UE4 슈퍼파월~!
모바일 게임 서비스 기능 알아보기

신광섭
Developer Relations Manager/Programmer
Epic Games Korea

목차

- 구글 플레이 서비스 + In-app Billing
- HttpRequest
- 안드로이드 패치

UNREAL ENGINE
UNREAL SUMMIT 2015

구글 플레이 서비스와 IN-APP BILLING

구글 플레이 서비스 + In-app Billing 사용설정

- 안드로이드 구글 플레이 서비스 지원
  - 리더보드
  - 업적
- 구글 플레이 In-app Billing 지원

- Google Play Developer Console에 새 애플리케이션 추가
- 추가한 애플리케이션에 APK를 업로드 필요
- 업로드 하는 앱은 Debug가 활성화 되지 않은 Release 모드로 Signing 된 앱 필요
  - Signing 관련 설정은 자세한 사용법은 "모바일 개발 설정과 패키징" 세션 ppt 참고

구글 플레이 서비스 + In-app Billing 사용설정

- Release모드로 Signing 된 앱 만들기
  - 장단이 있어서 기본적으로는 안되지만 혹시나 "실행" 버튼이 Release모드로 Signing된 앱이 생성되면 어떨지 생각하신다면?
    - In-App Purchase등등 테스트에는 Release모드로 Signing 된 앱이 있어야 하므로 코드 수정 및 테스트 iteration에 유용
  - 간단한 코드 수정을 통해서 이런 접근도 가능

구글 플레이 서비스 + In-app Billing 사용설정

Engine/Source/Developer/LauncherServices/LauncherServices.Build.cs
에 PrivateDependencyModuleNames 에 UnrealEd 추가

Engine/Source/Developer/LauncherServices/Private/Launcher/LauncherUATTask.h 에 UATCommandLine = FString::Printf 부분을 중심으로 아래와 같이

```
UProjectPackagingSettings* PackagingSettings = Cast<UProjectPackagingSettings>(UProjectPackagingSettings::StaticClass()->GetDefaultObject());
FString OptionalParams;
if (PackagingSettings->ForDistribution)
{
    OptionalParams += TEXT(" -distribution");
}

// base UAT command arguments
FString UATCommandLine;
FString ProjectPath = *ChainState.Profile->GetProjectPath();
ProjectPath = FPaths::ConvertRelativePathToFull(ProjectPath);

FString Configuration = FindObject<UEnum>(ANY_PACKAGE, TEXT("EProjectPackagingBuildConfigurations"))->GetEnumName(PackagingSettings->BuildConfiguration);
Configuration = Configuration.Replace(TEXT("PPBC_"), TEXT(""));
UATCommandLine = FString::Printf(TEXT("BuildCookRun -project=\"%s\" -noP4 -clientconfig=%s -serverconfig=%s"),
                                 *ProjectPath,
                                 *Configuration,
                                 *ConfigStrings[ChainState.Profile->GetBuildConfiguration()]);

UATCommandLine += NoCompile;
UATCommandLine += OptionalParams;
```

```
UATCommandLine += Rocket;
```

이런식으로 수정

Engine/Source/Developer/LauncherServices/Private/LauncherServicesPrivatePCH 에
#include "Settings/ProjectPackagingSettings.h"
추가

구글 플레이 서비스 + In-app Billing 사용설정

- Release Signing 된 앱 만들기
  - 이제 준비된 Release 모드 Singing 된 APK을 베타/알파 테스트에 업로드
  - 업로드된 앱을 출시
    - In-app Billing등을 테스트 하기 위해서는 꼭 출시됨 이여야 함
- Google Play Developer Console에서 Google Play 게임 서비스 활성화
  - https://developers.google.com/games/services/console/enabling 참고

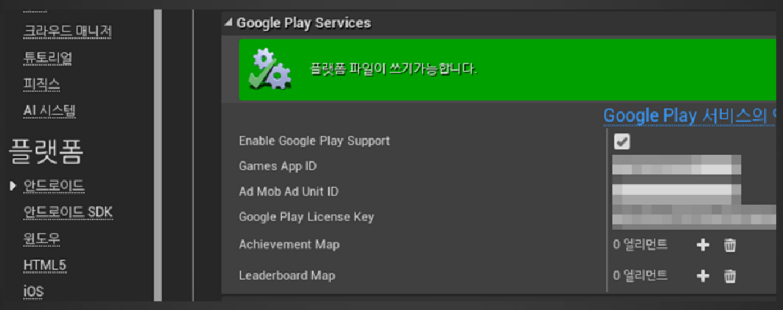
구글 플레이 서비스 + In-app Billing 사용설정

- 프로젝트에 Google Play Services 세팅하기
  - C++ 프로젝트를 사용하신다면 사용하시는 게임프로젝트모듈.Build.cs 에 아래 부분 추가해서 빌드

```
if (Target.Platform == UnrealTargetPlatform.IOS)
{
        PrivateDependencyModuleNames.AddRange(new string[] { "Core", "CoreUObject", "Engine", "OnlineSubsystem", "OnlineSubsystemUtils" });
        DynamicallyLoadedModuleNames.Add("OnlineSubsystemFacebook");
        DynamicallyLoadedModuleNames.Add("OnlineSubsystemIOS");
        DynamicallyLoadedModuleNames.Add("IOSAdvertising");
}
else if (Target.Platform == UnrealTargetPlatform.Android)
{
        PrivateDependencyModuleNames.AddRange(new string[] { "Core", "CoreUObject", "Engine", "OnlineSubsystem", "OnlineSubsystemUtils" });
        DynamicallyLoadedModuleNames.Add("AndroidAdvertising");
        DynamicallyLoadedModuleNames.Add("OnlineSubsystemGooglePlay");
}
```

구글 플레이 서비스 + In-app Billing 사용설정

- 프로젝트 세팅에 안드로이드에 Google Play Services 항목 설정



- Enable Google Play Support 체크
- Google Play 게임 서비스의 Games App ID 입력
- 추가한 애플리케이션의 Goole Play License Key 입력

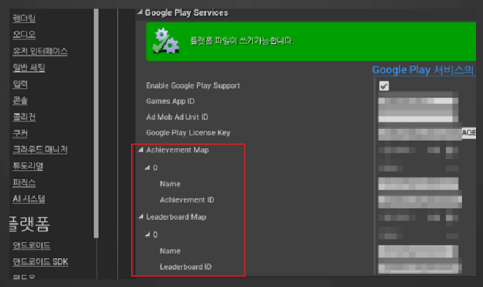
구글 플레이 서비스 + In-app Billing 사용설정

- In-app Billing 활성화를 위해서
  게임프로젝트₩Config₩Android₩AndroidEngine.ini
  에
  [OnlineSubsystemGooglePlay.Store]
  bSupportsInAppPurchasing=true
  추가

- 여기까지 설정하면 기본 준비는 완료!

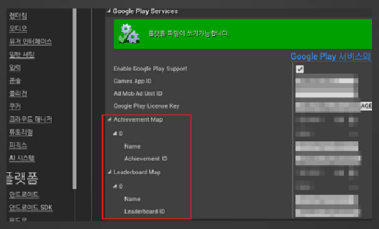
구글 플레이 서비스 사용하기

- 업적과 리더보드 ID 등록하기
  - 프로젝트 세팅에 안드로이드 항목에 Achievement Map과 Leaderboard Map
  - Name은 UE4 내부에서 사용되는 ID 개념
  - Achievement/Leardboard ID는 구글 플레이 콘솔에서 얻은 실제 ID 값



구글 플레이 서비스 사용하기

- 업적과 리더보드 ID 등록하기
  - 프로젝트 세팅에 안드로이드 항목에 Achievement Map과 Leaderboard Map
  - Name은 UE4 내부에서 사용되는 ID 개념
  - Achievement/Leardboard ID는 구글 플레이 콘솔에서 얻은 실제 ID 값

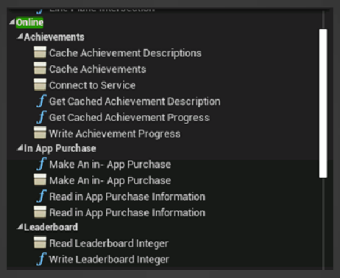


- 이렇게 준비하면 일단 업적과 리더보드 사용 준비는 완료.

UNREAL ENGINE
UNREAL SUMMIT 2015

구글 플레이 서비스 + In-app billing 사용

- 이렇게 준비된 서비스들을 블루프린트에서는 Online에 있는 항목들을 통해서



- 코드에서는
  IOnlineSubsystem::Get(FName(TEXT("GooglePlay")))
  로 OnlineSubsystem을 얻어서 필요한 서비스 인터페이스를 접근해서 함수 호출해서 사용가능

In-app billing

- 그런데 만약 구글 플레이 서비스가 아니라 In-app billing만 사용하고 싶다면?
- 현재는 모든 서비스들이 활성화 되는 것이 기본으로 코드 수정을 통해서
  구글 플레이 서비스는 사용되지 않도록 가능
- OnlineSubsystemGooglePlay.cpp 에 FOnlineSubsystemGooglePlay::Init() 함수에서

  // Queue up a task for the login so that other tasks execute after it.
  부터
  return true;
  전까지 코드를 제거하고, 거기서 추가로

  FJavaWrapper::OnActivityResultDelegate.AddRaw(this, &FOnlineSubsystemGooglePlay::OnActivityResult);

  만 호출하게 하시면 Games App ID등을 설정안하셔서도 크래쉬 없이 In-app billing만 사용 가능

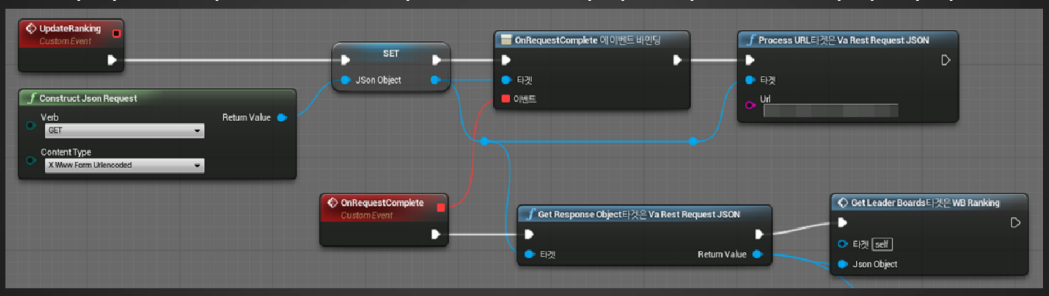
HTTP

HTTP Request

- HTTP 프로토콜을 쓰는 것이 게임에 많이 쓰이고 있음
- UE4에서는 HttpRequest 를 통해서 HTTP프로토콜을 사용 가능
- 단, 블루프린트에서 가능한 인터페이스는 아직 공식적으로 없지만 UE4 유저가 플러그인을 만들어서 공개한 상태 VaRest Plugin https://wiki.unrealengine.com/VaRest_Plugin
- VaRest Plugin을 통해서 HTTP 프로토콜 사용이 가능!

HTTP Request

- 소울 던전에서는 테스트로 HTTP 서버를 사용해서 랭킹 정보의 저장하고 가져오는데 VaRest 플러그인을 통해서 블루프린트에서 처리



- 이 플러그인 이외에도 JSON Query 플러그인도 존재
  - https://forums.unrealengine.com/showthread.php?7045-PLUGIN-JSON-Query&highlight=JSON+QUERY

HTTP Request

- 코드를 통한 사용법
  - FHttpModule을 사용하면 편하게 가능

```
TSharedRef<IHttpRequest> HttpRequest = FHttpModule::Get().CreateRequest();
HttpRequest->SetVerb("GET");
HttpRequest->SetURL(TEXT("http://www.google.com"));
HttpRequest->OnProcessRequestComplete().BindUObject(this, &TestClass::OnTestFuncCompleted);
HttpRequest->ProcessRequest();
```

Callback 함수는

```
void TestClass::OnTestFuncCompleted(FHttpRequestPtr Request, FHttpResponsePtr Response, bool bWasSuccessful)
```

안드로이드 패치

안드로이드 패치

- 구글 플레이의 APK 50MB 제한이기에 컨텐츠에 대한 해결 책 필요
- 구글은 컨텐츠 데이터를 위한 OBB를 지원하고 있음
- UE4도 기본적으로 패키지 프로젝트를 통해서 안드로이드 플랫폼 패키징시에 APK와 OBB 파일을 생성해줌
- 4.8 버전에서 OBB 다운로드 시스템이 지원되어 APK와 OBB를 통한 컨텐츠 데이터 서버 비용 없이 게임 서비스 가능

안드로이드 패치

- 하지만 패치의 용의함과 패치 사이즈 등의 여러가지 이슈로 자체 패치 서버와 패치 시스템을 사용하는 것이 한국에서는 일반적인 모바일 서비스 방법
- 그렇다면 UE4에 이런 솔루션은 없을까?

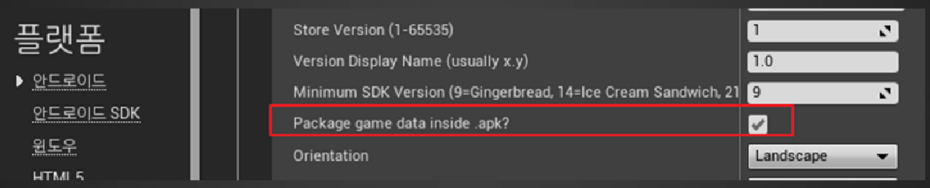
있습니다! 가능!

UE4 패치 시스템 시나리오

- UE4 안드로이드 패치 시스템은 엔진이 구동되어 실행한 상태에서 동작 가능
  - 즉, APK가 구동을 위한 컨텐츠들을 포함하여 배포가 되어야 함
  - 50MB으로 APK를 만들어야 하기 때문에 최소한의 컨텐츠만 포함해함
  - 무엇을 포함해야 하고 어떻게 만들어야 할까?

UE4 패치 시스템 시나리오

- APK 만들기
  - 컨텐츠가 포함된 APK를 만들기 위해서 프로젝트 세팅에 Package game data inside .apk? 옵션을 체크



  - 이 상태로 그냥 패키징을 하게 되면 전체 맵을 포함하게 되고, 프로젝트 런쳐를 쓰게 되도 우리가 원하는(?)을 다 하지 못 함

UE4 패치 시스템 시나리오

- APK 만들기
  - "모바일 개발 설정과 패키징" 세션을 보셨다면 익숙하실 AutomationTool(Engine₩Build₩BatchFiles₩RunUAT.bat) 을 이용해서 원하는 방식으로 패키징된 APK를 만듬
  - 우리가 원하는 것은 50MB 이하의 APK를 만들로 패치에 대한 정보만 보여주는 최소한의 시작 레벨만 가지고 모든 안드로이드 디바이스에서 구동이 되는 텍스쳐 포맷을 가지는 APK를 만들어야 하는 조건
  - 이 조건을 충족하기 위해서 텍스쳐 포맷은 ETC1 을 사용
    - 잘 알고 계시겠지만 알파 채널이 없는 완전 비추 텍스쳐 포맷
    - 하지만 모든 안드로이드 디바이스는 이 텍스쳐 포맷을 지원

UE4 패치 시스템 시나리오

- APK 만들기
  - 패치 관련해서 기본 배경화면이나 배치 여부를 묻는등 꼭 필요한 기본 UI만 표시하는 아주 간단한 패치 타이틀 맵을 만듬
    - 소울 던전 테스트에서는 아주 간단하게 Wifi 연결시에 패치 실행하는 것을 추천 한다는 메시지 창과 패치 시작시 프로그레스바 표시만 하는 UI 구성

UE4 패치 시스템 시나리오

- APK 만들기
  - 더 나아가서 UE4에는 사이즈를 줄이기 위한 좋은 기능이 있음
  - UE4 에서는 배포용 컨텐츠는 보안과 기타의 이유로 그냥 uasset 파일들이 아니라 .pak 파일이라는 컨텐츠들이 합쳐진 파일을 배포하는 것을 지원
  - 이 .pak 파일로 기본적으로 사이즈가 줄어 드는 것은 아니지만! 압축 옵션 지원!
  - 그래서 우리가 사용할 AutomationTool을 실행 할 commandline은

    RunUAT.bat BuildCookRun -project="프로젝트 파일 전체 경로" -noP4 -clientconfig=Development -platform=Android_ETC1 -targetplatform=Android -cookflavor=ETC1 -cook -map=패치를위한타이틀맵이름 -stage -package -cmdline=패치를위한타이틀맵이름 -compressed -pak

  - 이 조건이면 기본 APK 생성 가능!
    - 단, 실행 전에 프로젝트 세팅에 맵 & 모드에 Game Default Map을 패치를위한타이틀맵 으로 설정을 해야 추가적으로 필요없는 다른 Game Default Map 포함이 되지 않음

UE4 패치 시스템 시나리오

- 다음으로 패치 데이터 만들기
  - 패치 데이터는 먼저 APK에 포함된 데이터는 작더라도 빼고 만들어야 좋은데 그럼 어떻게 APK에 포함된 데이터를 분리 할 수 있을까?
  - UE4에 새로 추가중(?)인 DLC(Downloadable Content) 기능 이용!
  - DLC는 이미 배포된 컨텐츠를 제외하고 추가적인 컨텐츠를 배포 할때 유용한 기능
  - 패치 데이터 생성에도 이용하면 초기 APK에 포함된 데이터를 제외한 패치해야 하는 컨텐츠 데이터를 만들 수 있음

UE4 패치 시스템 시나리오

- DLC를 활성화한 쿠킹법
  - AutomationTool을 통해서 쿠킹
  - 유저가 실제 받는 다운로드 받는 패치 사이즈를 줄이기 위해서
    모든 텍스쳐 포맷에 대해서 배포버전에 포함된 컨텐츠 정보를 만들어 두어야 함
  - Android_DXT 포맷에 대한 실행 commandline 은

```
Engine₩Build₩BatchFiles₩RunUAT.bat BuildCookRun -project="프로젝트 파일 전체 경로" -noP4 -clientconfig=Development -serverconfig=Development -platform=Android_DXT -targetplatform=Android -cook -cookflavor=DXT -map=패치를위한타이틀맵이름 -stage -cmdline=패치를위한타이틀맵이름 -Messaging -nocompile -compressed -pak -CreateReleaseVersion=1.0 -newcook
```

  - -platform=Android_DXT 와 -cookflavor=DXT 부분을 지원할 안드로이드 텍스쳐 포맷들에 대해서 변경해서 전부 실행
    예) -platform=Android_ATC -cookflavor=ATC
    -platform=Android_ETC2 -cookflavor=ETC2

UE4 패치 시스템 시나리오

- DLC을 이렇게 쿠킹하고 나면

  게임프로젝트/Releases/1.0/

  아래에 플랫폼 이름의 폴더가 생성되어 있고, 그 안에 초기 배포된 어셋 정보들에 대한 AssetRegistry.bin 파일이 생성되어 있음
  (1.0 은 AutomationTool 실행시에 CreateReleaseVersion 에 설정한 버전명)

- DLC 기반 쿠킹시에 이 정보를 기초로 초기 배포 패키지는 제외한 쿠킹 컨텐츠들을 만들게 됨

UE4 패치 시스템 시나리오

- 이제 실제 배포할 DLC 컨텐츠 쿠킹을 먼저 하면

  Engine₩Build₩BatchFiles₩RunUAT.bat BuildCookRun -project="프로젝트 파일 전체 경로" -noP4 -clientconfig=Development -serverconfig=Development -platform=Android_DXT -targetplatform=Android -cook -cookflavor=DXT -map=배포할맵+배포할맵 -stage -pak -nocompile -BasedOnReleaseVersion=1.0 -DLCName=DLC이름 -NEWCOOK

- 이 항목도 배포할 버전의 텍스쳐 포맷을 위해서 -platform 과 –cookflavor 항목을 수정해서 각 텍스쳐별로 쿠킹 필요
- -map 에는 배포할 맵 이름들을 + 를 이용해서 넣어주면 됨
- DLCName은 원하는 이름을 적당히 정해주시면 됨

UE4 패치 시스템 시나리오

- 이렇게 쿠킹을 완료 하면 초기에 APK에 포함된 컨텐츠를 제외한 컨텐츠만 쿠킹된 데이터들이

  게임프로젝트₩Plugins₩DLC이름₩Saved₩Cooked₩플랫폼

  아래에 쿠킹된 uasset들이

  게임프로젝트₩Plugins₩DLC이름₩Saved₩StagedBuilds₩플랫폼

  아래에는 쿠킹된 uasset들로 만들어진 Pak 파일이 생성되어 있음

- 우리가 실제 배포할 패치 데이터는 StagedBuilds₩플랫폼 폴더를 사용해서 만들게 됨

UE4 패치 시스템 시나리오

- 이제 배포될 컨텐츠까지 다 완성이 된 상태
- 이제 필요한 작업은 어떻게 만들어진 .pak 파일을 유저가 다운로드 받을 수
  있게 해줄 것 인가?

- 이제 핵심적인 패치 만들기 기능과 패치 다운로드 기능을 살펴볼 차례

다운로드 데이터 제작 - BuildPatchTool

- 다운로드 받을 데이터를 제작하기 위해서 사용하는 툴은
  Engine₩Binaries₩Win64₩BuildPatchTool.exe
- 기본적으로 언리얼 런쳐에서 엔진이나 기타 다른 컨텐츠 배포를 위해서
  만들어진 툴!
- 다운로드 받는 데이터를 만들기 위한 좋은 기능을 가지고 있고 계속 진화중!

다운로드 데이터 제작 - BuildPatchTool

- 그럼 전에 단계에서 만든 .pak 파일을 다운로드 받을 수 있는 파일들로 만들어 보면

  Engine₩Binaries₩Win64₩BuildPatchTool.exe -BuildRoot="프로젝트폴더경로₩Plugins₩DLC이름₩Saved₩StagedBuilds₩플랫폼이름" -CloudDir="만들어진다운로드데이터들저장경로" -AppID=0 -AppName="MyGame" -BuildVersion="MyGame-1" -AppLaunch=".₩Engine₩Binaries₩Win64₩UE4Editor.exe" -AppArgs="" -customint="ChunkID=1"

- 이렇게 실행하면 아까 만들어진 .pak 들을 청크 단위로 쪼개서 그 쪼개진 파일들과 그 정보를 가지고 있는 manifest 파일을 CloudDir 폴더에 생성해 줌
- AppName은 크게 중요한 것은 아니고, 추후에 버전이 올라갈 수 있으니

다운로드 데이터 제작 - BuildPatchTool

- 가장 좋은 것은 각 디바이스에서 자기가 지원하는 가장 좋은 텍스쳐 포맷 컨텐츠만 다운로드 하는 것이기 때문에 전에 텍스쳐 포맷 별로 만들어둔 .pak 파일들에 모두 실행해서 각각의 텍스쳐 별 버전을 만드는 것을 추천

- 청크가 만들어진 CloudDir 폴더를 살펴보면
  AppNameBuildVersion.binary.manifest
  AppNameBuildVersion.manifest
  ChunksV3 폴더
  이렇게 3개가 생성되어 있음

- Manifest파일은 어떤 파일이 어떤 청크들로 나누어져 있는지 정보가 있고, 나중에 이 파일을 다운로드해서 다운로드 받아야 하는 청크와 복구를 하는 정보로 사용됨
- ChunksV3 폴더 안에는 청크로 나누어진 폴더와 그 밑에는 청크 파일들이 있음

다운로드 데이터 제작 - BuildPatchTool

- 청크로 만드는 시스템의 큰 장점은 원본 파일을 청크 단위로 쪼개면서 각각
  청크 바이너리 데이터가 같은 값을 가지고 있으면 다시 만들지 않고 그 값을
  재활용해서 결국 원본 .pak가 청크로 나누어지면 실제 다운로드 사이즈가
  작어질 수 있음!
  - 소울 던전 테스트에서는 500MB가 300MB 정도로 작어진 경우도 보임

UNREAL ENGINE
UNREAL SUMMIT 2015

청크 다운로드 받기 - BuildPatchServices

- 이제 만들어진 청크들을 어떻게 다운받아서 복구해야 할까?
- Manifest 기반으로 필요한 청크를 다운받아서 복구하는 모듈인 BuildPatchServices 모듈 지원!
- Amazon S3 같은 웹호스팅 서버에 파일을 올리고 그 URL과 manifest파일을 통해서 필요한 청크를 알아서 다운로드 받아서 복구 해주는 아주 멋진 녀석!

청크 다운로드 받기 - BuildPatchServices

- 실제 모듈 클래스 이름은 IBuildPatchServicesModule
- 그럼 어떤 방식으로 사용하면 되는지 살펴보면
  - 다운로드 할 수 있도록 Manifest 파일과 그 청크들을 웹호스팅 서버에 올려둠
  - Manifest 파일을 먼저 다운로드 해서 IBuildPatchServicesModule에게 다운로드 받아야 하는 파일을 알려줄 준비를 함
  - Manifest들이 다운로드 받은 청크 파일들이 어디 있는지 웹서버 주소를 IBuildPatchServicesModule에게 세팅
  - 기타 디바이스에서 다운로드와 복구된 파일을 넣을 패스를 정해서 IBuildPatchServicesModule에 세팅
  - IBuildPatchServicesModule에게 다운로드 하라고 요청!

청크 다운로드 받기 - BuildPatchServices

- 그럼 이제 각각의 단계를 좀 더 자세히 살펴보면
  - 다운로드를 받기 위해서 BuildPatchServices 모듈 로딩

```
IBuildPatchServicesModule* BuildPatchServices =
&FModuleManager::LoadModuleChecked<IBuildPatchServicesModule>(TEXT("BuildPatchServices"));
```

  - HttpRequest 를 통해서 Manifest 파일을 다운로드
    - 웹서버를 구축하셔서 앱에서 어떤 Manifest 파일을 사용해야 하는지, 아니면 해당 Manifest 정보를 넘겨주는 시스템 필요
  - 웹서버에서 넘겨 받은 Manifest 파일 정보가 JSON이라면
    BuildPatchServices->MakeManifestFromJSON
    바이너리 데이터로 받아왔으면
    BuildPatchServices->MakeManifestFromData
    함수를 통해서 IBuildManifestPtr 생성

청크 다운로드 받기 - BuildPatchServices

- Manifest 파일에서 필요한 다운로드 받을 청크 폴더가 있는 URL 설정
  BuildPatchServices->SetCloudDirectory(CloudURL);
- 디바이스에서 다운로드 받은 청크파일들로 복구된 파일들을 완전히 복구되어 최종 위치에 복사되기 전에 임시로 저장될 폴더 위치 지정
  - 안드로이드 디바이스에서 앱이 사용하는 경로를 가져와야 하고, 그 경로는 AndroidFile.cpp 에 이미 선언되어 있고, GExternalFilePath 이며 사용하시는 소스 코드에서
    extern FString GExternalFilePath; 이렇게 해서 사용하게 가져오시면 됨
  - 이 GExternalFilePath 경로에 예를 들면
    StageDir = FPaths::Combine(*PatchFilePath, TEXT("Patches"), TEXT("Staged"));이 런식으로 해서 설치 경로를 설정
  - 설정된 설치 경로를 세팅
    BuildPatchServices->SetStagingDirectory(StageDir);

청크 다운로드 받기 - BuildPatchServices

- 이제 다운로드 받은 청크를 기반으로 필요한 파일이 있으면 어느 폴더에 설치가 될지 정하는 폴더를 지정하는 것이 필요
  - GExternalFilePath 를 기반으로 적당한 폴더 패스를 만듬. 예를들면 InstallDir = FPaths::Combine(*GExternalFilePath, TEXT("Patches"), TEXT("Installed"));
- 이제 실제 청크를 다운로드 받고, 복구하는 일을 지시하기!
  - BuildInstaller = BuildPatchServices->StartBuildInstall(nullptr, InstallManifest, InstallDir, FBuildPatchBoolManifestDelegate::CreateRaw(this, &FMobilePatchManager::OnDownloadCompleted));
  - StartBuildInstall 함수 호출로 작업 지시를 할 수 있음
  - 만들어둔 IBuildManifestPtr 인스턴스, 설치될 경로, 다운로드와 복구 완료시에 이벤트를 받을 delegate 함수를 지정

청크 다운로드 받기 - BuildPatchServices

- IBuildInstallerPtr BuildInstaller = BuildPatchServices->StartBuildInstall(…) 호출로 넘겨 받은 IBuildInstallerPtr 인스턴스는 유용한 인스턴스

- IBuildInstallerPtr를 통해서 다운로드와 복구 작업을 포함한 전체 진행 상황 얻어오기
  BuildInstaller->GetUpdateProgress();
  앱이 백그라운드로 가고 다시 복귀 했을 때 다운로드와 복구를 멈춤/재시작 할 수 있는 함수 호출
  BuildInstaller->TogglePauseInstall()
  (참고: 앱이 백그라운드로 가고 다시 돌아오고는
  FCoreDelegates::ApplicationWillEnterBackgroundDelegate.AddRaw
  FCoreDelegates::ApplicationHasEnteredForegroundDelegate.AddRaw
  등의 Delegate를 통해서 알 수 있음)

청크 다운로드 받기 - BuildPatchServices

- 다운로드와 복구가 잘 완료되어서 완료 Delegate를 받으면 이제 필요한 것은 우선 사용된 Manifest 파일을 저장하는 것
- 그리고 필요한 것은 다운로드 받아서 복구한 .pak 파일을 시스템에서 찾을 수 있도록 등록하는 것!
  - 다운로드 받은 .pak 파일을 IPlatformFile::FDirectoryVisitor 상속받은 간단한 헬퍼 클래스를 만들어서 InstallDir 에서 .pak 파일을 찾도록 함
  - 찾아진 .pak 파일을 FCoreDelegates::OnMountPak.Execute(PakPath, 0) 를 호출하면 시스템에 .pak 파일이 등록됨
- 이제 실제 타이틀 레벨을 로딩 하면 다운로드 .pak 에서 찾아서 로딩되고 게임 가능!

청크 다운로드 받기 - BuildPatchServices

- 그럼 마지막으로 사용한 Manifest 파일을 저장하는 이유는 무얼까?
- 그것을 살펴보기 전에 그렇다면 처음 만들어서 배포한 .pak에 업데이트가 필요해서 패치를 해야 한다면 어떻게 될까를 먼저 생각해보면?
- 예를들어 .pak 버전 1을 받은 유저가 한동안 업데이트 하지 않아서 한달 후에 접속해서 버전 100이(?) 되었다면? 그 유저는 2-100까지 다 받아야 할까?
- 우리가 원하는 것은 마지막 버전의 .pak 파일을 복원하는 것이다 그 전의 단계는 중요하지 않음
- 그렇기 때문에 버전 100의 manifest 파일에 필요한 청크 파일들만 서버에 있다면 그 버전 100의 Manifest 파일을 통해서 StartBuildInstall 호출에 넘겨서 다운로드 받으면 됨

UNREAL ENGINE UNREAL SUMMIT 2015

청크 다운로드 받기 - BuildPatchServices

- 그런데 그렇다면 우리는 버전 100에 해당하는 모든 청크를 다운로드 받아야 하는 것일까? 전에 받은 마지막 .pak 파일에서 일부분만 바뀌었는데도?
- 그렇지 않음! 이 이유로 마지막 다운로드 받은 Manifest 파일을 저장한 것
- BuildPatchServices->StartBuildInstall(nullptr, InstallManifest, InstallDir, FBuildPatchBoolManifestDelegate::CreateRaw(this, &FMobilePatchManager::OnDownloadCompleted))
  호출 시에 제일 첫 파라미터로 nullptr를 준 곳에 저장했던 Manifest 파일을 로딩해서 그 정보를 넘겨 주면 이미 받은 청크 데이터를 알 수 있기 때문에 받아둔 .pak 파일에는 없는 새로운 청크만 받아서 버전 100의 .pak을 복구 하기 때문에 다운로드 사이즈를 줄 일 수 있음!

패치 시스템 정리

- APK 를 50MB 이하로 만들어서 배포
  - 이 APK는 초기 패치를 받기 위한 최소한의 리소스만 포함
- RunUAT.bat commadline을 통해서 배포해야할 .pak 파일을 각 텍스쳐 포맷 별로 만들어 두고, 이때 DLC 기능을 이용해서 APK 에 포함된 컨텐츠 제외
- BuildPatchTool을 이용해서 만들어진 .pak 파일을 청크단위로 쪼개고, 그 정보를 가진 Manifest 파일 생성
- 웹서버에 Manifest파일과 청크 파일을 올려두고, 게임에서 적절한 Manifest파일을 HttpRequest를 통해서 다운로드 받아서 준비를 해둠
- BuildPatchServices를 생성하고 거기에 청크를 받을 URL 세팅과 복구될 파일 경로등을 정해두고, 다운로드 받은 Manifest파일을 넘겨서 다운로드와 복구 실행
- 복구가 완료되면 사용한 Manifest파일을 저장하고, 복구된 .pak 파일을 찾아서 시스템에
  등록
- 이제 실제 게임 플레이를 시작할 타이틀 맵을 열면 끝!

감사합니다!
Q/A