Analytics > Log & Crash Search > Unity Android SDK 사용 가이드

> [Deprecated] Log & Crash Unity Android SDK 버전은 더 이상 지원되지 않습니다. TOAST SDK를 이용해 주시기 바랍니다.

Log & Crash Unity SDK는 Log & Crash Search 수집 서버에 로그를 보내는 기능을 제공합니다. Log & Crash Unity SDK 특·장점은 다음과 같습니다.

- 로그를 수집 서버로 보냅니다.
- 앱에서 발생한 크래시 로그를 수집 서버로 보냅니다.
- Log & Crash Search 에서 전송된 로그를 조회 및 검색이 가능합니다.

지원 환경

- 공통 - Unity3D v4.0 이상
- Android - Andorid SDK 2.3.3 API 이상

다운로드

TOAST Document에서 Unity SDK를 받을 수 있습니다.

```
[DOCUMENTS] > [Download] > [Analytics > Log & Crash Search] > [Unity SDK]
```

설치

- 다운받은 toast-logncrash-android-unity-sdk.unitypackage을 더블 클릭하여 Import합니다.

샘플 설명

샘플의 실행은 Assets > LogNCrash > Sample > SampleScene을 더블클릭하여 실행합니다. 샘플에는 초기화, 로그 전송, 에러 발생에 대한 예제가 기술되어 있습니다.

사용 예제

1. LogNCrashSettings를 통한 초기화

Unity 메뉴바에서 LogNCrash> Edit Settings를 선택하여 LogNCrashSettings를 생성합니다.
LogNCrashSettings는 AssetDatabase로 사용자 앱키와 SDK 동작을 정의 합니다.

- Appkey: 사용자 앱키
- URL: 콜렉터 주소, https://api-logncrash.cloud.toast.com를 사용합니다.
- Version: 로그 버전
- Send Warning: Unity에서 발생한 Warning 로그의 수집 여부
- Send Error: Unity에서 발생한 Error 로그의 수집 여부

- Send Debug Warning: Unity에서 사용자가 Debug 객체를 이용해 발생시킨 Warning 로그의 수집 여부
- Send Debug Error: Unity에서 사용자가 Debug 객체를 이용해 발생시킨 Error 로그의 수집 여부

LogNCrashSettings에 정보를 입력하고 LogNCrash객체의 파라미터가 없는 Initialize 함수를 호출하면 LogNCrashSettings에서 정보를 읽어와 초기화를 시도 합니다.

```
using Toast.LogNCrash;
namespace Toast.LogNCrash
{
    public class SampleScript : MonoBehaviour
    {
        void Start ()
        {
            LogNCrash.Initialize ();
        }
    }
}
```

2. Script를 통한 초기화 LogNCrash.Initialize에 파라미터를 입력하여 초기화를 시도 합니다. 파라미터는 서버 주소, 앱키, 버전, 포트, Send Thread Lock 실행 여부에 대한 정보를 넘겨줍니다.

```
using Toast.LogNCrash;
namespace Toast.LogNCrash
{
    public class SampleScript : MonoBehaviour
    {
        void Start ()
        {
            LogNCrash.Initialize ("https://api-logncrash.cloud.toast.com",
"appkey", "1.0.0", 80,  true);
            LogNCrash.StartSendThread ();
        }
    }
}
```

- Appkey: 사용자 앱키
- URL: 콜렉터 주소, http, https의 콜렉터 정보를 설정
- Version: 로그 버전
- Port: 프로토콜에 따라 80, 443을 설정
- SendThreadLock: true인 경우 발생한 로그들은 StartSendThread가 호출되기 전까지 서버에 전송하지 않고, 큐에 저장합니다. 단 Native Crash가 발생한 경우 ThreadLock을 해제하고 로그를 전송합니다.

상세 API

커스텀 필드 지정하기

```
public static void AddCustomField(string key, string val)
public static void RemoveCustomField(string key)
public static void RemoveAllCustomFields()
```

- Parameters
  - key: string
    - [in] custom field의 key, custom key는 “A~Z, a~z, 0~9, - _” 문자를 포함하며 반드시 알파벳이나 숫자로 시작해야 합니다.
  - value: string
    - [in] custom field의 값
- Note
  - 다음 keyword는 SDK에서 사용 중이므로 사용 할 수 없습니다.
    - projectName
      - projectVersion
      - host
      - body
      - logLevel
      - userID
      - Platform
      - DmpData
      - Unity3D
      - Locale
      - CountryCode
      - SessionID
      - ExceptionType
      - NeloSDK
      - NetworkType
      - DeviceModel
    - DeviceID
      - @logType
  - custom filed의 값이 NULL이나 비어있는 경우, SDK 는 해당 필드를 server로 전송 하지 않습니다.

기본 설정 관리

```
public static void SetLogSource(string value)
public static string GetLogSource()
```

- 로그소스를 구하거나 새로 지정합니다.

```
public static void SetLogType(string value)
public static string GetLogType()
```

- 로그 타입을 구하거나 새로 지정합니다.

LEVEL 필터

- Unity SDK에서는 Default 설정으로 FATAL 레벨의 로그만 전송 합니다. Error, Warning 레벨의 로그에는 변수값(시간, 경로, 진행도 등)의 삽입으로 인해 많은 로그들이 발생 할 수 있습니다.
    - Send Error: 시스템에서 발생한 ERROR 레벨의 로그를 전송 합니다.
    - Send Warning: 시스템에서 발생한 WARN 레벨의 로그를 전송 합니다.
    - Send Debug Error: 사용자가 발생시킨 ERROR 레벨의 로그를 전송 합니다.
    - Send Debug Warning: 사용자가 발생시킨 WARN 레벨의 로그를 전송 합니다.

API 사용 예제

```
- html > index.html을 참고해 주시기 바랍니다.
```

IP Address 수집 설정

```
public static void SetEnableHost:(bool flag)
```

- true인 경우 ip address를 구하여 host 필드에 저장합니다.
- false인 경우 host 필드에 "-" 저장합니다.

로그 전송

```
//send info log message
public static void Info(string strMsg)

//send debug log message
public static void Debug(string strMsg)

//send warn log message
public static void Warn(string strMsg)

//send fatal log message
public static void Fatal(string strMsg)

//send error log message
public static void Error(string strMsg)
```

- Parameters
    - strMsg: string
        - [in] 전송할 log 메세지

Handled Exception

```
//send Handled info log message
public static void Info(string strMsg, Exception e)
```

```
//send Handled debug log message
public static void Debug(string strMsg, Exception e)

//send Handled warn log message
public static void Warn(string strMsg, Exception e)

//send Handled fatal log message
public static void Fatal(string strMsg, Exception e)

//send Handled error log message
public static void Error(string strMsg, Exception e)
```

```
try{
        // Exception code
}catch(Exception e){
        LogNCrash.Info("handled exception message", e)
}
```

- try&catch에서 발생한 Exception을 전송합니다.

크래시 콜백

```
public void Crash_Send_Complete_Callback(string message) {
        Debug.Log("Crash_Send_Complete_Callback : " + message);
}

void Start() {
        LogNCrashCallBack.ExceptionDelegate += Crash_Send_Complete_Callback;
}
```

- ExceptionDelegate는 Unity CSharp에서 발생한 Crash를 서버로 전송한 이후 호출되는 콜백 입니다.
  네이티브 Crash의 경우 호출되지 않습니다.

유저 아이디 설정

```
public static void SetUserId(string userID)
public static string GetUserID()
```

- 사용자별 통계 자료를 얻으려면 반드시 설정해주어야 합니다.
- Parameter
  - userID: string
    - [in] 각 사용자를 구분할 user id

중복 제거 모드 설정

2.4.0 이상 SDK 부터 일반 로그에 중복 제거 로직이 적용되었습니다. 초기화 시 중복 제거 로직이 활성화됩니다.

일반 로그의 경우 body와 logLevel이 같은 로그가 발생한 경우 전송하지 않습니다.

크래시 로그의 경우 stackTrace와 condition 값이 같은 로그가 발생한 경우 전송하지 않습니다.

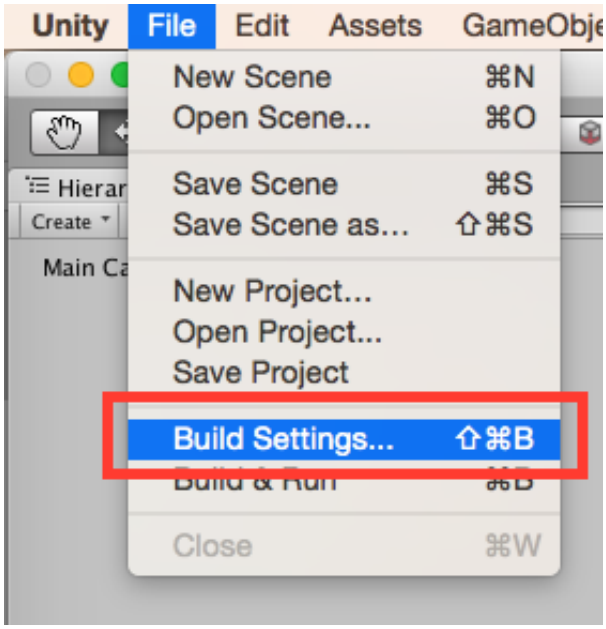
원하지 않는 경우 초기화 이후, 아래 함수를 통해 기능을 비활성화시킬 수 있습니다.

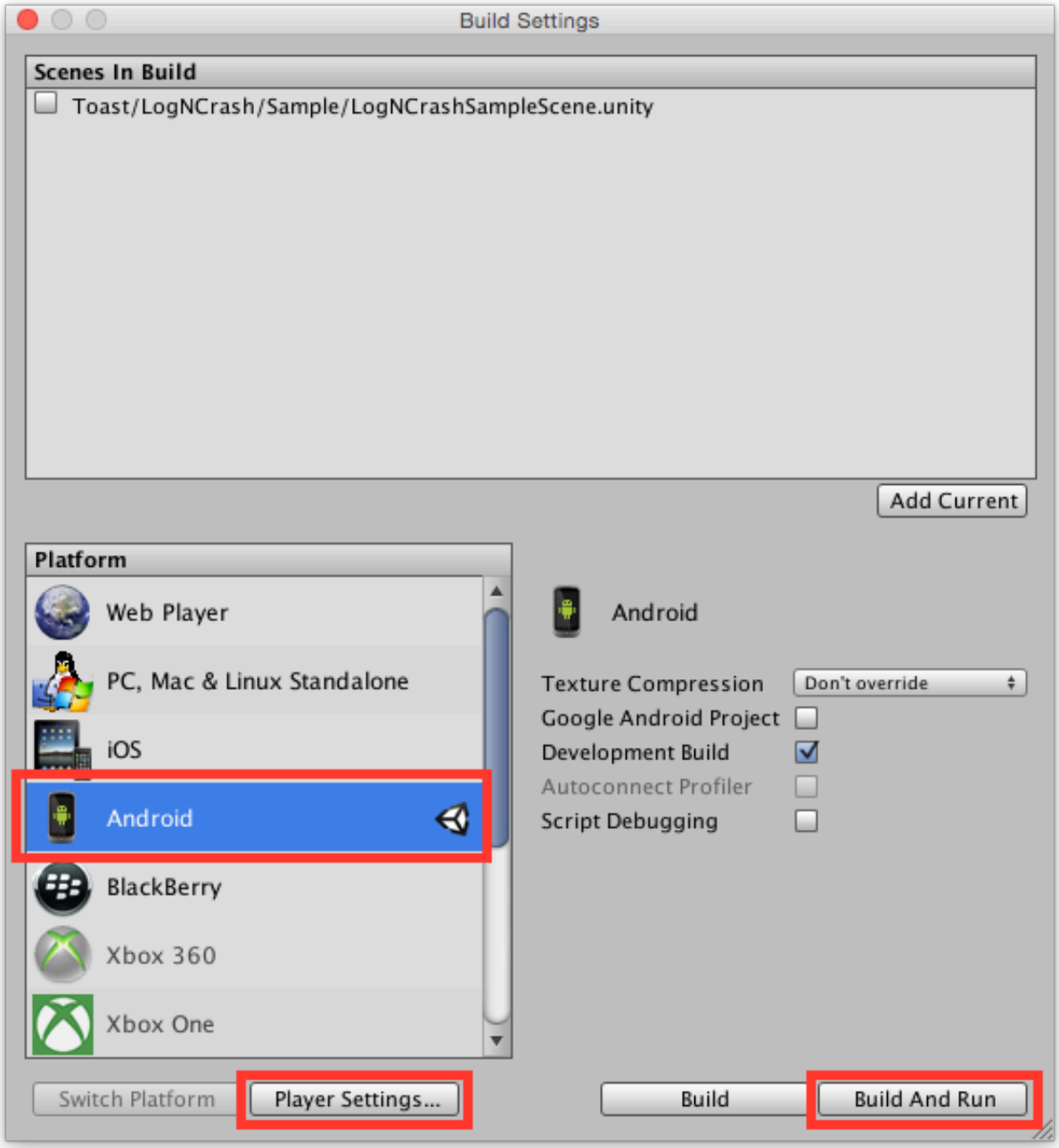
```
public static void SetDeduplicate(bool flag)
```

true :(Default 값) 중복 제거 로직 활성화
false: 중복 제거 로직 비활성화

Android Build 하기

1.File->Build Settings 클릭합니다.





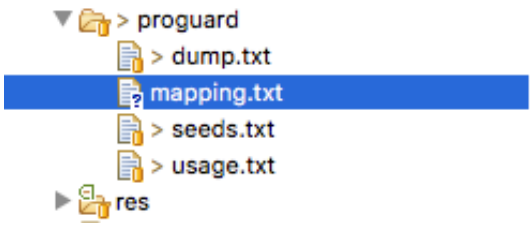
- Android Platform 선택 한 뒤 Player Settings 클릭합니다.

- Internet Access는 Require, Write Access는 External(SDCard)로 설정합니다.

2.Build settings에서 Build And Run 클릭합니다.

Android Unity Crash 해석하기

- Unity의 Crash는 Unity Engine에서 발생하는 Crash와 Android Naitve에서 발생하는 Crash로 구분됩니다.
- Proguard가 적용되지 않은 경우 별도의 Symbol 등록 과정이 필요하지 않습니다.
- Proguard가 적용된 경우 Native 레벨의 Crash 해석를 위하기 위해서는 mapping.txt 파일을 웹 콘솔 > Analytic > Log & Crash Search > Settings > 심볼 파일 탭에 등록해야 합니다.
- mapping.txt 파일은 proguard 폴더 하위에 생성됩니다.



Android Unity Crash 주의 사항

- 심볼이 없어 해석되지 않은 Crash 로그는 일반 로그로 취급됩니다.

외부 CrashHandler 사용하기

- 기존 SDK에서는 초기화 단계에서 logMessageReceived 등을 사용하여 Unity의 CrashHandler를 LogNCrash 전용 Callback 함수에 등록하여 사용하였습니다.
- 외부 CrashHandler와 같이 사용하는 경우가 있어, 같이 적용할 수 있도록 구조를 수정하였습니다. (MultihandlerSample 참고)

적용방법

- LogNCrash.SetCrashHandler 함수에 false를 파라미터로 넘겨 자동으로 CrashHandler가 등록되는 것을 막습니다.
- 반드시 Initialize 함수 이전에 설정되어야 합니다.

```
LogNCrash.SetCrashHanlder (false);
LogNCrash.Initialize ();
```

- 이후 LogNCrash.unity3dHandleException 함수를 사용하여 CrashHandler의 파라미터를 LogNCrash 객체로 넘겨줍니다.

```
void OnEnable()
{
        Application.logMessageReceived += HandleLog;
}

void HandleLog(string logString, string stackTrace, LogType type)
{
        if (LogNCrash.isInitialized) {
            LogNCrash.unity3dHandleException (logString, stackTrace, type);
        }
}
```

AssetDataBase를 활용한 빌드 환경 분기

- 메뉴바의 LogNCrash > Edit Settings를 클릭하면 간단한 데이터를 저장할 수 있는AssetDataBase가 생성됩니다.
- BuildPipeline.BuildPlayer를 통한 Build를 진행하는 경우 LogNCrashSettings.Setter_BuildType와 LogNCrashSettings.Getter_BuildType를 활용하여 빌드 환경을 분기 합니다.

```
using UnityEditor;
using UnityEngine;
using Toast.LogNCrash.Implementation;

public class lncAndroidBuildPipeline: MonoBehaviour
{
    [MenuItem("Build/Build Android (Alpha)")]
    public static void AndroidAlphaBuildScript()
    {
        BuildPlayerOptions buildPlayerOptions = new BuildPlayerOptions();
        buildPlayerOptions.scenes = new[]
{"Assets/Toast/Sample/Scene/Command/commandScene.unity"};
        buildPlayerOptions.locationPathName = "AndroidBuild.apk";
        buildPlayerOptions.target = BuildTarget.Android;
        buildPlayerOptions.options = BuildOptions.AutoRunPlayer;

        LogNCrashSettings.Setter_BuildType =
LogNCrashSettings.BuildType.alpha;

        BuildPipeline.BuildPlayer(buildPlayerOptions);
    }

    [MenuItem("Build/Build Android (Real)")]
    public static void AndroidRealBuildScript()
    {
        BuildPlayerOptions buildPlayerOptions = new BuildPlayerOptions();
        buildPlayerOptions.scenes = new[]
{"Assets/Toast/Sample/Scene/Command/commandScene.unity"};
        buildPlayerOptions.locationPathName = "AndroidBuild.apk";
```

```
        buildPlayerOptions.target = BuildTarget.Android;
        buildPlayerOptions.options = BuildOptions.AutoRunPlayer;

        LogNCrashSettings.Setter_BuildType = LogNCrashSettings.BuildType.real;

        BuildPipeline.BuildPlayer(buildPlayerOptions);
    }
}
```

- 명령에 따라 AssetDataBase에 저장된 값을 통해 LogNCrash 동작을 결정합니다.

```
using Toast.LogNCrash.Implementation;

void Start () {
        if (LogNCrashSettings.Getter_BuildType ==
LogNCrashSettings.BuildType.real) {
            SetReal ();
        } else if (LogNCrashSettings.Getter_BuildType ==
LogNCrashSettings.BuildType.alpha) {
            SetAlpha ();
        } else {
            UnityEngine.Debug.Log ("Default Type");
        }
}
```

- build type은 총 5개로 구성되어 있습니다.

```
public enum BuildType{
        real, alpha, beta, development, test
    }
```