Runtime

Main data structure for the Runtime phase of the transformation pipeline.
This is effectively the output of the pipeline, which the final dungeon can be constructed from.
This instance can also be serialized into an asset.

```
[BurstCompile]
public struct RuntimeGraph : IDisposable
```

Implements
IDisposable

Inherited Members
ValueType.Equals(object) , ValueType.GetHashCode() , ValueType.ToString() , object.Equals(object, object) , object.GetType() , object.ReferenceEquals(object, object)

Fields

AdjacencyOccupiedConnectionIdx

Connection index of the node's room data an adjacency occupies in the dungeon.
Required to know which connections are used by doors and which need to spawn block doors.

```
public NativeArray<int> AdjacencyOccupiedConnectionIdx
```

Field Value

NativeArray<int>

Remarks

This object owns this native memory and frees it in Dispose().
This array is in parallel with NodeAdjacencies and AdjacencySlotToEdge meaning the arrays contain information about the same adjacency at a given index.

AdjacencySlotToEdge

Mapping from adjacency to edge.
Every valid adjacency belongs to an edge. Since adjacencies are always bidirectional and thus take up two slots in NodeAdjacencies, an edge combines two adjacency slots into a single edge, helping to reduce data duplication.
Every index in the list is either an index for an edge collection or InvalidEdge.

```
public NativeArray<int> AdjacencySlotToEdge
```

Field Value

NativeArray<int>

Remarks

This object owns this native memory and frees it in Dispose().
This array is in parallel with NodeAdjacencies and AdjacencyOccupiedConnectionIdx meaning the arrays contain information about the same adjacency at a given index.

EdgeDoorDataIdx

Door identifier of each node.
The identifier stems from the DBPRegistry.

```
public NativeArray<int> EdgeDoorDataIdx
```

Field Value

NativeArray<int>

Remarks

This array is in parallel with EdgeNames, NodePositions and NodeRotations meaning the arrays contain information about the same node at a given index.
This object owns this native memory and frees it in Dispose().

See Also
NodeNames, NodePositions, NodeRotations

EdgeDoorDirections

Direction of each edge.
Determines where and with what rotation each door will be spawned in the final dungeon.

```
public NativeArray<GridDirection> EdgeDoorDirections
```

Field Value

NativeArray<GridDirection>

Remarks

This array is in parallel with EdgeNames, EdgeDoorDataIdx and EdgeDoorPositions meaning the arrays contain information about the same edge at a given index.
This object owns this native memory and frees it in Dispose().

See Also
EdgeNames, EdgeDoorDataIdx, EdgeDoorPositions

EdgeDoorPositions

Position of each edge.
This is the global grid position, not a position local to a room.
This position determines where each door will be spawned in the final dungeon.

```
public NativeArray<GridPosition> EdgeDoorPositions
```

Field Value

NativeArray<GridPosition>

Remarks

This array is in parallel with EdgeNames, EdgeDoorDataIdx and EdgeDoorDirections meaning the arrays contain information about the same edge at a given index.
This object owns this native memory and frees it in Dispose().

See Also
EdgeNames, EdgeDoorDataIdx, EdgeDoorDirections

EdgeNames

Name of each edge.
The name is carried all the way into the spawned dungeon and will then be queryable with GetDoorByName(FixedString64Bytes, int).

```
public NativeArray<FixedString64Bytes> EdgeNames
```

Field Value

NativeArray<FixedString64Bytes>

Remarks

This array is in parallel with EdgeDoorDataIdx, EdgeDoorPositions and EdgeDoorDirections meaning the arrays contain information about the same edge at a given index.
This object owns this native memory and frees it in Dispose().

See Also
EdgeDoorDataIdx, EdgeDoorPositions, EdgeDoorDirections

NodeAdjacencies

Adjacencies between nodes.
Adjacencies of a given node `n`, are in the range `[n*NrAdjacenciesPerNode, n*NrAdjacenciesPerNode+NrAdjacenciesPerNode)`.
Total length of this collection is the `number of nodes * NrAdjacenciesPerNode`.
Every index is either a node or InvalidNode.

```
public NativeArray<int> NodeAdjacencies
```

Field Value

NativeArray<int>

Examples

If `NrAdjacenciesPerNode=3`, the adjacencies for node `0` are at the indices 0, 1 and 2 (the range is `[0*3=0, 0*3+3=3)`).

If `NrAdjacenciesPerNode=2`, the adjacencies for node `4` are at the indices 8 and 9 (the range is `[42=8, 42+2=10)`).

Remarks

Adjacencies are always bidirectional, so if a node has an adjacency to another node, the second node must also have an adjacency to the first node.
This object owns this native memory and frees it in Dispose().
This array is in parallel with AdjacencySlotToEdge and AdjacencyOccupiedConnectionIdx, meaning the arrays contain information about the same adjacency at a given index.

See Also
NrAdjacenciesPerNode

NodeEntryPortalNames

Additional data about some nodes.
This data only exists for nodes that were connected to a portal node with portal type Entrance in the mission graph before the topology conversion.
Portals do not exist past the mission graph stage, but we keep track of the names so we can query them later.

```
public NativeHashMap<int, FixedString64Bytes> NodeEntryPortalNames
```

Field Value

NativeHashMap<int, FixedString64Bytes>

Remarks

This object owns this native memory and frees it in Dispose().

See Also
NodeExitPortalNames

NodeExitPortalNames

Additional data about some nodes.
This data only exists for nodes that were connected to a portal node with portal type Exit in the mission graph before the topology conversion.

Portals do not exist past the mission graph stage, but we keep track of the names so we can query them later.

```
public NativeHashMap<int, FixedString64Bytes> NodeExitPortalNames
```

Field Value

NativeHashMap<int, FixedString64Bytes>

Remarks

This object owns this native memory and frees it in Dispose().

See Also
NodeEntryPortalNames

NodeNames

Name of each node.
The name is carried all the way into the spawned dungeon and will then be queryable with GetRoomByName(FixedString64Bytes, int).

```
public NativeArray<FixedString64Bytes> NodeNames
```

Field Value

NativeArray<FixedString64Bytes>

Remarks

This array is in parallel with NodeRoomDataIdx, NodePositions and NodeRotations meaning the arrays contain information about the same node at a given index.
This object owns this native memory and frees it in Dispose().

See Also
NodeRoomDataIdx, NodePositions, NodeRotations

NodePositions

Grid position of each node.
This position determines where each room will be spawned in the final dungeon.

```
public NativeArray<GridPosition> NodePositions
```

Field Value

NativeArray<GridPosition>

Remarks

This array is in parallel with NodeNames, NodeRoomDataIdx and NodeRotations meaning the arrays contain information about the same node at a given index.
This object owns this native memory and frees it in Dispose().

See Also
NodeNames, NodeRoomDataIdx, NodeRotations

NodeRoomDataIdx

Room identifier of each node.
The identifier stems from the DBPRegistry.

```
public NativeArray<int> NodeRoomDataIdx
```

Field Value

NativeArray<int>

Remarks

This array is in parallel with NodeNames, NodePositions and NodeRotations meaning the arrays contain information about the same node at a given index.
This object owns this native memory and frees it in Dispose().

See Also
NodeNames, NodePositions, NodeRotations

NodeRotations

Grid rotation of each node.
This rotation determines how each room will be spawned in the final dungeon.

```
public NativeArray<GridRotation> NodeRotations
```

Field Value

NativeArray<GridRotation>

Remarks

This array is in parallel with NodeNames, NodeRoomDataIdx and NodePositions meaning the arrays contain information about the same node at a given index.
This object owns this native memory and frees it in Dispose().

See Also
NodeNames, NodeRoomDataIdx, NodePositions

NrAdjacenciesPerNode

Maximum number of adjacencies any node in the graph has.
This is used to determine how many slots are reserved for each node in the adjacency collection.

```
public int NrAdjacenciesPerNode
```

Field Value

int

Properties

IsCreated

Checks whether this instance currently holds the necessary memory to be valid.
This is a necessary condition for a valid instance, but not a sufficient one.
This means that if this is false, it's definitely not a valid instance to run logic on, but if this is true, that is not a guarantee that it is a valid instance.

```
public readonly bool IsCreated { get; }
```

Property Value

bool

Methods

AsReadOnly()

Creates a new read-only wrapper and returns it.

```
public RuntimeGraph.ReadOnly AsReadOnly()
```

Returns

RuntimeGraph.ReadOnly

Read-only view of this instance.

See Also
RuntimeGraph.ReadOnly

CreateAlloc(int, int, int, out RuntimeGraph)

Creates a new runtime graph instance and allocates all native collections for the given number of nodes and adjacencies.

```
[BurstCompile]
public static void CreateAlloc(int nrNodes, int nrAdjacenciesPerNode, int nrEdges,
out RuntimeGraph runtimeGraph)
```

Parameters

`nrNodes` int

Number of nodes the runtime graph will have.

`nrAdjacenciesPerNode` int

Number of adjacencies per node the runtime graph will have.

`nrEdges` int

Number of edges the runtime graph will have.

`runtimeGraph` RuntimeGraph

Output parameter that will contain the created RuntimeGraph instance.

Remarks

This allocates native memory with Unity.Collections.Allocator.Persistent. You must call Dispose() to free it.

Dispose()

Frees native memory. You must not continue using this instance after calling this.

```
public void Dispose()
```

See Also

BakedRuntimeGraphsAsset

Struct RuntimeGraph.ReadOnly

Namespace: PWS.DungeonBlueprint.Runtime

Read-only wrapper around a runtime graph instance.
This is only a view over the existing memory and not a deep copy.

```
public readonly struct RuntimeGraph.ReadOnly
```

Inherited Members
ValueType.Equals(object) , ValueType.GetHashCode() , ValueType.ToString() , object.Equals(object, object) , object.GetType() , object.ReferenceEquals(object, object)

Constructors

ReadOnly(ref RuntimeGraph)

Creates a new read-only wrapper instance around the given runtime graph.

```
public ReadOnly(ref RuntimeGraph original)
```

Parameters

`original` RuntimeGraph

Runtime graph to wrap.

Properties

AdjacencyOccupiedConnectionIdx

Read-only view on AdjacencyOccupiedConnectionIdx.

```
public NativeArray<int>.ReadOnly AdjacencyOccupiedConnectionIdx { get; }
```

Property Value

NativeArray<int>.ReadOnly

AdjacencySlotToEdge

Read-only view on AdjacencySlotToEdge.

```
public NativeArray<int>.ReadOnly AdjacencySlotToEdge { get; }
```

Property Value

NativeArray<int>.ReadOnly

EdgeDoorDataIdx

Read-only view on EdgeDoorDataIdx.

```
public NativeArray<int>.ReadOnly EdgeDoorDataIdx { get; }
```

Property Value

NativeArray<int>.ReadOnly

EdgeDoorDirections

Read-only view on EdgeDoorDirections.

```
public NativeArray<GridDirection>.ReadOnly EdgeDoorDirections { get; }
```

Property Value

NativeArray<GridDirection>.ReadOnly

EdgeDoorPositions

Read-only view on EdgeDoorPositions.

```
public NativeArray<GridPosition>.ReadOnly EdgeDoorPositions { get; }
```

Property Value

NativeArray<GridPosition>.ReadOnly

EdgeNames

Read-only view on EdgeNames.

```
public NativeArray<FixedString64Bytes>.ReadOnly EdgeNames { get; }
```

Property Value

NativeArray<FixedString64Bytes>.ReadOnly

IsCreated

Mirrors IsCreated of the instance the readonly instance was created with.

```
public bool IsCreated { get; }
```

Property Value

bool

NodeAdjacencies

Read-only view on NodeAdjacencies.

```
public NativeArray<int>.ReadOnly NodeAdjacencies { get; }
```

Property Value

NativeArray<int>.ReadOnly

NodeEntryPortalNames

Read-only view on NodeEntryPortalNames.

```
public NativeHashMap<int, FixedString64Bytes>.ReadOnly NodeEntryPortalNames { get; }
```

Property Value

NativeHashMap<int, FixedString64Bytes>.ReadOnly

NodeExitPortalNames

Read-only view on NodeExitPortalNames.

```
public NativeHashMap<int, FixedString64Bytes>.ReadOnly NodeExitPortalNames { get; }
```

Property Value

NativeHashMap<int, FixedString64Bytes>.ReadOnly

NodeNames

Read-only view on NodeNames.

```
public NativeArray<FixedString64Bytes>.ReadOnly NodeNames { get; }
```

Property Value

NativeArray<FixedString64Bytes>.ReadOnly

NodePositions

Read-only view on NodePositions.

```
public NativeArray<GridPosition>.ReadOnly NodePositions { get; }
```

Property Value

NativeArray<GridPosition>.ReadOnly

NodeRoomDataIdx

Read-only view on NodeRoomDataIdx.

```
public NativeArray<int>.ReadOnly NodeRoomDataIdx { get; }
```

Property Value

NativeArray<int>.ReadOnly

NodeRotations

Read-only view on NodeRotations.

```
public NativeArray<GridRotation>.ReadOnly NodeRotations { get; }
```

Property Value

NativeArray<GridRotation>.ReadOnly

NrAdjacenciesPerNode

Read-only view on NrAdjacenciesPerNode.

```
public int NrAdjacenciesPerNode { get; }
```

Property Value

int

Methods

DeepCopyAlloc(out RuntimeGraph)

Creates a deep copy of this instance.
The copy will be a RuntimeGraph instance, meaning it will be mutable.
This can be useful to decouple a runtime graph that is the result of a pipeline run from that pipeline, since resetting the pipeline will dispose of the runtime graph that it created.

```
public void DeepCopyAlloc(out RuntimeGraph copy)
```

Parameters

`copy` RuntimeGraph

Output parameter that will contain the copied runtime graph when the method returns.

Remarks

This allocates native memory with Unity.Collections.Allocator.Persistent. You must call Dispose() to free it.
No memory is allocated if IsCreated is false and the output parameter will be `default`.

Namespace PWS.DungeonBlueprint.Topology

Structs

TopologyGraph

Main data structure for the Topology phase of the transformation pipeline.
This is the first phase where each node represents a room in the final dungeon and each edge represents a door in the final dungeon.

TopologyGraph.ReadOnly

Read-only wrapper around a topology graph instance.
This is only a view over the existing memory and not a deep copy.

Struct TopologyGraph

Namespace: PWS.DungeonBlueprint.Topology

Main data structure for the Topology phase of the transformation pipeline.
This is the first phase where each node represents a room in the final dungeon and each edge represents a door in the final dungeon.

```
[BurstCompile]
public struct TopologyGraph : IDisposable
```

Implements
IDisposable

Inherited Members
ValueType.Equals(object) , ValueType.GetHashCode() , ValueType.ToString() , object.Equals(object, object) , object.GetType() , object.ReferenceEquals(object, object)

Fields

AdjacencySlotToEdge

Mapping from adjacency to edge.
Every valid adjacency belongs to an edge. Since adjacencies are always bidirectional and thus take up two slots in NodeAdjacencies, an edge combines two adjacency slots into a single edge, helping to reduce data duplication.
Every index in the list is either an index for an edge collection or InvalidEdge.

```
public NativeList<int> AdjacencySlotToEdge
```

Field Value

NativeList<int>

Remarks

This object owns this native memory and frees it in Dispose().
This list is in parallel with AdjacencySlotToEdge, meaning the lists contain information about

the same adjacency at a given index.

EdgeDoorDataCollectionIdx

Door collection identifier of each edge.
Identifier stems from the DBPRegistry.

```
public NativeList<int> EdgeDoorDataCollectionIdx
```

Field Value

NativeList<int>

Remarks

This list is in parallel with EdgeNames, meaning the lists contain information about the same edge at a given index.
This object owns this native memory and frees it in Dispose().

See Also
EdgeNames

EdgeNames

Name of each edge. Only door nodes from the mission are turned into edges in the topology graph, so these are only the door names.
Room nodes are instead turned into nodes and their names are available in NodeNames.
The name is carried all the way into the spawned dungeon and will then be queryable with GetDoorByName(FixedString64Bytes, int).

```
public NativeList<FixedString64Bytes> EdgeNames
```

Field Value

NativeList<FixedString64Bytes>

Remarks

This list is in parallel with EdgeDoorDataCollectionIdx, meaning the lists contain information about the same edge at a given index.
This object owns this native memory and frees it in Dispose().

See Also
EdgeDoorDataCollectionIdx, NodeNames

NodeAdjacencies

Adjacencies between nodes.
Adjacencies of a given node `n`, are in the range `[n*NrAdjacenciesPerNode, n*NrAdjacenciesPerNode+NrAdjacenciesPerNode)`.
Total length of this collection is the `number of nodes * NrAdjacenciesPerNode`.
Every index is either a node or InvalidNode.

```
public NativeList<int> NodeAdjacencies
```

Field Value

NativeList<int>

Examples

If `NrAdjacenciesPerNode=3`, the adjacencies for node `0` are at the indices 0, 1 and 2 (the range is `[0*3=0, 0*3+3=3)`).
If `NrAdjacenciesPerNode=2`, the adjacencies for node `4` are at the indices 8 and 9 (the range is `[4*2=8, 4*2+2=10)`).

Remarks

Adjacencies are always bidirectional, so if a node has an adjacency to another node, the second node must also have an adjacency to the first node.
This object owns this native memory and frees it in Dispose().
This list is in parallel with AdjacencySlotToEdge, meaning the lists contain information about the same adjacency at a given index.

See Also
NrAdjacenciesPerNode

NodeEntryPortalNames

Additional data about some nodes.
This data only exists for nodes that were connected to a portal node with portal type Entrance in the mission graph before the topology conversion.
Portals do not exist past the mission graph stage, but we keep track of the names so we can query them later.

```
public NativeHashMap<int, FixedString64Bytes> NodeEntryPortalNames
```

Field Value

NativeHashMap<int, FixedString64Bytes>

Remarks

This object owns this native memory and frees it in Dispose().

See Also
NodeExitPortalNames

NodeExitPortalNames

Additional data about some nodes.
This data only exists for nodes that were connected to a portal node with portal type Exit in the mission graph before the topology conversion.
Portals do not exist past the mission graph stage, but we keep track of the names so we can query them later.

```
public NativeHashMap<int, FixedString64Bytes> NodeExitPortalNames
```

Field Value

NativeHashMap<int, FixedString64Bytes>

Remarks

This object owns this native memory and frees it in Dispose().

See Also

NodeEntryPortalNames

NodeNames

Name of each node. Only room nodes from the mission are turned into nodes in the topology graph, so these are only the room names.
Door nodes are instead turned into edges and their names are available in EdgeNames. The name is carried all the way into the spawned dungeon and will then be queryable with GetRoomByName(FixedString64Bytes, int).

```
public NativeList<FixedString64Bytes> NodeNames
```

Field Value

NativeList<FixedString64Bytes>

Remarks

This list is in parallel with NodeRoomDataCollectionIdx, meaning the lists contain information about the same node at a given index.
This object owns this native memory and frees it in Dispose().

See Also
NodeRoomDataCollectionIdx, EdgeNames

NodeRoomDataCollectionIdx

Room collection identifier of each node.
Identifier stems from the DBPRegistry.

```
public NativeList<int> NodeRoomDataCollectionIdx
```

Field Value

NativeList<int>

Remarks

This list is in parallel with NodeNames, meaning the lists contain information about the same node at a given index.
This object owns this native memory and frees it in Dispose().

See Also
NodeNames

NrAdjacenciesPerNode

Maximum number of adjacencies any node in the graph has.
This is used to determine how many slots are reserved for each node in the adjacency collection.

```
public int NrAdjacenciesPerNode
```

Field Value

int

Properties

IsCreated

Checks whether this instance currently holds the necessary memory to be valid.
This is a necessary condition for a valid instance, but not a sufficient one.
This means that if this is false, it's definitely not a valid instance to run logic on, but if this is true, that is not a guarantee that it is a valid instance.

```
public readonly bool IsCreated { get; }
```

Property Value

bool

Methods

AsReadOnly()

Creates a new read-only wrapper and returns it.

```
public TopologyGraph.ReadOnly AsReadOnly()
```

Returns

TopologyGraph.ReadOnly

Read-only view of this instance.

See Also
TopologyGraph.ReadOnly

CreateAlloc(int, int, out TopologyGraph)

Creates a new topology graph instance and allocates all native collections for the given number of nodes and adjacencies.

```
[BurstCompile]
public static void CreateAlloc(int estimatedNrNodes, int
estimatedNrAdjacenciesPerNode, out TopologyGraph topologyGraph)
```

Parameters

`estimatedNrNodes` int

Number of nodes the topology graph is estimated to have.

`estimatedNrAdjacenciesPerNode` int

Number of adjacencies per node the topology graph is estimated to have.

`topologyGraph` TopologyGraph

Output parameter that will contain the created TopologyGraph instance.

Remarks

This allocates native memory with Unity.Collections.Allocator.Persistent. You must call Dispose() to free it.

Dispose()

Frees native memory. You must not continue using this instance after calling this.

```
public void Dispose()
```

Struct TopologyGraph.ReadOnly

Namespace: PWS.DungeonBlueprint.Topology

Read-only wrapper around a topology graph instance.
This is only a view over the existing memory and not a deep copy.

```
public readonly struct TopologyGraph.ReadOnly
```

Inherited Members
ValueType.Equals(object) , ValueType.GetHashCode() , ValueType.ToString() , object.Equals(object, object) , object.GetType() , object.ReferenceEquals(object, object)

Constructors

ReadOnly(ref TopologyGraph)

Creates a new read-only wrapper instance around the given topology graph.

```
public ReadOnly(ref TopologyGraph original)
```

Parameters

`original` TopologyGraph

Topology graph to wrap.

Properties

AdjacencySlotToEdge

Read-only view on AdjacencySlotToEdge.

```
public NativeArray<int>.ReadOnly AdjacencySlotToEdge { get; }
```

Property Value

NativeArray<int>.ReadOnly

EdgeDoorDataCollectionIdx

Read-only view on EdgeDoorDataCollectionIdx.

```
public NativeArray<int>.ReadOnly EdgeDoorDataCollectionIdx { get; }
```

Property Value

NativeArray<int>.ReadOnly

EdgeNames

Read-only view on EdgeNames.

```
public NativeArray<FixedString64Bytes>.ReadOnly EdgeNames { get; }
```

Property Value

NativeArray<FixedString64Bytes>.ReadOnly

IsCreated

Mirrors IsCreated of the instance the readonly instance was created with.

```
public bool IsCreated { get; }
```

Property Value

bool

NodeAdjacencies

Read-only view on NodeAdjacencies.

```
public NativeArray<int>.ReadOnly NodeAdjacencies { get; }
```

Property Value

NativeArray<int>.ReadOnly

NodeEntryPortalNames

Read-only view on NodeEntryPortalNames.

```
public NativeHashMap<int, FixedString64Bytes>.ReadOnly NodeEntryPortalNames { get; }
```

Property Value

NativeHashMap<int, FixedString64Bytes>.ReadOnly

NodeExitPortalNames

Read-only view on NodeExitPortalNames.

```
public NativeHashMap<int, FixedString64Bytes>.ReadOnly NodeExitPortalNames { get; }
```

Property Value

NativeHashMap<int, FixedString64Bytes>.ReadOnly

NodeNames

Read-only view on NodeNames.

```
public NativeArray<FixedString64Bytes>.ReadOnly NodeNames { get; }
```

Property Value

NativeArray<FixedString64Bytes>.ReadOnly

NodeRoomDataCollectionIdx

Read-only view on NodeRoomDataCollectionIdx.

```
public NativeArray<int>.ReadOnly NodeRoomDataCollectionIdx { get; }
```

Property Value

NativeArray<int>.ReadOnly

NrAdjacenciesPerNode

Read-only view on NrAdjacenciesPerNode.

```
public int NrAdjacenciesPerNode { get; }
```

Property Value

int

Namespace PWS.DungeonBlueprint.Util

Classes

BurstDiscard

Collection of helper functions which are attributed with Unity.Burst.BurstDiscardAttribute, meaning it will not be executed in code compiled by the Burst compiler.

MiscExtensions

Some helper functions to extend the functionality of various types.

Structs

BucketPriorityQueue<TValue>

Bucket priority queue for queueing elements with a known number of discrete priorities. You must pass the maximum number of priorities into the constructor.

RandomPicker

Random number picker for picking numbers in an interval [0, max] where max is passed to the constructor. You may change the maximum number after construction with Change Range(int, ref Random).
Numbers are guaranteed to only be picked once, until ResetSelection() or Change Range(int, ref Random) is called.

Struct BucketPriorityQueue<TValue>

Namespace: PWS.DungeonBlueprint.Util

Bucket priority queue for queueing elements with a known number of discrete priorities. You must pass the maximum number of priorities into the constructor.

```
public struct BucketPriorityQueue<TValue> : IDisposable where TValue :
unmanaged, IEquatable<TValue>
```

Type Parameters

`TValue`

Type of the elements in the priority queue. Must implement IEquatable<T> for removal functionality.

Implements
IDisposable

Inherited Members
ValueType.Equals(object) , ValueType.GetHashCode() , ValueType.ToString() , object.Equals(object, object) , object.GetType() , object.ReferenceEquals(object, object)

Remarks

This queue is not stable, meaning if multiple elements are queued with the same priority, there is no guaranteed order they will be dequeued with.
This type encapsulates native memory. You must call Dispose() to free it.
This type contains custom Unity.Profiling.ProfilerMarker fields with the prefix "BucketPriorityQueue".

Constructors

BucketPriorityQueue(int, int, Allocator)

Creates a new instance of BucketPriorityQueue<TValue> with the given number of priorities.

```
public BucketPriorityQueue(int nrBuckets, int bucketCapacity, Allocator allocator)
```

Parameters

`nrBuckets` int

Number of buckets in this queue. You can not enqueue an element with a priority higher than nrBuckets - 1.

`bucketCapacity` int

Initial capacity of each priority bucket. Buckets will be resized dynamically if you enqueue more than this into one priority bucket.

`allocator` Allocator

Allocator used for allocating the buckets.

Remarks

This allocates O(nrBuckets * bucketCapacity) memory using the given allocator handle. You must call Dispose() to free it.

Exceptions

ArgumentException

If `nrBuckets` is below or equal to 0 and ENABLE_UNITY_COLLECTIONS_CHECKS is defined.

Properties

IsEmpty

Checks whether there is currently an element queued up.

```
public bool IsEmpty { get; }
```

Property Value

bool

Exceptions

ObjectDisposedException

If Dispose() was already called on this instance and ENABLE_UNITY_COLLECTIONS_CHECKS is defined.

Methods

Clear()

Removes all elements currently queued.

```
public void Clear()
```

Remarks

O(number of buckets) time.

Dequeue()

Dequeues the element with the highest priority currently queued. If multiple elements with the same priority exist, they will be dequeued without a guaranteed order.

```
public TValue Dequeue()
```

Returns

TValue

Element with the highest priority currently queued.

Remarks

Amortized O(1) time, O(number of buckets) worst case.

Exceptions

ObjectDisposedException

If Dispose() was already called on this instance and ENABLE_UNITY_COLLECTIONS_CHECKS is defined.

InvalidOperationException

If the queue is empty and ENABLE_UNITY_COLLECTIONS_CHECKS is defined. Check Is Empty to avoid this.

Dispose()

Frees native memory. You must not continue using this instance after calling this.

```
public void Dispose()
```

Enqueue(TValue, int)

Enqueues the given element with the given priority into the queue.

```
public void Enqueue(TValue item, int priority)
```

Parameters

`item` TValue

Element to enqueue.

`priority` int

Priority this element is enqueued with.

Remarks

Amortized O(1) time, O(n) worst case (happens if the bucket has to be resized to enqueue the item).
This may allocate memory using the allocator handle originally given to this instance. The amount of new memory allocated depends on how many elements are queued with this priority. See Unity.Collections.LowLevel.Unsafe.UnsafeList<T>.Add(in T). If new memory allocated, the old memory is freed automatically.

Exceptions

ObjectDisposedException

If Dispose() was already called on this instance and ENABLE_UNITY_COLLECTIONS_CHECKS is defined.

InvalidOperationException

If `priority` is below 0 or higher than the maximum priority passed to the constructor and ENABLE_UNITY_COLLECTIONS_CHECKS is defined.

Remove(TValue, int)

Removes an element equal to the given element from the given priority bucket. Equality is determined by Equals(T).

```
public void Remove(TValue item, int priority)
```

Parameters

`item` TValue

Element to remove from the queue.

`priority` int

Priority the given element is currently queued with.

Remarks

Amortized O(1) time, O(number of buckets) worst case.
If multiple elements equal to the given element are queued in the given priority bucket, only one of them is removed. There is no guarantee on which of the elements will be removed.

Exceptions

ObjectDisposedException

If Dispose() was already called on this instance and ENABLE_UNITY_COLLECTIONS_CHECKS is defined.

InvalidOperationException

If `priority` is below 0 or higher than the maximum priority passed to the constructor and ENABLE_UNITY_COLLECTIONS_CHECKS is defined.

InvalidOperationException

If no element equal to `item` is currently queued with the given priority and ENABLE_UNITY_COLLECTIONS_CHECKS is defined.

See Also

Bucket priority queue on Wikipedia

Class BurstDiscard

Namespace: PWS.DungeonBlueprint.Util

Collection of helper functions which are attributed with Unity.Burst.BurstDiscardAttribute, meaning it will not be executed in code compiled by the Burst compiler.

```
public static class BurstDiscard
```

Inheritance

object ← BurstDiscard

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Properties

AllowMessages

Controls whether log functions in this class are allowed to print messages.

```
public static bool AllowMessages { get; set; }
```

Property Value

bool

Class MiscExtensions

Namespace: PWS.DungeonBlueprint.Util

Some helper functions to extend the functionality of various types.

```
public static class MiscExtensions
```

Inheritance

object ← MiscExtensions

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Methods

WithAlpha(Color, float)

Changes the given color to keep its R,G,B values but use the given alpha value.

```
public static Color WithAlpha(this Color color, float alpha)
```

Parameters

`color` Color

Color to take the R,G,B from.

`alpha` float

Alpha of the new color.

Returns

Color

New color with the R,G,B of the given color and the given alpha.

Struct RandomPicker

Namespace: PWS.DungeonBlueprint.Util

Random number picker for picking numbers in an interval [0, max] where max is passed to the constructor. You may change the maximum number after construction with ChangeRange(int, ref Random).
Numbers are guaranteed to only be picked once, until ResetSelection() or ChangeRange(int, ref Random) is called.

```
public struct RandomPicker : IDisposable
```

Implements
IDisposable

Inherited Members
ValueType.Equals(object) , ValueType.GetHashCode() , object.Equals(object, object) , object.GetType() , object.ReferenceEquals(object, object)

Remarks

Pick probabilities are not guaranteed to be equally distributed.
This type encapsulates native memory. You must call Dispose() to free it.
This type contains custom Unity.Profiling.ProfilerMarker fields with the prefix "RandomPicker".

Constructors

RandomPicker(int, AllocatorHandle, ref Random)

Creates a new instance of RandomPicker to pick random numbers.

```
public RandomPicker(int length, AllocatorManager.AllocatorHandle allocator, ref
Random random)
```

Parameters

`length` int

Maximum number this picker can pick.

`allocator` AllocatorManager.AllocatorHandle

Allocator used to allocate buffer for picked indices.

`random` Random

Random state for random number generation.

Remarks

This allocates O(length) memory using the given allocator handle. You must call Dispose() to free it.

Exceptions

ArgumentException

If `length` is less or equal to 0 and ENABLE_UNITY_COLLECTIONS_CHECKS is defined.

Fields

MustChangeRange

Used to indicate that the range of this picker is invalid and should be changed. If this is true, you should call ChangeRange(int, ref Random) before continuing to use this instance.

```
public bool MustChangeRange
```

Field Value

bool

Remarks

Never set to true internally. You need to recognize that the range needs to change externally, and you also need to call ChangeRange(int, ref Random) externally.

Properties

Current

Gets the currently picked number.
This is idempotent. Getting the property multiple times in a row will not continue picking new numbers. Call Continue(ref Random) to pick the next number and then get this property again to get the new picked number.

```
public int Current { get; }
```

Property Value

int

Exceptions

ObjectDisposedException

If Dispose() was already called on this instance and ENABLE_UNITY_COLLECTIONS_CHECKS is defined.

InvalidOperationException

If MustChangeRange is true and ENABLE_UNITY_COLLECTIONS_CHECKS is defined. Call ChangeRange(int, ref Random) before using this property.

IndexOutOfRangeException

If no number is currently picked (i.e. all numbers were picked already and Continue(ref Random) has been called again) and ENABLE_UNITY_COLLECTIONS_CHECKS is defined. Check HasCurrent before calling this getter to avoid this exception.

See Also
HasCurrent, Continue(ref Random)

HasCurrent

Checks whether this picker has a currently picked number. You should check this before using Current.

```
public bool HasCurrent { get; }
```

Property Value

bool

IsInitialized

Checks whether the allocated buffer is still valid.
True between calling RandomPicker(int, AllocatorHandle, ref Random) and calling Dispose().

```
public bool IsInitialized { get; }
```

Property Value

bool

Methods

ChangeRange(int, ref Random)

Resizes the underlying array and resets the existing selection state.
After this method returns this instance looks the same as if it was originally created with the given random.

```
public void ChangeRange(int max, ref Random random)
```

Parameters

`max` int

New maximum number to pick randomly.

`random` Random

Random instance for random number generation.

Remarks

O(max) time, unless `max` is the same as the old range, then O(1).
This may additionally allocate O(max) memory using the allocator originally given to this

instance. See Resize(int, NativeArrayOptions). If new memory allocated, the old memory is freed automatically.

Exceptions

ObjectDisposedException

If Dispose() was already called on this instance and ENABLE_UNITY_COLLECTIONS_CHECKS is defined.

ArgumentException

If `max` is less or equal to 0 and ENABLE_UNITY_COLLECTIONS_CHECKS is defined.

Continue(ref Random)

Picks the next random number within the range of the picker.
A number will only ever be picked once until you call ResetSelection() or ChangeRange(int, ref Random).

```
public void Continue(ref Random random)
```

Parameters

`random` Random

Random state for random number generation.

Exceptions

ObjectDisposedException

If Dispose() was already called on this instance and ENABLE_UNITY_COLLECTIONS_CHECKS is defined.

InvalidOperationException

If MustChangeRange is true and ENABLE_UNITY_COLLECTIONS_CHECKS is defined. Call ChangeRange(int, ref Random) before using this property.

Dispose()

Frees native memory. You must not continue using this instance after calling this.

```
public void Dispose()
```

ResetSelection()

Resets the selection.

```
public void ResetSelection()
```

Remarks

Since the random indices are picked based on the state of the Unity.Mathematics.Random instance every time Continue(ref Random) is called, the picks can differ between resets.

Exceptions

ObjectDisposedException

If Dispose() was already called on this instance and ENABLE_UNITY_COLLECTIONS_CHECKS is defined.

InvalidOperationException

If MustChangeRange is true and ENABLE_UNITY_COLLECTIONS_CHECKS is defined. Call ChangeRange(int, ref Random) before using this property.

ToString()

Turns the RandomPicker into a human-readable string.

```
public override string ToString()
```

Returns

string

Human-readable version of the RandomPicker.

Namespace PWS.DungeonBlueprint.World

Classes

DBPWorld

Runtime interface of a generated dungeon. This is one of the main components of Dungeon Blueprint. World takes a previously generated PWS.DungeonBlueprint.World. DBPWorld.RuntimeGraph and spawns it into the Unity scene. After spawning, the world offers runtime feature like querying and events.

Delegates

BlockDoorSpawnEventHandler

Delegate for subscribing to block door spawns.

CanMoveCheckCallback

Delegate for checking traversability from one tile to another tile during pathfinding.

DoorPassHandler

Delegate for subscribing to door pass events.

DoorSpawnEventHandler

Delegate for subscribing to door spawns.

RoomEnterEventHandler

Delegate for subscribing to room enter events.

RoomExitEventHandler

Delegate for subscribing to room exit events.

RoomSpawnEventHandler

Delegate for subscribing to room spawns.

RoomStayEventHandler

Delegate for subscribing to room stay events.

Delegate BlockDoorSpawnEventHandler

Namespace: PWS.DungeonBlueprint.World

Delegate for subscribing to block door spawns.

```
public delegate void BlockDoorSpawnEventHandler(DBPWorld world,
GameObject spawnedGameObject)
```

Parameters

`world` DBPWorld

World that invoked the event.

`spawnedGameObject` GameObject

GameObject that was spawned for the block door.

Delegate CanMoveCheckCallback

Namespace: PWS.DungeonBlueprint.World

Delegate for checking traversability from one tile to another tile during pathfinding.

```
public delegate bool CanMoveCheckCallback(DBPWorld world, GridPosition from,
GridPosition to)
```

Parameters

`world` DBPWorld

World that is calling the move check.

`from` GridPosition

Start position.

`to` GridPosition

End position.

Returns

bool

Delegate for checking traversability from one tile to another tile during pathfinding.

Remarks

You can use this to check traversability of specific tiles but also of whole rooms and doors. Use the world's queries to get the rooms the given positions are in and also to check for the door between the rooms.

Class DBPWorld

Namespace: PWS.DungeonBlueprint.World

Runtime interface of a generated dungeon. This is one of the main components of Dungeon Blueprint. World takes a previously generated PWS.DungeonBlueprint.World.DBPWorld. RuntimeGraph and spawns it into the Unity scene. After spawning, the world offers runtime feature like querying and events.

```
public class DBPWorld : IDisposable
```

Inheritance

object ← DBPWorld

Implements

IDisposable

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Examples

Spawn a dungeon

```
void Awake() {
    world = new DBPWorld(registry);
    world.Setup(runtimeGraph, settings);
}

void Update() {
    // Tick spawns the dungeon and then checks registered events, so you can just
keep calling this
    world.Tick();
}
```

Constructors

DBPWorld(DBPRegistry)

Creates a new instance of DBPWorld.

```
public DBPWorld(DBPRegistry registry)
```

Parameters

`registry` DBPRegistry

Registry your rooms and doors are registered with.

See Also
DBPRegistry

Properties

DungeonOriginWorldPosition

Position of the dungeon grid origin when it is spawned into the world. See DungeonOrigin WorldPosition.

```
public Vector3 DungeonOriginWorldPosition { get; }
```

Property Value

Vector3

IsReady

Whether the world is fully set up and the dungeon is spawned.
Gameplay events will not be invoked and some queries are not available if this is false. See OnWorldIsReady for an event that notifies you when the world is ready. See also OnWorld Reset for an event that notifies you when the world is not ready anymore.

```
public bool IsReady { get; }
```

Property Value

bool

NrDoors

Number of doors in the current world.
This is not the number of already spawned doors, but the number of doors in the dungeon data.
0 if the world is not set up.

```
public int NrDoors { get; }
```

Property Value

int

NrRooms

Number of rooms in the current world.
This is not the number of already spawned rooms, but the number of rooms in the dungeon data.
0 if the world is not set up.

```
public int NrRooms { get; }
```

Property Value

int

RoomGridToWorld

World scale of rooms. A 1x1x1 room is expected to have exactly these bounds in world space. See RoomGridToWorld.

```
public Vector3 RoomGridToWorld { get; }
```

Property Value

Vector3

SpawnContainer

Parent transform of all GameObjects spawned for the dungeon. This is `null` when no dungeon is spawned.

```
public Transform SpawnContainer { get; }
```

Property Value

Transform

Remarks

The spawned GameObjects include not only the rooms and doors, but also the block doors spawned.
Note that this property does not specifically filter GameObjects for the objects it actually spawned. If your own code instantiates more GameObjects alongside the spawned rooms and doors, they will also be children of this container.

Methods

AreRoomsConnected(int, int)

Checks whether the two given rooms are connected through a door.
You can get the room indices using another query (like GetRoomByName(FixedString64Bytes, int)) or from gameplay events (see RegisterRoomEvents(Transform, RoomEnterEventHandler, RoomStayEventHandler, RoomExitEventHandler)).

```
public bool AreRoomsConnected(int room1, int room2)
```

Parameters

`room1` int

First room.

`room2` int

Second room.

Returns

bool

True if there is a direction connection between the two rooms, false otherwise.

Remarks

Since connections are always bidirectional, this check is commutative (i.e. AreRoomsConnected(room1, room2) == AreRoomsConnected(room2, room1)).

Exceptions

InvalidOperationException

If the world is not set up.

ArgumentOutOfRangeException

If `room1` does not exist.

ArgumentOutOfRangeException

If `room2` does not exist.

Dispose()

Frees native memory. You must not continue using this instance after calling this.

```
public void Dispose()
```

Remarks

This is the same as calling Reset(), except that OnWorldReset will be called with true as second parameter.
You may use this as indication that the world will not be reused, and it is time to clean up all resources related to it.

DoesDoorConnectRooms(int, int, int)

Checks whether the given door connects to the two given rooms. You can get the door index using another query (like GetDoorByName(FixedString64Bytes, int)) or from gameplay events (see RegisterDoorEvents(Transform, DoorPassHandler)). You can get the room indices using another query (like GetRoomByName(FixedString64Bytes, int)) or from gameplay events (see RegisterRoomEvents(Transform, RoomEnterEventHandler, RoomStayEventHandler, RoomExitEventHandler)).

```
public bool DoesDoorConnectRooms(int door, int room1, int room2)
```

Parameters

`door` int

Door to check for connection between the rooms

`room1` int

First room to check for connection.

`room2` int

Second room to check for connection.

Returns

bool

True if `room1` and `room2` are connected by `door`.

Remarks

The order in which you pass the rooms makes no difference. DoesDoorConnectRooms(door, room1, room2) == DoesDoorConnectRooms(door, room2, room1).

Exceptions

InvalidOperationException

If the world is not set up.

ArgumentOutOfRangeException

If the given door does not exist.

ArgumentOutOfRangeException

If `room1` does not exist.

ArgumentOutOfRangeException

If `room2` does not exist.

InvalidOperationException

If the door is not connected to any rooms (which is not a valid state in general).

GetDoorBetweenRooms(int, int)

Gets the door between two rooms.
You can get the room indices using another query (like GetRoomByName(FixedString64Bytes, int)) or from gameplay events (see RegisterRoomEvents(Transform, RoomEnterEventHandler, RoomStayEventHandler, RoomExitEventHandler)).

```
public int GetDoorBetweenRooms(int room1, int room2)
```

Parameters

`room1` int

First room.

`room2` int

Second room.

Returns

int

Door index connecting the two rooms or InvalidEdge if the rooms are not connected.

Remarks

This method is commutative (i.e. GetDoorBetweenRooms(room1, room2) == GetDoorBetweenRooms(room2, room1)).

Exceptions

InvalidOperationException

If the world is not set up.

ArgumentOutOfRangeException

If `room1` does not exist.

ArgumentOutOfRangeException

If `room2` does not exist.

GetDoorByGameObject(GameObject)

Gets the door index of the door with the given game object.

```
public int GetDoorByGameObject(GameObject gameObject)
```

Parameters

`gameObject` GameObject

Game object you want to get the door for.

Returns

int

Door index with the given game object or InvalidEdge if such a door does not exist.

Exceptions

InvalidOperationException

If the world is not set up.

InvalidOperationException

If the world is not ready.

GetDoorByGridPositionAndDirection(GridPosition, GridDirection)

Gets the door index of the door with the given position and direction.

```
public int GetDoorByGridPositionAndDirection(GridPosition position,
GridDirection direction)
```

Parameters

`position` GridPosition

Position you want to get the door for.

`direction` GridDirection

Direction you want to get the door for.

Returns

int

Door index with the given position and direction or InvalidEdge if such a door does not exist.

Remarks

There are 2 different combinations of grid position and direction that represent the same position (i.e. position (0, 0, 0) and direction HorizontalRight and position (1, 0, 0) and direction HorizontalLeft are the same). This function always checks both the given position and direction and its equivalent second representation for a door.

Exceptions

InvalidOperationException

If the world is not set up.

GetDoorByName(FixedString64Bytes, int)

Gets the door index of the door with the given name.
Door name is not necessarily unique, so you can pass an additional index.

```
public int GetDoorByName(FixedString64Bytes name, int repetitionIdx = 0)
```

Parameters

`name` FixedString64Bytes

Name you want to get the door for. Note that string can be implicitly converted into Unity.Collections.FixedString64Bytes, so you can pass string into this method.

`repetitionIdx` int

Optional index of the edge if it was inserted repeatedly through a group node.

Returns

int

Door index with the given name and index or InvalidEdge if such a door does not exist.

Exceptions

InvalidOperationException

If the world is not set up.

GetDoorConnectedToRoom(int, int)

Gets the door connected to the given room with the given index.
Useful for iterating through all connected doors of a room. Use GetRoomConnectionCount(int) to find out how many connected doors there are.
You can get the room index using another query (like GetRoomByName(FixedString64Bytes, int)) or from gameplay events (see RegisterRoomEvents(Transform, RoomEnterEventHandler, RoomStayEventHandler, RoomExitEventHandler)).

```
public int GetDoorConnectedToRoom(int room, int connectionIdx)
```

Parameters

`room` int

Room you want to get the connected door for.

`connectionIdx` int

Index of the connection.

Returns

int

Door connected to the given room at the given index, or InvalidEdge if such a door does not exist.

Exceptions

InvalidOperationException

If the world is not set up.

ArgumentOutOfRangeException

If the given room does not exist.

ArgumentOutOfRangeException

If `connectionIdx` is negative or larger than NrAdjacenciesPerNode.

GetDoorData(int)

Gets the door data of the given door. You can get the door index using another query (like GetDoorByName(FixedString64Bytes, int)) or from gameplay events (see RegisterDoor Events(Transform, DoorPassHandler)).

```
public DoorDataSO GetDoorData(int door)
```

Parameters

`door` int

Door you want to get the door data of.

Returns

DoorDataSO

Door data of the door.

Exceptions

InvalidOperationException

If the world is not set up.

ArgumentOutOfRangeException

If the given door does not exist.

ArgumentOutOfRangeException

If the door data of the given door is not registered with the DBPRegistry.

See Also
DBPRegistry

GetDoorGameObject(int)

Gets the game object of the given door.
You can get the door index using another query (like GetDoorByName(FixedString64Bytes, int)) or from gameplay events (see RegisterDoorEvents(Transform, DoorPassHandler)).

```
public GameObject GetDoorGameObject(int door)
```

Parameters

`door` int

Door you want to get the game object of.

Returns

GameObject

Game object of the door.

Exceptions

InvalidOperationException

If the world is not set up.

InvalidOperationException

If the world is not ready.

ArgumentOutOfRangeException

If the given door does not exist.

GetDoorGridPositionAndDirection(int)

Gets the grid position of the given door.
You can get the door index using another query (like GetDoorByName(FixedString64Bytes, int)) or from gameplay events (see RegisterDoorEvents(Transform, DoorPassHandler)).

```
public (GridPosition, GridDirection) GetDoorGridPositionAndDirection(int door)
```

Parameters

`door` int

Door you want to get the position of.

Returns

(GridPosition, GridDirection)

Grid position and direction of the door.

Remarks

There are 2 different combinations of grid position and direction that represent the same position (i.e. position (0, 0, 0) and direction HorizontalRight and position (1, 0, 0) and direction HorizontalLeft are the same). There is no way to predict which combination this function will return. However, the return value is stable, so this function will always return the same values for the same door if you do not reset the world.

Exceptions

InvalidOperationException

If the world is not set up.

ArgumentOutOfRangeException

If the given door does not exist.

GetDoorName(int)

Gets the name of the given door.
You can get the door index using another query (like GetDoorByGridPositionAndDirection(GridPosition, GridDirection)) or from gameplay events (see RegisterDoorEvents(Transform, DoorPassHandler)).

```
public FixedString64Bytes GetDoorName(int door)
```

Parameters

`door` int

Door you want to get the name of.

Returns

FixedString64Bytes

Name of the door. Use Unity.Collections.FixedString64Bytes.ToString() if you need a normal string.

Exceptions

InvalidOperationException

If the world is not set up.

ArgumentOutOfRangeException

If the given door does not exist.

GetGridPositionFromWorldPosition(Vector3)

Turns the given world position into the corresponding grid position using DungeonOrigin WorldPosition and RoomGridToWorld.

```
public GridPosition GetGridPositionFromWorldPosition(Vector3 worldPosition)
```

Parameters

`worldPosition` Vector3

World position to turn into a grid position.

Returns

GridPosition

Grid position calculated from `worldPosition`.

Remarks

No validation is done here and the grid position may be invalid (negative sizes or sizes larger than the grid) if the given world position is outside the bounds of the dungeon.

GetPathBetween(List<GridPosition>, GridPosition, GridPosition, CanMoveCheckCallback)

Gets the path between two rooms and fills it into the given list.

```
public bool GetPathBetween(List<GridPosition> pathOutput, GridPosition start,
GridPosition goal, CanMoveCheckCallback canMove = null)
```

Parameters

`pathOutput` List<GridPosition>

List that will be filled with all rooms along the path. Contains `start` at index 0 and `goal` at the last index. If no path exists between the two positions, the list will stay empty.

`start` GridPosition

Start position of the path.

`goal` GridPosition

End position of the path.

`canMove` CanMoveCheckCallback

Callback called during pathfinding to check if a move is legal. See CanMoveCheckCallback. If this is null, all moves are considered legal, as long as they stay inside a room or pass through a door between rooms.

Returns

bool

True if a valid path was found between the two positions, false otherwise.

Remarks

You must pass a valid list as `pathOutput`, which will be filled with the path in this method.
This is to avoid managed allocations on every pathfinding query.
You likely should reuse the list across queries.
`pathOutput` is cleared by the method.
If you pass a delegate instance as `canMove`, you should make sure its performance impact is as low as possible. It may be called very often.

Exceptions

ArgumentNullException

If `pathOutput` is null.

ArgumentOutOfRangeException

If `start` is not inside the dungeon's grid.

ArgumentOutOfRangeException

If `goal` is not inside the dungeon's grid.

GetRoomByEntryPortalName(FixedString64Bytes)

Gets the room that was connected to an entry portal with the given name in the mission graph the dungeon was generated on.

```
public int GetRoomByEntryPortalName(FixedString64Bytes portalName)
```

Parameters

`portalName` FixedString64Bytes

Entry portal name to find the room for. Note that string can be implicitly converted into Unity.Collections.FixedString64Bytes, so you can pass string into this method.

Returns

int

The room that was connected to the entry portal with the given name, or InvalidNode if such a room does not exist.

Exceptions

InvalidOperationException

If the world is not set up.

GetRoomByExitPortalName(FixedString64Bytes)

Gets the room that was connected to an exit portal with the given name in the mission graph the dungeon was generated on.

```
public int GetRoomByExitPortalName(FixedString64Bytes portalName)
```

Parameters

`portalName` FixedString64Bytes

Exit portal name to find the room for. Note that string can be implicitly converted into Unity.Collections.FixedString64Bytes, so you can pass string into this method.

Returns

int

The room that was connected to the exit portal with the given name, or InvalidNode if such a room does not exist.

Exceptions

InvalidOperationException

If the world is not set up.

GetRoomByGameObject(GameObject)

Gets the room index of the room with the given game object.

```
public int GetRoomByGameObject(GameObject gameObject)
```

Parameters

`gameObject` GameObject

Game object you want to get the room for.

Returns

int

Room index with the given game object or InvalidNode if such a room does not exist.

Exceptions

InvalidOperationException

If the world is not set up.

InvalidOperationException

If the world is not ready.

GetRoomByGridPosition(GridPosition)

Gets the room index of the room with the given grid position.

```
public int GetRoomByGridPosition(GridPosition gridPosition)
```

Parameters

`gridPosition` GridPosition

Grid position you want to get the room for.

Returns

int

Room index with the given grid position or InvalidNode if such a room does not exist.

Exceptions

InvalidOperationException

If the world is not set up.

GetRoomByName(FixedString64Bytes, int)

Gets the room index of the room with the given name.
The room name is not necessarily unique, so you can pass an additional index.

```
public int GetRoomByName(FixedString64Bytes name, int repetitionIdx = 0)
```

Parameters

`name` FixedString64Bytes

Name you want to get the room for. Note that string can be implicitly converted into Unity.Collections.FixedString64Bytes, so you can pass string into this method.

`repetitionIdx` int

Optional index of the node if it was inserted repeatedly through a group node.

Returns

int

Room index with the given name and index or InvalidNode if such a room does not exist.

Exceptions

InvalidOperationException

If the world is not set up.

GetRoomConnectedToRoom(int, int)

Gets the room connected to the given room with the given index.
Useful for iterating through all connected rooms of a room. Use GetRoomConnectionCount(int) to find out how many connected rooms there are.
You can get the room index using another query (like GetRoomByName(FixedString64Bytes, int)) or from gameplay events (see RegisterRoomEvents(Transform, RoomEnterEventHandler, RoomStayEventHandler, RoomExitEventHandler)).

```
public int GetRoomConnectedToRoom(int room, int connectionIndex)
```

Parameters

`room` int

Room you want to get the connected room for.

`connectionIndex` int

Index of the connection.

Returns

int

Room connected to the given room at the given index, or InvalidNode if such a room does not exist.

Remarks

The index has no inherent meaning and there is no way to predict which connected room is at which index unless you have in-depth knowledge of the runtime graph instance given to the world.

Exceptions

InvalidOperationException

If the world is not set up.

ArgumentOutOfRangeException

If the given room does not exist.

ArgumentOutOfRangeException

If `connectionIndex` is negative or larger than NrAdjacenciesPerNode.

GetRoomConnectionCount(int)

Gets the number of doors (and thus rooms) connected to the given room. You can get the room index using another query (like GetRoomByName(FixedString64Bytes, int)) or from gameplay events (see RegisterRoomEvents(Transform, RoomEnterEventHandler, RoomStayEventHandler, RoomExitEventHandler)).

```
public int GetRoomConnectionCount(int room)
```

Parameters

`room` int

Room you want to get the number of doors for.

Returns

int

Number of doors connected to the given room.

Exceptions

InvalidOperationException

If the world is not set up.

ArgumentOutOfRangeException

If the given room does not exist.

InvalidOperationException

If no doors are connected to this room (which is not a valid state in general).

GetRoomData(int)

Gets the room data of the given room.
You can get the room index using another query (like GetRoomByName(FixedString64Bytes, int)) or from gameplay events (see RegisterRoomEvents(Transform, RoomEnterEvent Handler, RoomStayEventHandler, RoomExitEventHandler)).

```
public RoomDataSO GetRoomData(int room)
```

Parameters

`room` int

Room you want to get the room data of.

Returns

RoomDataSO

Room data of the room.

Exceptions

InvalidOperationException

If the world is not set up.

ArgumentOutOfRangeException

If the given room does not exist.

ArgumentOutOfRangeException

If the room data of the given room is not registered with the DBPRegistry.

GetRoomEntryPortalName(int)

Gets the name of the entry portal the given room was connected to in the mission graph the dungeon was generated on.

```
public FixedString64Bytes GetRoomEntryPortalName(int room)
```

Parameters

`room` int

Room to get the entry portal name of.

Returns

FixedString64Bytes

The name of the entry portal this room was connected to, default(FixedString64Bytes) if it wasn't connected to an entry portal.

Exceptions

InvalidOperationException

If the world is not set up.

ArgumentOutOfRangeException

If the given room does not exist.

GetRoomExitPortalName(int)

Gets the name of the exit portal the given room was connected to in the mission graph the dungeon was generated on.

```
public FixedString64Bytes GetRoomExitPortalName(int room)
```

Parameters

`room` int

Room to get the exit portal name of.

Returns

FixedString64Bytes

The name of the exit portal this room was connected to, default(FixedString64Bytes) if it wasn't connected to an exit portal.

Exceptions

InvalidOperationException

If the world is not set up.

ArgumentOutOfRangeException

If the given room does not exist.

GetRoomGameObject(int)

Gets the game object of the given room.
You can get the room index using another query (like GetRoomByName(FixedString64Bytes, int)) or from gameplay events (see RegisterRoomEvents(Transform, RoomEnterEvent Handler, RoomStayEventHandler, RoomExitEventHandler)).

```
public GameObject GetRoomGameObject(int room)
```

Parameters

`room` int

Room you want to get the game object of.

Returns

GameObject

Game object of the room.

Exceptions

InvalidOperationException

If the world is not set up.

InvalidOperationException

If the world is not ready.

ArgumentOutOfRangeException

If the given room does not exist.

GetRoomGridPosition(int)

Gets the grid position of the given room.
You can get the room index using another query (like GetRoomByName(FixedString64Bytes, int)) or from gameplay events (see RegisterRoomEvents(Transform, RoomEnterEvent Handler, RoomStayEventHandler, RoomExitEventHandler)).

```
public GridPosition GetRoomGridPosition(int room)
```

Parameters

`room` int

Room you want to get the position of.

Returns

GridPosition

Grid position of the room.

Exceptions

InvalidOperationException

If the world is not set up.

ArgumentOutOfRangeException

If the given room does not exist.

GetRoomName(int)

Gets the name of the given room.
You can get the room index using another query (like GetRoomByGridPosition(GridPosition)) or from gameplay events (see RegisterRoomEvents(Transform, RoomEnterEventHandler, RoomStayEventHandler, RoomExitEventHandler)).

```
public FixedString64Bytes GetRoomName(int room)
```

Parameters

`room` int

Room you want to get the name of.

Returns

FixedString64Bytes

Name of the room. Use Unity.Collections.FixedString64Bytes.ToString() if you need a normal string.

Exceptions

InvalidOperationException

If the world is not set up.

ArgumentOutOfRangeException

If the given room does not exist.

GetRoomSize(int)

Gets the size of the given room.
You can get the room index using another query (like GetRoomByName(FixedString64Bytes,

int)) or from gameplay events (see RegisterRoomEvents(Transform, RoomEnterEvent Handler, RoomStayEventHandler, RoomExitEventHandler)).

```
public GridSize GetRoomSize(int room)
```

Parameters

`room` int

Room you want to get the size of.

Returns

GridSize

Size of the room.

Exceptions

InvalidOperationException

If the world is not set up.

ArgumentOutOfRangeException

If the given room does not exist.

ArgumentOutOfRangeException

If the room data of the given room is not registered with the DBPRegistry.

GetRoomsConnectedToDoor(int)

Gets the rooms connected to the given door. You can get the door index using another query (like GetDoorByName(FixedString64Bytes, int)) or from gameplay events (see RegisterDoorEvents(Transform, DoorPassHandler)).

```
public (int room1, int room2) GetRoomsConnectedToDoor(int door)
```

Parameters

`door` int

Door you want to get the connected rooms of.

Returns

(int room1, int room2)

Tuple with room indices of the connected rooms.

Remarks

All doors are bidirectional. Therefore, the order of the rooms in the returned tuple has no inherent meaning. However, the return value is stable, so this function will always return the room indices in the same order for the same door if you do not reset the world.

Exceptions

InvalidOperationException

If the world is not set up.

ArgumentOutOfRangeException

If the given door does not exist.

InvalidOperationException

If the door is not connected to any rooms (which is not a valid state in general).

GetWorldPositionFromGridPosition(GridPosition)

Turns the given grid position into the corresponding world position using DungeonOrigin WorldPosition and RoomGridToWorld.

```
public Vector3 GetWorldPositionFromGridPosition(GridPosition gridPosition)
```

Parameters

`gridPosition` GridPosition

Grid position to turn into a world position.

Returns

Vector3

World position calculated from `gridPosition`.

Remarks

No validation is done here and the world position may be outside the dungeon if the given grid position is outside the bounds of the dungeon.

IsDoorConnectedToRoom(int, int)

Checks whether the given room is connected to the given door.
You can get the room index using another query (like GetRoomByName(FixedString64Bytes, int)) or from gameplay events (see RegisterRoomEvents(Transform, RoomEnterEventHandler, RoomStayEventHandler, RoomExitEventHandler)).
You can get the door index using another query (like GetDoorByName(FixedString64Bytes, int)) or from gameplay events (see RegisterDoorEvents(Transform, DoorPassHandler)).

```
public bool IsDoorConnectedToRoom(int room, int door)
```

Parameters

`room` int

Room to check.

`door` int

Door to check.

Returns

bool

True if the room is connected to the door, otherwise false.

Exceptions

InvalidOperationException

If the world is not set up.

ArgumentOutOfRangeException

If the given room does not exist.

ArgumentOutOfRangeException

If the given door does not exist.

RegisterDoorEvents(Transform, DoorPassHandler)

Starts tracking the given transform for door events, which will cause the given delegate to be called whenever the transform passes through a door.

```
public void RegisterDoorEvents(Transform tracked, DoorPassHandler onPass)
```

Parameters

`tracked` Transform

Transform to track.

`onPass` DoorPassHandler

Callback for door pass events.

Exceptions

ArgumentNullException

If `tracked` is null.

ArgumentNullException

If `onPass` is null.

RegisterDoorEvents(Transform, int, DoorPassHandler)

Starts tracking the given transform for door events on the given door, which will cause the given delegate to be called whenever the transform passes through the given door.

```
public void RegisterDoorEvents(Transform tracked, int door, DoorPassHandler onPass)
```

Parameters

`tracked` Transform

Transform to track.

`door` int

Callback for door pass events.

`onPass` DoorPassHandler

Callback for door pass events.

Exceptions

ArgumentNullException

If `tracked` is null.

ArgumentOutOfRangeException

If the given door does not exist.

ArgumentNullException

If `onPass` is null.

RegisterRoomEvents(Transform, RoomEnterEventHandler, RoomStayEventHandler, RoomExitEventHandler)

Starts tracking the given transform for room events, which will cause the given delegates to be called whenever the transform enters/stays in/exits a room.

```
public void RegisterRoomEvents(Transform tracked, RoomEnterEventHandler onEnterRoom,
RoomStayEventHandler onStayRoom, RoomExitEventHandler onExitRoom)
```

Parameters

`tracked` Transform

Transform to track.

`onEnterRoom` RoomEnterEventHandler

Callback for room enter events.

`onStayRoom` RoomStayEventHandler

Callback for room stay events.

`onExitRoom` RoomExitEventHandler

Callback for room exit events.

Exceptions

ArgumentNullException

If `tracked` is null.

ArgumentNullException

If all three event handlers are null.

RegisterRoomEvents(Transform, int, RoomEnterEventHandler, RoomStayEventHandler, RoomExitEventHandler)

Starts tracking the given transform for room events on the given room, which will cause the given delegates to be called whenever the transform enters/stays in/exits the given room.

```
public void RegisterRoomEvents(Transform tracked, int room, RoomEnterEventHandler
onEnterRoom, RoomStayEventHandler onStayRoom, RoomExitEventHandler onExitRoom)
```

Parameters

`tracked` Transform

Transform to track.

`room` int

Room to check for the events.

`onEnterRoom` RoomEnterEventHandler

Callback for room enter events.

`onStayRoom` RoomStayEventHandler

Callback for room stay events.

`onExitRoom` RoomExitEventHandler

Callback for room exit events.

Exceptions

ArgumentNullException

If `tracked` is null.

ArgumentOutOfRangeException

If the given room does not exist.

ArgumentNullException

If all three event handlers are null.

Reset()

Resets the world to its initial state.
Any current dungeon will be despawned and any events and queries will stop working.
You can call Setup(ReadOnly, DBPWorldSettings) again to spawn a new dungeon.

```
public void Reset()
```

Remarks

Calling this will invoke OnWorldReset with false as second parameter, indicating that the world instance may be reused.

Setup(ReadOnly, DBPWorldSettings)

Sets up the world. You must call this before the world can spawn the dungeon.
Calling this resets the world so you can just call this again to spawn a new dungeon without calling Reset() manually.
However, this will lead to the current dungeon despawning immediately, even if the new dungeon spawns asynchronously.

```
public void Setup(RuntimeGraph.ReadOnly runtimeGraph, DBPWorldSettings settings)
```

Parameters

`runtimeGraph` RuntimeGraph.ReadOnly

Runtime graph that contains all information about the dungeon that will be spawned.

`settings` DBPWorldSettings

Additional settings needed by the world.

Remarks

CRITICAL MEMORY LIFETIME WARNING:
The world does not copy the memory in `runtimeGraph`.
If the runtime graph instance you pass is disposed, the memory is freed and this world becomes invalid.
This happens automatically if this instance was generated through a DBPPipeline and that pipeline is disposed, reset or set up again.
To keep the world active while reusing the pipeline, use DeepCopyAlloc(out RuntimeGraph):

```
pipeline.Runtime.DeepCopyAlloc(out RuntimeGraph copy); // Now you own the copy's
memory and must dispose it yourself
world.Setup(copy.AsReadOnly(), settings);
```

Exceptions

InvalidOperationException

If `runtimeGraph` is invalid.

InvalidOperationException

If `settings` is invalid.

Tick()

Runs a single tick of the world. If the world is not set up, this is a noop.
A tick consists of continuing the spawning process until IsReady is true, afterward it checks gameplay events you have subscribed to. Therefore, you must continue calling this even after the dungeon is fully spawned, unless you are not using any gameplay events.

```
public void Tick()
```

Remarks

The tick rate is determined by your code and may be both fixed or variable.
Calling this does nothing if the world is not set up.

UnregisterDoorEvents(Transform)

Stops tracking the given transform for door events.

```
public void UnregisterDoorEvents(Transform tracked)
```

Parameters

`tracked` Transform

Transform to stop tracking.

Exceptions

ArgumentNullException

If `tracked` is null.

InvalidOperationException

If `tracked` is not being tracked.

UnregisterDoorEvents(Transform, DoorPassHandler)

Stops tracking the given transform for door events with the given callbacks.
Should the transform be registered for door events with multiple different callbacks, this

will not unregister the transform tracking with other callbacks than the one given.

```
public void UnregisterDoorEvents(Transform tracked, DoorPassHandler onPass)
```

Parameters

`tracked` Transform

Transform to stop tracking.

`onPass` DoorPassHandler

Door pass event handler the given transform is being tracked with.

Exceptions

ArgumentNullException

If `tracked` is null.

ArgumentNullException

If `onPass` is null.

InvalidOperationException

If `tracked` is not being tracked.

UnregisterDoorEvents(Transform, int)

Stops tracking the given transform for door events for the given door.
Should the transform be registered for door events for multiple door, this will not unregister the transform tracking for other door than the one given.

```
public void UnregisterDoorEvents(Transform tracked, int door)
```

Parameters

`tracked` Transform

Transform to stop tracking.

`door` int

Door the given transform is being tracked for.

Exceptions

ArgumentNullException

If `tracked` is null.

ArgumentOutOfRangeException

If the given door does not exist.

InvalidOperationException

If `tracked` is not being tracked.

UnregisterDoorEvents(Transform, int, DoorPassHandler)

Stops tracking the given transform for door events for the given door with the given handler.
Should the transform be registered for door events for multiple doors with the same handler, this will not unregister the transform tracking for other door than the one given.

```
public void UnregisterDoorEvents(Transform tracked, int door,
DoorPassHandler onPass)
```

Parameters

`tracked` Transform

Transform to stop tracking.

`door` int

Door the given transform is being tracked for.

`onPass` DoorPassHandler

Door pass event handler the given transform is being tracked with.

Exceptions

ArgumentNullException

If `tracked` is null.

ArgumentNullException

If `onPass` is null.

ArgumentOutOfRangeException

If the given door does not exist.

InvalidOperationException

If `tracked` is not being tracked.

UnregisterRoomEvents(Transform)

Stops tracking the given transform for room events.

```
public void UnregisterRoomEvents(Transform tracked)
```

Parameters

`tracked` Transform

Transform to stop tracking.

Exceptions

ArgumentNullException

If `tracked` is null.

InvalidOperationException

If `tracked` is not being tracked.

UnregisterRoomEvents(Transform, RoomEnterEventHandler, RoomStayEventHandler, RoomExitEventHandler)

Stops tracking the given transform for room events with the given callbacks.
Should the transform be registered for room events with multiple different callbacks, this will not unregister the transform tracking with other callbacks than the one given.

```
public void UnregisterRoomEvents(Transform tracked, RoomEnterEventHandler onEnter,
RoomStayEventHandler onStay, RoomExitEventHandler onExit)
```

Parameters

`tracked` Transform

Transform to stop tracking.

`onEnter` RoomEnterEventHandler

Room enter event handler the given transform is being tracked with.

`onStay` RoomStayEventHandler

Room stay event handler the given transform is being tracked with.

`onExit` RoomExitEventHandler

Room exit event handler the given transform is being tracked with.

Exceptions

ArgumentNullException

If `tracked` is null.

InvalidOperationException

If `tracked` is not being tracked.

UnregisterRoomEvents(Transform, int)

Stops tracking the given transform for room events for the given room.
Should the transform be registered for room events for multiple rooms, this will not unregister the transform tracking for other rooms than the one given.

```
public void UnregisterRoomEvents(Transform tracked, int room)
```

Parameters

`tracked` Transform

Transform to stop tracking.

`room` int

Room the given transform is being tracked for.

Remarks

If the given transform is registered for events on all rooms (using RegisterRoomEvents(Transform, RoomEnterEventHandler, RoomStayEventHandler, RoomExitEventHandler)), using this will unregister the transform only for the given room. It will still receive events for the other rooms.

Exceptions

ArgumentNullException

If `tracked` is null.

InvalidOperationException

If `tracked` is not being tracked.

ArgumentOutOfRangeException

If the given room does not exist.

UnregisterRoomEvents(Transform, int, RoomEnterEventHandler, RoomStayEventHandler, RoomExitEventHandler)

Stops tracking the given transform for room events for the given room with the given callbacks.
Should the transform be registered for room events with multiple different callbacks on this room, this will not unregister the transform tracking with other callbacks than the one given.

```
public void UnregisterRoomEvents(Transform tracked, int room, RoomEnterEventHandler
onEnter, RoomStayEventHandler onStay, RoomExitEventHandler onExit)
```

Parameters

`tracked` Transform

Transform to stop tracking.

`room` int

Room the given transform is being tracked for.

`onEnter` RoomEnterEventHandler

Room enter event handler the given transform is being tracked with.

`onStay` RoomStayEventHandler

Room stay event handler the given transform is being tracked with.

`onExit` RoomExitEventHandler

Room exit event handler the given transform is being tracked with.

Exceptions

ArgumentNullException

If `tracked` is null.

InvalidOperationException

If `tracked` is not being tracked.

Events

OnBlockDoorSpawn

Event invoked when a block door is spawned. This event will be invoked for all block doors before OnWorldIsReady.

```
public event BlockDoorSpawnEventHandler OnBlockDoorSpawn
```

Event Type

BlockDoorSpawnEventHandler

Remarks

This GameObject is not otherwise available through a world query. The only other way to access them is through SpawnContainer, so this event is very helpful if you need to run initialization logic for block doors or want to keep track of them yourself.

OnDoorSpawn

Event invoked when a door is spawned. When this is invoked, the GameObject is already registered with the world. GameObjects of rooms or doors that have not been spawned yet will not be. This event will be invoked for all doors before OnWorldIsReady.

```
public event DoorSpawnEventHandler OnDoorSpawn
```

Event Type

DoorSpawnEventHandler

Remarks

The GameObject is also available through GetDoorGameObject(int) once spawning is complete, but this event can be helpful for initialization logic where something needs to happen for every GameObject that spawns.

OnRoomSpawn

Event invoked when a room is spawned. When this is invoked, the GameObject is already registered with the world. GameObjects of rooms or doors that have not been spawned yet will not be. This event will be invoked for all rooms before OnWorldIsReady.

```
public event RoomSpawnEventHandler OnRoomSpawn
```

Event Type

RoomSpawnEventHandler

Remarks

The GameObject is also available through GetRoomGameObject(int) once spawning is complete, but this event can be helpful for initialization logic where something needs to happen for every GameObject that spawns.

OnWorldIsReady

Event invoked when the world is ready for gameplay interactions like events and querying. The parameter of the delegate is the world object which is ready.

```
public event Action<DBPWorld> OnWorldIsReady
```

Event Type

Action<DBPWorld>

OnWorldReset

Event invoked when the world is reset. After this event is invoked, the world should not be used for gameplay events and queries until another OnWorldIsReady event is invoked.
The event is only invoked if the world was ready before. If you call Reset() before the world was ready, the event is not invoked.
The first parameter of the delegate is the world which was reset, the second parameter indicates whether the world is in the process of being disposed.

```
public event Action<DBPWorld, bool> OnWorldReset
```

Event Type

Action<DBPWorld, bool>

Delegate DoorPassHandler

Namespace: PWS.DungeonBlueprint.World

Delegate for subscribing to door pass events.

```
public delegate void DoorPassHandler(DBPWorld world, Transform tracked,
int passedDoor)
```

Parameters

`world` DBPWorld

World that invoked the event.

`tracked` Transform

Transform that passed the door.

`passedDoor` int

Door that was passed. Use door queries to get more information about the door (like Get DoorGameObject(int)).

Delegate DoorSpawnEventHandler

Namespace: PWS.DungeonBlueprint.World

Delegate for subscribing to door spawns.

```
public delegate void DoorSpawnEventHandler(DBPWorld world, int spawnedDoor,
GameObject spawnedGameObject)
```

Parameters

`world` DBPWorld

World that invoked the event.

`spawnedDoor` int

Door that was spawned. Use door queries to get more information about the door (like GetDoorData(int)).

`spawnedGameObject` GameObject

GameObject that was spawned for the door.

Delegate RoomEnterEventHandler

Namespace: PWS.DungeonBlueprint.World

Delegate for subscribing to room enter events.

```
public delegate void RoomEnterEventHandler(DBPWorld world, Transform tracked,
int enteredRoom)
```

Parameters

`world` DBPWorld

World that invoked the event.

`tracked` Transform

Transform that entered the room.

`enteredRoom` int

Room that was entered. Use room queries to get more information about the room (like GetRoomGameObject(int)).

Delegate RoomExitEventHandler

Namespace: PWS.DungeonBlueprint.World

Delegate for subscribing to room exit events.

```
public delegate void RoomExitEventHandler(DBPWorld world, Transform tracked,
int exitedRoom)
```

Parameters

`world` DBPWorld

World that invoked the event.

`tracked` Transform

Transform that exited the room.

`exitedRoom` int

Room that was exited. Use room queries to get more information about the room (like Get RoomGameObject(int)).

Delegate RoomSpawnEventHandler

Namespace: PWS.DungeonBlueprint.World

Delegate for subscribing to room spawns.

```
public delegate void RoomSpawnEventHandler(DBPWorld world, int spawnedRoom,
GameObject spawnedGameObject)
```

Parameters

`world` DBPWorld

World that invoked the event.

`spawnedRoom` int

Room that was spawned. Use room queries to get more information about the room (like GetRoomData(int)).

`spawnedGameObject` GameObject

GameObject that was spawned for the room.

Delegate RoomStayEventHandler

Namespace: PWS.DungeonBlueprint.World

Delegate for subscribing to room stay events.

```
public delegate void RoomStayEventHandler(DBPWorld world, Transform tracked,
int stayedRoom)
```

Parameters

`world` DBPWorld

World that invoked the event.

`tracked` Transform

Transform that stayed in the room.

`stayedRoom` int

Room that was stayed in. Use room queries to get more information about the room (like GetRoomGameObject(int)).