# SCRIPTING REFERENCE
**version 1.4.1**

*Note: in all Uniblocks functions arguments* ***x,y,z*** *can either be passed as three ints (int x, int y, int z) or as an Index class.*

## Index
A data structure which simply contains x,y and z integer values. This is like an equivalent of Vector3 with ints instead of floats, to avoid any precision errors.

**Index (x,y,z)**
Creates a new index with the given x,y,z values.

**Vector3 ToVector3 ()**
Returns a Vector3 using the x,y,z values of the Index.

**string ToString ()**
Returns the index as a String in the form of “x,y,z”.

**Index FromString ( string indexString )**
Returns a new index converted from a string in an "x,y,z" format (for example, "5,0,-5" will return a new Index (5,0,-5)).

**Index GetAdjacentIndex ( Direction direction )**
Returns a new index which is adjacent to the Index in the given direction.

## Voxel
Stores the properties of each voxel, and provides functions for directly accessing specific voxels.

*functions:*

**DestroyBlock ( VoxelInfo voxelInfo )**
Sets the voxel to an empty block (id 0), updates the chunk's mesh and triggers an OnBlockDestroy event. Sends the voxel change to other connected players if multiplayer is enabled.

**PlaceBlock ( VoxelInfo voxelInfo, ushort data )**
Sets the voxel to the specified id, updates the chunk's mesh and triggers an OnBlockPlace event. Sends the voxel change to other connected players if multiplayer is enabled.

**ChangeBlock ( VoxelInfo voxelInfo, ushort data )**
Sets the voxel to the specified id, updates the chunk's mesh and triggers an OnBlockChange event. Sends the voxel change to other connected players if multiplayer is enabled.

## VoxelInfo

Stores a chunk GameObject and an index of a specific voxel in that chunk. The chunk is stored in VoxelInfo.chunk, and the voxel index in VoxelInfo.index. This class can also store the index of one adjacent voxel, which is used in the VoxelRaycast function - VoxelInfo.index stores the index of the voxel hit by the raycast, and VoxelInfo.adjacentIndex stores the voxel adjacent to the hit face.

*functions:*

**VoxelInfo (x,y,z, GameObject chunk)**
Creates a new VoxelInfo with the voxel index (x,y,z) and the chunk object 'chunk'.

**VoxelInfo (x,y,z, x,y,z, GameObject chunk)**
Creates a new VoxelInfo with the voxel index (x,y,z), adjacent voxel index (x,y,z), and the chunk object 'chunk'.

**ushort GetVoxel ()**
Returns the voxel ID.

**Voxel GetVoxelType ()**
Returns the Voxel component corresponding to the voxel ID.

**ushort GetAdjacentVoxel ()**
Returns the voxel ID of the adjacent voxel.

**Voxel GetAdjacentVoxelType ()**
Returns the Voxel component of the adjacent voxel ID.

**SetVoxel (ushort data, bool updateMesh)**
Changes the voxel data of the voxel stored in the VoxelInfo, and flags the chunk mesh for update if updateMesh is true.
If the index exceeds the chunk's boundaries, this will change the voxel data in the appropriate chunk instead, if it is currently instantiated.

## <u>Engine</u>
Stores global engine settings and provides some static functions for data conversion, etc.

*Properties:*

*Each static variable of the Engine has a non-static equivalent. The non-static variables have the same name as their static counterparts except with a lowercase L at the beginning, and are used in order to enable the editing of these variables in Unity, including in the Engine Settings window. The non-static variables are applied to their static counterparts (through the Engine GameObject in the scene) in the Awake function of the Editor, so changing the non-static variables at runtime will have no effect.*

**(static) string WorldName**
The name of the currently active world. Corresponds to the folder in which the world data is stored.

**(static) string WorldPath**
The path of the world data files. The default path is /application_root/world_name/
You can change the world path by directly editing the UpdateWorldPath private function inside the Engine script.

**(static) string BlocksPath**
The path to the block prefabs inside the Unity project. This is used by the Block Editor to find the blocks.

**(static) int WorldSeed**
The seed of the currently active world which can be used for procedural terrain generation. The seed is stored in the world data folder.

**(static) GameObject Blocks**
Stores the block prefabs. The array index corresponds to the blocks' voxel ID.

**(static) int HeightRange**
The maximum positive and negative vertical chunk index of spawned chunks.

**(static) int ChunkSpawnDistance**
The horizontal distance (in chunks) from the origin point within which chunks will be spawned.

**(static) int ChunkSideLength**
The side length (in voxels) of one chunk.

**(static) float TextureUnit**
The ratio of side length of one block's texture to the side length of the texture sheet, used to calculate block textures.

**(static) float TexturePadding**
Padding between textures on the texture sheet, as a fraction of the size of an individual block's texture.

**(static) bool GenerateColliders**
If false, chunks will not generate any colliders.

**(static) bool ShowBorderFaces**
If true, the side faces of blocks will be visible if a chunk bordering them is not instantiated.

**(static) bool SendCameraLookEvents**
If true, the CameraEventsSender component will send events to the block in the center of the field of view of the main camera.

**(static) bool SendCursorEvents**
If true, the CameraEventsSender component will send events to the block currently under the mouse cursor.

**(static) bool SaveVoxelData**
If false, chunks will not load or save voxel data. Instead, new data will always be generated when chunks are spawned.

**(static) bool EnableMultiplayer**
If true, chunks will request voxel data from a server instead of generating or loading from hard drive. Also, Voxel.ChangeBlock, Voxel.PlaceBlock, and Voxel.DestroyBlock will send the voxel change to the server for redistribution to other connected players.

**(static) bool TrackPlayerPosition**
If true:
Server: the server will check for the player's position to determine whether a voxel change needs to be sent to that player.
Client: the client will send a player position update to the server through the ChunkLoader script.
(Default: true)

**(static) float ChunkTimeout**
If a chunk spawned through ChunkManager.SpawnChunk hasn't been accessed for this amount of time (in seconds), it will be despawned (after saving it's voxel data). Requests for voxel data from clients and voxel changes in the chunk will reset the timer. A value of 0 disables this functionality.

**(static) bool EnableChunkTimeout**
This variable is automatically set to true if Engine.ChunkTimeout is greater than 0.

**(static) int MaxChunkDataRequests**
The maximum number of chunk data requests that can be queued in the server for each client at one time (0=unlimited). Reduce this limit if

the clients are spawning chunks too fast and you find your server can't keep up with the data requests.

**(static) Vector3 ChunkScale**
The scale of the chunk prefab.

*Functions:*

**(static) <u>SetWorldName</u> (string worldName)**
Sets the active world name to 'worldName', and resets the active seed. Use this to change the world name at runtime.

**(static) <u>GetSeed</u> ()**
Reads the currently active world's seed from file, or if a seed file is not found generates a new seed randomly, and stores it in the Engine.WorldSeed variable.

**(static) <u>SaveWorld</u> ()**
Saves the data of all currently instantiated chunks to disk, at a maximum chunks per frame rate specified in Engine.MaxChunkSaves.

**(static) <u>SaveWorldInstant</u> ()**
Saves the data of all currently instantiated chunks to disk in a single frame. This will most likely freeze the game for a few seconds, so it is not recommended to use this during gameplay.

**(static) GameObject <u>GetvoxelGameObject</u> (ushort voxelId)**
Returns the block prefab with the voxel ID 'voxelId'.

**(static) Voxel <u>GetVoxelType</u> (ushort voxelId)**
Returns the Voxel component of the block prefab with the voxel ID 'voxelId'.

**(static) VoxelInfo <u>VoxelRaycast</u> (Vector3 origin, Vector3 direction, float range, bool ignoreTransparent)**
Performs a raycast using the specified origin, direction and range, and returns a VoxelInfo containing the hit chunk GameObject (as VoxelInfo.chunk), the index of the hit voxel (as VoxelInfo.index), and the index of the voxel adjacent to the hit face (as VoxelInfo.adjacentIndex). If 'ignoreTransparent' is true, the raycast will punch through blocks which are Transparent or Semi-transparent. If no blocks are hit, returns null.
Note: This function will not work if collider generation is disabled.

**(static) VoxelInfo <u>VoxelRaycast</u> (Ray ray, float range, bool ignoreTransparent)**
Same as above, with 'ray' replacing 'origin' and 'direction'.

**(static) Index PositionToChunkIndex (Vector3 position)**
Returns the chunk index corresponding to the given world position.

**(static) GameObject PositionToChunk (Vector3 position)**
Returns the chunk GameObject corresponding to the given world position. Returns null if the chunk is not instantiated.

**(static) VoxelInfo PositionToVoxelInfo (Vector3 position)**
Returns the VoxelInfo containing a voxel corresponding to the given world position. Returns null if the voxel's chunk is not instantiated.

**(static) Vector3 VoxelInfoToPosition (VoxelInfo voxelInfo)**
Returns the world position corresponding to the center point of the voxel specified in 'voxelInfo'.

## ChunkManager
Manages the spawning and despawning of chunks.

*Properties:*

**(static) Dictionary<string,Chunk> Chunks**
Stores the references to all Chunk instances in the scene. Chunks add themselves to the dictionary on Awake by calling the ChunkManager.RegisterChunk function. The string key corresponds to the Chunk's index, and follows the format "x,y,z".

*Functions:*

**(static) RegisterChunk (Chunk chunk)**
Adds the chunk to the Chunks list. This is done automatically by Chunks in their Awake function.

**(static) UnregisterChunk (Chunk chunk)**
Removes the chunk from the Chunks list. This is done automatically by Chunks when they are destroyed.

**(static) GameObject GetChunk (x,y,z)**
Returns the GameObject of the chunk with the given chunk index if it is currently instantiated, else returns null.
*Note: This can be somewhat slow. Avoid using too many times per frame.*

**(static) Chunk GetChunkComponent (x,y,z)**
Returns the Chunk component of the chunk with the given chunk index if it is currently instantiated, else returns null.
*Note: This can be somewhat slow. Avoid using too many times per frame.*

**(static) GameObject SpawnChunk (x,y,z)**
Spawns a single chunk with the index (x,y,z) and returns the chunk's GameObject. If the chunk was already spawned, simply returns the already spawned GameObject instead.

**(static) GameObject SpawnChunkFromServer (x,y,z)**
Spawns a single chunk with the index (x,y,z), disables the mesh generation and enables timeout for the spawned chunk, and returns the chunk's GameObject. If the chunk was already spawned, simply returns the

already spawned GameObject instead, without disabling the mesh generation and enabling timeout.

**(static) <u>SpawnChunks</u> (float x, float y, float z)**
Begins spawning chunks with the origin point at the world position (x,y,z).

**(static) <u>SpawnChunks</u> (Vector3 position)**
Begins spawning chunks with the origin point at the world position 'position'.

**(static) <u>SpawnChunks</u> (x,y,z)**
Begins spawning chunks with the origin point at the given chunk index.

*Note: If you use SpawnChunks with three floats or a Vector3, it will treat the arguments as world position and convert it to chunk index before spawning chunks. If you use three ints or an Index, no conversion will be performed and the supplied index will be used directly.*

## <u>Chunk</u>
Manages various basic functions of a chunk, stores voxel data, etc.

*Properties:*

**ushort[] VoxelData**
The main data array, which contains the block ID for every voxel in the chunk. Accessed with GetVoxel and SetVoxel functions.

**Index ChunkIndex**
The chunk index (x,y,z), which is directly related to it's position in the world. The position of a chunk is always ChunkIndex * Engine.ChunkSideLength.

**GameObject[] NeighborChunks**
An array containing references to all direct neighbor chunks of the chunk. The chunks are stored in the order of the Direction enum (up, down, right, left, forward, back), so for examle NeighborChunks[0] returns the chunk above this one.

This array is populated and updated only when a chunk needs to check the voxel data of it's neighbor chunks, such as when updating the chunk's mesh, which means that at some times the array will not be fully up-to-date. You can call GetNeighbors() to update this array immediately.

*Functions:*

**<u>SetVoxel</u> (x,y,z, ushort data, bool updateMesh)**
Changes the voxel data at the specified index, and flags the mesh for update if updateMesh is true.
If the index exceeds the chunk's boundaries, this will change the voxel data in the appropriate chunk instead, if it is currently instantiated.

**<u>SetVoxelSimple</u> (x,y,z, ushort data)**
Changes the voxel data at the specified index. Does not update the mesh. Also, unlike SetVoxel, the index cannot exceed the chunk's boundaries (for example, x cannot be less than 0 and greater than chunk side length -1).

**<u>SetVoxelSimple</u> (int rawIndex)**
Changes the voxel data at the specified array index (flat 1D array index

as opposed to the 3D space coordinates of x,y,z).

**ushort GetVoxel (x,y,z)**
Returns the voxel data at the specified index. If the index exceeds the chunk's boundaries, this will return the voxel data from the appropriate chunk instead if it is currently instantiated, or ushort.MaxValue if it's not.

**ushort GetVoxelSimple (x,y,z)**
Returns the voxel data at the specified index. Unlike GetVoxel, the index cannot exceed the chunk's
boundaries (for example, x cannot be less than 0 and greater than chunk side length -1).

**ushort GetVoxelSimple (int rawIndex)**
Returns the voxel data at the specified array index (flat 1D array index as opposed to the 3D space coordinates of x,y,z).

**Index GetAdjacentIndex (x,y,z, Direction direction)**
Returns the index that is adjacent to the given x,y,z in the given direction. For example, (0,0,0, Direction.left) will return (-1,0,0).

**Index PositionToVoxelIndex (Vector3 position)**
Returns the index of the voxel at the given world position. Note that the position and therefore the returned index can be outside of the chunk's boundaries.

**Index PositionToVoxelIndex (Vector3 position, Vector3 normal, bool returnAdjacent)**
Returns the index of the voxel at the given world position., offset by half a block based on the given normal direction and the returnAdjacent boolean.
This is normally used when raycasting against the blocks. When a raycast hits the wall of a block, the hit position will be pushed either into the block (returnAdjacent==false) or out into the adjacent block (returnAdjacent==true), therefore returning either the block that was hit with the raycast, or the block adjacent to the block wall that was hit.

**Vector3 VoxelIndexToPosition (x,y,z)**
Returns the absolute world position of the center of the given voxel index.

**LoadVoxelData ()**
Loads the voxel data from disk. This will throw an error if the data can't be found, so make sure to call CheckIfFileExists() first.

**GenerateVoxelData ()**
Calls GenerateVoxelData() in the script assigned in the TerrainGenerator variable.

**bool <u>CheckIfFileExists</u> ()**
Returns true if the voxel data for this chunk can be found on the hard drive. Else returns false.

**<u>ClearVoxelData</u> ()**
Clears the main data array.

**int <u>GetDataLength</u> ()**
Returns the length of the main data array.

**<u>FlagToRemove</u> ()**
Flags the chunk to destroy itself by the end of this frame.

**<u>FlagToUpdate</u> ()**
Flags the chunk to update it's mesh by the end of this frame.

**<u>RebuildMesh</u> ()**
Instantly rebuilds the chunk's mesh, even if Engine.GenerateMeshes is disabled.

**<u>UpdateNeighborsIfNeeded</u> (x,y,z)**
Checks whether the given index lies on the border of the chunk. If yes, the neighbor chunk at that border will be flagged for a mesh update. This can potentially trigger up to 3 neighbor updates, if the given index is at the corner of the chunk.

**<u>GetNeighbors</u> ()**
Fills the NeighborChunks array with references to neighbor chunks in all directions, if they are currently instantiated.

## <u>**ChunkMeshCreator**</u>
Handles the creation of a chunk's mesh.

*functions:*

**<u>RebuildMesh</u> ()**
Immediately rebuilds the chunk mesh. In most cases you should be using **Chunk.FlagToUpdate()** instead.

ChunkDataFiles
Handles the loading and saving of a chunk's voxel data.

*functions:*

**bool <u>DataExists</u> ()**
Returns true if the voxel data for this chunk can be found on disk, else returns false.

**bool <u>RegionExists</u> ()**
Returns true if the region file corresponding to this chunk can be found on disk, else returns false.

**<u>LoadData</u> ()**
Loads the voxel data from disk. This will throw an error if the data can't be found, so make sure to call DataExists() first.

**<u>SaveData</u> ()**
Saves the voxel data to disk.

**(static) string <u>CompressData</u> (Chunk chunk)**
Compresses the voxel data of a specific chunk and returns it as a string.

**(static) <u>DecompressData</u> (Chunk chunk, string data)**
Decompresses chunk data from a previously compressed string, and loads it into the chunk's VoxelData array.

**(static) <u>SaveAllChunksInstant</u>**
Immediately saves the voxel data of all active chunks to hard drive.

## VoxelEvents

Inherit from this class in order to create custom events.
The events have a voxelInfo data as argument which contains the voxel index and the chunk of the voxel which called the event, and in some cases the index of one adjacent voxel as well (in case of VoxelRaycast, the adjacent index will be of a voxel corresponding to the block face hit by the raycast).

**OnMouseDown (int mouseButton, VoxelInfo voxelInfo)**
If Engine.SendCameraLookEvents is enabled: Called when the main camera is looking directly at a block and a mouse button is pressed down.
If Engine.SendCursorEvents is enabled: Called when the cursor is on top of a block and a mouse button is pressed down.
voxelInfo.adjacentIndex will contain the voxel adjacent to the face being looked at.

**OnMouseUp (int mouseButton, VoxelInfo voxelInfo)**
If Engine.SendCameraLookEvents is enabled: Called when the main camera is looking directly at a block and a mouse button is released.
If Engine.SendCursorEvents is enabled: Called when the cursor is on top of a block and a mouse button is released.
voxelInfo.adjacentIndex will contain the voxel adjacent to the face being looked at.

**OnMouseHold (int mouseButton, VoxelInfo voxelInfo)**
If Engine.SendCameraLookEvents is enabled: Called when the main camera is looking directly at a block and a mouse button is being held down.
If Engine.SendCursorEvents is enabled: Called when the cursor is on top of a block and a mouse button is being held down.
voxelInfo.adjacentIndex will contain the voxel adjacent to the face being looked at.

**OnLook (VoxelInfo voxelInfo)**
If Engine.SendCameraLookEvents is enabled: Called when the main camera is looking directly at a block.
If Engine.SendCursorEvents is enabled: Called when the cursor is on top of a block.
voxelInfo.adjacentIndex will contain the voxel adjacent to the face being looked at.

**OnBlockPlace (VoxelInfo voxelInfo)**
Called when a block is placed using the Voxel.PlaceBlock function.

**<u>OnBlockPlaceMultiplayer</u> (VoxelInfo voxelInfo, NetworkPlayer sender)**
Called in addition to the previous function when a block place is sent from server. The sender variable stores the NetworkPlayer that originally performed the change.

**<u>OnBlockDestroy</u> (VoxelInfo voxelInfo)**
Called when a block is destroyed using the Voxel.DestroyBlock function.

**<u>OnBlockDestroyMultiplayer</u> (VoxelInfo voxelInfo, NetworkPlayer sender)**
Called in addition to the previous function when a block destroy is sent from server. The sender variable stores the NetworkPlayer that originally performed the change.

**<u>OnBlockChange</u> (VoxelInfo voxelInfo)**
Called when a block is changed using the Voxel.ChangeBlock function.

**<u>OnBlockChangeMultiplayer</u> (VoxelInfo voxelInfo, NetworkPlayer sender)**
Called in addition to the previous function when a block change is sent from server. The sender variable stores the NetworkPlayer that originally performed the change.

## UniblocksClient

Sends requests for voxel data and voxel changes to a server.

Functions:

**SendPlaceBlock (VoxelInfo voxelInfo, ushort data)**
Sends an RPC containing a voxel change to the server, which will then redistribute it to all players. This function is called from Voxel.PlaceBlock and Voxel.DestroyBlock.

**SendChangeBlock (VoxelInfo voxelInfo, ushort data)**
Sends an RPC containing a voxel change to the server, which will then redistribute it to all players. This function is called from Voxel.ChangeBlock.

**[RPC] ReceiveVoxelData (int chunkx, int chunky, int chunkz, byte[] data):**
An RPC sent by the server containing voxel data requested by the client. The voxel data is sent in the **data** byte array. Upon arrival the data is decompressed and passed to the chunk with the index of (chunkx,chunky,chunkz).

**[RPC] ReceivePlaceBlock (NetworkPlayer sender, int rawVoxelIndex, int chunkx, int chunky, int chunkz, int data)**
An RPC containing a voxel change, sent by the server to all connected players (including the original sender). **sender** is the player who originally performed the voxel change, and the other variables store the location and voxel id of the change.

**[RPC] ReceiveChangeBlock (NetworkPlayer sender, int rawVoxelIndex, int chunkx, int chunky, int chunkz, int data)**
An RPC containing a voxel change, sent by the server to all connected players (including the original sender). **sender** is the player who originally performed the voxel change, and the other variables store the location and voxel id of the change.

## <u>**UniblocksServer**</u>
Acts as a multiplayer server, sends voxel data and voxel changes to other players.

*Properties:*

**bool EnableDebugLog**
If true, most of the network events on the server will be printed in the console.

**float AutosaveTime**
If greater than 0, the server will automatically save the world periodically. The value of the property determines the autosave interval (in seconds).

*Functions:*

**[RPC] <u>SendVoxelData</u> (NetworkPlayer player, int chunkx, int chunky, int chunkz)**
A voxel data request sent by a client. The server will spawn the requested chunk (if it's not already spawned), compress it's data, and send it back to the client (through the UniblocksClient's ReceiveVoxelData RPC).

**[RPC] <u>ServerPlaceBlock</u> (NetworkPlayer sender, int rawVoxelIndex, int chunkx, int chunky, int chunkz, int data)**
Receives a voxel change sent by a client. By default, the server simply redistributes the change to all connected players. This is a good place to check whether the block change is valid before accepting it and sending to other players.

**[RPC] <u>ServerChangeBlock</u> (NetworkPlayer sender, int rawVoxelIndex, int chunkx, int chunky, int chunkz, int data)**
Receives a voxel change sent by a client. By default, the server simply redistributes the change to all connected players. This is a good place to check whether the block change is valid before accepting it and sending to other players.

**(private) <u>DistributeChange</u> (NetworkPlayer sender, int rawVoxelIndex, int chunkx, int chunky, int chunkz, int data, bool isChangeBlock)**
Sends a voxel change to all connected players (including the server). Checks whether the player is within range of the change if Engine.TrackPlayerPosition is enabled on the server.

**[RPC] <u>UpdatePlayerPosition</u> (NetworkPlayer player, Index chunkIndex)**
Sent by the client to tell the server it's current player position (as a chunk index). Updates the server's PlayerPositions array.

**[RPC] <u>UpdatePlayerRange</u> (NetworkPlayer player, int range)**
Sent by the client to tell the server it's chunk spawn distance setting. Updates the server's PlayerChunkSpawnRanges array.