

UNREAL ENGINE

徹底解説！UE4を使ったモバイルゲーム開発における
コンテンツアップデートの極意！

GAME CREATORS CONFERENCE '18

Epic Games Japan 岡田和也



自己紹介

Epic Games Japan サポートエンジニア
岡田和也(おかげ @pafuhana1213)



最近のお仕事

EGJのモバイル担当したり
コソコソ色々してます

本日のお品書き

- 本日のお題を選んだ理由
- UE4のコンテンツアップデート用機能の紹介
- Battle Breakersにおける運用例

Unreal Engine 4

モバイル対応に力を入れてます！

Epic Games製モバイルタイトル

Battle Breakers

- ・約1年前に、iOS / Android版をリリース
- ・数百体のキャラクタが登場するRPG



Fortnite Battle Royale

- ・iOS版をリリース済。数ヶ月後にAndroid版も
- ・PC, コンソール版と同じゲーム内容で
クロスプラットフォームプレイが可能



数多くのモバイル向けの機能・最適化が
日々追加されます！



参考: 4.20でに入るモバイル向けの最適化

MOBILE IMPROVEMENTS IN 4.20

- Minimum static mesh LOD per-platform
- Minimum skeletal mesh LOD per-platform
- Hardware **occlusion** improvements
- RHI **thread** for OpenGL on Android
- HLOD tools and workflow optimization
- Audio quality** node
- Audio **variation culling**
- Audio per-platform **downsampling**
- Audio per-platform **compression quality**
- Shading model tweaks to better match PC
- Reflection capture brightness fix
- Landscape support for four layers
- Landscape **tessellation** improvements

- 2-byte **strings** on iOS and Android
- No memory cost for **unused LODs**:
 - Static meshes
 - Skeletal meshes
 - Material quality levels
 - Grass and foliage
 - High detail **components** and meshes
 - High detail emitters in **Cascade**
- Settings based on **device memory**
- Material memory reduction
- Editor scriptability for bulk asset changes
- Particle component pooling
- Material parameter collection update cost

[Optimizing UE4 for Fortnite: Battle Royale - Part 2 | GDC 2017 | Unreal Engine](#)



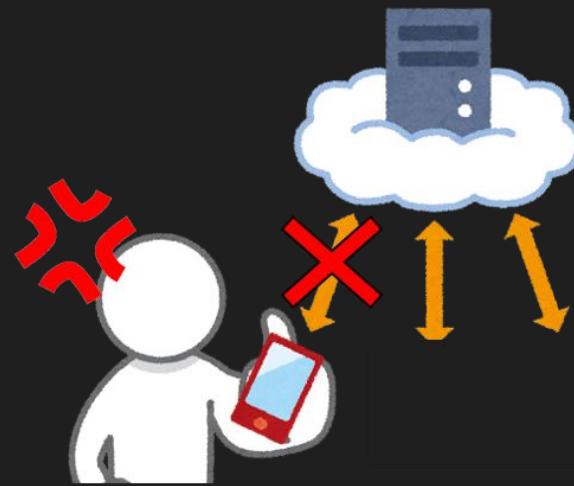
その中からコンテンツアップデートを
選んだのは何故？

モバイル開発において
コンテンツアップデートは
重要かつ難易度が高いから！

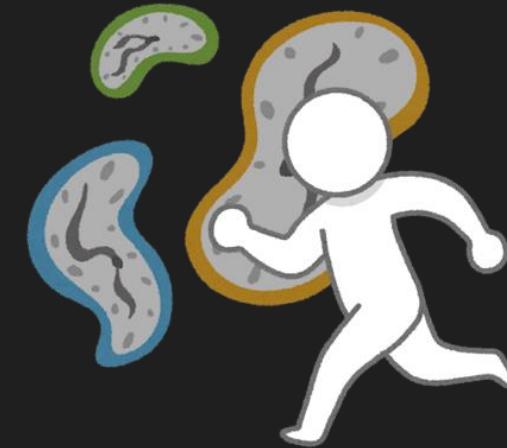
コンテンツアップデートの重要性

タイトルの寿命とユーザーの満足度に直結する部分！

- ・検証・設計が不十分な状態で作業を進めると
最後まで負債として抱えることに…



DLサイズによる負荷



人 / 時間的コスト

コンテンツアップデートの難しさ

プロジェクトによって最適解が異なる！

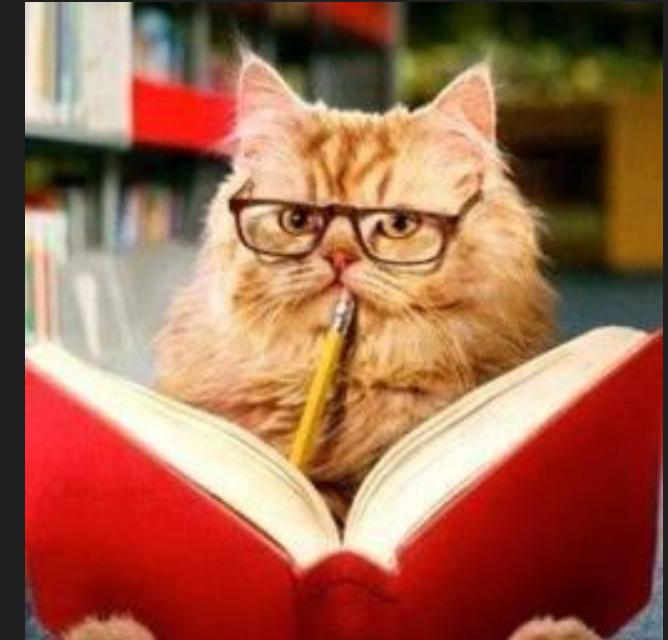
- ゲームの仕様
- 開発規模
- 用意可能な設備
- スケジュール
- 予算
- 予算
- 予算
- などなどなどなどなどなど



本講演の目的

コンテンツアップデートの検証・設計作業のお手伝い

- UE4のコンテンツアップデートに関する機能の紹介
 - 当時最新のUE4.19.0 準拠
 - Battle Breakersにおける運用事例の紹介
 - Battle Breakersを選んだ理由は後述



本日のお品書き

- 本日のお題を選んだ理由
- UE4のコンテンツアップデート用機能の紹介
- Battle Breakersにおける運用例



コンテンツアップデート用機能の紹介

- Chunk機能によるコンテンツ分割
- Chunk用パッケージの作成
- Chunkデータのダウンロード
- Chunkからのアセット取得

Chunkの日本語訳

- ・**大きいかたまり**
- ・厚切れ
- ・かなりの量
- ・たんまり
- ・どっさり
- ・(ずんぐりして)がっちりした人



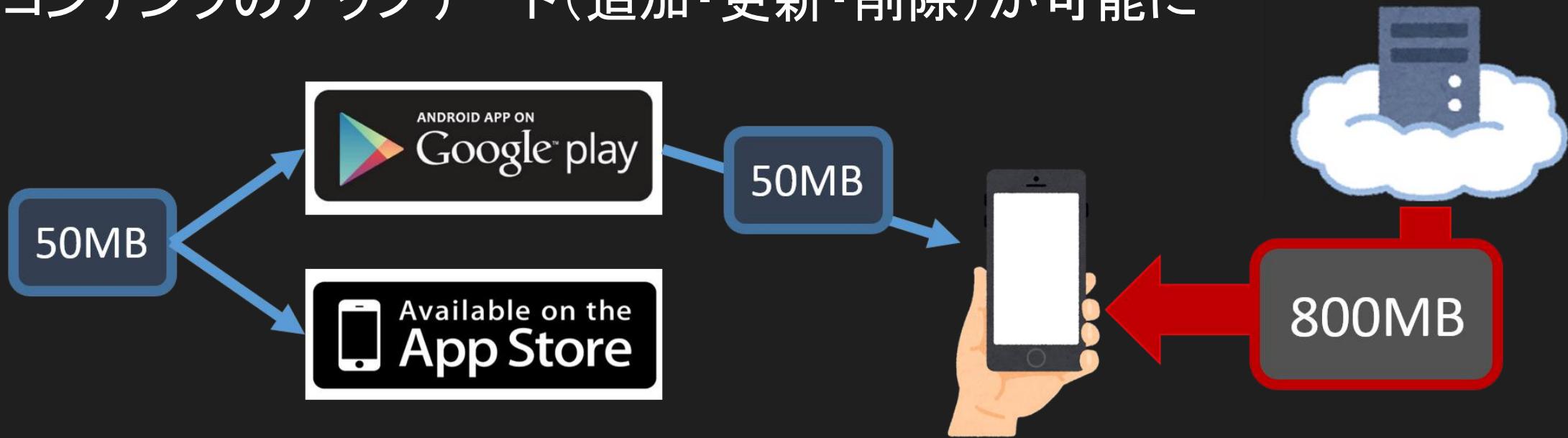
<https://ejje.weblio.jp/content/chunk>

コンテンツアップデート用機能の紹介

- Chunk機能によるコンテンツ分割
- Chunk用パッケージの作成
- Chunkデータのダウンロード
- Chunkからのアセット取得

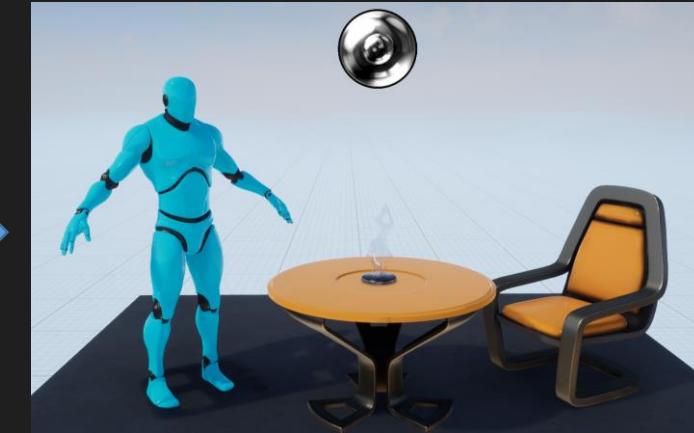
コンテンツ分割のメリット ①

- ・ストアからの初回ダウンロードサイズの削減
- ・ストアを介さない
コンテンツのアップデート(追加・更新・削除)が可能に



コンテンツ分割のメリット ②

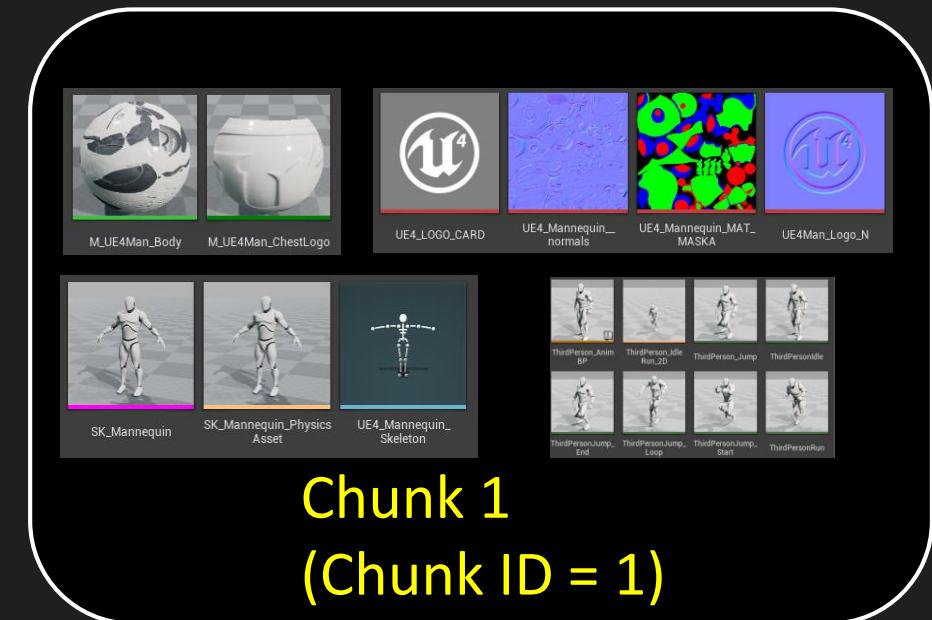
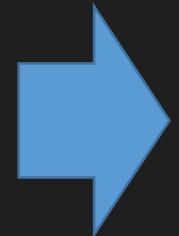
細かく分けることで
必要に応じた部分的なアップデートが可能に



Chunk機能によるコンテンツ分割

ゲームデータを複数のパッケージ(.pak)に分割

- 分割は 各アセットに設定された**Chunk ID**に基づく



Chunk IDは
どうやって設定するの？



Chunk IDを設定するには？

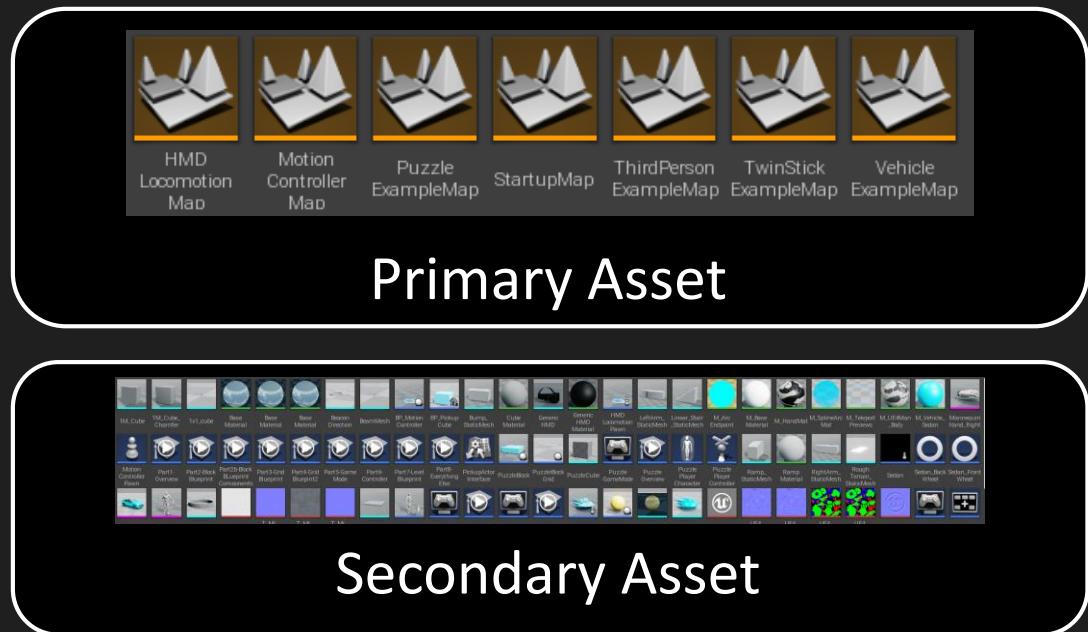
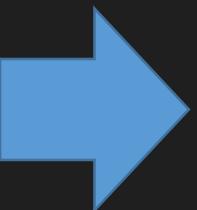
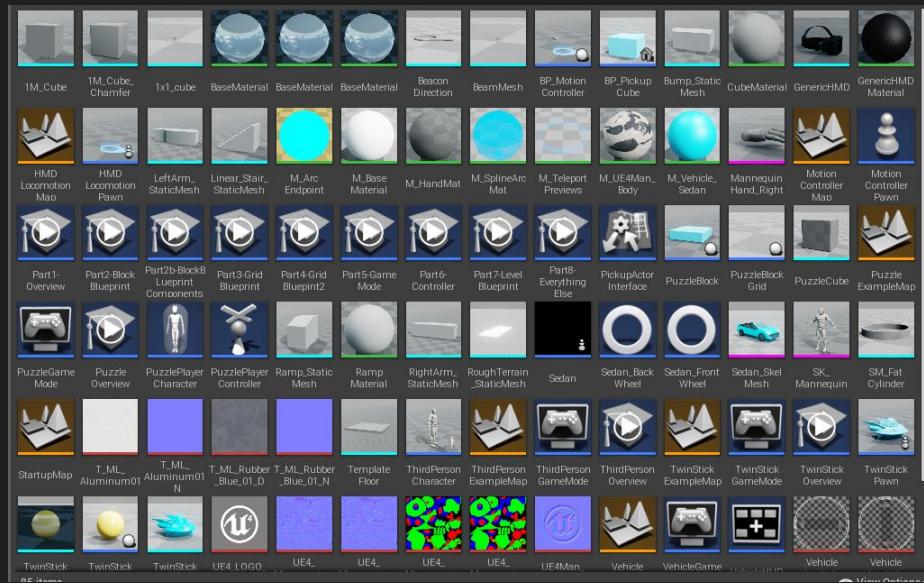
対象アセットを

設定したいChunk IDの情報を持つPrimary Asset として管理する
または、そのPrimary Asset の Secondary Asset として管理する

UE4のアセット管理システム（UE4.17～）

全アセットを Primary Asset と Secondary Asset に分類し

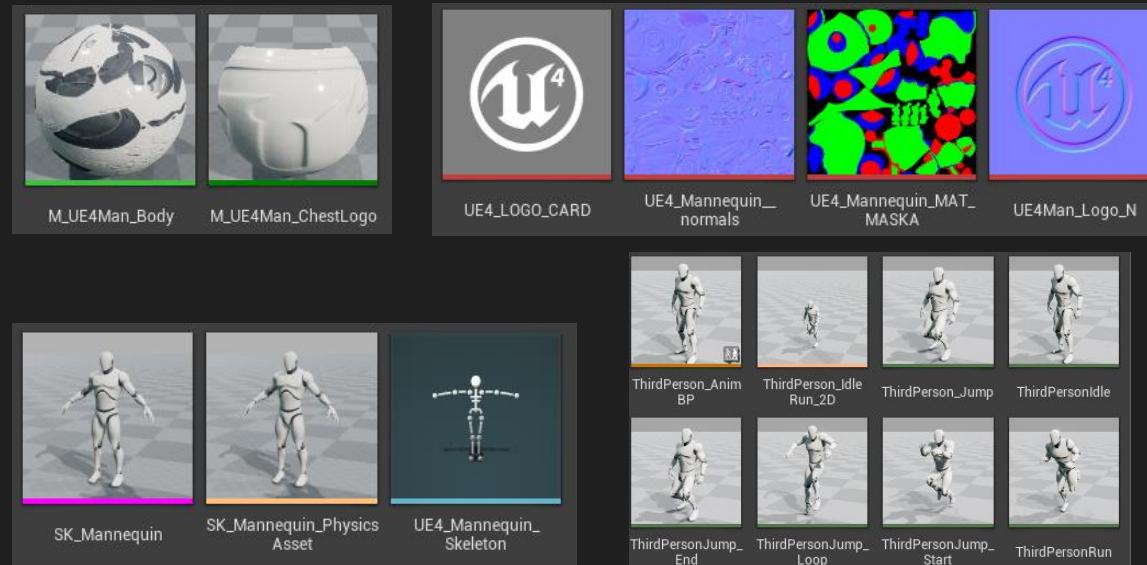
Asset Manager というシステムがそれらを管理



UE4のアセット管理システム（UE4.17～）

- Primary Asset :
Asset Managerによって直接管理されるアセット
- Secondary Asset :
Primary Assetsに紐づくアセット
(Primary Assetsが直接・間接参照しているアセット)
[公式ドキュメント:アセットの参照](#)

キャラクタのBP が Primary Assetの場合



Primary Asset

Secondary Asset

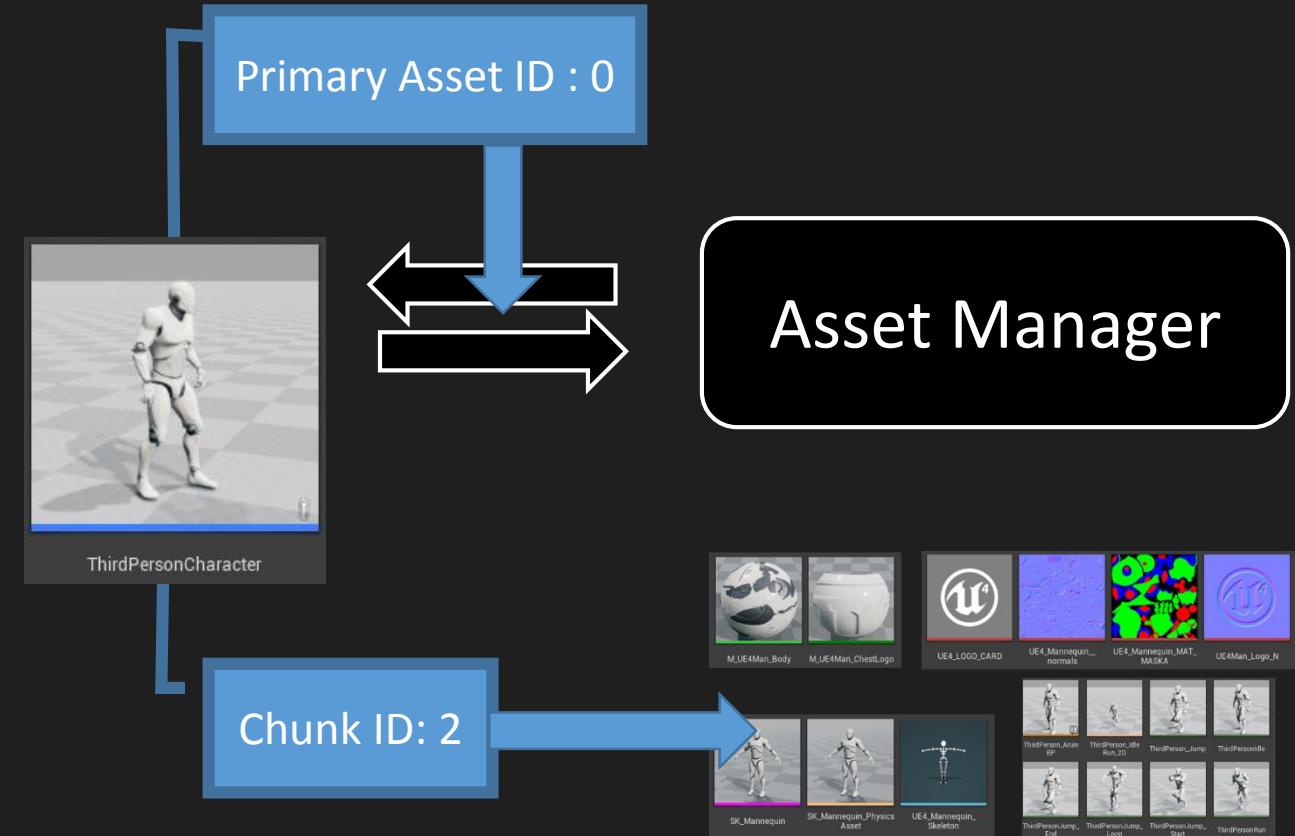
各 Primary Asset に割り当てる情報

Primary Asset ID

- Asset Managerの管理用

Chunk ID

- ・パッケージ時の分割 / 管理用
 - ・Secondary Assetにも
連鎖的に適用

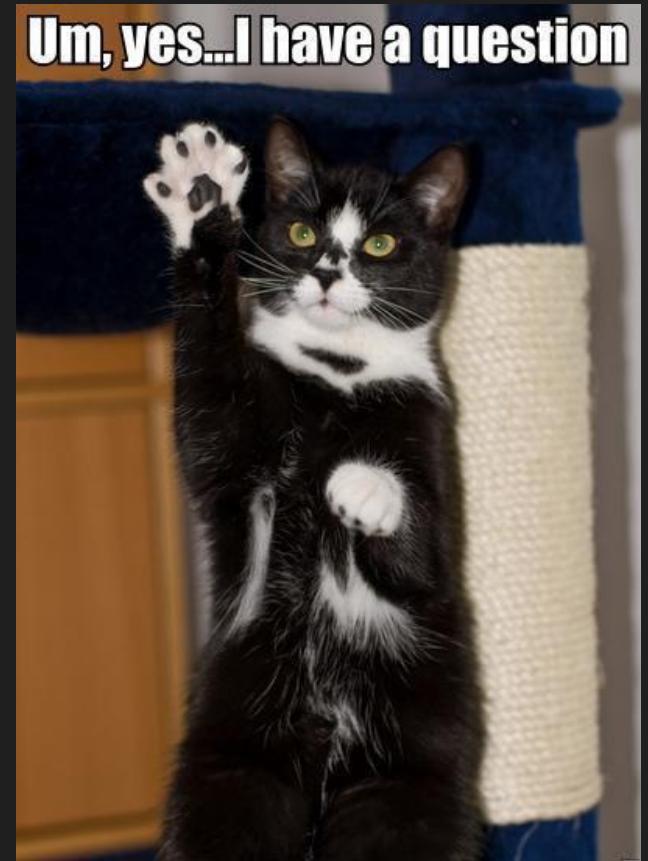


(6ページ前) Chunk IDを設定するには？

対象アセットを

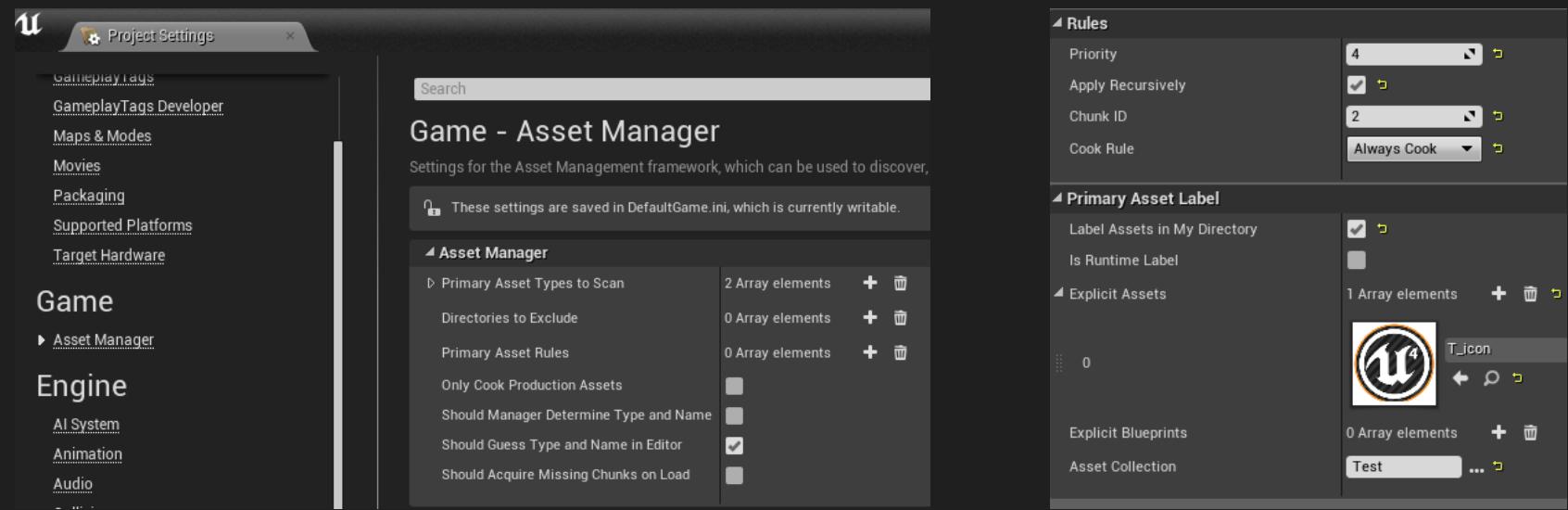
設定したいChunk IDの情報を持つPrimary Assetとして管理する
または、そのPrimary Asset の Secondary Assetとして管理する

対象アセットを
Primary / Secondary Assetとして
管理するにはどうすればいいの？



UE4が提供している機能

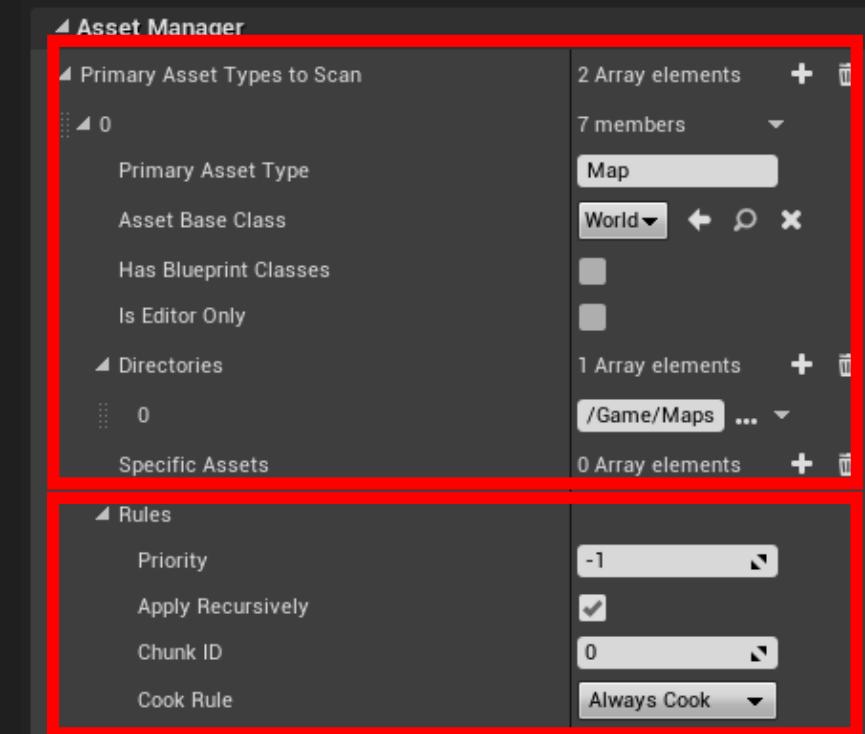
- Project Setting内のAsset Managerカテゴリ
- Primary Asset Label アセット



Project Settings / Asset Manager 編

Primary Asset Types to Scan

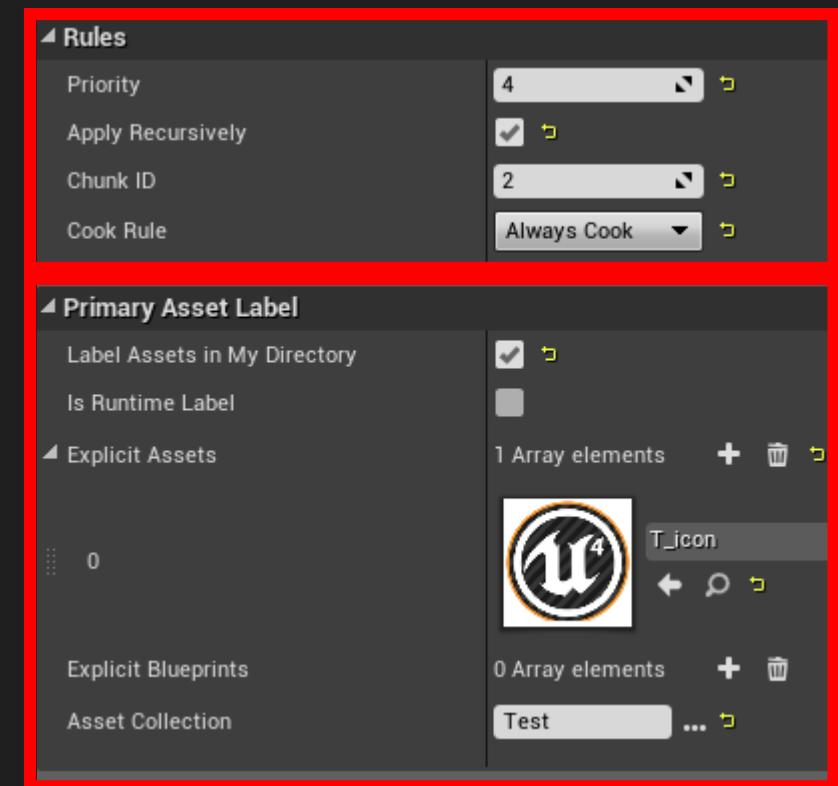
- Primary Assetとして管理するアセットを抽出する際のフィルタ設定
- 対象Primary Asset とそのSecondary AssetへのChunk IDの設定ルール



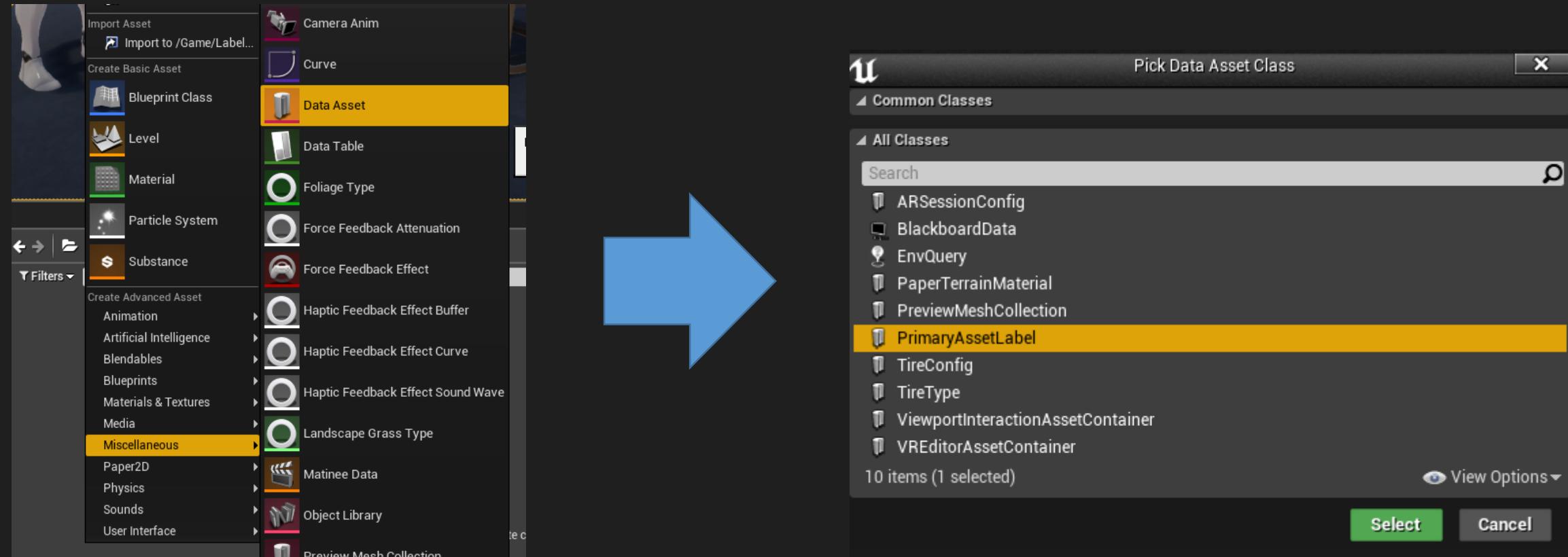
Primary Asset Label アセット編

Chunk IDの設定に特化したアセット

- このアセット自体はPrimary Asset
(デフォルト設定の場合)
- Secondary Assetを明示的に指定する機能
- 指定したSecondary Assetへの
Chunk IDの設定ルール



Primary Asset Labelアセットの作り方

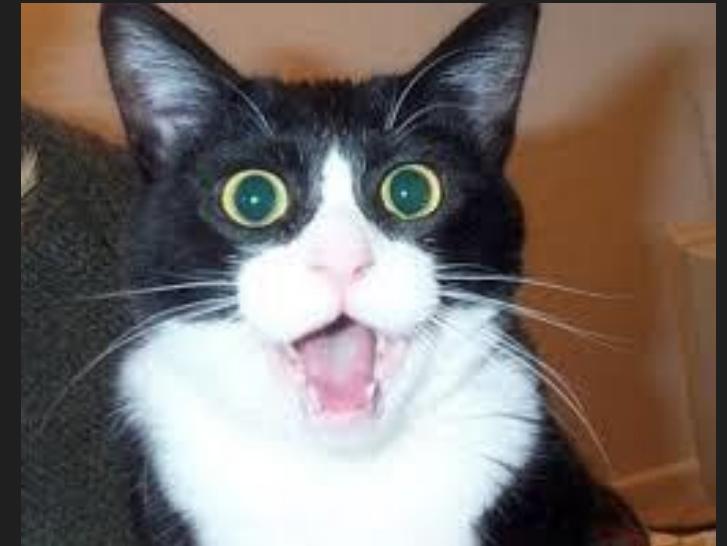


複数の設定(アセット)からChunk IDを指定されている場合

例 :

- Asset Manager からは Chunk ID = 2 で 設定
- Primary Asset Label からは Chunk ID = 3 で設定

対象のアセットは
Chunk 2, 3両方に含まれることに !



Chunk間のアセット競合を回避するために

Chunk間の親子関係を定義！

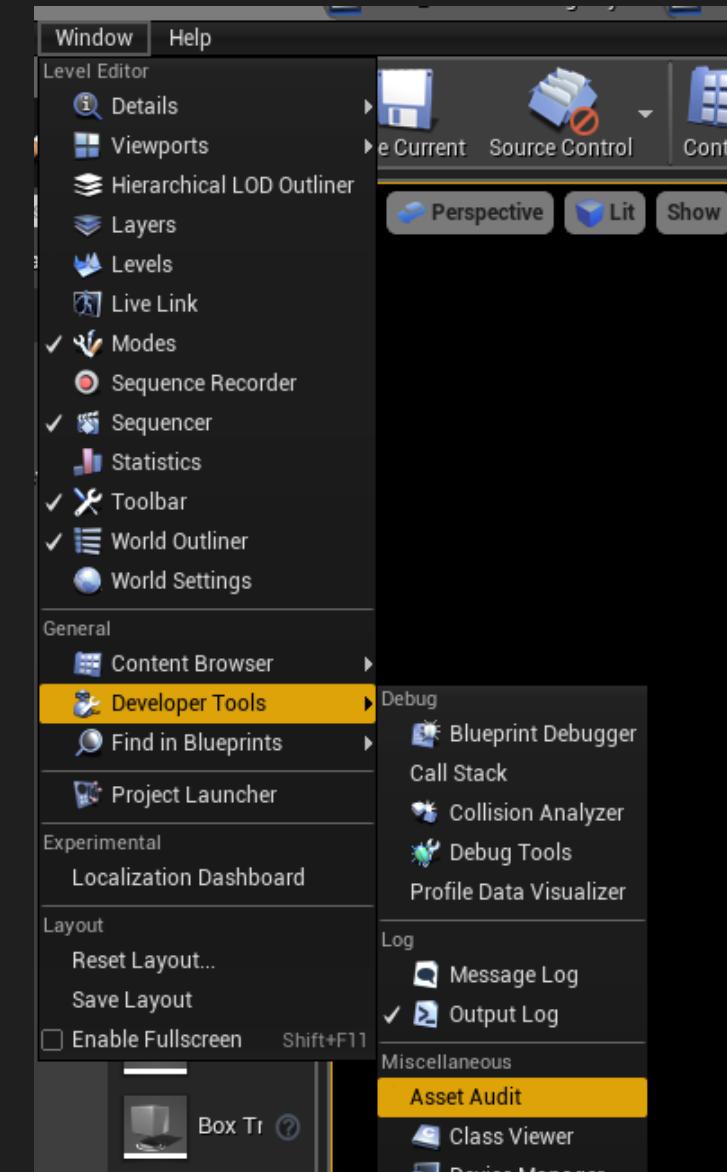
- 親Chunkに含まれるアセットは子Chunkには含まれない

DefaultEngine.ini

```
[/Script/UnrealEd.ChunkDependencyInfo]
+DependencyArray=(ChunkID=100,ParentChunkID=11)
+DependencyArray=(ChunkID=101,ParentChunkID=11)
```

設定したChunkの確認方法

4.19で改良された
Asset Audit の出番！



Asset Audit

Add Chunksボタンを押すことで、各Chunkのリストを表示

- ・アセット競合も確認可能
- ・Selected Platformから
実際に作成したパッケージからの情報を取得することも可能

The screenshot shows the Asset Audit window with a toolbar at the top containing buttons for Clear Assets, Add Primary Asset Type, Add Asset Class, Add Managed Asset, Add Chunks (which is highlighted with a red box), Refresh, and Selected Platform (set to Editor). Below the toolbar is a table with columns: Name, Primary Type, Primary Name, Disk Size, Exclusive Disk Size, Total Usage, Cook Rule, and Chunks. The table lists three items: Chunk_0, Chunk_1, and Chunk_2, each with its respective details. At the bottom left, it says "3 items".

Name	Primary Type	Primary Name	Disk Size	Exclusive Disk Size	Total Usage	Cook Rule	Chunks
Chunk_0			13.747 MB		0		
Chunk_1			454.154 MB		0		
Chunk_2			787.722 MB		0		

The screenshot shows a list view titled "Chunks" with columns: Js, Cook, Ru, Chunks, Dimensions, and H. It displays one item: "2+5 4,000 > T 2".

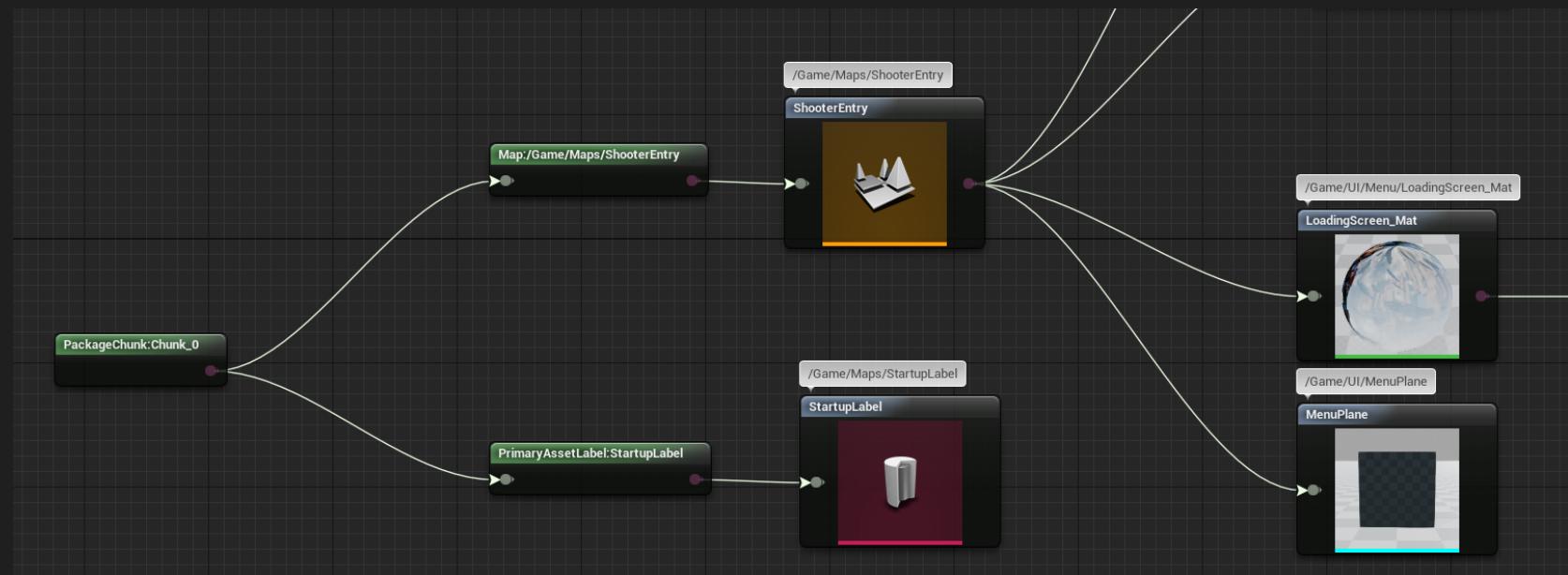
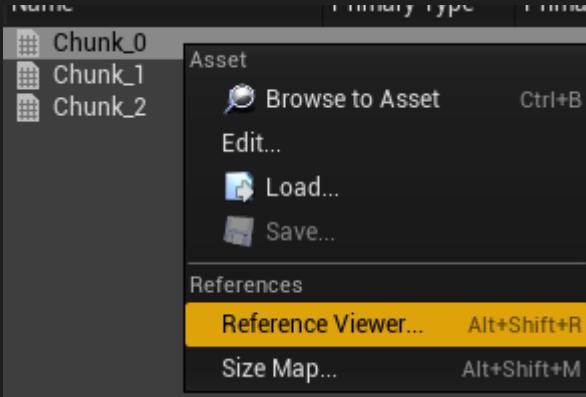
Js	Cook	Ru	Chunks	Dimensions	H
2+5	4,000	>	T	2	



Reference Viewer

各アセット間の参照関係を可視化

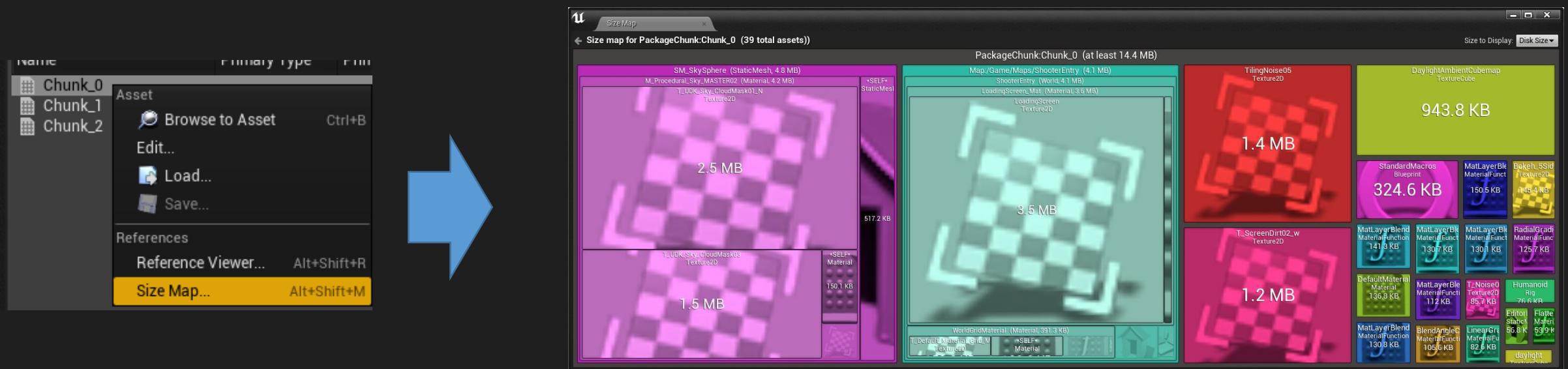
- ・該当アセットが何故このChunkに含まれているかの調査に便利！



Size Map

Chunkに含まれるアセットのサイズの割合を可視化

- Chunkの容量削減の調査に便利！



ここまでまとめ

Chunk IDを割り当てることで、コンテンツを分割することが可能

- Asset Manager
- Primary Asset Label

Chunkによる分割結果をエディタ上で確認可能！

- Asset Audit

もう少し詳細な解説はこちら



UE4のモバイル開発におけるコンテンツアップデートの話
- Chunk IDとの激闘編 -

コンテンツアップデート用機能の紹介

- Chunk機能によるコンテンツ分割
- Chunk用パッケージの作成
- Chunkデータのダウンロード
- Chunkからのアセット取得

Chunk用パッケージを作成するには？

Chunkに関する設定を事前にしておけば
ゲームの実行ファイル(exe ・ apk ・ ipaなど)を作るだけ！

- 実行ファイルと同時に
各Chunk毎のパッケージが完成

Chunkに関する事前設定を行う箇所

- Project Settings
- Project Launcher

Project Settings

Project - Packaging

Fine tune how your project is packaged for release.

These settings are saved in DefaultGame.ini, which is currently writable.

Use Pak File

Generate Chunks

Generate No Chunks

Chunk Hard References Only

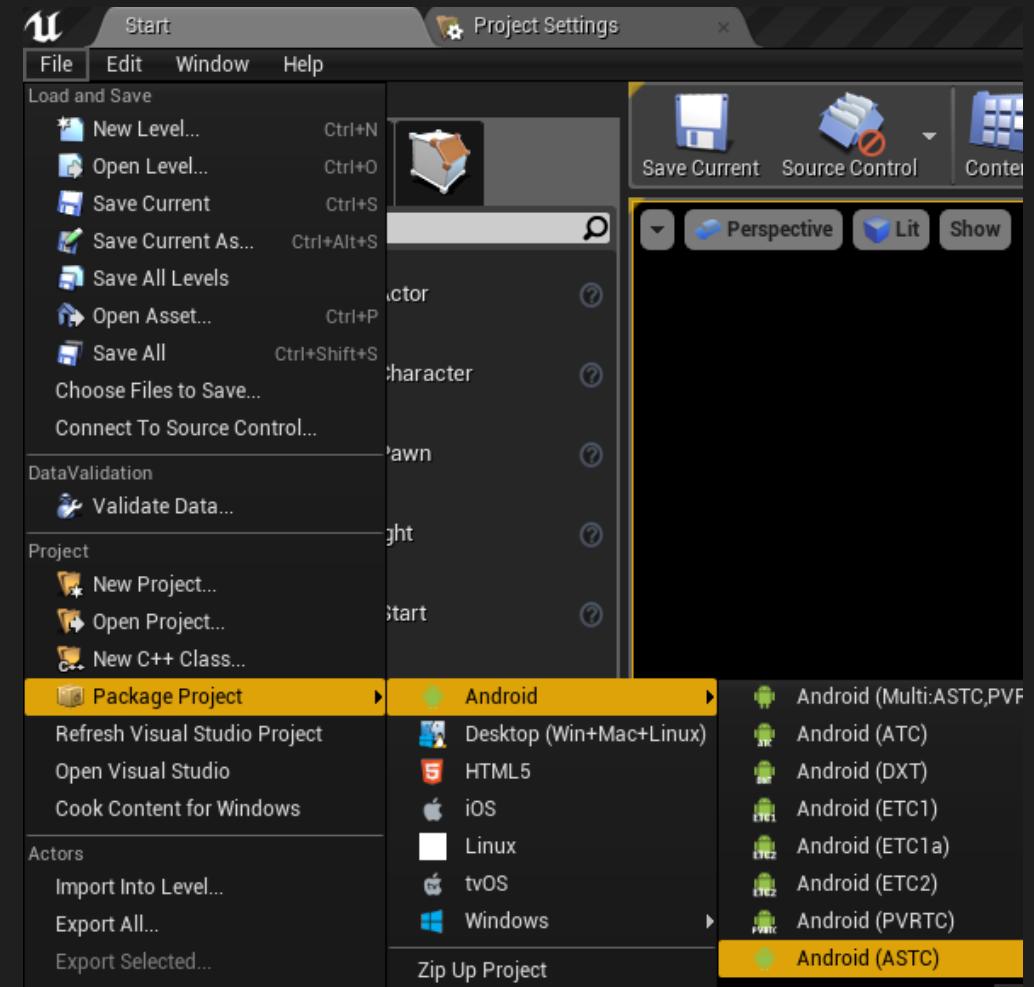
Build Http Chunk Install Data

Http Chunk Install Data Directory ...

Http Chunk Install Data Version

Share Material Shader Code

Shared Material Native Libraries



Project Launcher

The screenshot shows the Unreal Engine Project Launcher interface. On the left, a list of build profiles is displayed:

- Nightly (HTML5)
- Chrome (HTML5)
- Firefox (HTML5)
- D1146 (WindowsNoEditor)
- Nightly(64bit) (HTML5)

A message at the bottom says: "Don't see your device? Verify it's setup and claimed in the Device Manager."

On the right, the "Advanced" tab is selected in the settings panel. The "Advanced Settings" section contains the following options:

- Iterative cooking: Only cook content modified from previous cook
- Stage base release pak files
- Compress content
- Add a new patch tier
- Save packages without versions

The "Num cookers to spawn:" field is set to 0.

The "Generate Chunks" checkbox is checked and highlighted with a red rectangle.

The "Http Chunk Install Settings" section is also highlighted with a red rectangle and contains the following settings:

- Create Http Chunk Install data

Http Chunk Install Data Path: D:/UnrealProjects/HttpChunkUpdateTest/ChunkInstall/

Http Chunk Install Release Name: release1

Cooker build configuration: Development

Additional Cooker Options:

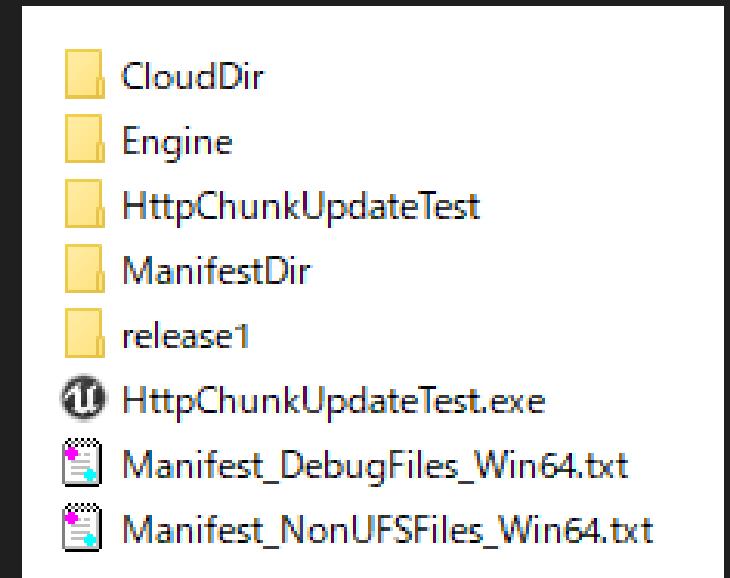


パッケージ作成処理の完了後

指定した出力先に生成される CloudDir フォルダに
コンテンツ配信時に必要なファイルが配置されている



CloudDir 内のファイルを
配信用サーバにアップすればOK！

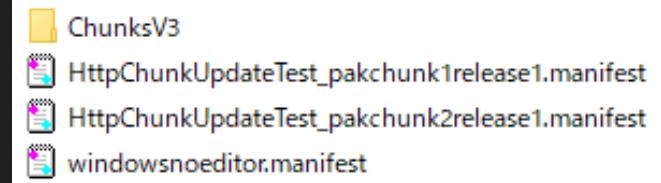


CloudDir フォルダの中身

各Chunkのデータ

Chunk管理用のManifestデータ

- Chunkのダウンロード時に使用

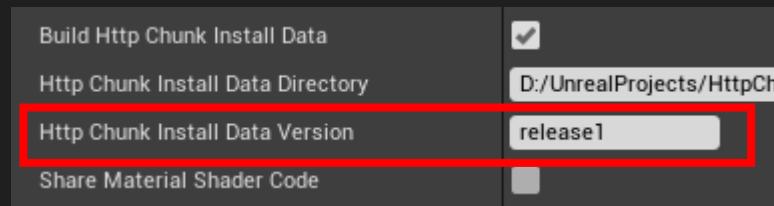


```
1 {  
2     "AppID": "001000000000",  
3     "AppNameString": "HttpChunkUpdateTest_pakchunk1",  
4     "BuildVersionString": "release1",  
5     "ChunkFilesizeList": [  
6         "0A2A1152407C1EBCA586899B496630E7": "20401000000000000000000000000000"  
7     ],  
8     "ChunkHashList": [  
9         "0A2A1152407C1EBCA586899B496630E7": "017068074075094142219010"  
10    ],  
11    "ChunkShaList": [  
12        "0A2A1152407C1EBCA586899B496630E7": "2718C12D630D2C5B49F05DFAB9D1AFF1D33C23DF"  
13    ],  
14    "CustomFields": [  
15        "ChunkID": "00100000000000000000000000000000",  
16        "PakReadOrdering": "00000000000000000000000000000000",  
17        "bisPatch": "false"  
18    ],  
19    "DataGroupList": [  
20        "0A2A1152407C1EBCA586899B496630E7": "064"  
21    ],  
22    "FileManifestList": [  
23        "FileChunkParts": [  
24            {  
25                "Guid": "0A2A1152407C1EBCA586899B496630E7",  
26                "Offset": "000000000000",  
27                "Size": "013190000000"  
28            }  
29        ],  
30        "FileHash": "140059193155159049126220040013243240082064166083151126168175",  
31        "Filename": "pakchunk1-WindowsNoEditor.pak"  
32    ],  
33 ],  
34 },  
35     "LaunchCommand": "",  
36     "LaunchExeString": "",  
37     "ManifestFileVersion": "013000000000",  
38     "PrereqArgs": "",  
39     "PrereqIds": [],  
40     "PrereqName": "",  
41     "PrereqPath": "",  
42     "bisFileData": false  
43 } [EOF]
```

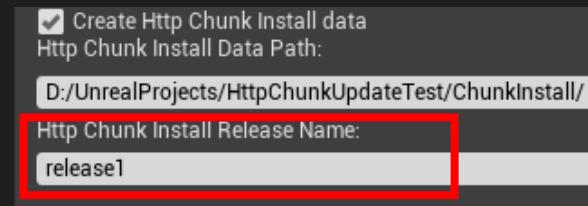


Manifestの役割

紐付いているChunkの
バージョン情報とハッシュ値の管理



Project Settings



Project Launcher

```
"AppID": "001000000000",  
"AppNameString": "HttpChunkUpdateTest",  
"BuildVersionString": "release1",  
"ChunkFileSizeList": {  
    "0A2A1152407C1E60A586899B496630E  
},
```

Chunk毎のManifest



バージョン情報とハッシュ値の役割

既にダウンロード済みのChunkを
再ダウンロードするか否かの判定に使用

- ローカルのManifestとサーバ上のManifestの比較

判定の流れ

1. バージョンが異なっている場合は再ダウンロード
2. バージョンが同じだが、
ハッシュ値が異なっている場合は再ダウンロード



Chunkの更新フローの方針

特定のChunkのみ修正したい場合は
バージョンを変えずに、Chunkパッケージを作成・差し替え

- 変更がないChunkはハッシュ値が変化しないので
再ダウンロードは行われない



更新対象のChunk1のハッシュ値のみ変化

Chunkの更新フローの方針

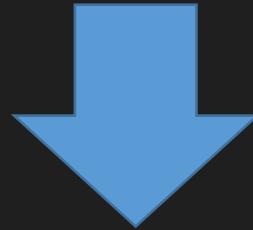
全Chunkの再ダウンロードを強制したい場合は
バージョンを変えて、Chunkパッケージを作成・差し替え
• エンジンバージョンの更新や
ゲームシステムの変更などによる大型アップデート時 など

コンテンツアップデート用機能の紹介

- Chunk機能によるコンテンツ分割
- Chunk用パッケージの作成
- Chunkデータのダウンロード
- Chunkからのアセット取得

まず結論から

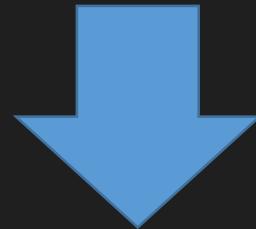
起動時に全てのChunkを読み込む仕様の場合



iniに設定追加 + Chunkのダウンロードをリクエストするだけで
サーバから全Chunkを自動ダウンロード！

まず結論から

ゲームの進行度などに応じて
必要になったChunkをその都度ダウンロードする仕様の場合



Asset Managerのロード関数を使用すれば
必要になったアセットを持つChunkを自動ダウンロード！



Chunk一括ダウンロードの機能を使うには

事前準備

- Chunk用パッケージをサーバに配置
- iniファイルにサーバに関する設定を追加
- `HttpChunkInstaller`プラグインを有効に

ランタイムで必要な処理

- `HttpChunkInstaller`の関数呼び出し

Chunk用パッケージをサーバに配置

<http://192.168.0.11/ChunkData/> 以下に配置する場合

CloudDirフォルダ内のファイルを
<http://192.168.0.11/ChunkData/> に配置

Index of /ChunkData			
	Name	Last modified	Size Description
	Parent Directory		-
	ChunksV3/	2018-03-27 02:47	-
	Master.manifest	2018-03-27 02:47	828
	Test_pakchunk1releas..>	2018-03-27 02:47	949
	Test_pakchunk2releas..>	2018-03-27 02:47	5.8K
Apache/2.4.29 (Win32) OpenSSL/1.1.0g PHP/7.2.3 Server at 192.168.0.11			



iniファイルへの設定追加

<http://192.168.0.11/ChunkData/> 以下に配置する場合

[HTTPChunkInstall]

TitleFileSource=Http

CloudProtocol=http

CloudDomain=192.168.0.11

CloudDirectory= ChunkData

[HTTPOnlineTitleFile]

BaseUrl="192.168.0.11"

EnumerateFilesUrl= ChunkData

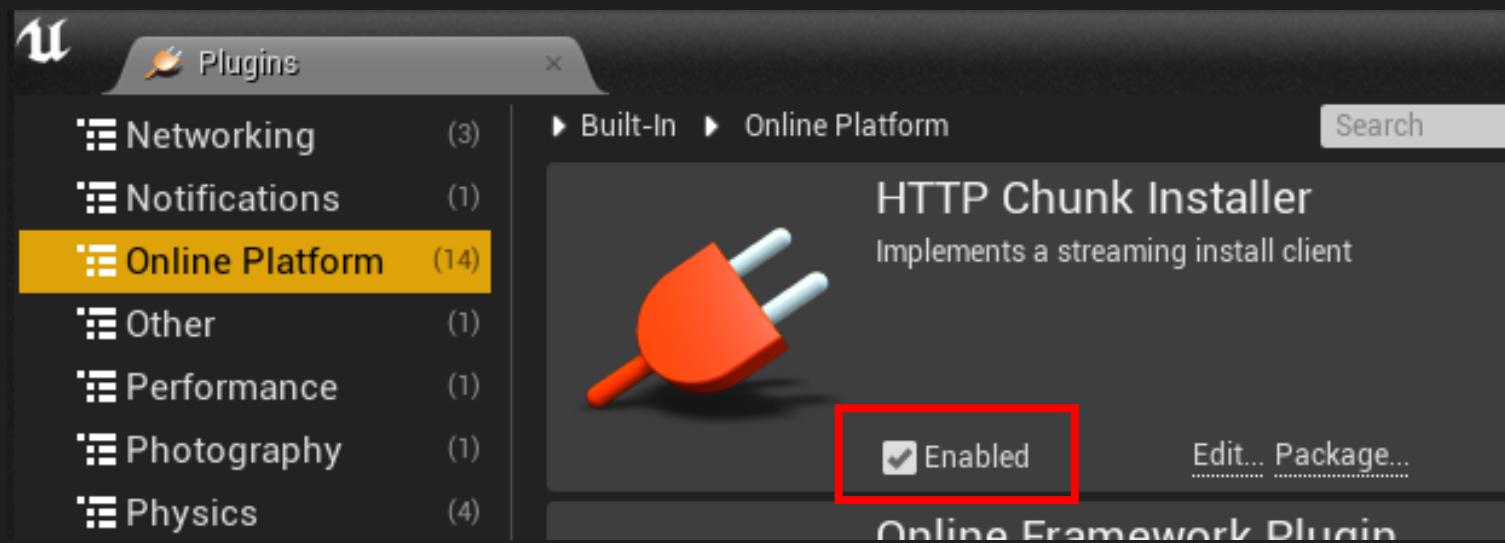
[StreamingInstall]

DefaultProviderName=HTTPChunkInstaller



HttpChunkInstallerプラグインを有効に

デフォルトでは無効になっているので注意



【注意】

ランチャー版の場合、HttpChunkInstallerプラグインを有効にすると
パッケージングに失敗する不具合が報告されています

この不具合はUE4.19.2で修正予定です

UE4.19.1以前のエンジンでHttpChunkInstallerプラグインを
使用する場合は、GithubからDLしてビルドしたUE4をご使用ください



HttpChunkInstallerの関数呼び出し

FHTTPChunkInstall::SetInstallSpeed を使うことで

Chunkのダウンロードをリクエスト可能

- 与える引数の値によって
ダウンロード時の処理負荷を調整可能



HTTPChunkInstallerの関数を使うには

1. Build.cs の ModuleNames に “HTTPChunkInstaller” を追加
2. Ini に以下の文字列を追加
[StreamingInstall]
DefaultProviderName=HTTPChunkInstaller
3. #include "HTTPChunkInstaller.h"
4. FHTTPChunkInstall* ChunkInstall =
static_cast<FHTTPChunkInstall*>
(FPlatformMisc::GetPlatformChunkInstall());

起動時に
全てのChunkを読み込む仕様を実現！



Tips

プラットフォーム固有のiniファイルで設定しておくと
Chunkに関するプラットフォーム別対応が楽になります！

- [ProjectDirectory]/Config/[Platform]/[Platform]Engine.ini
- [公式ドキュメント:コンフィギュレーションファイル](#)



WindowsEngine.ini



AndroidEngine.ini



IOSEngine.ini

ロード時に必要なChunkをDLする機能を使うには？

事前準備

- Chunk用パッケージをサーバに配置
- iniファイルにサーバに関する設定を追加
- Asset Managerの設定を一部変更

ランタイムで必要な処理

- HttpChunkInstallerのSetInstallSpeed関数の呼び出し
- Asset Managerによるロード処理を活用



Asset Managerの設定を一部変更

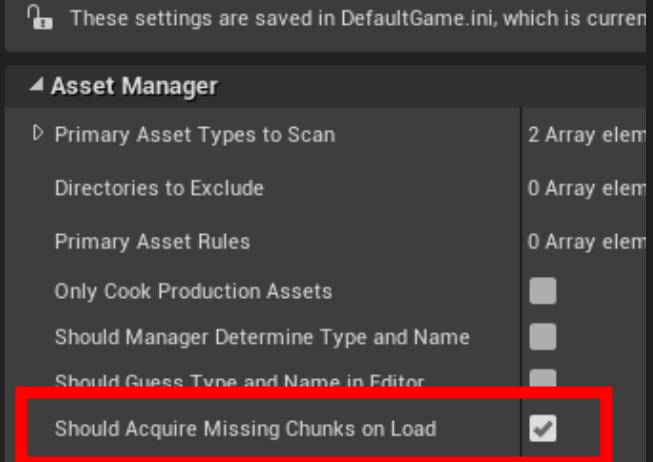
Project SettingsのAsset Managerカテゴリにある
“Should Acquire Missing Chunks on Load” を有効に！



Asset Managerを使ってアセットをロードする際、
もしロード対象のアセットがローカルに存在しない場合、
そのアセットを含むChunkを自動ダウンロードするように！

Game - Asset Manager

Settings for the Asset Management framework, which can be used to...



These settings are saved in DefaultGame.ini, which is currently being edited.

Asset Manager

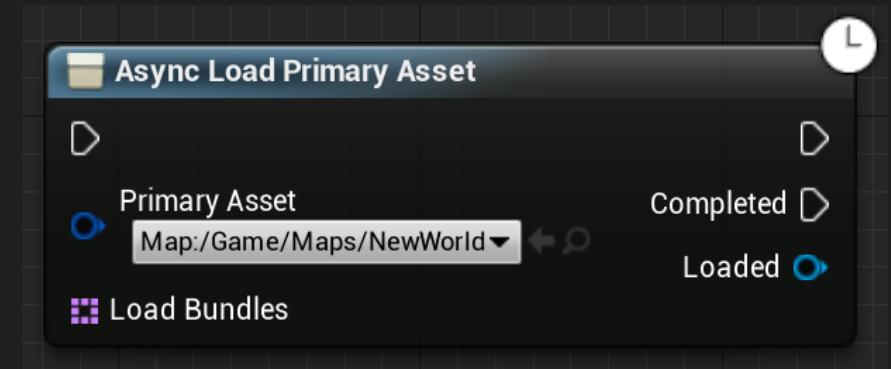
- Primary Asset Types to Scan
- Directories to Exclude
- Primary Asset Rules
- Only Cook Production Assets
- Should Manager Determine Type and Name
- Should Guess Type and Name in Editor
- Should Acquire Missing Chunks on Load**



つ、つまり、どういうこと...？

Asset Managerを経由すれば、ロード対象のアセットが
サーバからダウンロード済みか否かを意識しなくてもOK

- 未ダウンロードの場合、
ロード時間にChunkのダウンロード時間が含まれるので注意！

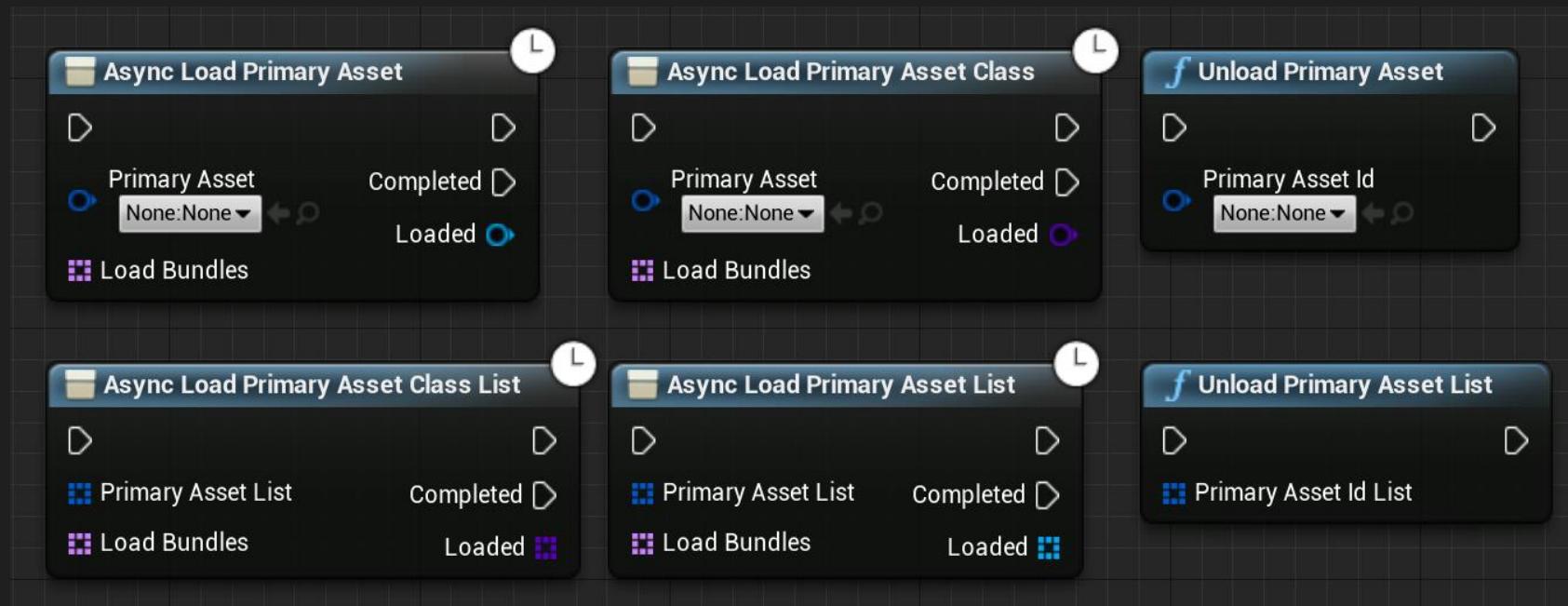


Asset Managerが用意しているロード処理

- Primary Asset の非同期ロード処理
- Primary Asset に紐づく Asset Bundle の非同期ロード処理

Primary Asset の非同期ロード処理

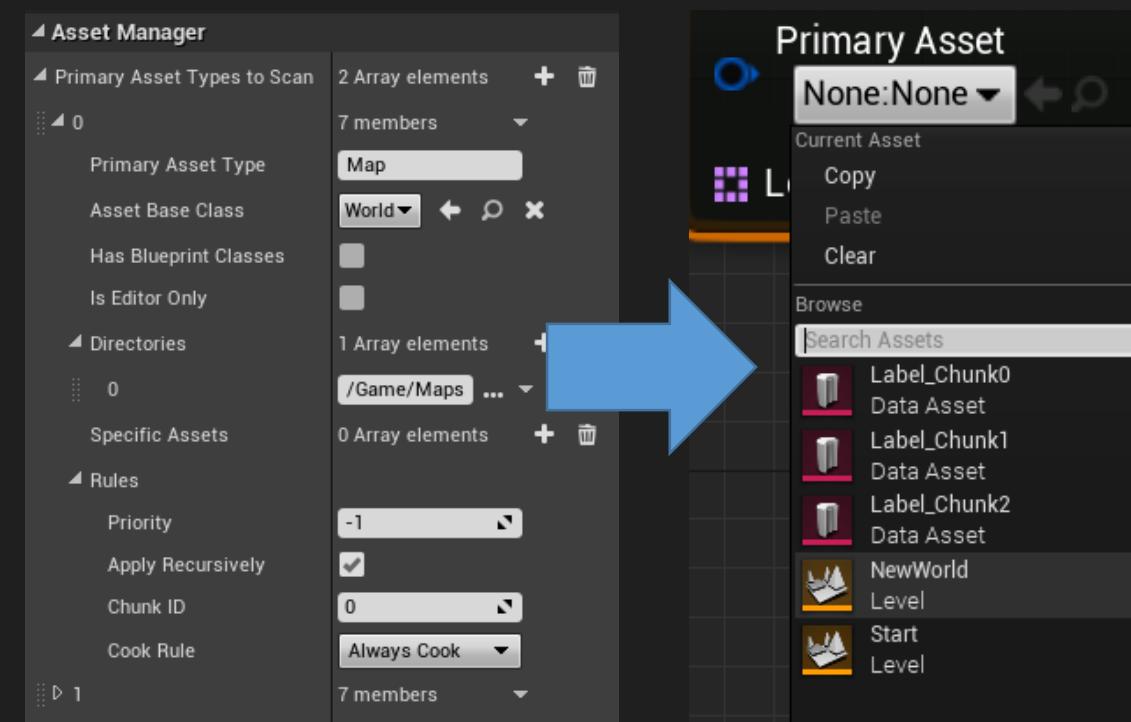
基本的な処理はBPノードで提供済み！

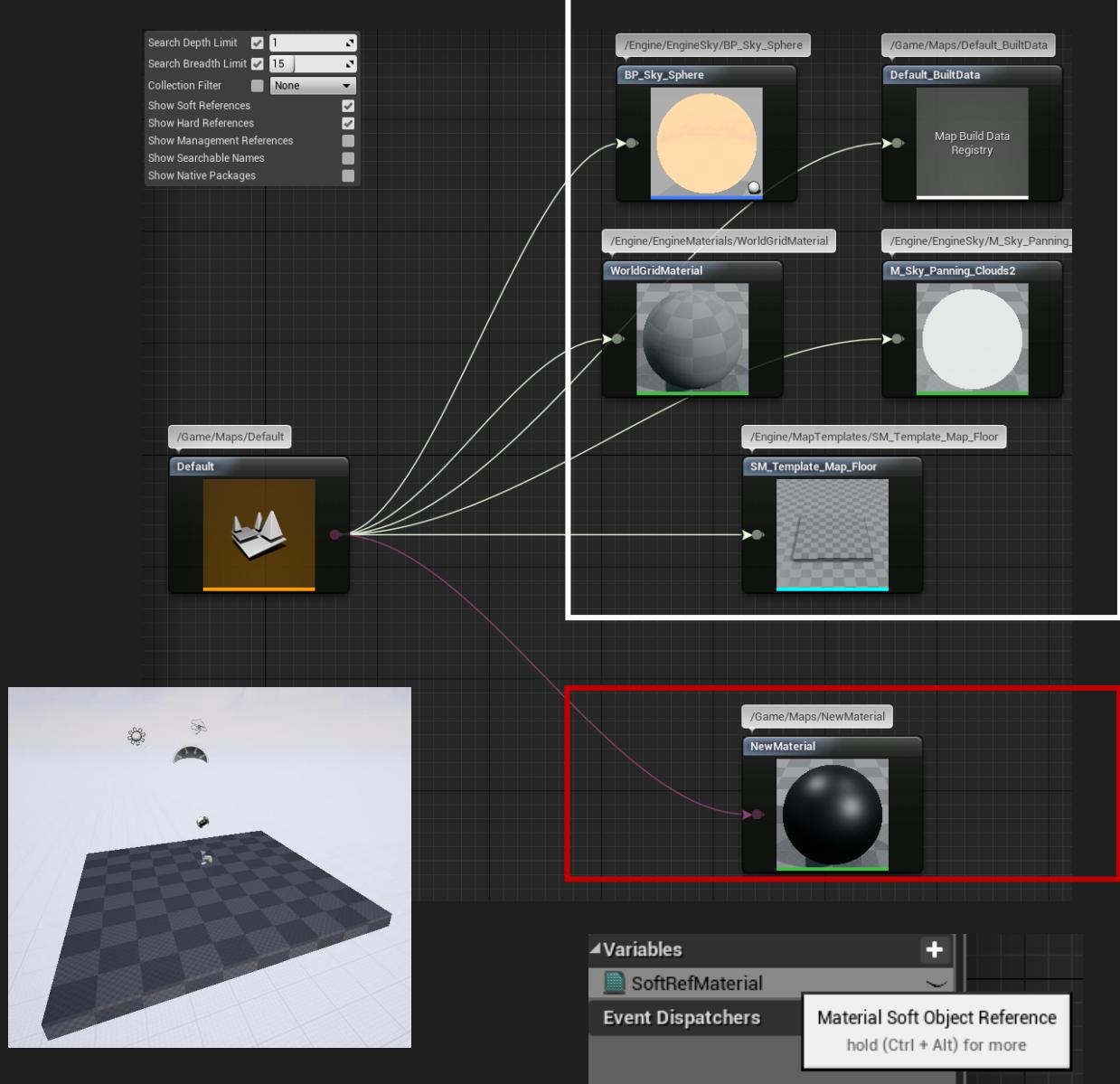


Primary Asset の非同期ロード処理

Primary Asset Types to Scanで
抽出されたPrimary Assetがロード対象

ロード対象のPrimary Assetが
直接参照している Secondary Assetも
同時にロードされます





Secondary Asset(直接参照)
 → 同時ロード対象

Secondary Asset(間接参照)
 → 同時ロード対象外



明示的にロード処理を呼ぶ必要あり

間接参照のアセットも一緒にロードして欲しいなあ...
でも、直接参照にすると...色々と困ることあるしなあ...

ハア...ツライ...

そんな貴方のために
Asset Bundle があります！！！

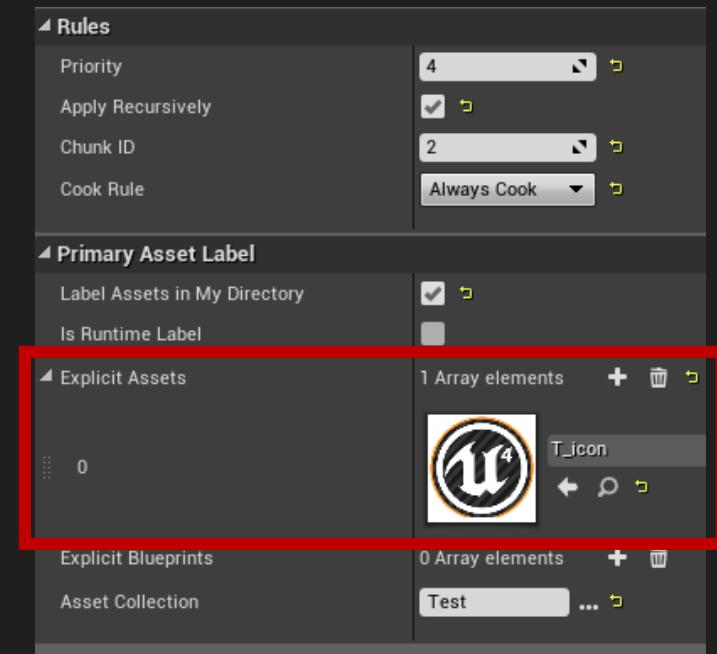
Asset Bundleとは？

専用のタグ名が設定された Secondary Asset(間接参照)

- 現在はC++からのみ利用可能

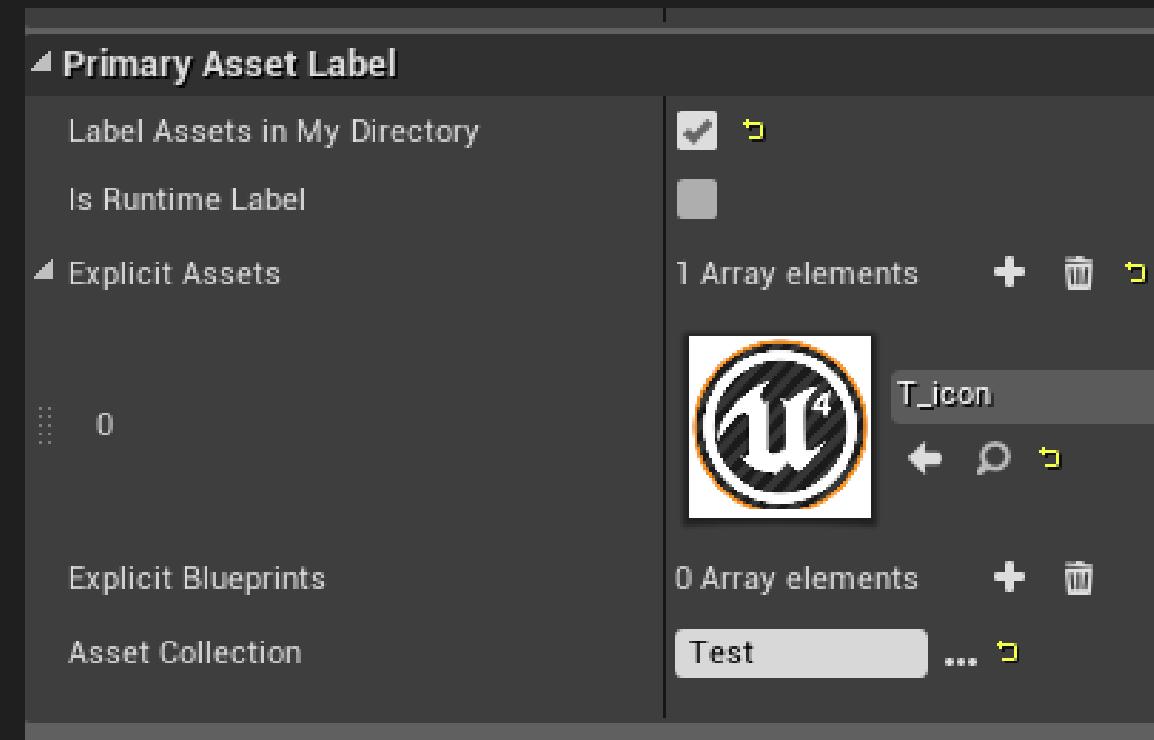
例: Primary Asset Label の Explicit Assets

```
UPROPERTY(EditAnywhere,  
Category = PrimaryAssetLabel, meta = (AssetBundles = "Explicit"))  
TArray<TSoftObjectPtr<UObject>> ExplicitAssets;
```



Primary Asset Label が持つAssetBundle

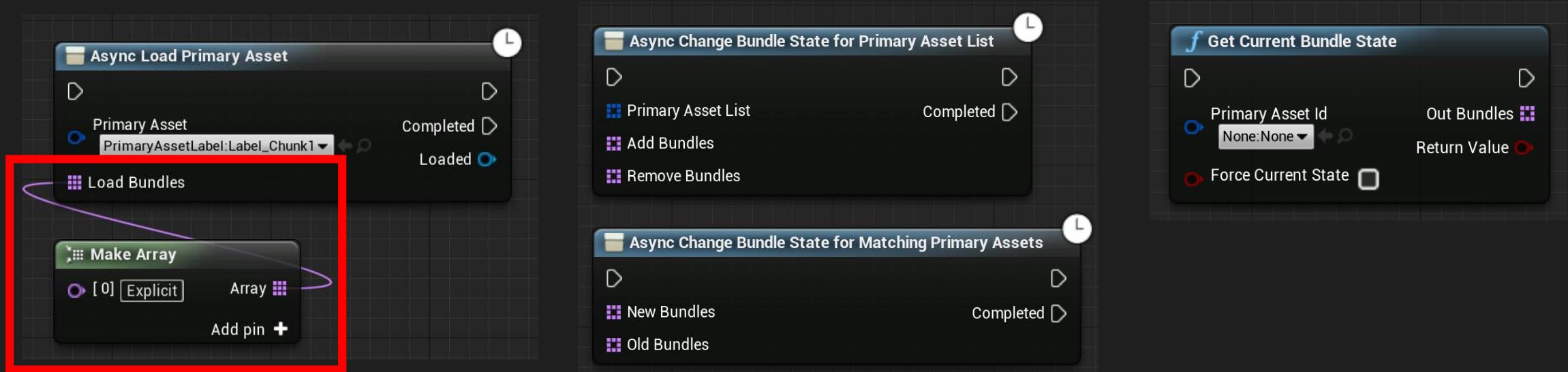
- Explicit Assets / Blueprint
→ **Explicit**
- Label Assets in My Directory
→ **Directory**
- Asset Collection
→ **Collection**



Asset Bundleの非同期ロード処理

Primary Assetのロード時に
ロードしたいAsset Bundleのタグ名を渡せばOK！

- Asset Bundleの「追加ロード / アンロード」や「ロード済か確認」も可能！



Asset Bundleに関するTips

Primary AssetへのAsset Bundleの登録は
エディタ / ランタイム上で動的に実行可能！

- エディタ:

UPrimaryAssetLabel::UpdateAssetBundleData() を参考に

- ランタイム:

UAssetManager::AddDynamicAsset を使用



Asset Bundleの運用例

1. キャラクタに関するアセットを管理するPrimary Asset を用意
2. 用途別にタグを分けたAsset Bundleを登録
3. 状況に応じて、必要なAsset Bundleのみをロード

ここまでまとめ

Chunkのダウンロードは事前準備をしておけば
エンジン側で自動的に行なってくれる！

Asset Managerのロード関数を使用すれば
必要に応じたChunkの自動ダウンロードを簡単に実現可能

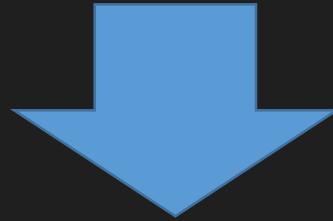
Asset Bundle機能を使えば、
間接参照しているアセットのロード管理を柔軟に行える！

コンテンツアップデート用機能の紹介

- Chunk機能によるコンテンツ分割
- Chunk用パッケージの作成
- Chunkデータのダウンロード
- Chunkからのアセット取得

非同期ロード処理の注意点

Asset Managerによる非同期ロード処理は
そのアセットのデータをメモリに載せる所まで！

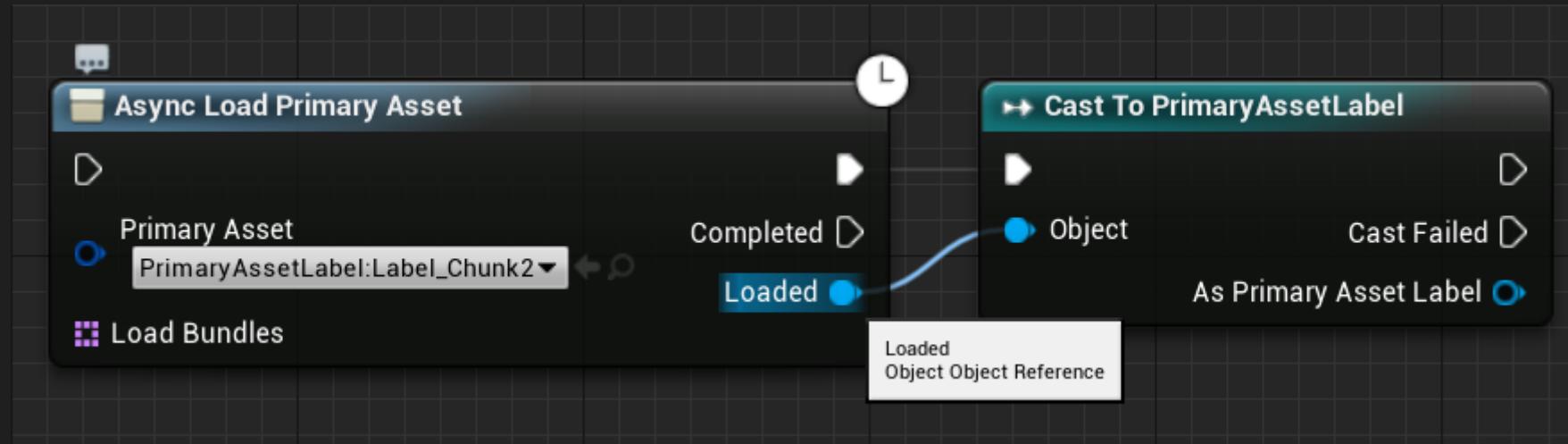


レベル上で使用するには
ロードしたアセット自身を取ってくる必要がある！

Primary Assetからの取得

Async Load Primary Asset (List)の返り値から取得

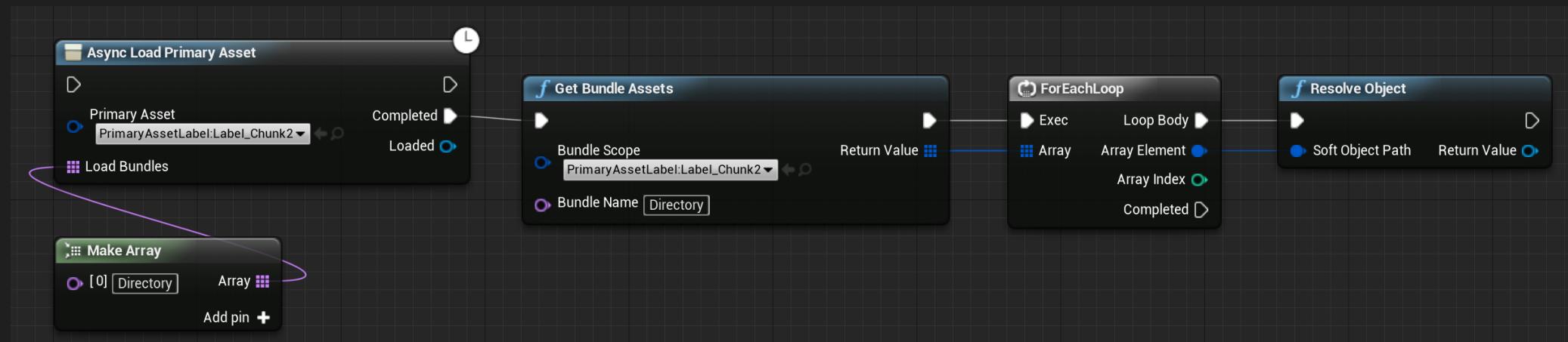
- UObject型なので注意



Asset Bundleからの取得

UAssetManager::GetAssetBundleEntry を使用

- ・ 現状C++での提供のみ
- ・ 自作BP関数ライブラリから呼べるようにすると便利！



ここまでで取得できないアセット

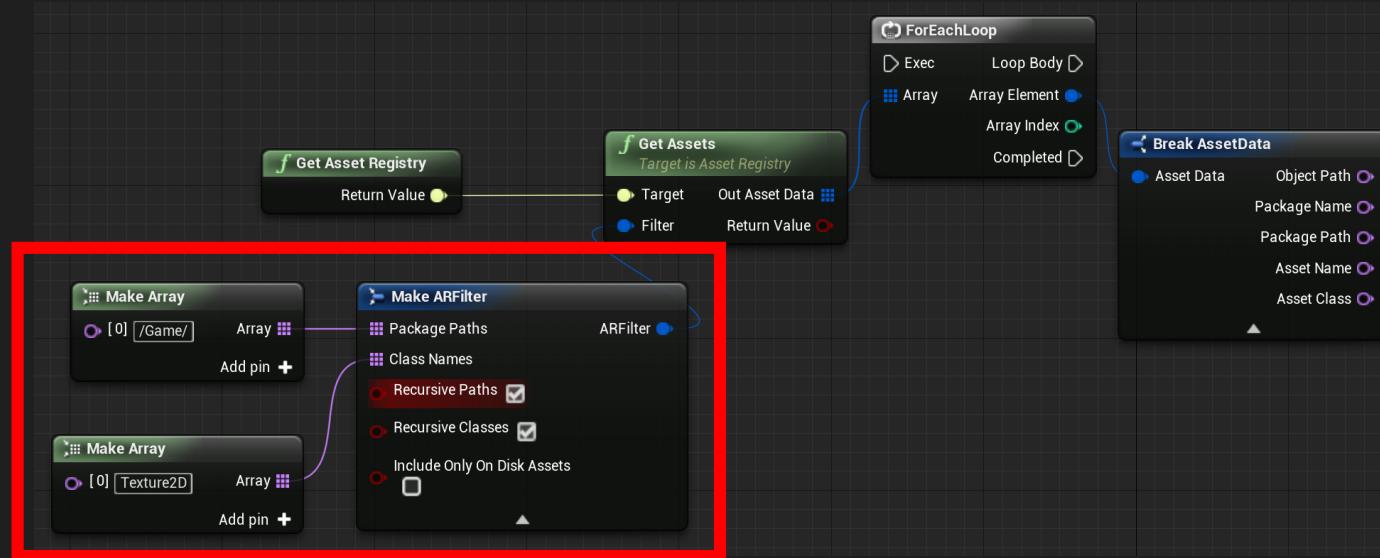
Primary Asset 又は Asset Bundleが持つ
プロパティや関数などを使っても
アクセスできないアセット



Asset Registryを活用する！

Asset Registryからアセットを検索可能

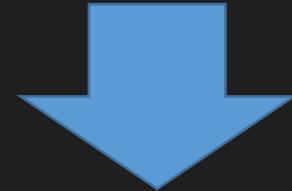
- ・ クラス名・パス名などで検索フィルタを利用可能
- ・ 検索結果(FAssetData) の GetAsset関数からアセットを取得



Asset Registryで抽出したアセットのロード

Asset Registryによる検索結果は
未ダウンロードのアセットも含むので注意！

- ・未ダウンロードの場合、FAssetData::GetAsset の結果はNULLに…



Chunkを自動ダウンロードしてくれる
UAssetManager::LoadAssetList でロードしましよう！



Asset Registryによるアセット取得に関して

Asset Managerが「いい感じに」してくれてた処理を
自分で実装することに...

- Asset Registryによる検索コストの問題も

Primary Asset 又は Asset Bundleから
アクセスできるように設計した方が安全！



ここまでまとめ

- Asset Managerのロード処理だけでは
レベル上にはまだ表示されない
 - メモリに載っているアセットを取得する必要がある
- アセットの取得に関する処理の設計は
Asset Managerのシステムを活用する形にする
 - Primary Asset
 - Asset Bundle



本日のお品書き

- 本日のお題を選んだ理由
- UE4のコンテンツアップデート用機能の紹介
- Battle Breakersにおける運用例

こちらも合わせてご確認ください

UE4公式ブログ

「チャンク化による高速ダウンロードを可能にするツールと最適化の考察」



Battle Breakersを選んだ理由

日本で主流の
モバイルゲームの形式に近いから！



Battle Breakersの仕様

300体以上のキャラクタが登場

- ・さらに、各キャラクタにレア度が存在

起動すると、DLを挟まずに
チュートリアルをすぐプレイすることが可能

- ・チュートリアル終了後や
各ステージの初プレイ時などのタイミングでDL

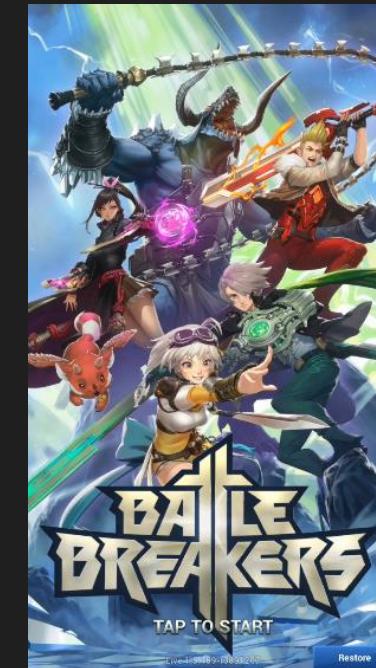
Battle Breakersにおける運用テクニック

- Chunk IDの設定ルールについて
 - タイトル固有のPrimary Asset Label
- Chunk IDの自動設定
 - Chunk間のアセット重複の回避

Chunk ID = 0

実行ファイル (exe ・ apk ・ ipa)に含まれるアセットを管理

- ・ゲームの起動
- ・ログイン
- ・初回起動時のチュートリアル
- ・などに必要なアセット



Chunk ID = 2

Chunk IDの設定漏れ検出用

どのChunkにも登録されていないアセットは

Chunk 2 に含ませる処理を追加

- Chunk IDがデフォルト値のままだと
Chunk ID = 0 に設定されて(実行ファイルに含まれて)しまう



Chunk ID = 5 ~ 17

キャラクタの表示・音声以外のアセットを管理

- ステージデータ
- メニューで使用するUI

キャラクタなどで使用される共通アセットもここで管理

- 戦闘演出 など

Chunk ID = 50~99

ローカライズが必要なアセットを管理

- 使用言語に応じたChunkのみダウンロード

こんにちは

Hello

Bonjour

مرحبا هناك

Chunk ID = 50

Chunk ID = 51

Chunk ID = 52

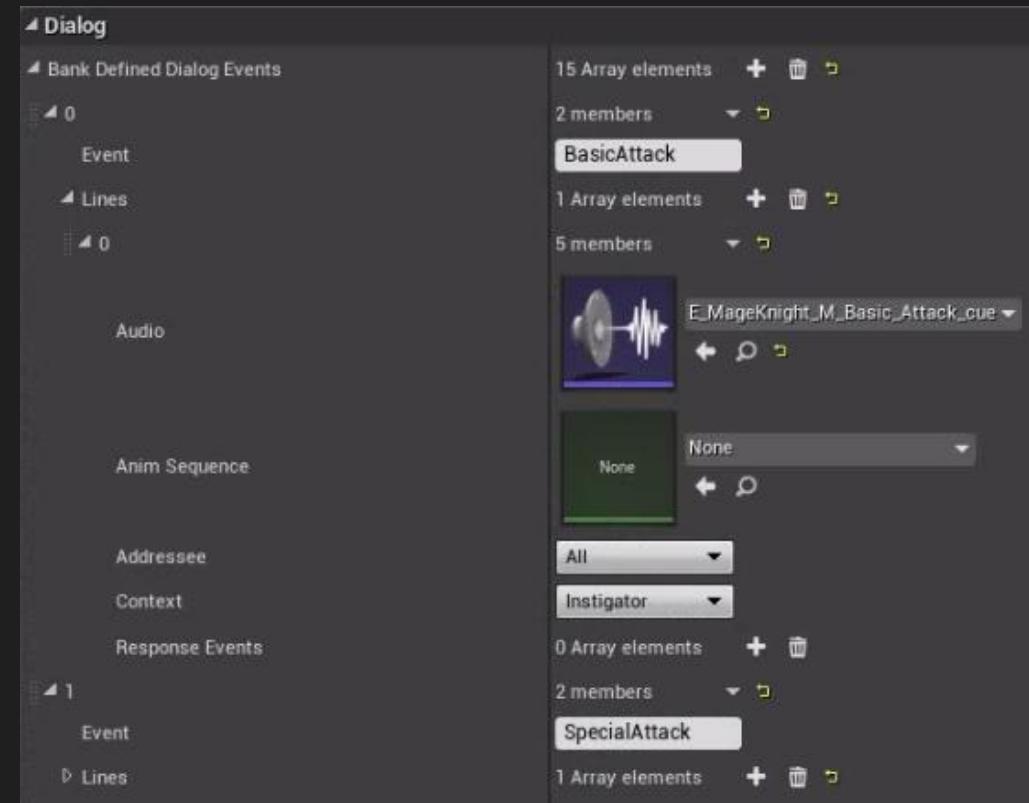
Chunk ID = 53



Chunk ID = 100 ~ 999

キャラクタ毎のボイス管理

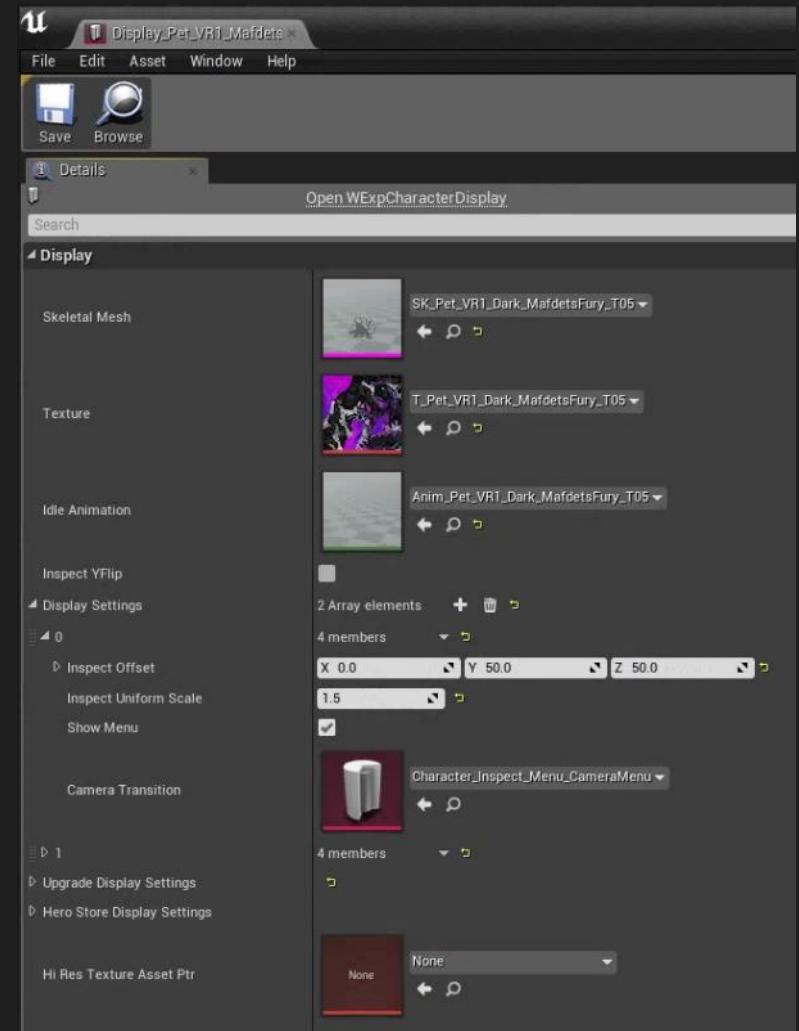
- キャラクタ固有のイベント 每の音声データ
(攻撃時, ダメージ時など)



Chunk ID = 1000 ~ 1300+

キャラクタ毎の表示に必要なアセットの管理

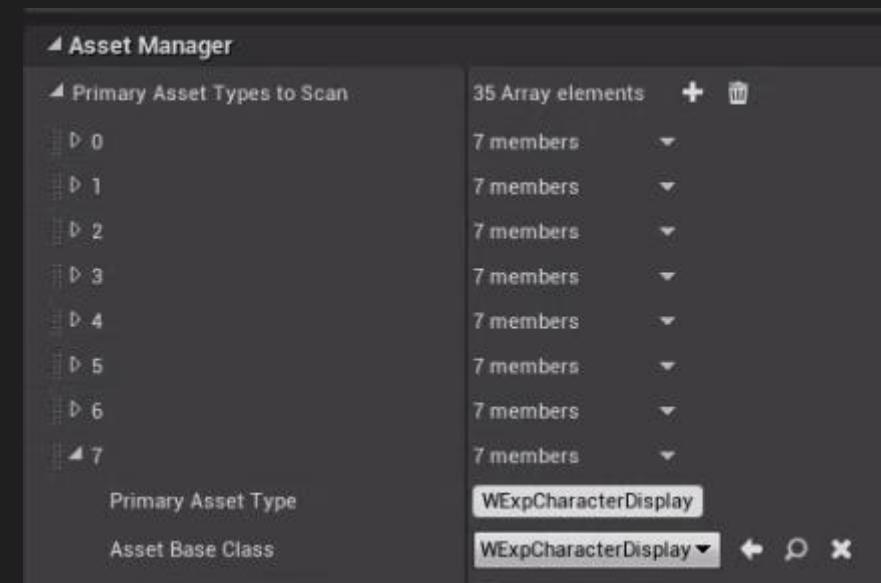
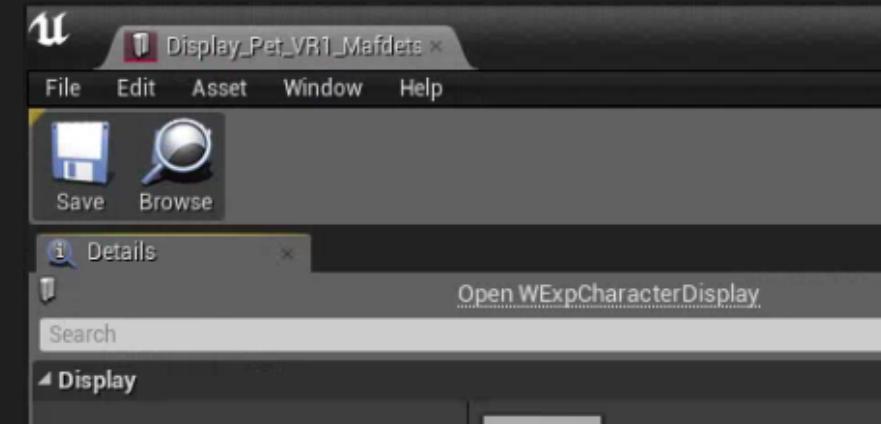
- Skeletal Mesh
- Animation
- Texture



どのように管理しているのか

設定・管理項目毎に
タイトル固有のPrimary Asset Labelを作成

Project SettingsのAsset Managerで
それらをScan対象として登録



Chunk IDの
手動管理は超大変！！！

現在のChunk ID設定フローの課題

Asset Manager / Primary Asset Labelの機能は提供されているが
手動設定なので、ヒューマンエラーが発生しやすい

「数百 * レア度」体分のキャラクタの
Chunk IDをミスなく正確に管理できるか？



もしChunk IDの設定を間違った場合

ハッシュ値が変わるので
全ユーザが該当Chunkを強制再ダウンロード！

もしChunk IDで制御している箇所があった場合
最悪なケースとして、ユーザ毎の管理情報が壊れる可能性も...！

やばい！



なので

Battle Breakersでは
Chunk IDの設定を一部自動化します！

Chunk IDの設定を自動化するには？

UAssetManager::UpdateManagementDatabase に

Chunk IDの自動設定処理を追加する！

- ・アセット管理システムのデータベースの更新処理用
- ・データベースに関連する

Asset ManagerやPrimary Asset Labelの設定変更時に実行



Asset Managerをカスタマイズする場合

プロジェクト用に拡張したAsset Managerのクラスを
エンジンに認識させる事が可能です

- ・つまり、エンジンコードに手を入れなくていい！

詳細な手順は以下のドキュメントをご確認ください

[アセット管理 / プライマリアセットを登録しディスクからロードする](#)

Battle Breakersで行っている処理の例 ①

課題

- ・使用用途・タイミングが同じ場合は同じChunkに配置したい
- ・1つのPrimary Assetで管理するのは非効率
- ・手動でChunk IDを制御したくない

対象のPrimary Assetがとあるカテゴリに属する場合は
そのカテゴリ用のChunk IDを自動設定



Battle Breakersで行っている処理の例 ②

数十年前のスライドより

Chunk間の親子関係設定も
手動で行うのは辛すぎる！

Chunk間のアセット競合を回避するために

Chunk間の親子関係を定義！

- ・親Chunkに含まれるアセットは子Chunkには含まれない

DefaultEngine.ini

```
[/Script/UnrealEd.ChunkDependencyInfo]  
+DependencyArray=(ChunkID=100,ParentChunkID=11)  
+DependencyArray=(ChunkID=101,ParentChunkID=11)
```

Twitter icon #UE4 | #gcconf2018

UNREALENGINE.COM 

Battle Breakersで行っている処理の例 ②

Chunk間の親子関係をコード内で設定し
その結果を ini ファイルに出力する処理を実装

- ・共通アセットの使用が確定している箇所は
共通アセット用のChunk ID、親子関係を自動設定
- ・エディタ上から親子関係を制御できるように
プロパティを追加



Battle Breakersで行っている処理の例 ③

課題

- ・数百体のキャラクタ * オプション(レア度など)という膨大な数を管理する必要がある
- ・キャラクタ毎にユニークなChunk IDを設定
 - ・オプション違いは同じChunk IDに設定



新規キャラ用Primary Assetに対する設定ルール

1. Primary Asset ID名は
キャラクタ名 + オプションというルールで設定
2. Chunk IDの自動設定処理時に
Primary Asset IDからキャラクタ名を抽出
3. 抽出したキャラクタ名が既存のものなら
同じChunk IDを設定
新規なら新しいChunk IDを設定



中学生_N



中学生_SSR

ここまでまとめ

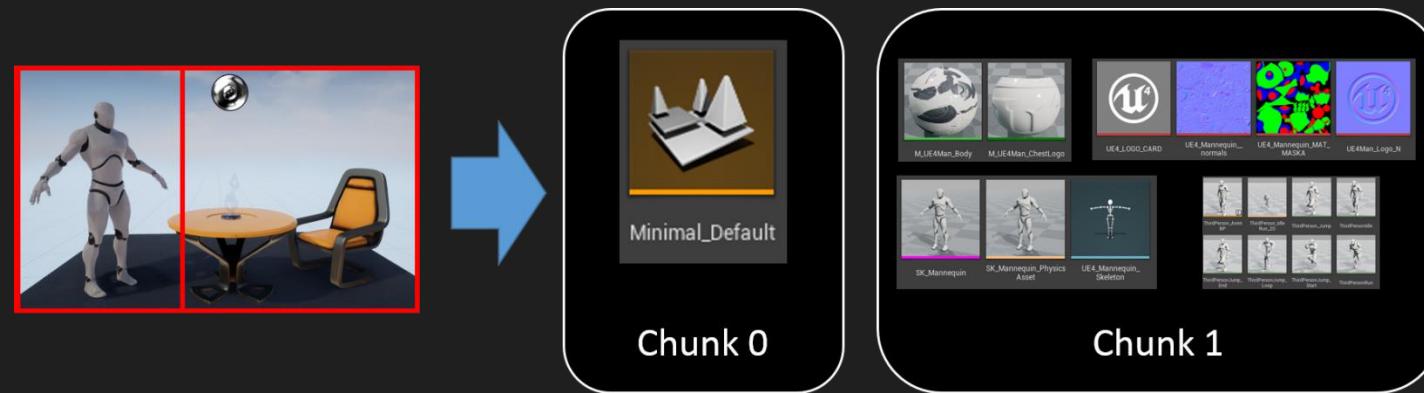
Chunk IDの手動管理は
ゲームの規模が大きくなるにつれてリスクが高まる

Chunk IDの自動設定の仕組みを用意することで
ヒューマンエラーを回避しつつ、作業効率を向上できる



本日のまとめ

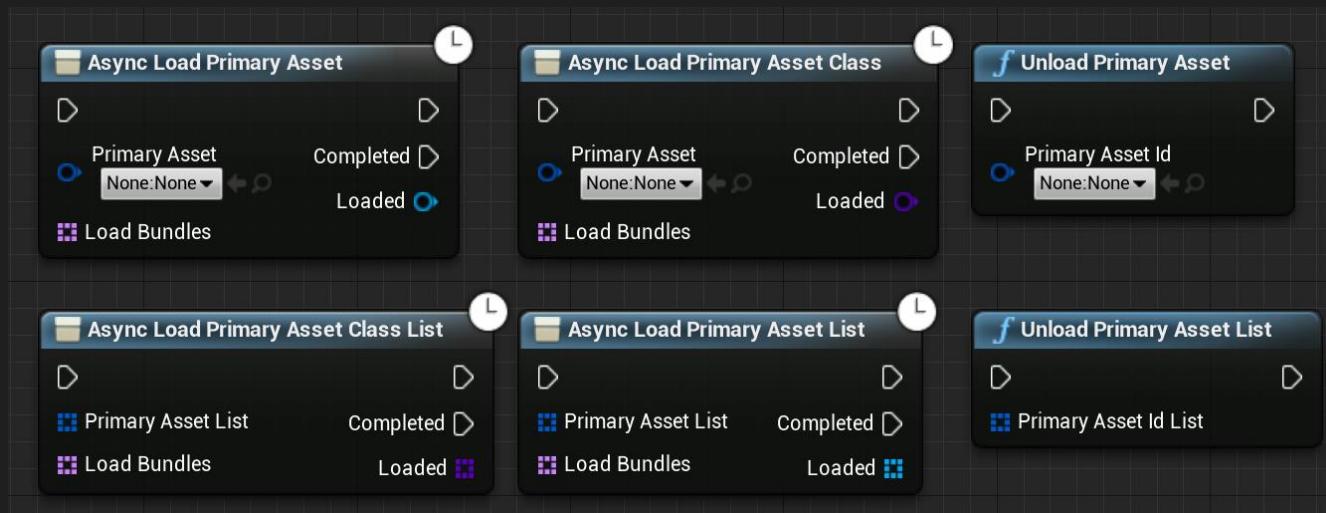
- より良いコンテンツアップデートの仕組みを構築するには
Chunkによるコンテンツ分割が不可欠
- Chunk IDの設定方法は複数存在
 - Asset Manager / Primary Asset Label / C++コード



本日のまとめ

Chunkのダウンロード・データ取得に関して
UE4の標準機能が一通りカバー済み

- Asset Manager, Manifest, HttpChunkInstaller, Asset Bundle



Index of /ChunkData			
Name	Last modified	Size	Description
Parent Directory		-	
ChunksV3/	2018-03-27 02:47	-	
Master.manifest	2018-03-27 02:47	828	
Test_pakchunk1releas..>	2018-03-27 02:47	949	
Test_pakchunk2releas..>	2018-03-27 02:47	5.8K	

Apache/2.4.29 (Win32) OpenSSL/1.1.0g PHP/7.2.3 Server at 192.168.0.11

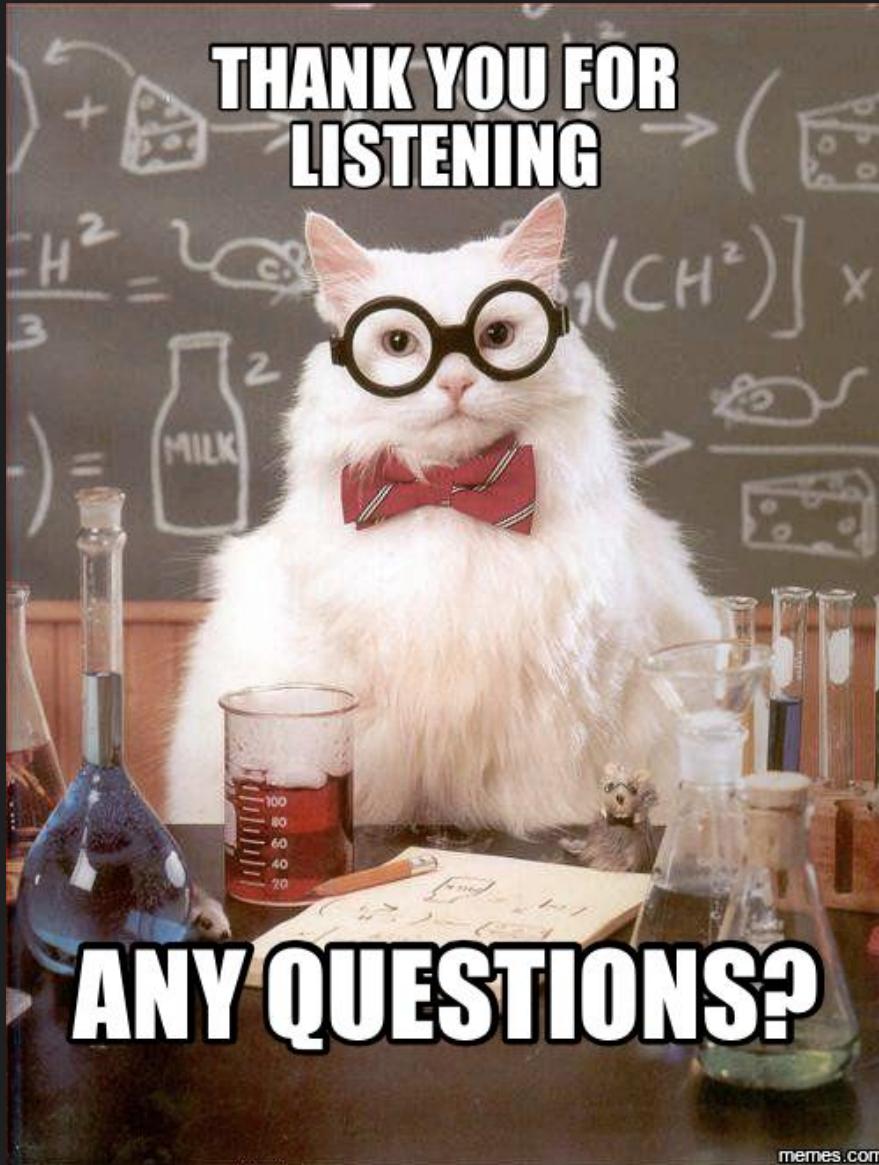


本日のまとめ

- Chunk IDの設定ミスは重大な問題を引き起こす可能性がある
- Chunk IDの自動設定処理を用意することでヒューマンエラーの回避・作業効率の向上を実現
 - Battle Breakersにおける運用事例

各仕様を把握した上で
タイトルに適したフローを構築することが重要！

お困りの際は是非ご相談ください！



ご清聴
ありがとうございました！

本講演に関する質問はこちらからどうぞ！

E-mail : kaz.okada@epicgames.com

Twitter : @pafuhana1213

展示ブースにEGJのスタッフがいますので
本講演に関すること以外の
質問も是非どうぞ！