## 枚举字符串互转

```
template<typename T>
static T Conv_StringToEnum(const TCHAR* InEnumName, const TCHAR* InValueString)
{
    UEnum* EnumPtr = FindObject<UEnum>(ANY_PACKAGE, InEnumName, true);
    if(!EnumPtr)
    {
        return INDEX_NONE;
    }

    return static_cast<T>(EnumPtr->GetValueByNameString(InValueString));
}

template<typename T>
static FString Conv_EnumToString(const TCHAR* InEnumName, T InEnum)
{
    UEnum* EnumPtr = FindObject<UEnum>(ANY_PACKAGE, InEnumName, true);
    ensure(EnumPtr);
    if(!EnumPtr)
    {
        return FString();
    }

    return EnumPtr->GetNameStringByValue(static_cast<int64>(InEnum));
}

#define CONV_ENUM_TO_STRING(EnumName, EnumValue) Conv_EnumToString<EnumName>
(TEXT(#EnumName), EnumValue)
#define CONV_STRING_TO_ENUM(EnumName, ValueString) Conv_StringToEnum<EnumName>
(TEXT(#EnumName), ValueString)
```

# Json处理

## 字符串与Json对象互转

```
void FPatcherEditorMisc::Conv_JsonToString(const TSharedRef<FJsonObject>
InJsonObject, FString& OutString)
{
    const TSharedRef<TJsonWriter<>> Writer =
TJsonWriterFactory<>::Create(&OutString);
    FJsonSerializer::Serialize(InJsonObject, Writer);
}

void FPatcherEditorMisc::Conv_StringToJson(const FString& InJsonString,
TSharedPtr<FJsonObject>& OutJson)
{
    const TSharedRef<TJsonReader<TCHAR>> Reader =
TJsonReaderFactory<TCHAR>::Create(InJsonString);
    FJsonSerializer::Deserialize(Reader, OutJson);
}
```

## 在Json对象中添加数组

```
TSharedPtr<FJsonObject> NewJson = MakeShareable(new FJsonObject);
TArray<TSharedPtr<FJsonValue>> ValueArray;
{
    TSharedPtr<FJsonObejct> ValueItemObj = MakeShareable(new FJsonObject);
    ValueArray.Add(MakeShareable(new FJsonVlaueObject(ValueItemObj)));
}
NewJson->SetArrayField(TEXT("NewKey"), ValueArray);
```

# 图片操作

## PNG转JPG

```
bool ConvertPNGtoJPG(const FString& InSourcePath, const FString& InTargetPath)
{
    check(InSourcePath.EndsWith(TEXT(".png")) &&
InTargetPath.EndsWith(TEXT(".jpg")));

    IImageWrapperModule& ImageWrapperModule =
FMoudleManager::LoadModuleChecked<IImageWrapperModule>(FName("ImageWrapper"));
    IImageWrapperPtr SourceImageWrapper =
ImageWrapperModule.CreateImageWrapper(EImageFormat::PNG);
    IImageWrapperPtr TargetImageWrapper =
ImageWrapperModule.CreateImageWrapper(EImageFormat::JPEG);
    TArray<uint8> SourceImageData;
    TArray<uint8> TargetImageData;
    int32 Width, Height;
    const TArray<uint8>* UncompressedRGBA = nullptr;
    if(!FPlatformFileManager::Get().GetPlatformFile().FileExists(*InSourcePath))
    {
        return false;
    }

    if(!FFileHelper::LoadToArray(SourceImageData, *InSourcePath))
    {
        return false;
    }

    if(SourceImageWrapper.IsValid() && SourceImageWrapper-
>SetCompressed(InSourceImageData.GetData(), SourceImageData.Num()))
    {
        if(SourceImageWrapper->GetRaw(ERGBFormat::RGBA, 8, UncompressedRGBA))
        {
            Height = SourceImageWrapper->GetHeight();
            Width = SourceImageWrapper->GetWidth();
            if(TargetImageWrapper->SetRaw(UncompressedRGBA->GetData(),
UnCompressedRGBA->Num(), Width, Height, ERGBFormat::RGBA, 8))
            {
                TargetImageData = TargetImageWrapper->GetCompressed();
                if(!FFileManagerGeneric::Get().DirectoryExists(*InTargetPath))
                {
```

```
 FFileManagerGeneric::Get().MakeDirectory(*FPaths::GetPath(InTargetPath), true);
                }
                return FFileHelper::SaveArrayToFile(TargetImageData,
*InTargetPath);
            }
        }
    }
    return false;
}
```

## PNG转UTexture2D

```
Texture* LoadTexture2DFromBytesAndExtension(const FString& InImagePath, uint8*
InCompressedData, int32 InCompressedSize, int32& OutWidth, int32& OutHeight);

UTexture2D& LoadTexure2DFromFilePath(const FString& InImagePath, int32&
OutWidth, int32& OutHeight)
{
    if(!FPlatformFielManager::Get().GetPlatformFile().FileExists(*InImagePath))
    {
        return nullptr;
    }

    TArray<uint8> CompressedData;
    if(!FFileHelper::LoadFileToArray<CompressedData, *InImagePath))
    {
        return nullptr;
    }

    return LoadTexture2DFromBytesAndExtension(InImagePath,
CompressedData.GetData(), CompressedData.Num(). OutWidth, OutHeight);
}

Texture* LoadTexture2DFromBytesAndExtension(const FString& InImagePath, uint8*
InCompressedData, int32 InCompressedSize, int32& OutWidth, int32& OutHeight)
{
    UTexture2D* Texture = nullptr;
    IImageWrapperModule& ImageWrapperModule =
FMoudleManager::LoadModuleChecked<IImageWrapperModule>(FName("ImageWrapper"));
    IImageWrapperPtr ImageWrapper =
ImageWrapperModule.CreateImageWrapper(EImageFormat::PNG);

    if(ImageWrapper.IsValid() && ImageWrapper->SetCompressed(InCompressedData,
InCompresedSize))
    {
        const TArray<uint8>* UnCompressedRGBA = nullptr;
        if(ImageWrapper->GetRaw(ERGBFormat::RGBA, 8, UncompressedRGBA))
        {
            Texture = UTexture2D::CreateTransient(ImageWrapper->GetWidth(),
ImageWrapper->GetHeight(), FP_R8G8B8A8);
            if(Texture != nullptr)
            {
```

```
                OutWidth = ImageWrapper->GetWidth();
                OutHeight = ImageWrapper->GetHeight();
                void* TextureData = Texture->PlatformData-
>Mips[0].BulkDData.Lock(LOCK_READ_WRITE);
                FMemory::Memcpy(TextureData, UcompressedRGBA->GetData(),
UncompressedRGBA->Num());
                Texture->PlatformData->Mips[0].BulkData.Unlock();
                Texture->UpdateResource();
            }
        }
    }
    return Texture;
}
```