Runtime Worlds and Active Grids

By Kyle Gillen

(Last Updated 11/4/15)

Runtime vs Inspector Components

- Inspector Worlds and Active Grids are fully realized as components on active game objects in your scene, which you add and setup in the Unity Editor.
- The Component Manager automatically detects their existence and initializes them when the Component Manager is initialized.
- They provide a simple, script free way to create dynamic worlds.

Runtime vs Inspector Components

- Runtime worlds/active grids are created and initialized at runtime via scripting.
- The base settings of each runtime world/active grid is based on a prototype.
- A prototype is a world/active grid component placed on a deactivated game object and referenced by the Component Manager via the inspector (more on this later).

Why use Runtime Components?

- The most common situation in which you'll be using a runtime component is when your main player (the one used to drive the dynamic loading) cannot be assigned in the inspector.
- This is common in multiplayer games where your player is created at runtime via some special networking method, though other situations may also call for a runtime player.

Why use Runtime Components?

- Creating non persistent Worlds/Active Grids
  - All Inspector Worlds/Active Grids are persistent (unless they are destroyed), meaning their state is saved and loaded whenever the Component Manager's Save and Load methods are called.
  - Runtime Worlds/Active Grids, however, can be non persistent so their state is not saved (this is useful if you want to save/load the state of your worlds/active grids yourself, rather than relying on the built in save/load methods).

Why use Runtime Components?

- Allowing the user to choose settings
  - You may not want to have a default World/Active Grid that is loaded at the start of your game.
  - Using runtime Worlds/Active Grids, you can have the user choose from a variety of settings and than select a World/Active Grid from a list of prototypes based on the user's choices.
- If you find additional uses for runtime components, please let me know!

Creating the Prototypes

- To utilize runtime components, the first thing you need to do is create one or more prototypes.
- Note that each prototype should be on its own game object (an empty game object can be used), **and that game object should be in a deactivated state** (otherwise the Component Manager will detect the existing of the component and treat is as an inspector World/Active Grid).
- Every prototype should be on its own game object (necessary for dragging its reference onto the appropriate field of the Component Manager's inspector).

Creating the Prototypes

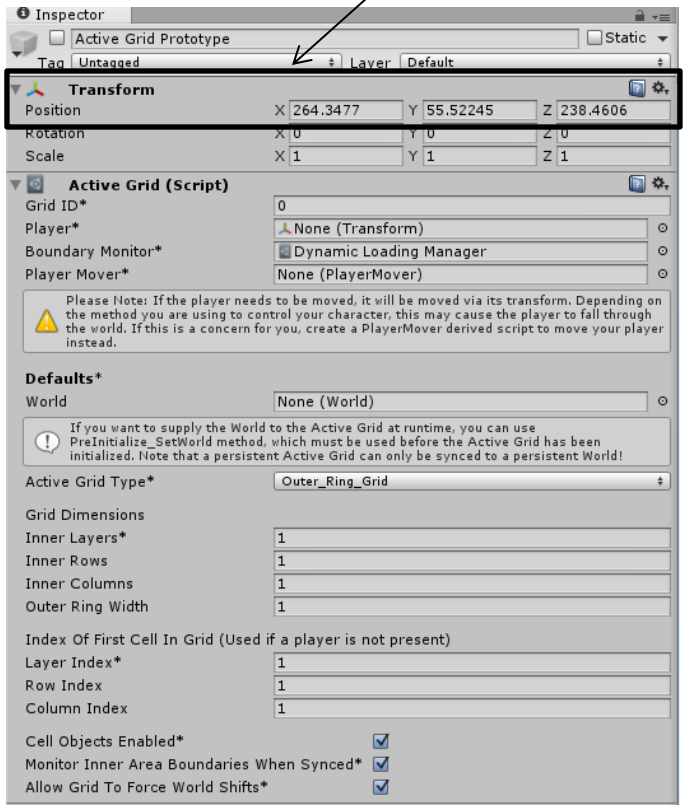
- Once you have a prototype, adjust it's settings to suit whatever type of Active Grid/World you want to create at runtime.
- For Active Grid prototypes, The Player, Player Mover and World fields can be left blank, though you will need to set them when creating your runtime Active Grid (more on this later).
- The Grid ID field can be ignored, as it will not be used (an ID that does not conflict with any other ID of the same component type will automatically be generated at runtime for the runtime component).

Creating the Prototypes

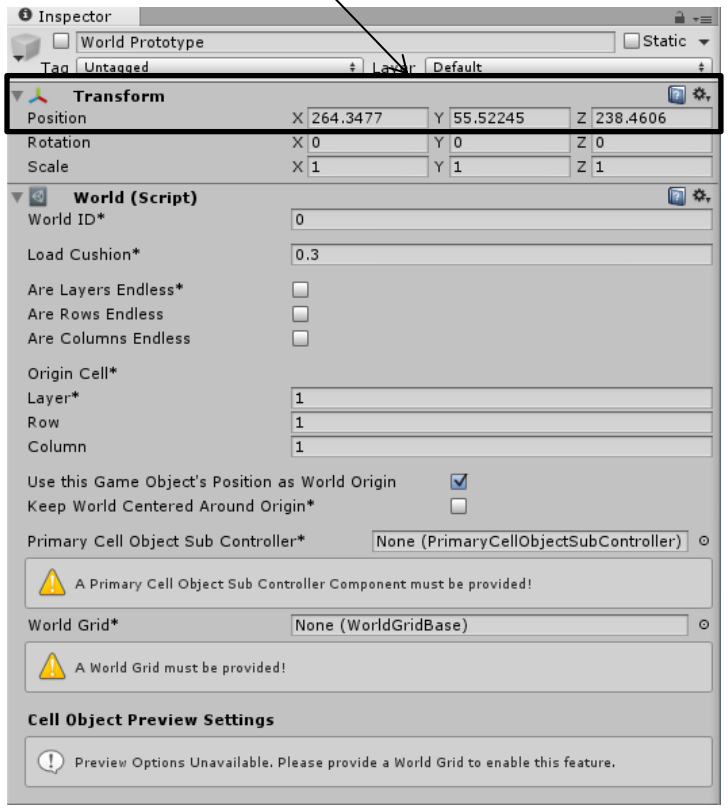
- When creating your runtime World, you have the option to specify and alternate World Origin and Origin Cell. If you're planning on providing these details at runtime, you can ignore these fields on the World prototype. Otherwise, make sure to set them!
- Ensure a Primary Cell Object Sub Controller and World Grid are provided, and that all other fields (besides the World ID) are set.

Creating the Prototypes

Position does
Not matter



Position matters if “Use this Game Object’s Position as World Origin” is checked and no alternate World Origin is provided during runtime creation of the World.

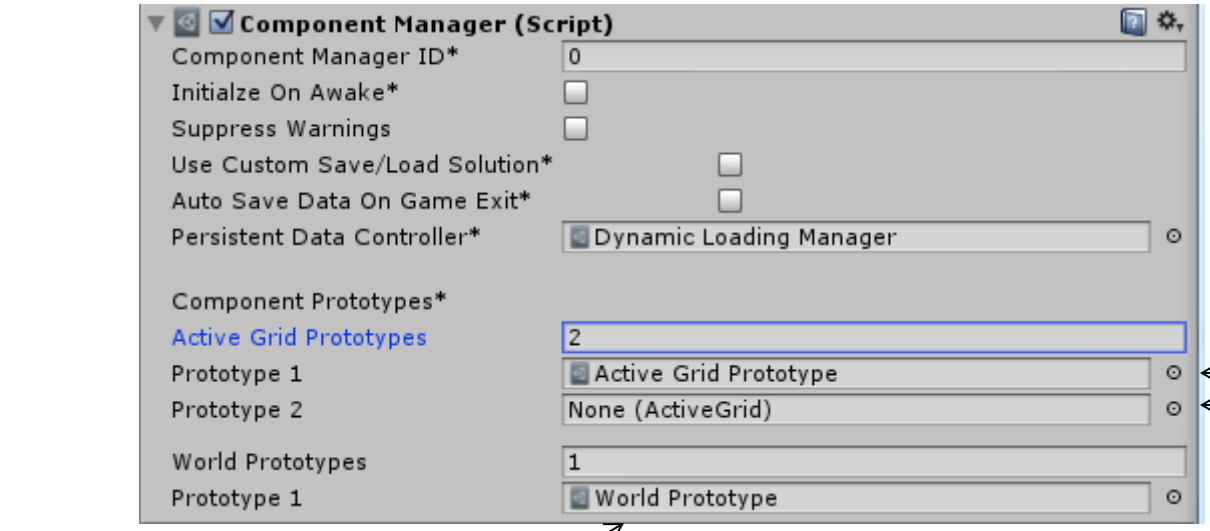


Referencing the Prototypes

- Once you've created the prototypes, you need to reference them via the Component Manager's inspector.
- Set the "Active Grid Prototypes" value to however many Active Grid Prototypes you have.
- Set the "World Prototypes" value to however many World Prototypes you have.

Referencing the Prototypes

- A number of additional fields will be displayed based on the values you entered.
- Drag the game objects that contains the correct prototypes onto each field.



Creating the Components at Runtime

- Things to keep in mind before creating a runtime component:
  - When a create method is called, the Component Manager will run through it's load process.
  - If using a custom save/load solution and you need to load data manually, you should call the Load method before this load process is run (i.e., before creating a runtime component).
  - You will need to manually load the cells of a runtime created Active Grid after creation (more on this later).

Creating the Components at Runtime

- Things to keep in mind before creating a runtime component:
  - Generally speaking, you will want to create the World/Active Grid from another scripts Awake method (and also load the cells of the Active Grid). This will ensure the cell objects are loaded before the first Update method is called (you can ignore this directive if creating Worlds/Active Grids mid game [i.e., during the update cycle]; in these instances, you'll want to load the objects of the Active Grid gradually).

Creating the Components at Runtime

- To create an Active Grid or World, create a new script or modify an existing one. First things first, add a using statement of the form using DynamicLoadingKit; to the top of your script. This will allow you to more easily reference all of the classes of the kit.

Creating the Components at Runtime

- Next, get a reference the Component Manager object in the scene. There are many ways to do this; simply using the line ComponentManager componentManager = FindObjectOfType<ComponentManager>(); should work, though you can also expose a ComponentManager field in the inspector and drag the reference onto the field manually.

Creating the Components at Runtime

- To create an Active Grid:
  - Use the CreatePersistentActiveGrid or (less common) CreateNonPersistentActiveGrid methods, which are instance methods of the ComponentManager class (this is why we needed a reference to the component manager object).
  - The prototypeToConstructGridFrom argument is the value found in the inspector of the ComponentManager under Active Grid Prototypes.

Creating the Components at Runtime

- To create an Active Grid:
  - For example, if you want to create a grid that utilizes the settings found on Prototype 2, pass in 2 as the value for prototypeToConstructGridFrom.
  - All other parameters are optional (null or true will be used for the value if none is provided)
  - You can find information about both methods and their parameters here.

Special Note*

If you pass in an alternate player when creating a non persistent Active Grid and are planning on using a PlayerMover component, **you must also pass in the PlayerMover you wish to use!** Otherwise, the PlayerMover will be set to null (even if your Active Grid prototype has a PlayerMover object assigned).

Creating the Components at Runtime

- To create a World:
  - Use the CreatePersistentWorld or (less common) CreateNonPersistentWorld methods, which are instance methods of the ComponentManager class (this is why we needed a reference to the component manager object).
  - The prototypeToConstructWorldFrom argument is the value found in the inspector of the ComponentManager under World Prototypes.

Creating the Components at Runtime

- To create a World:
  - For example, if you want to create a world that utilizes the settings found on Prototype 2, pass in 2 as the value for prototypeToConstructWorldFrom.
  - All other parameters are optional (null or true will be used for the value if none is provided)
  - You can find information about both methods and their parameters here.

Additional Info

- As stated previously, you will need to manually load the cells for a runtime created Active Grid.
- For this purpose, you can use the TryLoadCellObjectsASAP, TryLoadCellObjects or TryLoadCellObjectsAndWaitForLoadToComplete methods of the ActiveGrid class.
- The ASAP method can be used to ensure the objects are loaded before the first Update cycle is run, though this will only work if called from another scripts Awake method (and that script precedes the Component Manager in the compilation order).

Additional Info

- The other two methods will load the objects over multiple frames and are suitable for use mid-game, when performance matters.
- The TryLoadCellObjectsAndWaitForLoadToComplete method can be called as a coroutine, allowing you to yield on it. This will allow you to sequence some other action (such as removing a loading screen) to run only after the objects have been fully loaded.

Additional Info

- It is possible to link a runtime World to a runtime Active Grid. Simply create the World first, then pass a reference to the newly created world into the Active Grid creation method (whichever one you use).

Additional Info

- A persistent World can be synced to by both persistent and non persistent Active Grids.
- On the other hand, a non persistent World can only be synced to by a non persistent Active Grid.
- Without this rule, it would be possible for the link between an Active Grid and World to be broken between sessions (if the AG was persistent but World was not).

Additional Info (Player Assignment)

- If you wish to assign a player to an Active Grid at runtime, you will need to create a non persistent Active Grid.
- This ensures that the link between active grid and player will not fail silently (since the Dynamic Loading Kit will not be able to automatically assign the player when a game is loaded).

Additional Info (Player Assignment)

- Because the grid is non persistent, its data will not be saved automatically when the Component Manager's Save method is called.
- **You will need to handle saving/loading manually**!
- Thankfully, this is quite simple.
- When you wish to save the grids data, use the ActiveGrid's GetPersistentStringSaveData and save the string that is returned.

Additional Info (Player Assignment)

- Loading the save data is as simple as passing in this string as the persistentDataToSetStateFrom argument when creating the Active Grid during the next session (using the CreateNonPersistentActiveGrid method).

Persistent vs Non Persistent

- Here's some additional information that might be useful regarding persistent and non persistent Worlds/Active Grids.
  - Persistent Worlds/Active Grids are created once. Every time you load a Component Manager's save data, it recreates persistent Worlds/Active Grids automatically, so you don't have to.
  - For persistent components, you don't have to manually call the GetPersistentStringSaveData method. The Component Manager does this automatically when it's Save method is called.

Persistent vs Non Persistent

- Non persistent components need to be manually recreated every time the game is loaded (assuming you want them to be).
- Both Active Grids and Worlds have a GetPersistentStringSaveData method, allowing you to manually save/load non persistent Worlds/Active Grids at runtime.