Mod SDK changelog

This is the changelog for the Core Keeper Mod SDK. It will contain changes to the SDK itself as well as internal changes to the project that isn't covered in the regular changelog.

It is still very much recommended to also read what is in the changelog for the Core Keeper application on Steam: https://store.steampowered.com/news/app/1621690?updates=true

1.0.1

- The following components have been merged into one component named DropLootAuthoring:
  - DropsLootAuthoring
  - DropsLootFromLootTableAuthoring
  - SeasonalLootAuthoring
  - DropsLootOnUseAuthoring
  - DropsLootWhenDamagedAuthoring
- HealthRegenerationAuthoring has been merged into HealthAuthoring.
- AnimationOrientationAuthoring has been merged into AnimationAuthoring.
- GrowingAuthoring options has been merged into SeedAuthoring and PlantAuthoring.
- LastAttackerAuthoring has been merged into EnemyAuthoring.
- Removed canBeHitAtSameTimeAsEnemy bool in MineableAuthoring as it was only used by PlayerGrave.
- Chests no longer need to be adjacent to the crafting station to pick materials from the chests but can be anywhere within a 10 tiles radius of the crafting station. The code that handles this has been reworked to function very differently from before.
- The ClientInput component isn't set outside of the prediction loop anymore. If you need to fetch or modify some input value outside of the prediction loop, fetch the data from ClientInputData. See updated InvertPlayerMovementSystem in SystemExample.

1.0

- Core Keeper is now available on consoles as well as PC. Modding is only available on PC, and we will most likely never support mods on consoles in the same way we do on PC. The platform owners (i.e. Sony, Microsoft and Nintendo) have strict requirements for all games that are released on their consoles, and there is no reasonable way to comply with them if players can modify the game to any meaningful extent.
- There is a known issue with mods not working properly on non-steam platforms due to some differences in the game initialization flow. We are working on a fix for this.
- Player controller has been changed to be server authoritative instead. This was also mentioned in the game changelog, but this is worth bringing up here as well as one of the biggest refactorings we've done. This means the player controller consists of a number of ECS systems. The code is still similar in places, but for example PlayerController have more static functions instead called from ECS.

- Apart from the player most enemies and objects are interpolated (shown with a server delay to avoid client-side computation). Some player interactions can still make them be predicted instead to improve responsiveness which means that a bunch of logic are now supporting prediction meaning a lot more systems are designed to run both on server and client.
- Completely new interaction system which is triggered from ECS. This part had to be completely remade instead of ported. Callbacks on EntityMonoBehaviours like OnUse should still be called in the same way via the new system.
- All SinglePugMap methods have been refactored to use world coordinates instead of render coordinates. Use EntityMonoBehaviour.ToWorldFromRender or Manager.camera.RenderOrigo to convert from render to world coordinates.
- RenderOrigo has been renamed to the more accurate RenderAnchor. You can access these using Manager.camera.GetRenderAnchor. If you don't need it for the full session, call Manager.camera.ReturnRenderAnchor to return it so that it can be cleaned up.
- PugSprite has been completely removed. This component was used to create MeshRenderers from SpriteRenderer to get the batching support from Unity, but has since been replaced mostly by using SpriteObject instead of SpriteRenderer.
- Even more enemies and objects have been converted from SpriteRenderer to using our own sprite rendering solution SpriteObject.
- More parts of the code is now using Burst, especially after the player controller rework. This is mainly done to improve performance. We are aware that Burst-compiled code is a lot harder to patch, and we are already looking into ways to make this easier for modding.
- The new world generation uses a new GPU-driven framework to generate the terrain. This data is processed by an ECS system to create the ECS data that describes a world. The rules for this processing are described in the `TileTypeMapping` asset. You can also get direct access to the data produced by world generation shaders via the helper struct created by `Manager.worldgen.CreateProceduralDataRequester`. Each position in-game is described by a RGBA pixel. The current format is as follows:
    - Red channel: encodes the tile type. Use `CoreKeeperWorld.GetTileType` to get a more meaningful enum value.
    - Green channel: encodes the biome. Use `CoreKeeperWorld.GetBiome` to get a more meaningful enum value.
    - Blue channel: bit 0 (least significant) encodes if the position has only floor and no wall, bit 1 if there is a roof hole, and bit 3 if the Great Wall occupies this position. All other bits are currently unused.
    - Alpha channel: bits 0-2 encodes the ore type at this position. All other bits are currently unused.
- SerializeWorldSystem has been reworked to only load in things to the "live" world when the player is close enough (200 tiles). Previously, entities outside this range would be disabled but would still be kept in the live world. Some entities are always loaded like electricity, automation objects and bosses. The unloaded objects are kept in a separate ECS world (SerializeWorld) with a minimal representation. This world is what the world save file contains.

- AnimationAuthoring script was moved inadvertently, if you used this in any custom GameObject, you'll have to reassign it. We try to not break references like that, but this one slipped the cracks in some refactor.
- WindUpAuthoring removed and same settings are moved to SecondaryUseAuthoring.
- Custom EntityMonoBehaviours aren't updated as automatically as before, you might have to call some functions manually via GraphicalUpdate (see EnemyExample).
- Addressables are used in various places to load and unload assets to minimize memory footprint.
- A bunch of TimerSimple fields have been changed to use TickTimer. The typical reason is that this timer needs to be synchronized between server and client meaning that it needs to use NetworkTick instead of a regular floating point value to measure time.

This update contains lots of other small internal changes. Too many to list all of them here, but feel free to ask in the Discord and we'll do our best to answer your questions.

0.7.5

- Updated Unity version to 2022.3.20f1. This is mostly in preparation for 1.0 and while it means there will be some changes we don't expect anything noticeable but you will need the new version for the Mod SDK as well.
- For performance reasons, the update frequency of the ECS server world has been decreased from 30 to 20 updates per second. Any modded ECS code that uses the number of elapsed ticks as a metric for when to update will consequently run slower and may need to be adjusted. Similarly, MonoBehaviour.FixedUpdate is now called 30 times per second, down from 60 previously. Client world systems, objects implementing IGraphicalObject, and MonoBehaviour that only call the non-fixed update methods are unaffected, as they are all still called once per frame.
- All EntityUtility methods that previously accepted a ComponentType parameter have been refactored to instead take the relevant IComponentData type as a generic type parameter. Previously, each call to these functions resulted in a C# reflection lookup, which accumulated to have a significant performance impact.

0.7.4

- The mods that are loaded from mod.io will have an extra check that verifies the current version is supported by the mod, indicated by the Game Version tag. If this isn't set by the mod to the current version, it will still be possible to load the mod, but the player will get a warning first. This means it is more important to set this version tag correctly than before.
- The HealthChangeBuffer now has an entity field which indicates which entity is affected by the change and is stored in a singleton instead of on each entity.
- Expanded usage of the PugProperties system to avoid storing as much data on entities which instead can be shared. Each object entity now has an

ObjectPropertiesCD component that can be used to look up shared data for that object. Added SetProperty functions to PugConverter where this data can be set.
- Some of the tile functions in Manager.multiMap have been replaced with Manager.multiMap.GetTileLayerLookup which gives you a job-compatible struct which can be used to access the same functions.

0.7.3

- You can choose a dedicated server install when choosing an install location in the SDK. Mods that are loaded via the SDK will also be detected properly by any connecting client even if the client uses the mod from mod.io. There is no automatic install of any missing mod that isn't installed via mod.io though.
- We are retiring the Translation component in favor of the LocalTransform component introduced in ECS 1.0. The main motivation for this change is to improve performance, as the Translation and LocalTransform must currently be kept in sync by copying the values back and forth. The Translation component will still be available in the project in order not to break serialized worlds, but no runtime entities are expected to have this component. This is one of the most commonly used components, but the following steps made the upgrade process fairly pain-free when we upgraded our own code:
    1. Use the following case-sensitive find-and-replace patterns. This took care of the vast majority of our usages.
        - `Translation translation` -> `LocalTransform transform`
        - `translation.Value` -> `transform.Position`
        - `, translation) in` -> `, transform) in`
        - `<Translation>` -> `<LocalTransform>`
        - `typeof(Translation)` -> `typeof(LocalTransform)`
    2. Fix all resulting compiler errors. The most common cause is that the patterns don't catch variable names in some cases.
    3. Replace all usages of `new Translation` with calls to `LocalTransform.FromPosition`.
    4. Find and replace any remaining usages of the Translation component.

    Unity's official ECS 1.0 upgrade guide might provide useful information on the new LocalTransform component: https://docs.unity3d.com/Packages/com.unity.entities@1.0/manual/upgrade-guide.html#update-transforms-in-your-project.
- Completely removed the Rotation component, as rotation of entities is not used anywhere in Core Keeper. Setting the Rotation field on LocalTransform also has no effect.
- EntityMonoBehaviour is the base class for all of our graphical objects, and over the years it has grown to an unmanageable size as it has to support all our objects' specific use cases. We are making a push to return to a more component-based approach. Our aim is to make it less daunting and error-prone to add new graphical objects to the game. For this update specifically, we have:
    1. Added support for multiple IGraphicalObject components on the same object. All such components **at the object root** are picked up and have their methods invoked as appropriate.

    2. Added separate interfaces for components that only provide spawn/despawn behavior. Prefer using these when possible to avoid the overhead of empty GraphicalUpdate methods being called each frame.
    3. Added several new components that control aspects of graphical objects that were previously managed with settings on EntityMonoBehaviour. We encourage you to when possible use these components instead of the corresponding EntityMonoBehaviour settings. The old settings have not been removed, but they will not be developed further and may be removed at a future time. These components are listed below, for more information see their in-editor tooltips and how they are used on existing assets. `SpriteSkinFromPaintColor`, `SpriteGradientMapFromPaintColor`, `SpriteVariationFromEntityDirection`, `SpriteVariationFromEntityVariation`, `SpriteSkinFromEntityAndSeason`, `EnableEffectsForEntityVariation`, `OffsetFromEntityDirectionOrVariation`
- The hierarchies of a lot of our graphical objects have also had an overhaul to improve performance. In most cases this means porting an older object to use our in-house sprite rendering solution (SpriteObject). Apart from this we have also done lots of small changes to increase performance in various ways, see the performance section of the 0.7.3 changelog for the Core Keeper application for more information on this.

0.7.2

- Upgraded Unity version to 2022.3.10f1.
- Upgraded DOTS/ECS to 1.0. See links for upgrade instructions. The Translation component should still work as before, and any baking changes aren't relevant because we are using our own baking system (PugConversion).
    - https://docs.unity3d.com/Packages/com.unity.entities@1.0/manual/upgrade-guide.html
    - https://docs.unity3d.com/Packages/com.unity.netcode@1.0/manual/upgrade-guide.html
    - https://docs.unity3d.com/Packages/com.unity.physics@1.0/manual/upgrade-guide.html
- Burst version upgraded to 1.8.9.
- A lot of ECS systems have been converted to use ISystem instead of SystemBase to improve performance.
- Added an option for you to bypass the security check, adding a notice about it to the user: "Caution: 'Elevated Access' mods have increased access to resources outside the game like user files and internet. For the best experience, install only from reputable sources." The reflection API might be removed in the future in favor of using this option instead.
- The Unity.Collections package is embedded in the SDK, but the only change is that we have added the old FixedString32/64 types which are still used for some serialized components. Changing these to the new versions would have broken serialization.
- The DOTS version uses a very different serialization format for ECS worlds. As a result there is a separate application in CoreKeeper_Data/StreamingAssets/Patcher

which is used to convert saves from before 0.7.2 to an interim format that the new version can read.