

목차

소개	1.1
시작하기	1.2
엑셀 하우투	1.2.1
구글 하우투	1.2.2
문제 해결	1.2.3
뛰어나다	1.2.3.1
Google	1.2.3.2
용법	1.3
런타임 로딩	1.3.1
LINQ와 함께 사용	1.3.2
지원되는 셀 유형	1.4
열거형	1.4.1
배열 유형	1.4.2
수식 셀 유형	1.4.3
사례 연구	1.5
공식 계산: 파트 I	1.5.1
공식 계산: 파트 II	1.5.2
공식 계산: 파트 III	1.5.3
팁 및 알려진 문제	1.6
자주하는 질문	1.7
부록	1.8
구글 OAuth2 설정	1.8.1
코드 템플릿	1.8.2
참조	1.9

Unity 퀵시트

Unity-QuickSheet를 사용하면 Unity 편집기 내에서 Google 및 Excel 스프레드시트 데이터를 사용할 수 있습니다. Unity-QuickSheet를 사용하면 스프레드시트에서 데이터를 검색하고 [ScriptableObject](#) 를 사용하여 자산 파일로 저장할 수 있습니다. 코드를 한 줄도 작성하지 않고도 형식을 지정할 수 있습니다.

특징

- 쉬운! 코드를 한 줄도 작성할 필요가 없습니다.
- 편리한! 그것은 엑셀 파일에서 데이터를 검색할 수 있습니다. (xls 및 xlsx 형식은 Windows에서 모두 지원되고 OSX에서는 xls만 지원됩니다.)
- 유연한! Google 스프레드시트에서 데이터를 검색할 수 있습니다.
- 빠른! 데이터를 검색하기 위해 파서를 작성할 필요가 없으며 검색된 데이터를 Unity3D의 [ScriptableObject](#) 로 자동 직렬화합니다. 바이너리 형식이므로 일반적으로 XML을 사용하는 것보다 빠릅니다. ASCII 형식.

다시 말하지만, 코드 한 줄도 작성할 필요가 없습니다!

시작하기

Excel 스프레드시트로 작업하는 경우 '[Excel Howto](#)' 페이지를 참조 하고 Unity-Quicksheet 로 시작하려면 '[Google Spreadsheet Howto](#)' 페이지를 참조하십시오.

또한 기타 정보에 대한 [FAQ](#) 페이지가 있습니다.

여기 에서 Unity 포럼 페이지를 찾을 수도 있습니다. 질문이나 제안이 있으면 포럼에 게시하거나 [github 프로젝트 페이지](#)에 문제로 남겨주세요. 모든 피드백을 환영합니다!

용법

스프레드시트 데이터를 Unity 에디터로 임포트해야 하는 모든 곳에서 'Unity-QuickSheet'를 사용할 수 있습니다. 쉽고 빠릅니다.

- 클라이언트 측에서 데이터베이스로 사용하십시오. json이나 xml에 비해 쉽고 빠릅니다.
- LINQ와 함께 사용하십시오. 훨씬 쉽게 데이터를 처리할 수 있습니다.
- int, float 등의 기본형 뿐만 아니라 enum, array 등 다양한 형태를 지원합니다.
- 현지화.
- 공식 계산의 자동화. [사례 연구](#) 섹션 을 참조하십시오 .

특허

이 코드는 MIT 라이선스 조건에 따라 배포됩니다 .

일부 코드는 [GDataDB에서 빌렸습니다](#). 그들의 라이선스를 따릅니다.

Copyright (c)2013 김현우

엑셀로 작업하는 방법

이 게시물은 [Unity-QuickSheet](#) 를 설정하고 사용하는 방법을 보여주고 도와줍니다. 엑셀로.

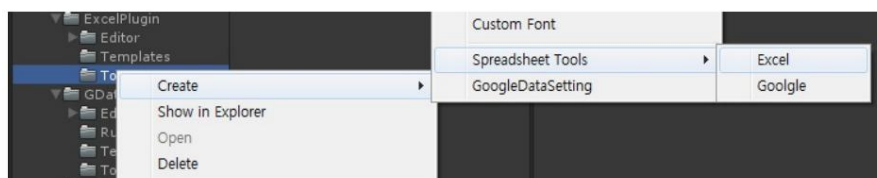
Excel 버전 97/2000/XP/2003은 '.xls' 에 대해 지원 됩니다(해당 [NPOI로 인해](#), Unity-QuickSheet 용 Excel 스프레드시트를 읽는 데 사용되는 오픈 소스 프로젝트 는 '.xls' 용 Excel 버전 5.0/95를 지원 하지 않습니다. '.xlsx' 와 상관없습니다 .

	A	B	C	D	E	F	G
1	MeleeSkillLevel	STR	DEX	INTL	TotalDamage	TotalArmor	Health
2	0	10	10	10	8	16	56
3	1	13	11	11	10	20	66.6
4	2	16	12	12	12	24	77.2
5	3	19	13	13	14	28	87.8
6	4	22	14	14	16	32	98.4
7	5	25	15	15	18	36	109
8	6	28	16	16	20	40	119.6
9	7	31	17	17	22	44	130.2
10	8	34	18	18	24	48	140.8
11	9	37	19	19	26	52	151.4
12	10	40	20	20	28	56	162
13	11	43	21	21	30	60	172.6
14	12	46	22	22	32	64	183.2
15	13	49	23	23	34	68	193.8
16	14	52	24	24	36	72	204.4

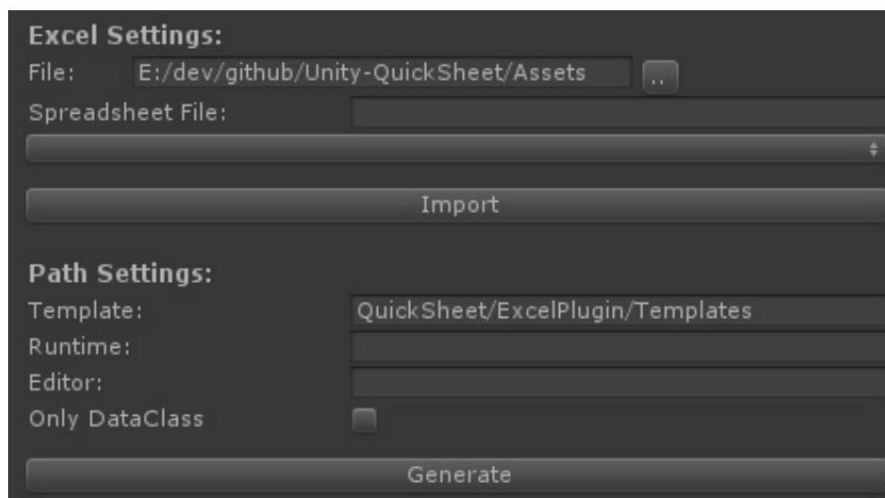
시작하기 전에 스프레드시트 페이지를 다시 확인하세요. 첫 번째 행이 빈 행이 아니어야 함을 의미하는 빈 행 없이 시작해야 합니다.

1단계) 엑셀 설정 파일 생성

먼저 엑셀 설정 파일을 생성해야 합니다. 프로젝트 보기를 마우스 오른쪽 버튼으로 클릭하고 '만들기 > 스프레드시트 도구 > Excel'을 선택하기만 하면 됩니다. 스크립트 파일을 생성하고 지정된 엑셀 파일에서 데이터를 가져오기 위한 다양한 설정을 보여주는 새 파일을 생성합니다.



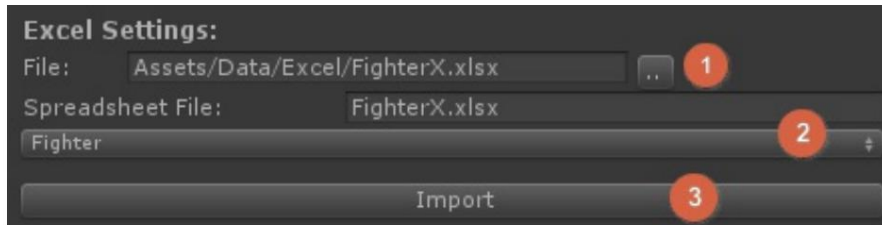
Excel 메뉴 항목을 선택 하면 설정 파일이 생성됩니다. 다음과 같이 표시될 수 있습니다.



설정을 시작합니다.

파일 설정

파일 설정은 가져올 엑셀 파일과 시트 페이지를 설정하는 것입니다.



먼저 가져올 엑셀 파일을 지정해야 합니다.

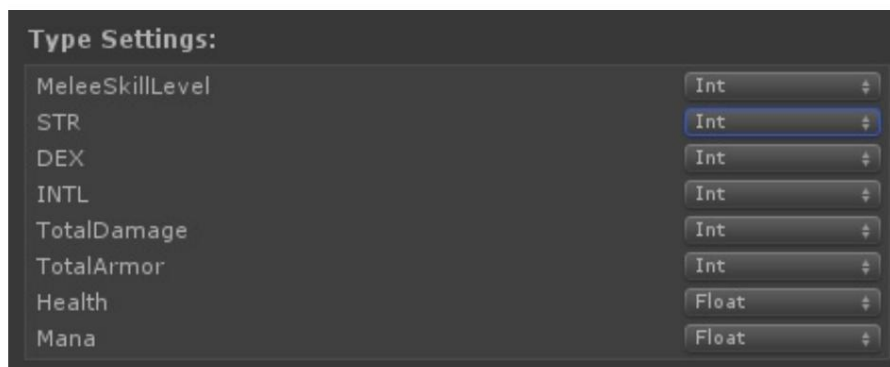
1. 파일 열기 다이얼로그에서 불러올 엑셀 파일을 선택합니다.

Excel 파일에는 하나 이상의 시트 페이지가 있을 수 있으므로 어떤 시트를 선택하고
에서 데이터를 검색합니다.

1. 가져올 시트 페이지를 선택합니다.
2. 가져오기 버튼을 누르면 지정된 엑셀 파일을 가져오고 모든 열 머리글을 표시합니다.

유형 설정

지정된 시트 페이지를 가져오면 페이지의 모든 열 머리글이 표시됩니다. 이는 각 셀의 유형을 설정하는 데 필요합니다.



적절한 유형의 셀을 설정합니다.

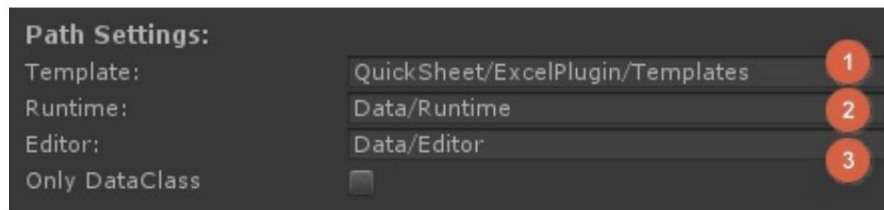
현재 다음 유형이 지원됩니다.

- 문자열
- 정수
- 뜨다
- 더블
- 열거
- 부울

경로 설정

경로 설정은 생성된 스크립트 파일이 저장되는 경로를 지정하는 것과 관련이 있습니다.

참고: 모든 경로는 'Assets/' 없이 상대적이어야 합니다. `



1. 템플릿 은 스크립트 파일을 생성하기 위해 필요한 템플릿 파일이 있는 경로를 나타냅니다. ~ 안에 대부분의 경우 변경할 필요가 없습니다.
2. 런타임 은 런타임에 사용되는 생성된 스크립트 파일이 위치할 경로를 나타냅니다.
3. Editor 는 Editor 모드에서 사용되는 생성된 스크립트 파일이 들어갈 경로를 나타냅니다.

2단계) 스크립트 파일 생성

필요한 모든 설정을 완료했다면 Excel 파일의 시트 페이지에서 데이터를 읽고 프로젝트 보기 에서 자산 파일로 사용되는 ScriptableObject 에 저장하는 데 필요한 일부 스크립트 파일을 생성해야 합니다.

생성 버튼을 누릅니다 .

일부 스크립트 파일을 생성한 후 Unity 에디터는 해당 파일을 컴파일하기 시작합니다. Unity가 컴파일을 종료할 때까지 기다린 다음 지정된 편집기 및 런타임 경로를 확인하여 필요한 모든 스크립트 파일이 올바르게 생성되었는지 확인합니다.

Editor 폴더 에는 두 개의 파일이 있어야 합니다.

- your-sheetpage-nameAssetPostProcessor.cs your-
- sheetpage-nameEditor.cs

런타임 에서 폴더에는 견인 파일이 포함되어야 합니다.

- 귀하의 시트 페이지 이름.cs 귀하
- 의 시트 페이지 이름 데이터.cs

your-sheetpage-nameData.cs 파일 을 참조하십시오 . 파일의 클래스 멤버는 시트 페이지의 각 셀을 나타냅니다.

```

UnityEngine 사용
System.Collections 사용 ;

[System.Serializable] 공개 클래스
래스 FighterData {

    [SerializeField] int
    meleeskilllevel;

    [노출 속성]
    공개 int Meleeskilllevel {
        get {return meleeskilllevel; } set
        { meleeskilllevel = 값; }
    }

    [직렬화 필드] int str;

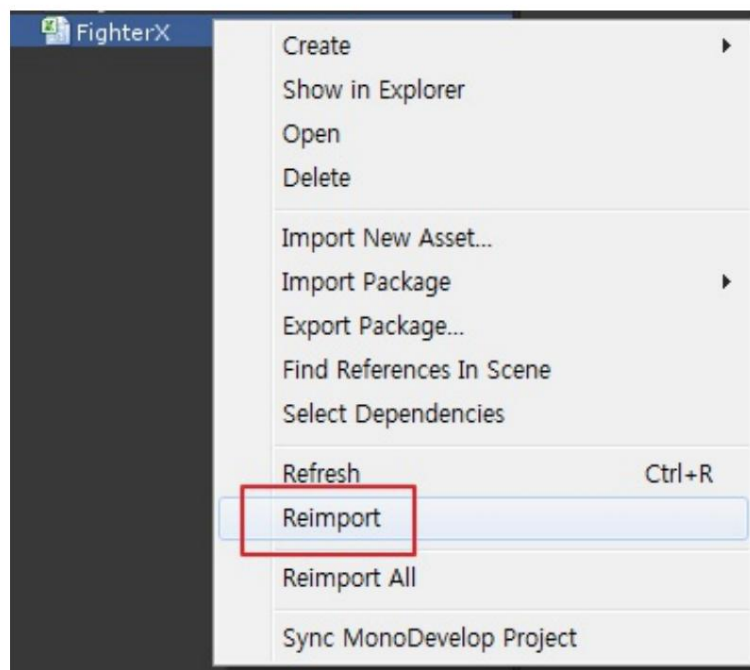
    [ExposeProperty]
    public int STR { get {return str; } 세트 { str = 값; } }

    ...
}

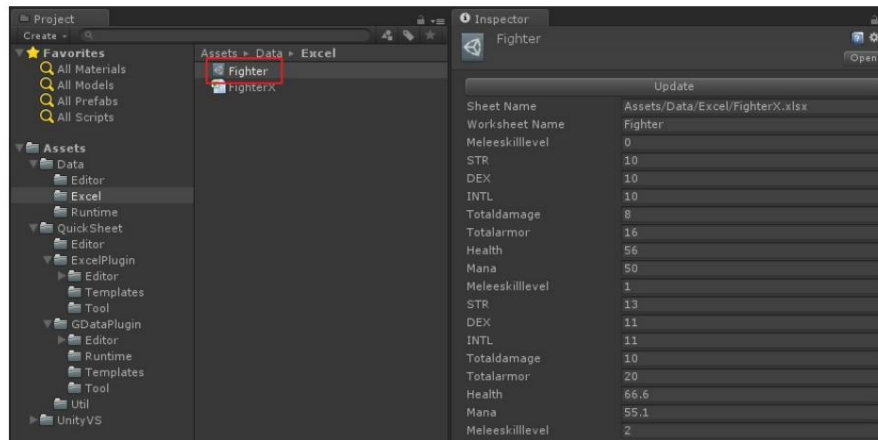
```

3단계) 스프레드시트 데이터 가져오기

자산 파일을 생성하고 스프레드시트 파일에서 생성된 자산 파일로 데이터를 가져오는 것은 프로젝트 보기 내에서 xls 또는.xlsx 파일을 다시 가져오기만 하면 됩니다.



xls 또는.xlsx 파일을 다시 가져 오면 시트 페이지 이름과 동일한 파일 이름(엑셀 파일 이름 아님) 을 가진 자산 파일이 자동으로 생성되고 Excel 파일 의 시트 페이지에서 생성된 자산 파일로 데이터를 자동으로 가져옵니다.



끝났다. 당신이 그것을 즐기시기 바랍니다!

Google 스프레드시트로 작업하는 방법

이 문서 페이지에서는 [Unity-QuickSheet](#) 설정 및 사용에 대해 설명하고 도움을 줍니다. '구글 스프레드시트' 로

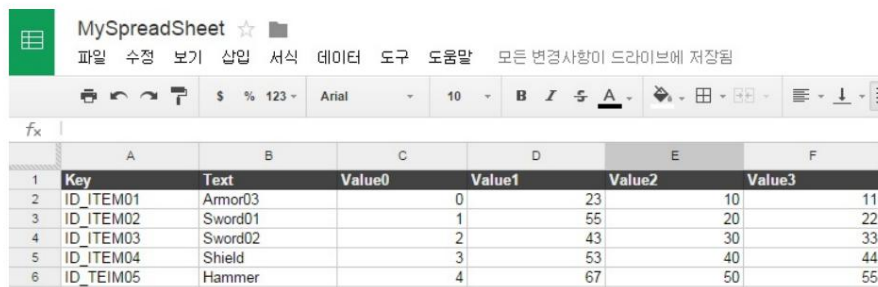
Google 드라이브에 액세스하기 위한 OAuth2 설정

Google은 2015년 5월 5일부터 인증 체계를 변경했습니다. 이제 Google 스프레드시트에 액세스하려면 OAuth2를 설정해야 합니다. 이를 설정하려면 [Google 개발자 콘솔](#) 을 방문하세요. 그런 다음 새 프로젝트를 만들고 Drive API를 활성화하고 "서비스 계정" 유형의 새 클라이언트 ID를 만들고 json 파일을 다운로드합니다.

자격 증명 을 설정하고 Google spreadsheet 설정 파일을 설정하는 데 필요한 OAuth2 'client_ID' 및 'client_secret' 을 가져오는 방법은 [이 페이지](#) 를 참조하세요 .

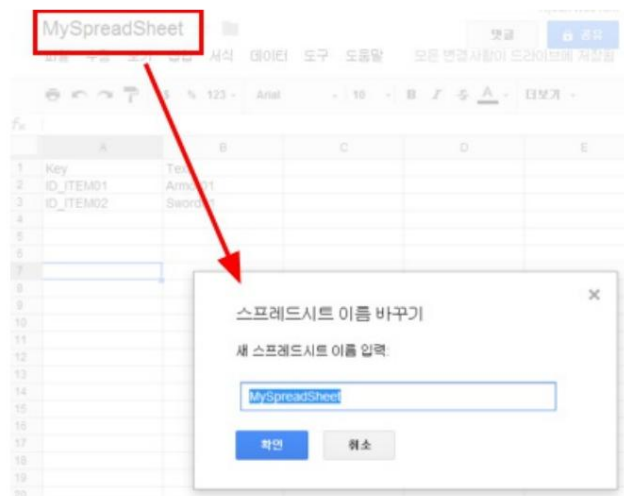
스프레드시트 및 워크시트 만들기

'Google 드라이브'에 로그인한 후 'Google 드라이브'에 Google 스프레드시트를 만듭니다.
계정.

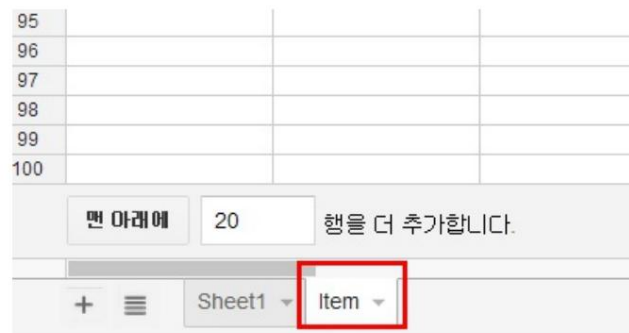


	A	B	C	D	E	F
	Key	Text	Value0	Value1	Value2	Value3
2	ID_ITEM01	Armor03	0	23	10	11
3	ID_ITEM02	Sword01	1	55	20	22
4	ID_ITEM03	Sword02	2	43	30	33
5	ID_ITEM04	Shield	3	53	40	44
6	ID_ITEM05	Hammer	4	67	50	55

생성된 스프레드시트의 제목을 다음과 같이 'MySpreadSheet' 로 변경합니다.



다음으로 새 워크시트를 만들고 다음 이미지와 같이 원하는 이름으로 이름을 바꿉니다.



이제 스프레드시트의 셀을 편집해야 합니다. 생성된 첫 번째 행에 'Key'와 'Text' 삽입
다음과 같은 워크시트:

	fx Key	
	A	B
1	Key	Text
2	ID_ITEM01	Armor01
3	ID_ITEM02	Sword01
4		
5		

중요 첫 번째 행의 셀 이름이 멤버 필드의 이름으로 사용됩니다.

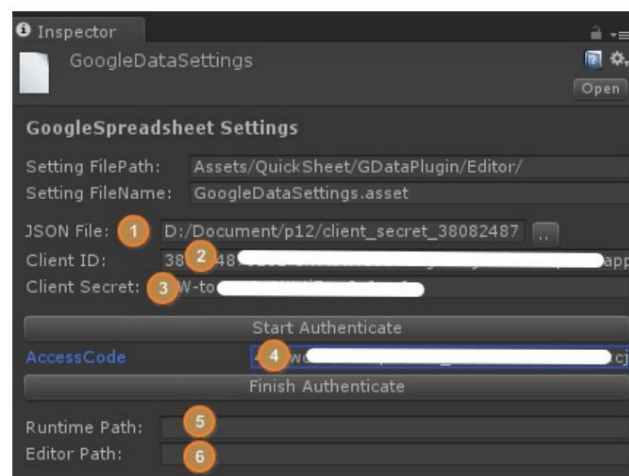
생성된 클래스는 이 페이지의 뒷부분에서 볼 수 있습니다. 따라서 C#의 키워드인 'int', 'string'과 같은 이름을 사용하지 않도록 주의하십시오.

Google OAuth2 서비스 계정

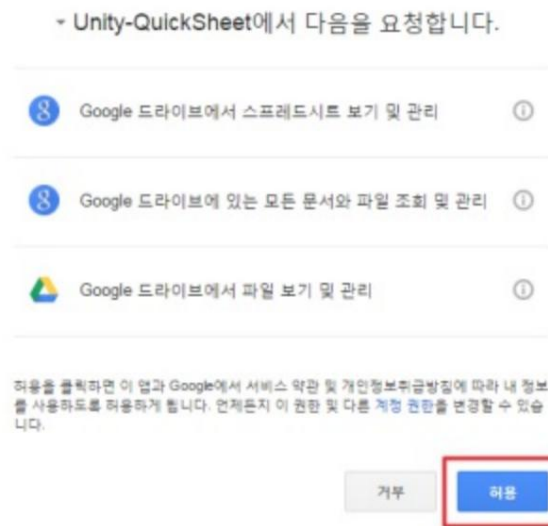
계속 진행하기 전에 'Google OAuth2 계정'을 만들어 계정을 확인하고 Google 스프레드시트에서 Unity에 액세스할 수 있도록 해야 합니다.

아직 'OAuth2 서비스 계정'을 설정하지 않았다면 이 페이지에 있는 [Google API 페이지의 OAuth2 설정](#)을 참조하세요. Unity 에디터로 데이터를 가져오기 전에 먼저 설정해야 합니다.

json 개인 키를 성공적으로 가져 오면 'Assets/QuickSheet/GDataPlugin/Editor' 폴더 아래에 있는 'GoogleDataSettings.asset' 파일을 선택합니다.

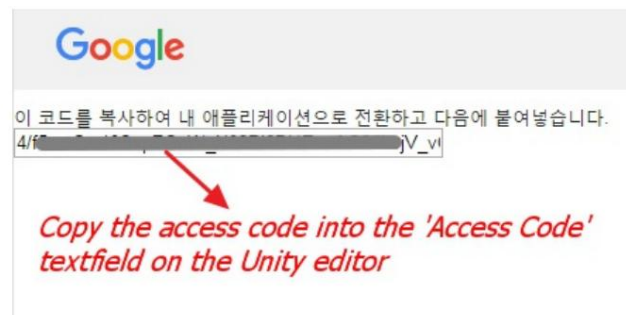


1. 먼저 다운로드한 json 개인키를 'JSON 파일'로 설정합니다.
2. 이제 2) '클라이언트 ID' 와 3) '클라이언트 비밀번호' 가 자동으로 지정되는 것을 볼 수 있습니다. 3. '인증 시작' 버튼을 클릭합니다. 다음 이미지와 함께 브라우저가 시작됩니다.



'허용' 버튼을 선택하고 다음으로 이동합니다.

이제 '액세스 코드'가 표시됩니다. 복사하여 Unity의 4) '액세스 코드' 설정에 붙여넣습니다.



마지막 단계는 '인증 완료' 버튼을 클릭하여 자격 증명을 확인하는 것입니다.

그리고 프로젝트의 '런타임 경로' 및 '에디터 경로'와 같은 다른 설정을 지정합니다. 다음과 같이 설정하면 충분합니다.

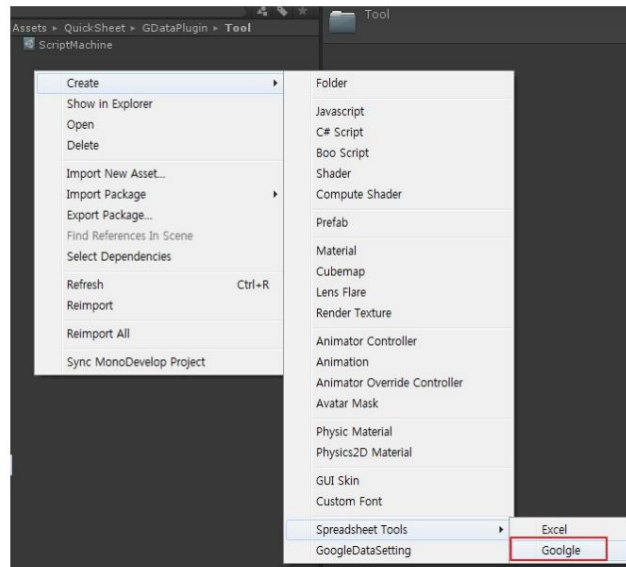
런타임 경로: 데이터/런타임

에디터 경로: 데이터/에디터

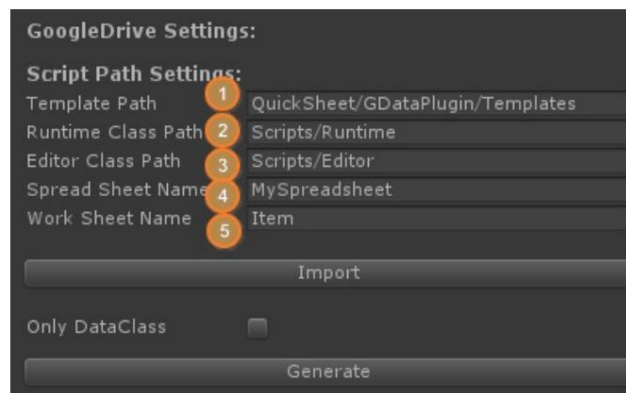
'Data' 폴더가 'Assets' 폴더 아래에 있다고 가정 합니다.

1단계) 구글 스프레드시트 설정 파일 생성

먼저 구글 스프레드시트 설정 파일을 생성해야 합니다. 프로젝트 보기를 마우스 오른쪽 버튼으로 클릭하고 '만들기 > 스프레드시트 도구 > Google'을 선택하기만 하면 됩니다. 스크립트 파일을 생성하고 지정된 Google 스프레드시트에서 데이터를 가져오기 위한 다양한 설정을 보여주는 새 파일을 생성합니다.



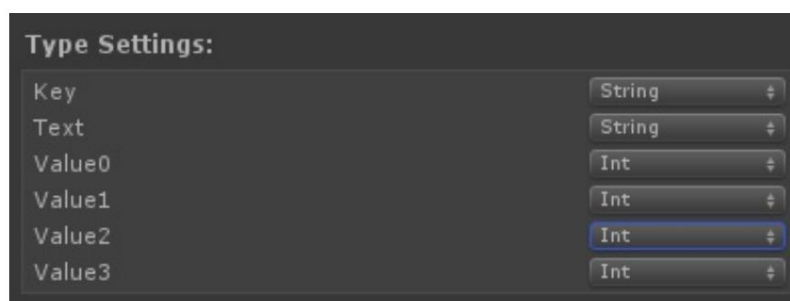
Google 메뉴 항목을 선택 하면 설정 파일이 생성됩니다. 다음과 같이 표시될 수 있습니다.



스크립트 경로 설정

1. 템플릿 은 스크립트 파일을 생성하기 위해 필요한 템플릿 파일이 있는 경로를 나타냅니다. ~ 안에 대부분의 경우 변경할 필요가 없습니다.
2. 런타임 은 런타임에 사용되는 생성된 스크립트 파일이 위치할 경로를 나타냅니다.
3. Editor 는 Editor 모드에서 사용되는 생성된 스크립트 파일이 들어갈 경로를 나타냅니다.
4. 스프레드시트 이름 은 구글 드라이브에서 생성된 스프레드시트의 이름입니다.
5. 워크시트 이름 은 데이터를 가져오려는 스프레드시트의 워크시트 이름 중 하나입니다.에서.

모든 경로 설정을 완료한 후 가져오기 버튼을 누르면 페이지의 모든 열 헤더가 표시됩니다. 이는 각 셀의 유형을 설정하는 데 필요합니다.



적절한 유형의 셀을 설정합니다.

현재 다음 유형이 지원됩니다.

- 문자열
- 정수
- 뜨다
- 더블
- 열거
- 부울

2단계) 스크립트 파일 생성

필요한 모든 설정을 완료했다면 Excel 파일의 시트 페이지에서 데이터를 읽고 프로젝트 보기 에서 자산 파일로 사용되는 ScriptableObject 에 저장하는 데 필요한 일부 스크립트 파일을 생성해야 합니다.

생성 버튼을 누릅니다 .

일부 스크립트 파일을 생성한 후 Unity 에디터는 해당 파일을 컴파일하기 시작합니다. Unity가 컴파일을 종료할 때까지 기다린 다음 지정 된 편집기 및 런타임 경로를 확인하여 필요한 모든 스크립트 파일이 올바르게 생성되었는지 확인합니다.

Editor 폴더 에는 두 개의 파일이 있어야 합니다.

- your-sheetpage-nameAssetCreator.cs your-
- sheetpage-nameEditor.cs

런타임 에서 폴더에는 건인 파일이 포함되어야 합니다.

- 귀하의 시트 페이지 이름.cs 귀하
- 의 시트 페이지 이름 데이터.cs

your-sheetpage-nameData.cs 파일 을 참조하십시오 . 파일의 클래스 멤버는 시트 페이지의 각 셀을 나타냅니다.

UnityEngine 사용

System.Collections 사용 ;

///

/// !!! 기계 생성 코드 !!!

/// !!! 탭을 공백으로 변경하지 마세요!!!

///

[System.Serializable] 공개 클래스

스 PlayerItemData {

[SerializeField] 문자열

키;

공개 문자열 키 { get { 반환 키; } 설정 { 키 = 값; } }

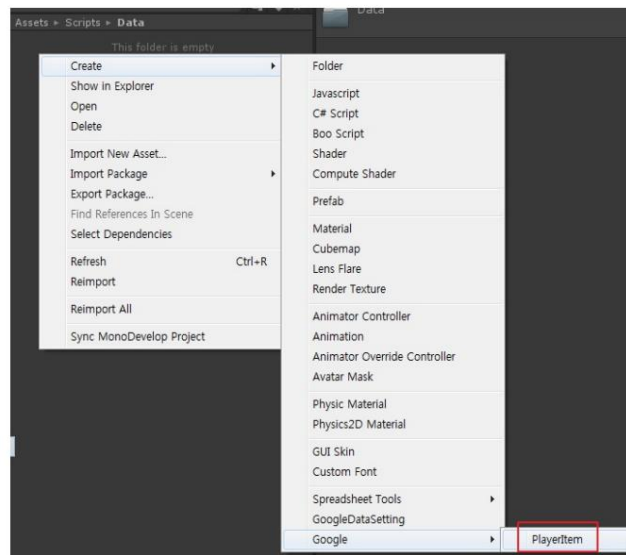
[SerializeField] 문자열

텍스트;

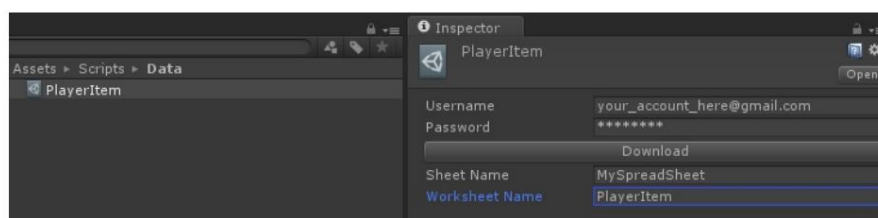
공개 문자열 텍스트 { get { 반환 텍스트; } 세트 { 텍스트 = 값; } }

3단계) 스프레드시트 데이터 가져오기

이제 Google 드라이브에서 모든 데이터를 가져오고 저장할 자산 파일을 만들어야 합니다. 프로젝트 보기를 마우스 오른쪽 버튼으로 클릭하고 '만들기 > Google'을 선택하기만 하면 됩니다. 이제 Google 스프레드시트의 워크시트 이름과 동일한 이름을 가진 새 메뉴 항목이 있습니다.



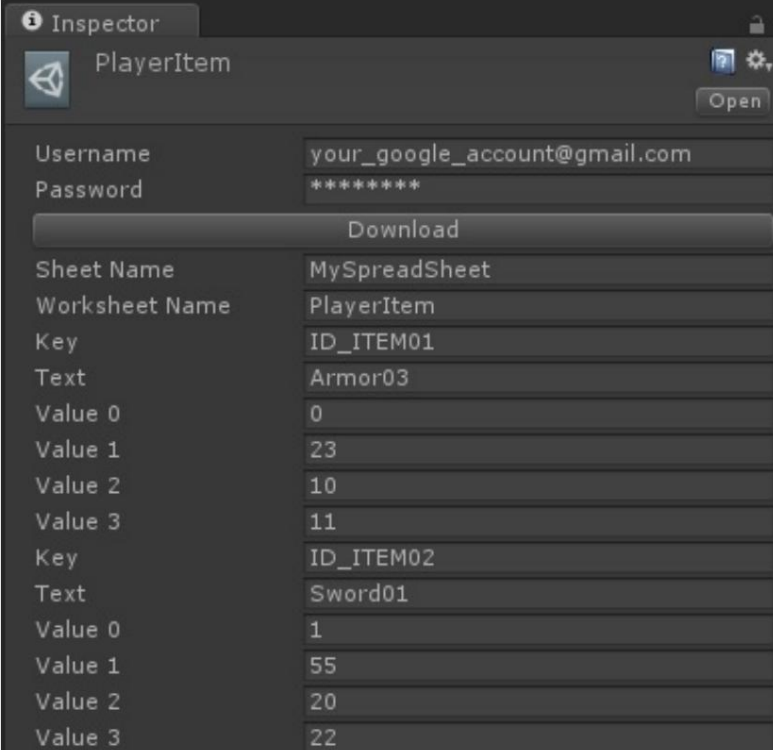
메뉴 항목을 선택하면 자산 파일이 생성됩니다. 다음과 같이 표시될 수 있습니다.



생성된 자산 파일은 지정된 워크시트 이름과 동일한 파일 이름을 갖습니다.

1. Google 계정의 사용자 이름 과 비밀번호 를 입력합니다.
2. Google 설정과 동일한 스프레드시트 및 워크시트 이름을 지정합니다.
3. 다운로드 버튼을 누르면 구글 드라이브에서 데이터 가져오기가 시작됩니다. (몇몇 시간이 소요될 수 있습니다. 초.)

다운로드가 완료되면 다음과 같이 Inspector View에 가져온 데이터가 표시됩니다.



The image shows a software interface titled 'Inspector' with a sub-header 'PlayerItem'. It contains a form for user authentication and data entry. The 'Username' field is filled with 'your_google_account@gmail.com' and the 'Password' field is masked with asterisks. Below these is a 'Download' button. The main section of the form is a table with two data entries. The first entry has a 'Key' of 'ID_ITEM01' and a 'Text' of 'Armor03', followed by four 'Value' fields containing 0, 23, 10, and 11. The second entry has a 'Key' of 'ID_ITEM02' and a 'Text' of 'Sword01', followed by four 'Value' fields containing 1, 55, 20, and 22. There is also an 'Open' button in the top right corner.

Field	Value
Username	your_google_account@gmail.com
Password	*****
Download	
Sheet Name	MySpreadSheet
Worksheet Name	PlayerItem
Key	ID_ITEM01
Text	Armor03
Value 0	0
Value 1	23
Value 2	10
Value 3	11
Key	ID_ITEM02
Text	Sword01
Value 0	1
Value 1	55
Value 2	20
Value 3	22

끝났다. 당신이 그것을 즐기시기 바랍니다!

문제나 문제가 발생하면 여기의 [TroubleShoting 페이지](#) 를 참조하십시오.

문제 해결

뛰어나다

'콘텐츠 유형 부분을 읽을 수 없습니다!' Mac의 오류

Mac 컴퓨터에서 .xlsx 파일 을 열면 '콘텐츠 유형 부분을 읽을 수 없습니다!' 오류가 발생합니다.

'xls' 로 저장 한 다음 다시 엽니다. 문제를 해결합니다.

지원되는 버전의 '.xls' 파일

Excel 버전 97/2000/XP/2003은 '.xls' 에 대해 지원 됩니다(해당 [NPOI로 인해](#), Unity-QuickSheet 용 Excel 스프레드시트를 읽는 데 사용되는 오픈 소스 프로젝트 는 '.xls' 에 대해 Excel 버전 5.0/95를 지원하지 않습니다 . 하지만 '.xlsx' 에 대해서는 신경 쓰지 않습니다 .

Excel Deserialize 예외 예외 오류

Unity 콘솔에는 'Excel Deserialize Exception: Object reference not set to instance of objectRow[5], Cell[6] Is that cell empty?'과 같은 예외 오류가 있을 수 있습니다. 스프레드시트의 셀이 비어 있는 경우.

주어진 행과 오류의 셀 인덱스가 있는 셀을 확인합니다. 그런 다음 셀이 문자열 유형으로 정의되고 비어 있지 않은 것을 확인하십시오. 셀이 비어 있으면 적절한 데이터를 채워십시오.

Google

잘못된 자격 증명 오류

'다운로드' 버튼을 클릭하여 데이터를 가져오려고 할 때 잘못된 자격 증명으로 표시되는 오류가 발생한 경우 Google 계정 페이지와 2단계 인증이 있는지 확인하세요.

Google 2단계 인증이 켜져 있으면 Google 비밀번호가 무엇이든 상관없이 승인되지 않습니다. 애플리케이션 특정 비밀번호(ASP)라고 하는 것을 (Google에서) 생성해야 합니다. [Google 계정 페이지](#) 로 이동 그리고 ASP를 설정하고 생성한 비밀번호를 코드에 비밀번호로 입력하면 완료됩니다.

보안 오류

Unity 웹 플레이어의 보안 샌드박스에서 Google 스프레드시트 플러그인이 작동하지 않습니다. 빌드 설정 에서 플랫폼 을 'Stand Alone' 또는 'iOS' 또는 'Android' 플랫폼으로 변경 해야 합니다.

런타임 로딩

내보낸 .asset 스크립팅 가능한 객체 파일을 런타임에 장면으로 로드하는 방법은 Unity3D의 다른 방법과 다르지 않습니다.

아래 이미지와 같이 간단한 스프레드시트가 있다고 가정해 보겠습니다.

	A	B	C	D	E	F
1	Key	Text	Value0	Value1	Value2	Value3
2	ID_ITEM01	Armor03	0	23	10	11
3	ID_ITEM02	Sword01	1	55	20	22
4	ID_ITEM03	Sword02	2	43	30	33
5	ID_ITEM04	Shield	3	53	40	44
6	ID_ITEM05	Hammer	4	67	50	55

그리고 이미 .asset scriptableObject로 내보냈습니다.

.asset scriptableObject를 로드하기 위해 해야 할 일은 조립식 파일의 경우와 동일합니다.

먼저 새 MonoBehaviour 스크립트 파일을 만든 다음 PlayerItem 클래스 인스턴스를 public으로 선언합니다.

```

공개 클래스 SimpleLoader : MonoBehaviour {

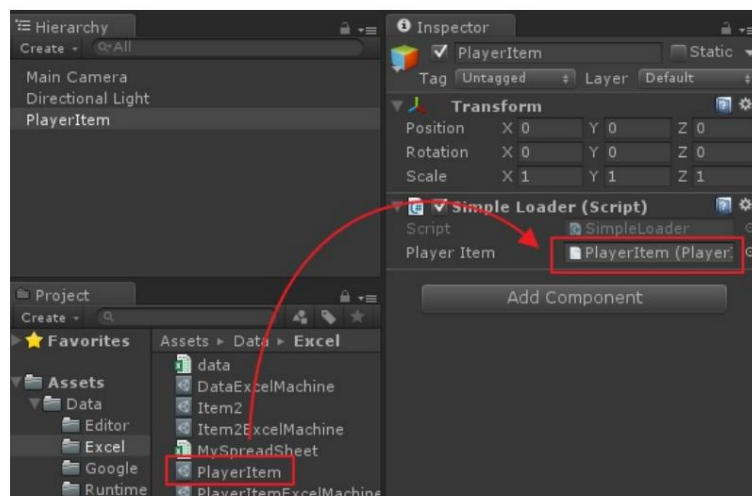
    // 이미 PlayerItem 클래스가 있습니다. public PlayerItem
    playerItem;

    무효 시작 () {

        // PlayerItem.dataArray 배열은 내보낸 모든 데이터를 포함합니다.
        // 콘솔에 Value0 값, 0을 출력합니다.
        Debug.LogFormat("값0: {0}", playerItem.dataArray[0].Value0);
    }
}

```

이제 .asset 파일을 끌어서 Inspector 보기에 놓기만 하면 됩니다.



이것이 런타임에 .asset 파일을 로드하는 모든 것입니다. 이제 플레이를 시작하면 'Value0' 값이 입력됩니다.
콘솔에서.

링크 사용

문서의 이 섹션에서는 LINQ(Language Integrated Query) 를 사용하여 복잡한 스프레드시트에서 내보낸 특정 데이터 세트를 쿼리하는 방법을 설명합니다.

[101-LINQ 샘플](#) 을 보는 것이 좋습니다. 아직 LINQ에 익숙하지 않다면. 또한 LINQ에 대한 문서가 많이 있으므로 인터넷 검색을 수행하십시오.

좋습니다, 시작합니다.

Excel 과 같은 스프레드시트는 이미 알고 있듯이 게임 디자이너가 게임 내에서 사용되는 다양한 테이블과 데이터 세트를 만들기 위해 매일 사용하는 가장 강력한 도구 중 하나입니다.

다음 이미지는 게임의 모든 무기 아이템에 대한 테이블 데이터를 보여줍니다. 표시된 열이 테이블의 모든 열이 아니라는 점에 유의하십시오. 특히 게임이 RPG 장르인 경우 일반적인 게임의 경우 훨씬 더 많을 수 있습니다.

	A	B	C	D	E	F	G	H	I	Q	R	S	T	U	V
1	ID	Name	STR	DEX	INT	FTH	Uni	Cr	Ph	S	Dx			WeaponType	DamageType
2	1000	Battle Axe	12	8	0	0	4	100	250	C	D			Axe	Standard
3	1001	Hand Axe	9	8	0	0	2.5	100	220	C	D			Axe	Standard
4	1002	Thrall Axe	8	8	0	0	1.5	100	208	C	C			Axe	Standard
5	1003	Dragonslayer's Axe	18	14	0	0	4	100	180	D	D			Axe	Standard
6	1004	Butcher Knife	24		0	0	7	100	190	S				Axe	Slash
7	1005	Brigand Axe	14	8	0	0	3	100	248	C	D			Axe	Standard
8	1006	Winged Knight Twinaxes	20	12	0	0	8.5	100	244	C	D			Axe	Standard
9	1007	Elonora	20	8	0	0	6.5	100	284	D	D			Axe	Standard
10	1008	Man Serpent Hatchet	16	13	0	0	4	100	250	C	D			Axe	Standard
11	1009	Short Bow	7	12	0	0	2	100	154	E	D			Bow	Projectile

주어진 스프레드시트의 테이블이 아무리 복잡하더라도 Unity-Quicksheet를 통해 테이블을 Unity 편집기로 쉽게 가져올 수 있습니다. 따라서 Unity로 테이블 가져오기가 성공적으로 완료되었다고 가정합니다.

이제 클래스를 다음과 같이 표시하는 WeaponTable.cs 스크립트 파일을 자동으로 생성했습니다.

```

공개 클래스 WeaponData {

    공개 정수 ID { 가져오기; 세트; }
    공개 문자열 이름 { get; 세트; } 공개 int STR { 가져오기; 세트; }
    ...
}

```

그리고 .asset 파일 을 위한 ScriptableObject 파생 클래스도 있습니다.

```

공개 클래스 WeaponTable : ScriptableObject {

    ...
    공개 무기 데이터[] dataArray;
    ...
}

```

항목을 쉽게 관리할 수 있도록 스프레드시트의 모든 항목에는 해당 범위에서 항목 유형을 식별하는 데 사용되는 고유 ID(첫 번째 열 참조)가 있습니다.

유형	범위
무기	1000 ~ 1999
갑옷	2000년 ~ 2999년
소모품	3000 ~ 3999

범위는 항목 유형의 수 또는 항목 자체의 각 수에 따라 다양할 수 있습니다.

따라서 항목 ID의 정수 배열을 갖는 것만으로도 인벤토리에 있는 항목을 설명하기에 충분합니다.

```
공개 클래스 인벤토리 {
    ...
    // 현재 인벤토리에 속한 아이템 int[] items = {
        1000, 1001, 1002, 1003, 2002, 2003, 2004, 2005,
        2006, 2007, 2008, 2009,
        2010년, 2011년, 2012년, 2013년, 2014년, 2015년,
        2016, 2017, 2018, 4014, 4015, 4016,
        4017, 4018, 4019, 4020, 4021, 4022,
        4023, 4024, 4025, 4026, 4027, 4028, 4029,
        4030, 4031, 4032, 4033, 4034, 4035, 4036
    };
    ...
}
```

그런 다음 쿼리를 수행할 수 있는 인벤토리에 있는 실제 항목 데이터를 검색해야 합니다.
항목 테이블에.

ItemManager 는 인벤토리의 다양한 항목을 처리하고 액세스하는 메서드를 제공하는 클래스입니다.
Unity-Quicksheet에서 자동으로 생성되는 인스턴스로 무기 멤버 필드를 기록해 두십시오 .

```

공용 클래스 ItemManager : 싱글톤<ItemManager> {

    공개 WeaponTable WeaponTable;
    ...
    // 무기 ID: 1000~1999 public
    List<weaponData> GetWeaponList(int[] ids)
    {
        List<weaponData> 항목 = new List<weaponData>();

        var 무기 ID = ids.where(x => x >= 1000 && x < 2000).ToList();
        var WeaponResult = 무기 테이블.dataArray의 w에서 // 스프레드시트의 모든 무기 유형 항목

        .
        무기 ID의 n에서 // 인벤토리에 있는 무기.
        여기서 w.ID == n
        선택 w; 반환 무

        기 결과.ToList();
    }
}

```

위의 코드에서 알 수 있듯이 GetWeaponList 메서드와 같이 LINQ를 사용하여 무기 유형의 항목을 쉽게 검색할 수 있습니다.

스프레드시트의 테이블을 일종의 데이터베이스로 간주합니다. 그런 다음 편집 시간에 Unity-Quicksheet를 사용하여 데이터를 추출하고 Unity로 가져오고 LINQ를 사용하여 런타임에 테이블에서 데이터를 쿼리하는 것을 선호합니다. 테이블이 아무리 복잡해도 케이크 조각입니다.

Quicksheet 내에서 ENUM 유형 사용

데이터 클래스에 대한 열거형 유형을 지정하는 것은 쉽습니다. 다음과 같이 스프레드시트에서 'RareType'에 대한 열거형 유형을 설정하려고 한다고 가정해 보겠습니다.

	A	B	D	E	F
1	Id	Id	SkillType	RareType	Card
2	1	Smack2	Active	Normal	2
3	2	Strike2	Passive	Rare	2
4	3	Smack3	Active	Legend	3
5	4	Strike3	Passive	Normal	3

'RareType'은 Normal, Rare 및 Legend의 세 가지 유형 중 하나의 값만 가질 수 있습니다.

QuickSheet는 열거형 자체를 생성할 수 없으므로 스크립트 파일을 생성하기 전에 먼저 열거형 유형을 선언해야 합니다.

일반적으로 다양한 열거형을 포함하는 빈 .cs 파일을 만든 다음 스프레드시트에서 설정한 'RareType' 열거형 유형을 선언합니다.

```
공개 열거형 희귀 유형 {
```

```
    정상,  
    희귀한,  
    전설,
```

```
}
```

이제 오류 없이 필요한 스크립트 파일을 생성할 수 있습니다!

QuickSheet에서 배열 유형 사용

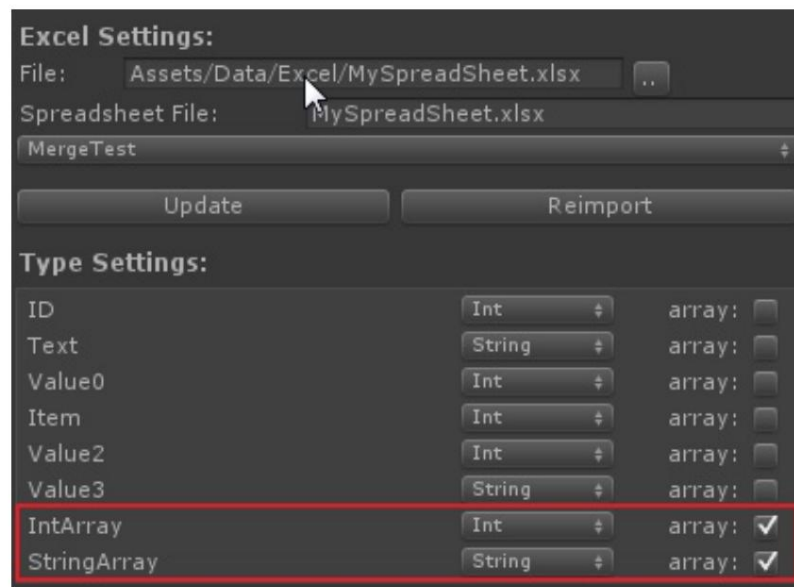
다음과 같이 셀에서 심표로 구분된 값과 함께 배열 유형을 사용할 수 있습니다.

	A	B	C	D	E	F	G	H
1	ID	Text	Value0	Item	Value2	Value3	IntArray	StringArray
2	1	Armor03	0	23	10	a	1,2,3,4,5,	a,b,c,d,e,
3	2	Sword01	1	55	20	b	1,2,3,4,5,	a,b,c,d,e,
4	3	Sword02	2	43	30	c	1,2,3,4,5,	a,b,c,d,e,
5	4	Shield	3	53	40	d	1,2,3,4,5,	a,b,c,d,e,
6	5	Hammer	4	67	50	e	1,2,3,4,5,	a,b,c,d,e,
7								

셀의 마지막 값 뒤에 있어야 하는 마지막 심표를 놓치지 마십시오. (v.1.0.0.0에서 변경됨)

1,2,3,4,5, -> v.1.0.0.0 이후에는 '5' 바로 뒤에 오는 심표가 더 이상 필요하지 않습니다.

주어진 엑셀 파일로 import한 후, 각 column-header의 타입을 지정 하고 배열 타입에 대한 배열 옵션을 확인한다.



다음과 같이 지정된 유형을 가진 데이터 클래스의 배열 유형 멤버 필드를 생성합니다.

암호:

```
[SerializeField] int[]
intarray = 새로운 int[0];

public int[] Intarray { get {return intarray; } 세트 { 내부 배열 = 값; } }

[필드 직렬화]
문자열[] 문자열 배열 = 새 문자열[0];

공개 문자열[] 문자열 배열 { get {반환 문자열 배열; } 세트 { 문자열 배열 = 값; } }
```

참고: 열거형 배열은 아직 Excel이 아닌 Google 스프레드시트에서만 지원됩니다.

수식 셀 유형

스프레드시트에서 셀 유형을 공식화할 수도 있습니다.

C2

:

✕

✓

fx

=AbitiliesBase[@cooldown]/Metadata[Pacing]

	A	B	C	D
1	AbilityType	castTime	cooldown	power
2	Heal	0.666666667	6.666666667	
3	Poke	0	2.666666667	15
4	Barr	1.333333333	4	0
5	Nuke	6.666666667	40	6
6	Stun	0	6.666666667	15
7	Swap	0	6.666666667	0
8	Copy	0	0	0
9	Taunt	0	13.33333333	0
10	Buff	0	6.666666667	2

스프레드시트에서 하는 일반적인 방법으로 수식을 사용하여 셀의 값을 계산합니다. 또한 Unity Quicksheet는 숫자 또는 문자열과 같은 다른 셀 유형과 차이 없이 이를 직렬화합니다.

참고: 현재 Excel에서만 사용할 수 있습니다. (v.1.0.0.1)

공식 계산 자동화

RPG의 캐릭터는 일반적으로 HP, MP, 레벨 등과 같은 다양한 스탯을 가지며 일반적으로 다양한 주어진 공식을 통해 계산됩니다. 예를 들어 캐릭터의 위력은 레벨이 오를수록 증가하며 이를 공식의 형태로 나타낼 수 있다.

그러나 공식을 다루는 방법에 문제가 있습니다. 코드 내에서 수식을 작성할 수 있습니다. 하지만 수식이 바뀔 때마다 다시 작성해야 하는 문제가 있다. 게임 디자이너가 전체 개발 기간 동안 계속해서 변경하는 것은 놀라운 일이 아닙니다.

따라서 코드 조각이 아닌 데이터 형식으로 수식을 최대한 유연하게 처리할 수 있는 방법이 필요합니다.

문서의 이 섹션에서는 완전한 데이터 기반 방식으로 스프레드시트와 Unity-Quicksheet 플러그인을 사용하여 수식 계산을 쉽게 수행할 수 있는 방법을 설명합니다.

더 읽기 전에 RPG의 일반적인 통계 및 공식에 익숙하지 않은 경우 [게임 시스템의 기술: 실제 사례](#) 기사를 참조하는 것이 좋습니다. 첫 번째.

스프레드시트의 수식

가장 먼저 할 일은 스프레드시트에 다양한 통계와 공식을 정의하여 게임에서 사용할 수 있도록 만드는 것입니다.

Excel 또는 Google 스프레드시트를 원하는 대로 사용할 수 있지만 이 문서의 설명은 Excel 스프레드시트를 기반으로 합니다.

당신이 만들려고 하는 게임에서 캐릭터가 레벨이 올라감에 따라 파워가 증가한다고 가정해 봅시다. 그는 자신의 힘을 설명하는 세 가지 기본 통계를 가지고 있습니다. 각각 힘(STR), 손재주(DEX), 지능(ITL)입니다. 스탯이 올라갈수록 더 많은 체력(HP)과 마나(MP)를 얻습니다.

다음 이미지는 스프레드시트의 각 통계 및 해당 공식을 보여줍니다.

	A	B
1	Stat	Formula
2	STR	$\text{Round}((5 * (\text{SkillLevel} * 0.6)), 0) + 10$
3	DEX	$\text{Round}((5 * (\text{SkillLevel} * 0.2)), 0) + 10$
4	ITL	$\text{Round}((5 * (\text{SkillLevel} * 0.2)), 0) + 10$
5	HP	$(5 * ((\text{STR} * 0.5) + (\text{DEX} * 0.39) + (\text{ITL} * 0.23)))$
6	MP	$(5 * ((\text{STR} * 0.01) + (\text{DEX} * 0.12) + (\text{ITL} * 0.87)))$

스프레드시트의 데이터는 이 문서의 예에 사용됩니다.

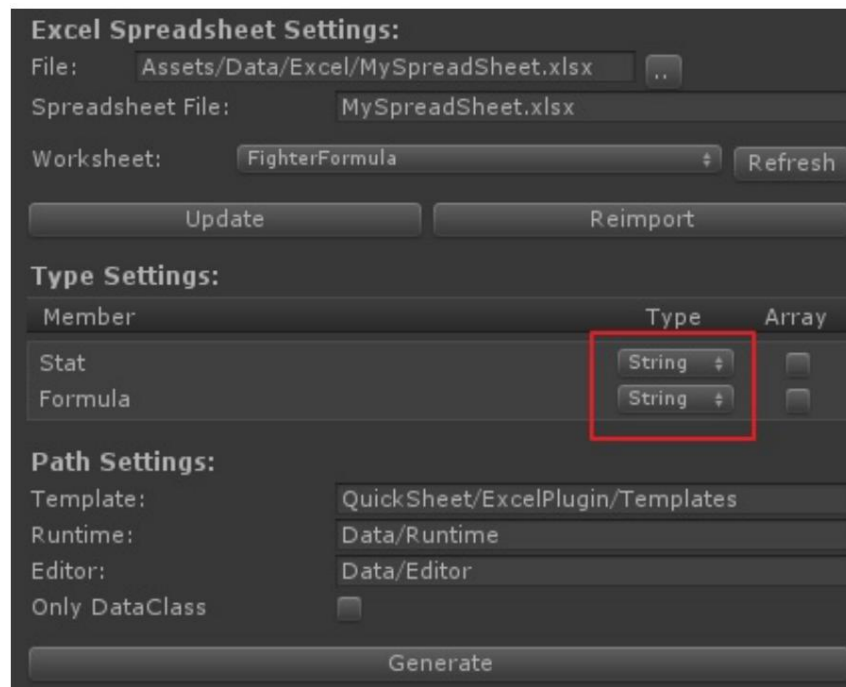
위 이미지의 Stat 과 Formula 는 [craft of game systems: Practical examples](#) 기사에서 차용한 것입니다.

데이터 가져오기

먼저 워크시트의 모든 데이터를 Unity 편집기로 가져옵니다.

가져오기 설정 자산 파일을 생성하고 스프레드시트를 지정하여 파일을 가져옵니다. (자세한 단계는 [Excel로 작업하는 방법](#) 페이지를 참조하십시오.)

'유형 설정'에서 Stat 및 Formula 유형을 모두 문자열 유형으로 설정합니다. Stat은 플레이어에 속한 각 stat의 이름일 뿐이므로 유형을 문자열로 설정하면 괜찮은 것 같지만 Formula도 문자열로 설정되는 이유는 무엇입니까? 답은 나중에 찾을 수 있습니다. 지금은 다시 생각해보면 문자열 외에 다른 적절한 형식이 없다는 것을 알게 될 것입니다.

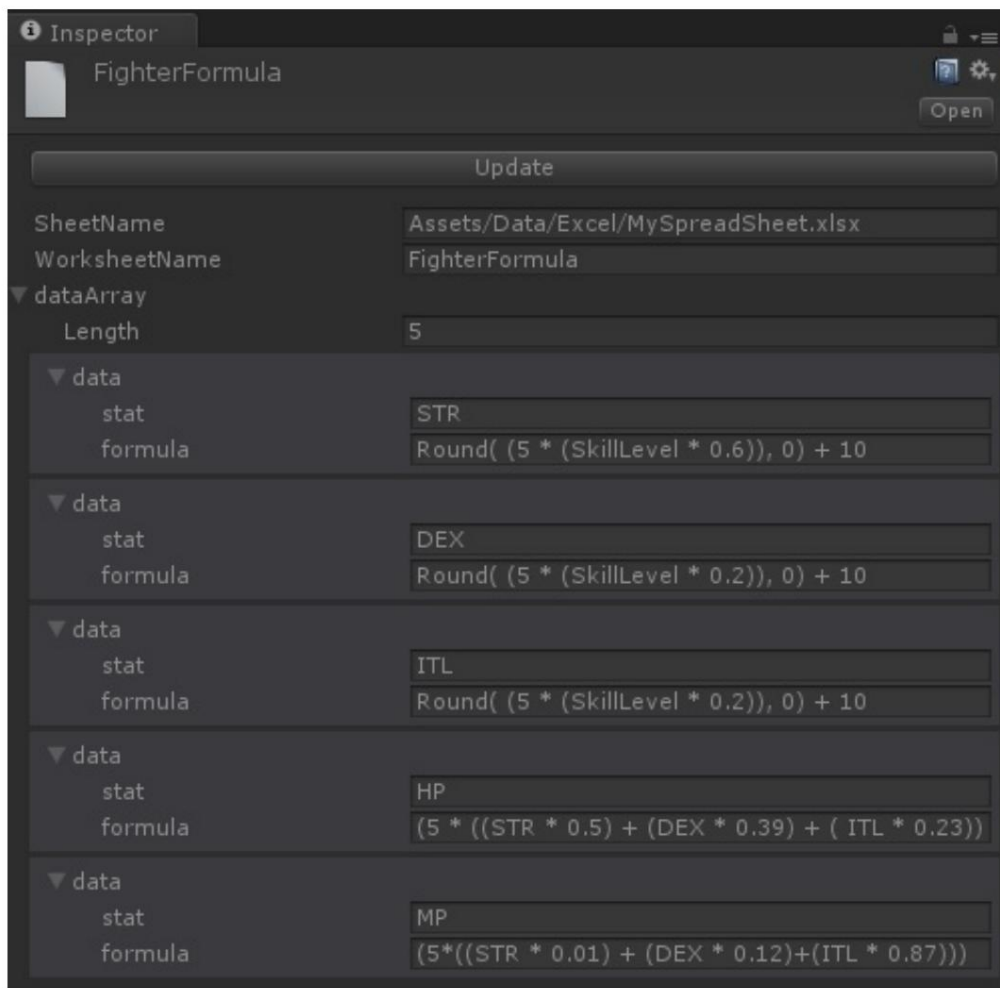


지정된 디렉토리 아래에 필요한 모든 스크립트 파일을 생성하는 생성 버튼을 클릭 합니다.

'에디터' 및 '런타임' 텍스트 필드에서.

그런 다음 프로젝트 창에서 .xlsx Excel 파일을 선택 하고 마우스 오른쪽 버튼을 클릭하여 표시되는 다시 가져오기 메뉴 를 선택합니다. 선택한 엑셀 파일

엑셀 파일을 다시 불러 오면서 'ScriptableObject' 를 워크시트 이름의 자산 파일로 생성하여 모든 데이터를 불러옵니다. 생성된 자산 파일을 선택하고 개선된 데이터가 올바른지 확인하십시오. Unity 편집기의 Inspector 보기에서 다음 이미지를 볼 수 있습니다.



생성된 코드, 특히 런타임 설정 디렉토리에서 찾을 수 있는 FighterFormuladata 클래스를 볼 수 있습니다 .

```
[System.Serializable] 공개 클래스
래스 FighterFormulaData {

    [SerializeField] 문자열
    통계; 공개 문자열 통계
    { get {return stat;} 세트 { 통계 = 값;}}

    [SerializeField] 문자열
    수식; 공개 문자열 수식
    { get {수식 반환 ;} 세트 { 공식 = 값;}}
}
```

스프레드시트에서 데이터를 가져오는 데 익숙하지 않은 경우 스프레드 시트 유형에 따라 [Excel 작업 방법](#) 또는 [Google 스프레드시트 작업 방법](#) 페이지를 참조하세요.

계산 코드 작성

이제 스프레드시트에서 가져온 주어진 공식으로 각 통계를 계산하는 코드를 작성해 보겠습니다.

우선, 게임에서 파이터 유형의 플레이어의 모든 통계를 나타내는 클래스가 필요합니다. 다음과 같은 간단한 POCO 클래스인 PlayerStat 을 사용할 수 있습니다. SkillLevel 속성 을 제외하고 스프레드시트에 속성으로 표시되는 모든 통계가 있습니다.

```
공개 클래스 PlayerStat
{
    공개 int SkillLevel { 가져오기; 세트; } 공개 float STR
    { get; 세트; } 공개 float DEX { get; 세트; }

    공개 부동 ITL { 가져오기; 세트; } 공개 부동 HP
    { get; 세트; } 공개 부동 MP { get; 세트; }

}
```

다음으로 해야 할 일은 각각의 통계를 해당 공식으로 계산하고 설정하는 것입니다. 결과는 각각의 통계로 돌아갑니다.

일반적으로 RPG의 공식은 다항식이며 가져온 공식은 문자열 데이터입니다. 따라서 다항식을 문자열 데이터 형식으로 계산할 수 있는 계산기가 필요합니다.

그렇다면 문자열 데이터 형식으로 수식을 계산하는 방법은 무엇입니까?

간단한 계산기를 작성하여 수행할 수 있지만 운 좋게도 C#으로 작성된 계산기가 이미 많이 존재합니다. 물론 바퀴를 재발명할 필요는 없습니다.

그 중 주목할만한 것 중 하나는 [.NET용 계산 엔진입니다](#). 사용하기 쉽고 이미 공식 계산을 위한 대부분의 기능이 있습니다.

또한 [Zirpl CalcEngine](#)이 있습니다. [.NET용 계산 엔진](#) 포스트 이 [github 프로젝트 페이지](#) 에서 찾을 수 있는 이식 가능한 .NET 라이브러리(PCL)로 이동 합니다.

CalcEngine 이 주어진 수식을 문자열 유형으로 사용하여 계산 하는 방법을 살펴보겠습니다 .

```
CalcEngine.CalcEngine 계산기 = new CalcEngine.CalcEngine(); 계산기.변수["a"] = 1;

// 예상대로 2를 출력합니다. var result
=calculator.Evaluate("a + 1");
```

위에 표시된 것처럼 평가하기 전에 수식의 변수를 지정하고 Evaluate 를 호출 하면 결과를 얻을 수 있습니다. 충분히 간단합니까?

'Player.cs' 가 Unity 게임 개체의 구성 요소에 대한 MonoBehaviour 클래스를 파생 하므로 스크립트 파일을 만듭니다 .

공개 클래스 플레이어 : MonoBehaviour {

// ScriptableObject는 스프레드시트에서 가져온 데이터를 포함합니다. public FighterFormula 전투기
공식;

// 지속성 데이터를 위한 클래스 인스턴스 private PlayerStatus
playerStatus = new PlayerStatus();
...

이제 해당 공식으로 각 통계를 계산할 차례입니다.

CalcEngine의 편리하고 강력한 기능 중 하나는 .NET 개체를 CalcEngine의 평가 컨텍스트에 연결하는 바인딩 메커니즘인 'DataContext'입니다.

CalcEngine에 각 변수를 지정하는 대신 'PlayerStatus' 클래스 인스턴스를 CalcEngine의 DataContext에 설정하는 것만으로 계산을 수행할 준비가 된 것입니다.

앞서 설명한 것처럼 CalcEngine의 'Evaluate' 함수를 호출하면 주어진 공식을 계산하고 해당 통계 값을 반환합니다. 공식이 무엇이든 계산하고 결과를 얻는 데 차이가 없습니다.

```

무효 시작 () {

    // 스킬 레벨 지정
    playerStatus.SkillLevel = 4;

    CalcEngine.CalcEngine 계산기 = new CalcEngine.CalcEngine();

    // CalcEngine은 Reflection을 사용하여 PlayderData 개체의 속성에 액세스합니다. // 따라서 표현식에서 사용할 수 있습니다. 계산
    기.DataContext = 플레이어 상태;

    // 플레이어의 각 통계를 계산합니다. playerStatus.STR =
    Convert.ToSingle(calculator.Evaluate(GetFormula("STR"))); Debug.LogFormat("STR: {0}", playerStatus.STR);

    playerStatus.DEX = Convert.ToSingle(calculator.Evaluate(GetFormula("DEX"))); Debug.LogFormat("DEX:
    {0}", playerStatus.DEX);

    playerStatus.ITL = Convert.ToSingle(calculator.Evaluate(GetFormula("ITL"))); Debug.LogFormat("ITL: {0}",
    playerStatus.ITL);

    playerStatus.HP = Convert.ToSingle(calculator.Evaluate(GetFormula("HP"))); Debug.LogFormat("HP:
    {0}", playerStatus.HP);

    playerStatus.MP = Convert.ToSingle(calculator.Evaluate(GetFormula("MP"))); Debug.LogFormat("MP:
    {0}", playerStatus.MP);
}

// 주어진 수식 이름으로 수식 데이터를 검색하는 도우미 함수입니다. 문자열 GetFormula(문자열 수식 이름) {

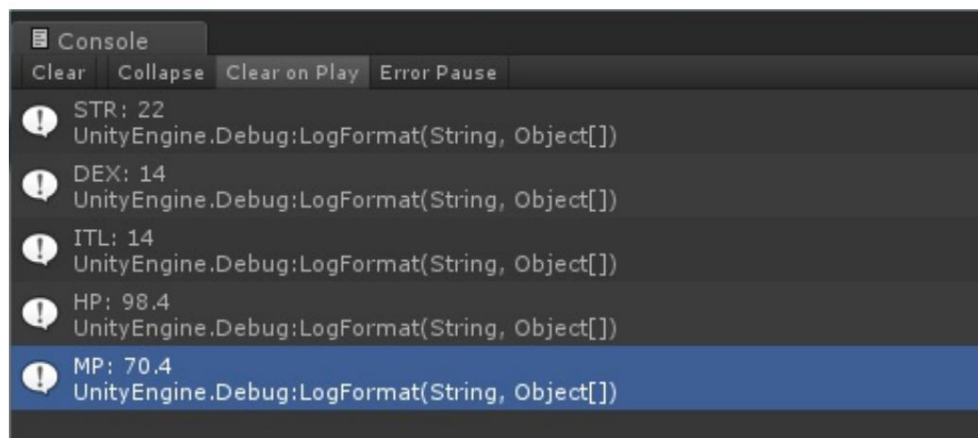
    반환 전투기Formula.dataArray.Where(e => e.Stat == FormulaName)
                                .FirstOrDefault().공식;

}
}

```

그게 다야. 스프레드시트에서 수식을 변경하더라도 코드를 변경할 필요가 없습니다. 가져온 scriptableObject.asset 파일을 업데이트하기만 하면 코드 쪽 변경 없이 런타임에 마지막 작업이 완료됩니다.

이제 Unity 에디터를 실행하면 콘솔에서 다음과 같은 결과를 얻을 수 있습니다.



위의 계산된 통계 결과를 [Dungeon Siege 2 시스템 스프레드시트](#)에 있는 스프레드시트의 원본과 비교하십시오. 페이지. 보시다시피 결과는 같습니다.

	A	B	C	D	E	F
1	FIGHTER STATISTICS					
2	Melee Skill	0	1	2	3	4
3	STR	10	13	16	19	22
4	DEX	10	11	12	13	14
5	INT	10	11	12	13	14
6	Total Damage	8.0	10.0	12.0	14.0	16.0
7	Total Armor	16	20	24	28	32
8	Health	56.0	66.6	77.2	87.8	98.4
9	Mana	50.0	55.1	60.2	65.3	70.4

개발 주기에서 게임 플레이 밸런스를 조정하면서 디자이너는 여러 번 공식을 변경하려고 합니다. 자연스러운 과정이지만 수식이 변경될 때마다 코드를 변경하고 싶지는 않습니다.

문서에서 볼 수 있듯이 이 접근 방식은 디자이너가 수식을 변경할 때마다 코드를 변경하지 않아도 되므로 시간을 크게 절약할 수 있을 뿐만 아니라 오류를 방지할 수 있습니다.

공식 계산 자동화: 파트 II

이전 문서에서는 각각의 통계가 별도로 계산됩니다. 지루한 작업입니다. 더 나쁜 것은 PlayerStatus 클래스에 새로운 통계가 추가될 때마다 계산하고 다시 설정해야 한다는 것입니다.

```
playerStatus.SkillLevel = 4;

계산기 = 새로운 CalcEngine.CalcEngine(); 계산기.DataContext
= 플레이어 상태;

playerStatus.STR = Convert.ToSingle(calculator.Evaluate(GetFormula("STR"))); playerStatus.DEX =
Convert.ToSingle(calculator.Evaluate(GetFormula("DEX"))); playerStatus.ITL =
Convert.ToSingle(calculator.Evaluate(GetFormula("ITL"))); playerStatus.HP =
Convert.ToSingle(calculator.Evaluate(GetFormula("HP"))); playerStatus.MP =
Convert.ToSingle(calculator.Evaluate(GetFormula("MP")));
```

예를 들어 캐릭터의 행운 통계에 사용되는 새로운 통계 'LUK' 를 추가 하면 다음 코드와 같이 계산 코드를 다시 작성해야 합니다.

```
playerStatus.MP = Convert.ToSingle(calculator.Evaluate(GetFormula("LUK")));
```

별거 아닐 수도 있지만 지루합니다.

그렇다면 다음 코드와 같이 모든 계산을 한 번에 수행할 수 있다면 어떻게 될까요?

FormulaEnhanced.cs

```
playerStatus = 새로운 PlayerStatus();
playerStatus.SkillLevel = 4;

// 봄!
FormulaCalculator = 새로운 FormulaCalculator();
FormulaCalculator.Calculate<PlayerStatus>(playerStatus, FighterFormula.dataArray);
```

매력 없어 보이나요? 단순할 뿐만 아니라 유연합니다. 자, 어떻게 가능한지 봅시다.

가장 먼저 할 일은 공식을 추출하고 해당 통계를 해결하는 것입니다. FormulaCalculator.GetFormulaTable 메서드 를 호출하여 처리할 수 있습니다 . 스프레드시트에서 가져와 첫 번째 매개변수에 대한 ScriptableObject 파생 클래스 인 'FighterFormula' 클래스에 있는 지정된 'FighterFormulaData' 배열을 사용하여 사전을 생성한다고 생각하십시오 . 두 번째 매개변수는 'stat' 이름 이고 마지막 매개변수는 문자열 형식인 'formula' 자체입니다.

FormulaCalculator.cs


```
public void Calculate<T>(System.Object 데이터, System.Object[] obj, 문자열 키, 문자열 값) {

    // obj: FighterFormulaData 배열 // 키:
    FighterFormulaData 클래스의 Stat 속성 // 값: FighterFormulaData 클래스의
    Formula 속성
    사전<문자열, 문자열> 공식표 = GetFormulaTable(obj, 키, 값);
    계산<T>(데이터, 공식표);
}
```

그런데 수식에 필요한 속성만 계산해야 하는 문제가 있습니다. 'PlayerStatus' 클래스를 참조하세요. 모든 properties가 공식 계산을 위한 변수로 사용되는 것은 아닙니다. SkillLevel 속성은 그 목적을 위해 필요하지 않습니다. 따라서 변수에 사용되는 속성과 사용하지 않는 속성을 구분하는 방법이 필요합니다. 하지만 어떻게?

수식 변수를 사용하는 모든 속성에 사용자 정의 속성을 설정하여 해결할 수 있습니다. 그만큼

이를 위해 다음 코드에서 보여지는 FormulaVariable 속성이 사용됩니다.

PlayerStatus.cs

```
공개 클래스 PlayerStatus {

    공개 int SkillLevel { 가져오기; 세트; }

    // 'FormulaVariable'이 있는 stat 멤버 필드는 자동으로 인식됩니다.
    // 공식 변수로.
    [FormulaVariable] public
    float STR { get; 세트; }
    [공식변수]
    공개 부동 소수점 DEX { 가져오기; 세트; }
    [FormulaVariable] public
    float ITL { get; 세트; }
    [공식변수]
    공개 플로트 HP { get; 세트; }
    [FormulaVariable] public
    float MP { get; 세트; }
}
```

'FormulaVariableAttribute'는 C#의 단순 속성입니다.

FormulaVariableAttribute.cs

```
시스템 사용 ;

[AttributeUsage(AttributeTargets.Property)] 공개 클래스
FormulaVariableAttribute : 속성 {

}
```

주어진 수식을 계산하기 전에 GetFormulaProperties 메서드는 'PlayerStatus' 클래스에서 'FormulaVariable' 속성이 있는 모든 속성을 수집합니다. 반사 메커니즘을 사용하면 쉽게 할 수 있습니다.

FormulaCalculator.cs

```

    개인 PropertyInfo[] GetFormulaProperties<T>()
    {
        var _유형 = typeof(T);

        List<PropertyInfo> FormulaPropList = new List<PropertyInfo>();

        // 반사. 주어진 클래스 T의 모든 속성을 가져옵니다.
        PropertyInfo[] 속성 = _type.GetProperties(BindingFlags.Public | BindingFlags.Instance)
;

        foreach (속성의 PropertyInfo p) {

            if (!(p.CanRead && p.CanWrite))
                계속하다;

            개체[] 속성 = p.GetCustomAttributes(true); foreach (속성 의 개체 o) {

                // 'FormulaVariable' 사용자 정의 속성이 있는 속성을 얻습니다. if (o.GetType() ==
                typeof(FormulaVariableAttribute))
                    FormulaPropList.Add(p);
            }
        }
        반환 FormulaPropList.ToArray();
    }

```

이제 실제 계산을 할 차례입니다. 제네릭 메서드 'Calculate' 를 호출하여 수행하므로 'PlayerStatus' 클래스 뿐만 아니라 모든 유형의 클래스에서 계산을 수행할 수 있습니다 .

FormulaCalculator.cs

```

public void Calculate<T>(System.Object 데이터 인스턴스, 사전<문자열, 문자열> 수식표) {

    var _유형 = typeof(T);

    // CalcEngine에 변수 값을 알려줍니다. calcEngine.DataContext
    = 데이터 인스턴스;

    // 'FormulaVariable' 속성을 가지는 각각의 속성을 평가한다.
    PropertyInfo[] 속성 = GetFormulaProperties<T>(); foreach (속성의 PropertyInfo
    p)
    {
        문자열 수식 = null; if
        (formulaTable.TryGetValue(p.Name, out 공식)) {

            if (string.IsNullOrEmpty(수식)) {

                var 값 = calcEngine.Evaluate(공식); p.SetValue(dataInstance,
                Convert.ChangeType(값, p.PropertyType), null);

            }
        }
    }
}

```

계산해야 하는 속성의 선언 순서가 중요합니다.

'PlayerStatus' 클래스를 다시 참조 하십시오. HP 속성 에 대한 변수 유지를 평가하려면 먼저 STR, DEX 및 ITL의 올바른 값을 평가하고 설정해야 합니다.

```
HP = (5 ((STR 0.5) + (DEX 0.39) + (ITL 0.23)))
```

PlayerStatus.cs

```

공개 클래스 PlayerStatus {

    ...

    [공식변수]
    공개 부동 STR { 가져오기; 세트; }

    [공식변수] public float
    DEX { get; 세트; }

    [공식변수]
    공개 플로트 ITL { 가져오기; 세트; }

    [FormulaVariable]
    public float HP { get; 세트; }

    ...

}

```

그게 다야.

한 가지 명심해야 할 점은 공식 계산을 단순화하기 위해 리플렉션을 많이 사용했다는 것입니다. 이미 알고 있듯이 리플렉션은 런타임에도 빠른 방법이 아닙니다. 따라서 이 접근 방식으로 작업할 때 게임 루프의 속도가 중요한 위치에서 신중하게 수행해야 합니다. 예를 들어 MonoBehaviour의 업데이트에서 계산을 하지 마십시오.

다음은 FormulaCalculator 클래스 코드의 전체 라인을 보여줍니다.

FormulaCalculator.cs

```

공개 클래스 FormulaCalculator {

    CalcEngine.CalcEngine calcEngine = 새로운 CalcEngine.CalcEngine();

    공개 무효 계산<T>(System.Object 데이터, System.Object[] obj) {

        PropertyInfo[] 정보 = obj[0].GetType().GetProperties(BindingFlags.Public | BindingFlags.Ins
탄스);

        문자열 키 = infos[0].이름; 문자열 값 =
infos[1].이름;

        계산<T>(데이터, 개체, 키, 값);
    }

    public void Calculate<T>(System.Object 데이터, System.Object[] obj, 문자열 키, 문자열 값) {

        사전<문자열, 문자열> 공식표 = GetFormulaTable(obj, 키, 값);
        계산<T>(데이터, 수식표);
    }

    public void Calculate<T>(System.Object 데이터 인스턴스, 사전<문자열, 문자열> 수식표) {

        var _유형 = typeof(T);

        calcEngine.DataContext = 데이터 인스턴스;

        PropertyInfo[] 속성 = GetFormulaProperties<T>(); foreach (속성의 PropertyInfo
p) {

            문자열 수식 = null; if
(formulaTable.TryGetValue(p.Name, out 공식))
            {
                if (!string.IsNullOrEmpty(수식)) {

                    var 값 = calcEngine.Evaluate(공식); p.SetValue(dataInstance,
Convert.ChangeType(값, p.PropertyType), null);
                }
            }
        }
    }

    공개 사전<문자열, 문자열> GetFormulaTable (System.Object[] obj, 문자열 _key, 문자열 _값
(이름))
    {
        사전<문자열, 문자열> 결과 = 새 사전<문자열, 문자열>();

        foreach (obj 의 System.Object o)
        {
            PropertyInfo[] 정보 = o.GetType().GetProperties(BindingFlags.Public | BindingFlags.Inst
안스);

            // 사전의 키

```

```

문자열 statName = null; var 발견
= infos.Where(e => e.Name == _key).FirstOrDefault(); if (발견 != null) {

    개체 값 = found.GetValue(o, null); 통계 이름 = 값 .ToString();

}

// 사전의 값
문자열 수식 = null; 찾은 =
infos.Where(e => e.Name == _value).FirstOrDefault(); if (발견 != null) {

    개체 값 = found.GetValue(o, null); 수식 = 값 .ToString();

}

if (string.IsNullOrEmpty(statName) == false && string.IsNullOrEmpty(formula) == false)
    result.Add(statName, 수식);
또 다른
{
    // 오류
}
}

반환 결과;
}

개인 PropertyInfo[] GetFormulaProperties<T>() {

    var _유형 = typeof(T);

    List<PropertyInfo> FormulaPropList = new List<PropertyInfo>();

    PropertyInfo[] 속성 = _type.GetProperties(BindingFlags.Public | BindingFlags.Instance)
;

    foreach (속성의 PropertyInfo p) {

        if (!(p.CanRead && p.CanWrite))
            계속하다;

        개체[] 속성 = p.GetCustomAttributes(true); foreach (속성 의 객체 o)

        {
            if (o.GetType() == typeof(FormulaVariableAttribute)) FormulaPropList.Add(p);

        }

    }

    반환 FormulaPropList.ToArray();
}
}

```

공식 계산 자동화: 파트 III

(작업중...)

팁 및 알려진 문제

뛰어나다

하나의 엑셀 파일에 적은 수의 시트를 보관하십시오. 하나의 Excel 파일에 20개 이상의 시트가 포함된 Excel 파일이 있고 각 시트에 대해 모든 ScriptableObject 자산 파일을 생성한 경우를 생각해 보겠습니다. 그리고 같은 Excel 파일에 새 시트를 만든 다음 필요한 스크립트 파일을 생성해 봅니다. 무슨 일이야? 시트를 하나만 생성하고 새 시트로만 데이터를 가져오려고 하지만 Quicksheet는 모든 시트에서 모든 데이터를 다시 가져오려고 합니다. 왜냐하면 Excel에서 데이터를 쿼리 하여 Unity의 다시 가져오기를 통해 새로 생성된 ScriptableObject로 쿼리하기 때문입니다. 따라서 작동한다는 점을 명심하십시오.

시트가 너무 많은 Excel 파일을 사용하면 속도가 느려질 수 있습니다.

Google 드라이브에 액세스하기 위한 OAuth2 설정

Google은 2015년 5월 5일부터 인증 체계를 변경했습니다. 이제 OAuth2가 필요합니다. 이를 설정하려면 [Google 개발자 콘솔](#) 을 방문하세요. 새 프로젝트를 만들고 Drive API를 활성화하고 "서비스 계정" 유형의 새 클라이언트 ID를 만들고 json 파일을 다운로드합니다. [Google 스프레드시트 Howto](#) 에서 OAuth2 Google 서비스 계정 섹션을 참조하세요. 자세한 내용은 페이지를 참조하십시오.

자격 증명 설정 및 OAuth2 'client_ID' 및 'client_secret' 가져오기 [페이지](#) 를 참조하세요 .

하위 트리를 통해 QuickSheet 추가

다음과 같이 github 프로젝트에 하위 트리를 통해 QuickSheet를 추가할 수 있습니다.

```
자식 하위 트리 추가 --prefix=Assets/QuickSheet https://github.com/your_github_account/your_project.git Q
uick 시트
```

Assets unity 프로젝트 폴더 아래 에 QuickSheet 폴더를 만든 다음 필요한 모든 파일을 QuickSheet 폴더 아래에 넣습니다.

원격 저장소에 대한 모든 변경 사항 은 다음과 같이 git subtree pull 을 사용하여 쉽게 가져올 수 있습니다.

```
git subtree pull --prefix=Assets/QuickSheet https://github.com/kimsama/Unity-QuickSheet.git QuickShe
외
```

자주하는 질문

Q: 런타임에 Google 스프레드시트에서 데이터를 가져올 수 있나요?

A: 아니요. Unity-Quicksheet 는 편집 시 데이터를 쉽게 처리할 수 있는 도구입니다. 기술적인 관점에서 Excel이나 Google 스프레드시트에서 가져온 모든 데이터는 **ScriptableObject** 로 자산 파일 형식으로 저장됩니다. 직렬화 및 역직렬화를 쉽게 하기 위해 파생 클래스 객체를 사용합니다.

Unity의 **ScriptableObject** 지속성 데이터용이 아니므로 런타임 변경 데이터 를 직렬화할 방법이 없습니다. Unity-Quicksheet 는 이러한 목적으로 개발되지 않았습니다. 따라서 보유한 데이터가 런타임에 변경될 수 있고 지속성이어야 하는 경우 해당 목적을 위한 json과 같은 다른 것을 고려해야 합니다.

JSON, BSON 또는 XML 등과 같은 양방향 직렬화를 위한 형식이 많이 있습니다. 결정은 전적으로 플랫폼 및/또는 요구 사항에 따라 다릅니다.

반면에 대부분의 경우 런타임에 Excel에서 데이터를 가져올 이유가 없습니다. Excel은 아니지만 런타임에 Google 드라이브 에서 데이터를 연결하고 가져오는 것이 유용할 수 있지만 Excel과 동일한 인터페이스를 유지하기 위해 Unity-Quicksheet 는 Google 스프레드시트에서도 이를 제공하지 않습니다.

(미래에 변경될 수 있지만)

Q: 모든 플랫폼을 지원하니까?

A: 네, Unity의 빌드 대상 플랫폼으로 지정된 모든 플랫폼에서 실행할 수 있습니다. 단, Web Player 빌드 대상 설정의 Google 스프레드시트 제한이 있습니다.

Unity 측의 보안 제한으로 인해 Unity 웹 플레이어의 보안 샌드박스에서 Google 스프레드시트 플러그인이 작동하지 않습니다. 따라서 플랫폼 설정을 '독립형'으로 변경 하거나 '빌드 설정' 에서 'iOS' 또는 'Android' 플랫폼 과 같은 다른 것으로 변경해야 작동합니다.

Q: Unity 에디터에서 변경된 데이터를 Excel 또는 Google 스프레드시트에 다시 저장할 수 있습니까?

A: 아니요, 할 수 없습니다. Excel 또는 Google 드라이브에 변경 사항을 유지하고 데이터 흐름에 대한 한 가지 방법이 오류 방지에 더 좋습니다.

Q: Mac 시스템의 OSX에서 Excel 파일에서 데이터를 가져올 수 있습니까?

A: 예, 문제 없이 Mac 컴퓨터에서 실행됩니다. Excel 파일을 Mac 컴퓨터에서 '.xlsx' 파일이 아닌 '.xls'로 저장하십시오. 그렇지 않으면 '콘텐츠 유형 부분을 읽을 수 없습니다!'라는 오류가 발생할 수 있습니다.

Q: 엑셀 대신 '오픈오피스'를 사용할 수 있나요?

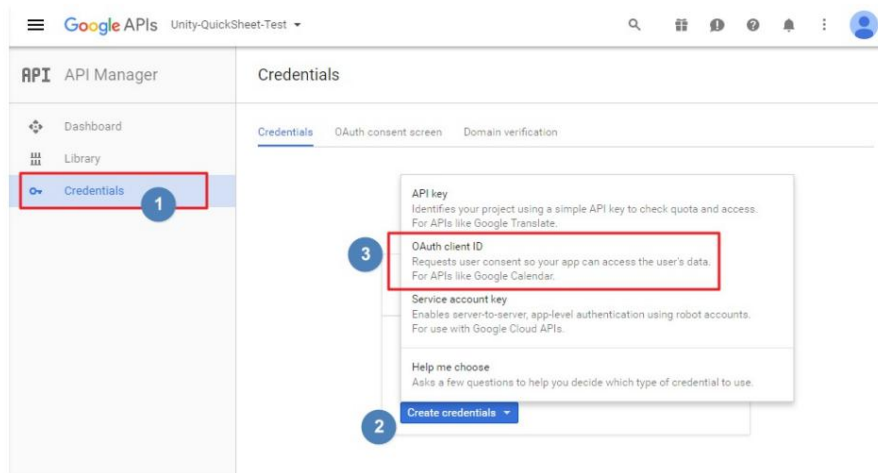
A: 예, 둘 다 '.xls' 파일에 대해 동일한 결과를 얻습니다.

Google 드라이브에 액세스하기 위한 OAuth2 설정

Google API의 OAuth2 설정

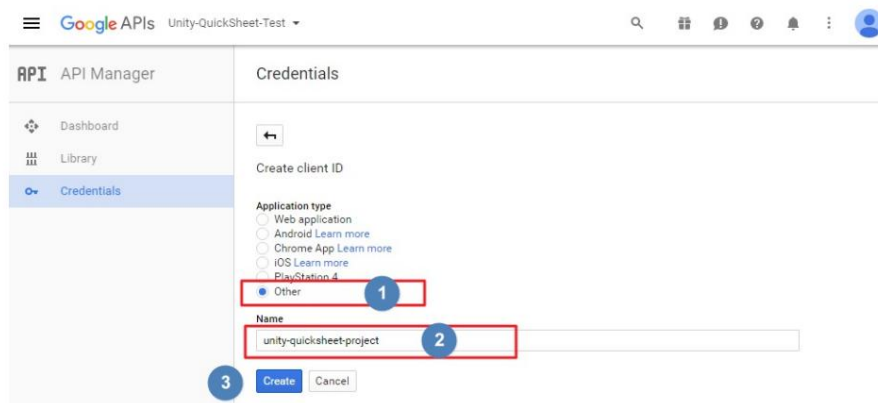
Google 드라이브, 특히 Unity-Quicksheet용 Google 스프레드시트에 액세스하려면 OAuth2 인증이 필요합니다. 이를 설정하려면 [Google API](#) 를 방문하세요. 페이지를 열고 새 프로젝트를 만듭니다.

다음은 oauth2 자격 증명에 필요한 'client_ID' 및 'client_secret' 토큰을 얻는 방법을 단계별로 설명합니다.

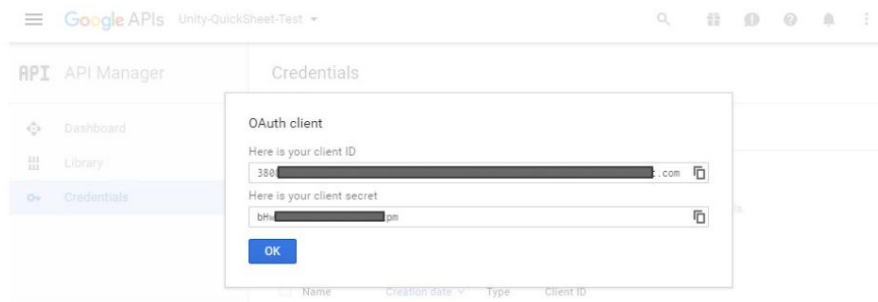


왼쪽 패널에서 1) 자격 증명 을 선택 하고 상단 메뉴에서 자격 증명 탭을 선택한 다음 2) 자격 증명 만들기 버튼을 클릭하면 생성할 수 있는 자격 증명 유형이 표시된 대화 상자가 열립니다.

3) OAuth 클라이언트 ID 를 선택해야 합니다 . 그렇지 않으면 json 파일 형식이 잘못될 수 있습니다.

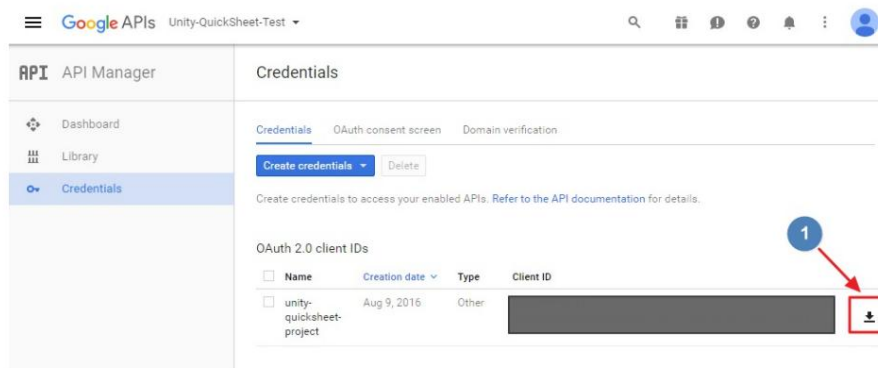


1) 신청 유형 에서 기타 를 선택 하고 2) 이름 필드 에 적절한 이름을 입력 한 다음 3) 을 누릅니다. '만들기' 버튼을 누르면 자격 증명이 생성됩니다.



이제 새로 생성된 자격 증명에 대한 'client_ID' 및 'client_secret' 을 볼 수 있습니다 .

이때 해당 'client_ID' 와 'client_secret' 을 Unity Quicksheet의 Google 설정에 직접 복사하여 붙여넣거나 Unity-Quicksheet 설정에 필요한 'client_ID' 와 'client_secret' 이 포함된 oauth2 json 파일을 다운로드할 수 있습니다 .



마지막으로 1) 'client_ID' 와 'client_secret' 이 포함된 json 파일을 다운로드하기 위한 다운로드 아이콘을 클릭합니다.

Unity-Quicksheet의 OAuth2 설정

Google 측에서 인증을 위한 작업이 완료되었으며 Unity 측에서 Google 스프레드시트에 액세스하기 위한 나머지를 설정할 때입니다.

[Google 스프레드시트 방법 페이지](#) 의 OAuth2 Google 서비스 계정 섹션을 참조하세요. 자세한 사항은.

코드 템플릿

Unity-Quicksheet는 다양한 템플릿 파일을 사용 하여 스프레드시트에서 데이터를 가져오는 데 필요한 '.cs' 스크립트 파일을 자동으로 생성합니다.

모든 템플릿 파일은 간단한 ASCII '.txt' 파일입니다. 따라서 다른 형태의 스크립트를 생성하려는 경우 쉽게 변경할 수 있습니다.

생성된 스크립트는 목적에 따라 두 가지로 나뉩니다. 하나는 런타임용이고 다른 하나는 편집기 스크립트용입니다.

실행 시간	편집자
데이터 클래스.txt	PostProcessor.txt
속성.txt	ScriptableObjectEditorClass.txt
ScriptableObjectClass.txt	해당 없음

에디터 스크립트는 Unity 전용 폴더 인 'Editor' 라는 폴더 아래에 배치됩니다 .

사용자 정의 템플릿 사용

Unity-QuickSheet의 유연한 기능 중 하나는 템플릿 파일을 생성된 스크립트 파일로 원하는 대로 변경할 수 있다는 것입니다.

하지만 한 가지 명심해야 할 것이 있습니다.

접두어에 '\$' 가 있는 단어는 Unity-QuickSheet가 스크립트 파일을 생성할 때 대체됩니다.

따라서 '\$' 가 없는 단어를 변경 하면 생성된 스크립트 파일이 컴파일되지 않고 구문 오류가 발생하므로 주의해야 합니다.

Unity 편집기 스크립트에 익숙하다면 간단한 스프레드시트에서 데이터를 가져오고 생성된 스크립트 파일을 살펴보세요. 마지막은 자기 설 명입니다.

참조

데이터 클래스.txt

예어	설명
\$클래스이름	DataClass 클래스의 클래스 이름입니다. 스프레드시트의 이름에 해당합니다. (또는 Google 스프레드시트의 경우 '워크시트' 이름)
\$MemberFields	스프레드시트 의 '열 헤더' 이름에 해당합니다.

속성.txt

예어	설명
\$필드이름	'DataClass' 클래스의 멤버 필드 이름입니다.
\$CapitalFieldName	\$FieldName과 동일하지만 대문자입니다.

ScriptableObjectClass.txt

예어	설명
\$클래스이름	ScriptableObject 클래스를 파생시키는 '클래스 이름'
\$ 데이터 클래스 이름	'DataClass.txt' 템플릿 파일의 \$ClassName과 동일해야 합니다.

ScriptableObjectEditorClass.txt

예어	설명
\$WorkSheetClassName	스프레드시트의 이름에 해당합니다. (또는 '워크시트' 이름 구글 스프레드시트의 경우)
\$클래스이름	'ScriptableObjectClass' 클래스의 \$ClassName과 동일합니다.
\$ 데이터 클래스 이름	'DataClass' 클래스의 \$ClassName과 동일합니다.

PostProcessor.txt

예어	설명
\$AssetPostprocessorClass	
\$클래스 이름	'ScriptableObjectClass' 클래스의 \$ClassName과 동일합니다.
\$ 데이터 클래스 이름	'DataClass' 클래스의 \$ClassName과 동일합니다.

참고문헌

- [유니티 직렬화](#) 직렬화 메커니즘에 대한 자세한 내용은 Unity 포럼을 참조하세요.
- [GDataDB](#) Google 스프레드시트에서 데이터를 검색하는 데 사용됩니다. [GDataDB](#) _ enum 유형 을 지원하도록 약간 수정되었습니다.
- [속성 노출](#) Unity3D의 인스펙터 뷰에서 스프레드시트의 변수를 쉽게 노출하고 [GDataDB](#)가 get/set 접근자를 통한 접근 (v.1.0.0에서 삭제됨)
- [NPOI](#) xls 및 xlsx 파일을 읽는 데 사용됩니다.
- Google Data SDK의 모든 "*.dll" 파일은 Google [Data API SDK](#) 에서 사용할 수 있습니다. net
- 2.0용 Newtonsoft.Json 소스 코드는 [여기](#) 에서 사용할 수 있습니다.
- [Unity-Google 데이터](#), 스프레드시트 데이터를 Unity로 가져오려는 이전의 노력입니다.