PATIGAMES Publish SDK v1.4.1  
(for Unity)

(\*) 이 문서는 Unity를 위한 설정 방법과 C#에서의 SDK 사용법만 안내하고 있으므로 해당 문서를 볼 때는 반드시 Android 문서를 같이 참고할 것.

# API 사용 전 협의 사항

파티게임즈에서 제공하는 관리페이지에 게임이 등록되어 있는지, SDK/API 메뉴에서 gameid / ClientData(manifest에 추가하는 encryptedClientData) / serverKey등이 발급되어 있는지 확인한다.

관리 페이지 : <https://bo.patigames.com>(라이브) / <https://botest.patigames.com> (스테이지)

# 플러그인 설치

## [공통 작업]

(\*) 각 플랫폼 설정작업도 참고하면서 볼 것.

1. 적당한 위치에 압축을 푼 뒤, patisdk-unity-core.unitypackage를 Unity에서 Import Package로 가져온다. 경로에 한글이나 공백이 포함되지 않도록 주의한다.
2. 게임의 초기화 부분에서 다음 코드를 실행한다.

|  |
| --- |
| if (PatiSDK.Instance.InitSDK()) {  PatiSDK.Instance.Kakao.InitKakao(clientID, clientSecret)  // Kakao에서만 실행. 카카오에서 받은 clientID와 clientSecret을 등록한다.  PatiSDK.Instance.SetTestMode(); // 개발 버전에서만 실행  PatiSDK.Instance.GameLaunched();  } else {  // 에러 처리  } |

InitSDK()에서는 다음과 같은 작업이 진행된다.

1. Permission 확인 작업.
2. Manifest에서 meta-data로 넣은 항목 점검.
3. deviceID 또는 WIFI의 Mac address(WIFI 전용 기기인 경우)를 확인한다. WIFI 전용 기기에서 WIFI가 꺼진 상태라면 Mac Address를 얻어올 수 없기에 에러가 발생한다. 이 경우 게임을 시작하면 안 된다. 유저에게 “인터넷에 접속할 수 없습니다”와 비슷한 에러메시지를 보여주고 게임을 종료한다. 예외적으로 블루투스 테더링으로 인터넷이 가능한 경우가 있는데 극소수라 판단한다.

*(\*) C#의 PatiSDK 관련 함수는 namespace Pati에 있다.*

## [Android]

1. patisdk-unity-android.unitypackage를 유니티에서 Import Package로 가져온다.
2. PatiSDK가 필요로 하는 라이브러리들의 jar를 Assets/Plugins/Android에 추가한다. 라이브러리들이 필요로 하는 res들은 Assets/Plugins/Android/res 경로에 넣으면 된다.
3. AndroidManifest.xml, 필요에 따라 res/values/strings.xml을 수정한다. (수정 방법은 Android 문서 참고)
4. Build Settings의 Android에서 Google Android Project를 한 뒤 Export한다.
5. 유니티에서 Export한 프로젝트를 ADT(이클립스)에서 연다.
6. src/**%{PackageName}**/UnityPlayerNativeActivity.java를 열고 다음과 같이 수정한다.
   1. onCreate 함수 마지막에 아래와 같이 추가한다.

|  |
| --- |
| Pati.createInstance(**this**);  Pati.*getInstance*().initFacebook(savedInstanceState); // 페이스북 사용시에만 |

* 1. onDestroy, onPause, onResume에 해당하는 함수 각각에 맞는 Pati의 함수를 호출한다.

|  |
| --- |
| (예시)  @Override **protected** **void** onDestroy ()  {  Pati.*getInstance*().onDestroy();  mUnityPlayer.quit();  **super**.onDestroy();  } |

* 1. onSaveInstanceState, onStop, onActivityResult

역시 각각에 맞는 Pati의 함수를 호출하면 되지만, 기본적으로 Override되지 않은 상태이므로, Source -> Override/Implement Methods에서 추가한뒤 작업한다.

onSaveInstanceState, onStop은 NativeActivity, onActivityResult는 Activity에서 override할 수 있다.

|  |
| --- |
| (예시)  @Override  **protected** **void** onSaveInstanceState(Bundle outState) {  Pati.*getInstance*().onSaveInstanceState(outState);  **super**.onSaveInstanceState(outState);  } |

1. 프로젝트를 우클릭 한뒤, Export – Java – JAR file을 선택한 뒤, src와 gen을 선택하고 jar로 저장한다.
2. 저장된 jar를 유니티 프로젝트의 Assets/Plugins/Android에 넣는다.
3. 안드로이드로 빌드 및 실행한다.

## [iOS]

1. patisdk-unity-iOS-(kakao/facebook/googleplus)-#(version).unitypackage를 유니티에서 imoprt한다. 여러 플랫폼을 사용하는 경우 모두 import 한다.
2. Build한 뒤, Xcode 프로젝트를 연다.
3. 프로젝트의 Build Phases – Link Binary With Libraries에서 아래의 라이브러리를 추가한다.
   1. Security.framework
   2. Storekit.framework
   3. libc++.dylib
   4. ImageIO.framework
4. 프로젝트의 Info.plist를 열고, 아래 내용을 추가한다.
   1. PatiEncryptedClientData (string) : 발급받은 EncryptedClientData
5. 이후에는 카카오/페이스북 사용에 따라 작업이 다르다. (둘다 사용하지 않으면 필요없음.)
   1. 카카오 사용시
      1. Build Phases – Copy Bundle Resources에 KAAuth.bundle을 추가한다.
      2. Info.plist에 아래 내용을 추가한다.
         * URL types-URL Schemes-item 0 (string) : kakao***%{KakaoClientID}***
      3. Build Settings – Other Linker Flags에 –all\_load를 추가한다.
      4. UnityAppController.mm을 열고, 아래 내용과 같이 추가 및 수정한다.
         * #include 아래에 추가.

|  |
| --- |
| extern "C" BOOL \_patiunity\_handleOpenURLScheme(const char\* url, const char\* application); |

* + - * @implementation UnityAppController 안에 추가.

|  |
| --- |
| - (BOOL)application:(UIApplication \*)application handleOpenURL:(NSURL \*)url  {  return \_patiunity\_handleOpenURLScheme([[url absoluteString] UTF8String], "");  } |

* + - * application:(UIApplication\*)application openURL의 return을 아래와 같이 수정.

|  |
| --- |
| return \_patiunity\_handleOpenURLScheme([[url absoluteString] UTF8String], [sourceApplication UTF8String]); |

* 1. 페이스북 사용시
     1. Build Phases – Link Binary With Libraries에 FacebookSDK.framework를 추가한다.
     2. Info.plist에 아래 내용을 추가한다.
        + FacebookAppID (string) : ***%{FacebookAppID}***
        + FacebookDisplayName (string) : 페이스북 개발자 센터의 App 이름
        + URL types-URL Schemes-item 0 (string) : fb***%{FacebookAppID}***
     3. UnityAppController.mm을 열고, 아래 내용과 같이 추가 및 수정한다.
        + applicationDidBecomeActive:(UIApplication \*)application에 아래 내용 추가.

|  |
| --- |
| [FBAppEvents activateApp];  [FBAppCall handleDidBecomeActive]; |

* + - * applicationWillTerminate:(UIApplication \*)application에 아래 내용 추가.

|  |
| --- |
| [FBSession.activeSession close]; |

* + - * application:(UIApplication \*)application openURL의 return을 아래와 같이 수정.

|  |
| --- |
| return [FBSession.activeSession handleOpenURL:url]; |

* 1. 구글플러스 사용시
     1. Build Phases – Link Binary With Libraries에 GooglePlus.framework를 추가한다.
     2. Info.plist에 아래 내용을 추가한다.
        + URL types-URL Schemes-item 0 (string) : 앱 BundleId
        + URL types-URL Identifier (string) : 앱 BundleId
        + GooglePlusClientId (string) : 발급받은 google\_auth\_ios키
     3. UnityAppController.mm을 열고, 아래 내용과 같이 추가 및 수정.
        + #import <GooglePlus/GooglePlus.h>를 포함.
        + application:(UIApplication \*)application openURL에 아래 코드를 추가.

|  |
| --- |
| [[GPPSignIn sharedInstance] handleURL:url sourceApplication:sourceApplication annotation:annotation]; |

## [Windows]

Unity Pro버전에서만 사용가능한 기능.

원활한 개발을 위해, 윈도우 UnityEditor에서 테스트용도로 사용되는 SDK를 지원한다. 플랫폼에 의존적인 기능(페이스북, 카카오)는 간단한 테스트만 가능하고 상세한 기능은 생략되어있으며, 단순 개발 테스트를 위한 버전이므로 각 플랫폼 (안드로이드, iOS)에서의 추가적인 테스트가 필요하다.

1. patisdk-unity-desktop.unitypackage를 유니티에서 Import Package로 가져온다.
2. 게임의 초기화 부분에서 InitSDK를 실행하기 전에 다음 코드를 실행한다.

|  |
| --- |
| #if UNITY\_EDITOR || UNITY\_STANDALONE  PatiSDKDesktop.SetEncryptedClientData("***%{EncryptedClientData}***");  PatiSDKDesktop.SetMarketName("***%{Market}***");  PatiSDKDesktop.SetPackageName("***%{PackageName}***");  PatiSDKDesktop.SetUsingPrivacyInfo(***%{UsingPrivacyInfo}***);  // 아래의 코드를 통해 PatiSDK의 로그를 UnityEditor에서 확인할 수 있다.  PatiSDKDesktop.SetLogDelegate((type, log) => {  if (type == PatiSDKDesktop.LogType.Debug)  Debug.Log(log);  else  Debug.LogWarning(log);  });  #endif |

1. UnityEditor에서 실행한다.

\* 실행시 DllNotFoundException이 발생한다면 Assets/Plugins/x86경로가 존재하는지 확인한다. 만약 존재하면서 x86경로가 필요하다면 Assets/Plugins의 patisdk.dll을 x86경로로 이동한뒤 재실행 해본다. 필요하지 않으면 x86을 삭제한다.

# 게임 버전 정보

|  |
| --- |
| PatiSDK.Instance.GetVersionInfo((isMaintenance, maintenanceMsg, versionInfo) => {  // isMaintenance (bool)  // maintenanceMsg (string)  // versionInfo (Dictionary<string, object>)  }, (errorCode, errorMsg, otherInfos) => {  // errorCode (int)  // errorMsg (string)  // otherInfos (Dictionary<string, object>)  }); |

최신 게임 버전 정보를 받아온다. 현재 점검 상태 정보도 함께 내려온다.

versionInfo에는 dataVer, appVer, appupdate, marketurl 4개의 값이 담겨온다. 자세한 내용은 안드로이드 문서 참조.

만약 appupdate가 업데이트필수(FORCE)라면 deprecated된 앱으로 판단하여 콜백 이후로 호출되는 API request를 차단한다.

# 로그인

|  |
| --- |
| (수동 게스트 로그인 예시)  PatiSDK.Instance.Login(false, new AuthInfo().setGuest(), (uid, timestamp, authToken, otherInfos) => {  // uid (int)  // timestamp (int)  // authToken (string)  // otherInfos (Dictionary<string, object>)  }, (errorCode, errorMsg, otherInfos) => {  // errorCode (int)  // errorMsg (string)  // otherInfos (Dictionary<string, object>)  }); |
| (자동 로그인 예시)  PatiSDK.Instance.Login(true, null, (uid, timestamp, authToken, otherInfos) => {  // uid (int)  // timestamp (int)  // authToken (string)  // otherInfos (Dictionary<string, object>)  }, (errorCode, errorMsg, otherInfos) => {  // errorCode (int)  // errorMsg (string)  // otherInfos (Dictionary<string, object>))  }); |

게임 계정을 새로 만들거나 해당 플랫폼으로 로그인한다. (setGuest: 게스트, setPati(email, password(, nick)): PatiFriends 회원, setKakao(): 카카오 회원, setFacebook(): 페이스북 회원)

게스트 로그인의 경우 유저가 앱을 삭제하거나 앱 데이터를 삭제할 경우 다시 같은 게임 계정으로 로그인할 수 없게 된다. 따라서 게스트 로그인 기능을 쓰는 게임의 경우 게임에서 이런 사항을 잘 안내하고, 회원 가입을 최대한 유도해야 한다. 회원 로그인은 삭제되어도 자동 로그인만 풀릴 뿐, 다시 로그인하면 된다.

|  |
| --- |
| bool enableAutoLogin = PatiSDK.Instance.IsEnableAutoLogin; |

위 Property를 사용하면, 저장된 로그인 정보가 있는지 확인할 수 있다. 게임 초기화면에서 로그인 선택을 생략하고 싶을 때 호출하면 된다. 하지만 로그인 정보가 있어도 여러 가지 이유로 인해 로그인에 실패할 수도 있고, 게스트 로그인의 경우 신규 가입이 될 수도 있음에 주의하라.

플랫폼의 결제 기능을 사용하는 경우 otherInfos에 유저의 잔액 정보가 함께 제공된다. 캐시의 종류에는 paid, bonus, free가 있으며(결제 항목 참조) otherInfos내 키 값은 paid\_cash,bonus\_cash, free\_cash이다.

(\*) Android 문서의 로그인 보안 프로세스 항목을 반드시 확인할 것!

# 로그아웃

|  |
| --- |
| (로그아웃 예제)  PatiSDK.Instance.Logout(() => {  // success  }, (errorCode, errorMsg, otherInfos) => {  // errorCode (int)  // errorMsg (string)  // otherInfos (Dictionary<string, object>)  }); |

게임에서 로그아웃 할 때 호출한다. 회원 연동이 된 계정에서만 호출 가능하다. 성공하면 자동 로그인, 푸시 토큰정보가 삭제되며, 카카오나 페이스북 로그인이 되있는 경우 해당 플랫폼의 로그아웃이 호출된다. 다음 로그인에서 자동 로그인을 호출하는 경우 오류가 발생하거나 게스트로 새로 게임이 시작될 수 있으니 주의한다.

# 로컬 로그인 정보 제거

|  |
| --- |
| (로컬 로그인 정보 제거 예제)  PatiSDK.Instance.ClearLocalLoginInfo(); |

로그인 상태에선 실행되지 않으며 이 때는 false가 리턴된다.

게스트를 포함한 로컬에 저장되어있는 로그인 정보가 제거되므로 주의해서 사용해야만 한다.

# 탈퇴

|  |
| --- |
| (탈퇴 예제)  PatiSDK.Instance.Unregister("password", () => {  // success  }, (errorCode, errorMsg, otherInfos) => {  // errorCode (int)  // errorMsg (string)  // otherInfos (Dictionary<string, object>)  }); |

회원을 탈퇴할 때 호출한다. 파티 회원으로 로그인했을때는 패스워드 입력이 필요하며, 다른 경우에는 비워두어도 상관없다. 탈퇴시 7일의 유예기간이 생긴다.

## [탈퇴 철회]

|  |
| --- |
| (탈퇴 철회 예제)  PatiSDK.Instance.LoginCancelUnregister(new AuthInfo().setPati("email", "password"), (uid, timestamp, authToken, otherInfos) => {  // uid (int)  // timestamp (int)  // authToken (string)  // otherInfos (Dictionary<string, object>)  }, (errorCode, errorMsg, otherInfos) => {  // errorCode (int)  // errorMsg (string)  // otherInfos (Dictionary<string, object>)  }); |

탈퇴 유예기간인 경우 철회를 할 수 있다. 탈퇴한 계정으로 로그인하면 -130번 에러 코드가 내려오며, (이 에러코드는 계정 블록이 되었을 때도 사용되므로 에러코드 항목을 참고해서 작업할 것) 유저에게 철회 여부를 물은 뒤, 탈퇴 철회를 한 뒤 로그인을 할 수 있다. (이 때는 반드시 수동 로그인으로 처리된다.)

# 게임 친구 관련

|  |
| --- |
| (게임 친구 얻기 예제)  PatiSDK.Instance.GetGameFriends((friends) => {  // friends (List<FriendInfo>)  }, (errorCode, errorMsg, otherInfos) => {  // errorCode (int)  // errorMsg (string)  // otherInfos (Dictionary<string, object>)  }); |
| (게임 친구 요청 예제)  PatiSDK.Instance.RequestGameFriend(friend\_uid, (relation) => {  // relation (FriendRelation)  }, (errorCode, errorMsg, otherInfos) => {  // errorCode (int)  // errorMsg (string)  // otherInfos (Dictionary<string, object>)  }); |

GetGameFriends로 게임 친구 목록을 가져올 수 있다. FriendsInfo는 UserID와 Relation property를 가지고 있는데, UserID는 친구의 게임 ID, Relation은 친구와의 관계를 나타내는 FriendRelation enum값이다. FriendRelation은 네가지 값이 있는데, None(아무 관계도 아님), Following(해당 유저에게 친구 요청을 한 경우), Follower(다른 유저가 자신에게 친구 요청을 한 경우), Friend(서로 요청을 보내 친구인 상태)가 있다.

원본 JSON을 참조하고 싶은 경우, FriendInfo의 RawData 프로퍼티를 참조하면 원래의 JSON을 Dictionary<string, object> 형식으로 변환한 값을 얻을 수 있다.

친구 요청/취소는 RequestGameFriend, CancelGameFriend를 사용하는데, 두 함수는 기능만 다르고 parameter와 callback이 같기 때문에 똑같은 용법으로 사용하면 된다.

요청/취소엔 해당 유저의 UserID가 필요하며, 해당 동작이 처리된 후 해당 유저와의 FriendRelation이 콜백으로 반환된다.

# 게스트

## [회원 연동]

|  |
| --- |
| (Guest -> PatiFriends 회원 연동 예제)  PatiSDK.Instance.AddAuth(new AuthInfo().setPati("email", "password", "nick"), () => {  }, (errorCode, errorMsg, otherInfos) => {  // errorCode (int)  // errorMsg (string)  // otherInfos (Dictionary<string, object>)  }); |

로그인한 게임 계정에 플랫폼 회원정보를 연동한다. (setPati(email, password(, nick)): 파티 회원, setKakao(): 카카오 회원, setFacebook(): 페이스북 회원) 연동이 성공하면 다음 로그인부터는 연동한 회원정보로 로그인이 가능해진다.

한 게임 계정에 대해 파티도 연동하고 페이스북도 연동하는 등의 여러 플랫폼 연동은 가능하다. 그러나 한 플랫폼 계정에 여러 게임 계정 연동이라던가 한 게임계정에 같은 플랫폼 계정 여러 개를 연동하는 것은 불가능하다.

## [게스트 정보 삭제]

|  |
| --- |
| PatiSDK.Instance.RemoveGuestInfoPermanently(); |

이 함수의 호출은 반드시 유저에 충분한 안내와 함께 동의를 얻은 후 호출해야 한다. 함수를 호출하면 자동로그인을 위해 어플리케이션에 저장되어 있던 게스트 정보가 삭제된다. 한번 게스트 정보를 삭제하고 나면 SDK에서 다시 해당 계정을 복구할 방법은 없다.

# PatiFriends

## [회원 가입]

|  |
| --- |
| (파티 회원 가입 예제)  PatiSDK.Instance.SignupPati(new AuthInfo().setPati("email", "password", "nick"), (uid, timestamp, authToken, otherInfos) => {  // uid (int)  // timestamp (int)  // authToken (string)  // otherInfos (Dictionary<string, object>)  }, (errorCode, errorMsg, otherInfos) => {  // errorCode (int)  // errorMsg (string)  // otherInfos (Dictionary<string, object>)  }); |

파티게임즈에 회원으로 가입한다. 회원 아이디와 비밀번호로 사용할 email, password는 필수, nick은 선택이다. nick의 기본값은 빈 문자열이다. 초기화면에서 회원 가입을 시도할 때 SDK에 이미 게스트로 플레이 했던 정보가 있다면 에러가 발생한다. 이 경우 우선 로그인 후 회원가입을 진행해야 한다. (게스트 계정 유실 방지).

## [비밀번호 찾기]

|  |
| --- |
| (파티 회원 비밀번호 찾기 예제)  PatiSDK.Instance.ForgotPassword("email", () => {  // success  }, (errorCode, errorMsg, otherInfos) => {  // errorCode (int)  // errorMsg (string)  // otherInfos (Dictionary<string, object>)  }); |

파티 회원이 비밀번호 찾기를 원할 때 사용한다. email은 회원가입에 사용한 이메일 주소로 해당 이메일로 가입된 유저가 있으면 메일주소로 비밀번호를 바꿀 수 있는 웹페이지 링크가 전송된다. 호출이 성공하면 메일을 확인해보라는 안내를 해야 한다.

## [비밀번호 변경]

|  |
| --- |
| (파티 회원 비밀번호 변경 예제)  PatiSDK.Instance.ChangePassword("currentPW", "newPW", () => {  // success  }, (errorCode, errorMsg, otherInfos) => {  // errorCode (int)  // errorMsg (string)  // otherInfos (Dictionary<string, object>)  }); |

파티 회원의 비밀번호를 변경한다. 파티 회원으로 로그인한 상태에서만 가능하다.

# Kakao

카카오SDK를 지원하려면 초기화할 때 InitKakao 함수를 호출해야 하며, 관련 라이브러리를 추가해야한다. (플러그인 설치 항목 참고.)

## [친구목록 얻기]

|  |
| --- |
| (카카오 친구 가져오기 예제)  PatiSDK.Instance.Kakao.GetFriends((appFriends, nonAppFriends) => {  // appFriends (List<object>)  // nonAppFriends (List<object>)  foreach (Dictionary<string, object> v in appFriends)  {  int gameUserID = (int)v["game\_user\_id"];  string kakaoUserID = (string)v["user\_id"];  string kakaoNickname = (string)v["nickname"];  string kakaoProfileImageURL = (string)v["profile\_image\_url"];  bool kakaoMessageBlocked = (bool) v["message\_blocked"];  string kakaoHashedTalkUserId = (string)v["hashed\_talk\_user\_id"];  }  foreach (Dictionary<string, object> v in nonAppFriends)  {  string kakaoUserID = (string)v["user\_id"];  string kakaoNickname = (string)v["nickname"];  string kakaoProfileImageURL = (string)v["profile\_image\_url"];  bool kakaoMessageBlocked = (bool) v["message\_blocked"];  string kakaoHashedTalkUserId = (string)v["hashed\_talk\_user\_id"];  string kakaoSupportedDevice = (string)v["supported\_device"];  }  }, (errorCode, errorMsg, otherInfos) => {  // errorCode (int)  // errorMsg (string)  // otherInfos (Dictionary<string, object>)  }); |

카카오 친구 목록을 가져온다. appFriends는 게임에 가입된 친구, nonAppFriends는 게임에 가입하지 않은 친구이며, appFriends는 game\_user\_id를 추가로 받을 수 있다. NonAppFriends의 경우는 요청한 게임이 지원하는 OS를 사용하고 있는지를 알려주는 supported\_device가 Kakao로부터 추가로 내려온다.

카카오 user\_id의 경우 NonAppFriends이면 음수 형식이 되며 해당 사용자가 AppFriends가 되야 양수 형태로 바뀌며 절대값도 같지 않다. 하지만 hashed\_talk\_user\_id는 사용자가 기기를 변경하거나 카카오톡을 탈퇴하는 경우가 아니면 앱 가입 전/후와 관계없이 일정한 값을 가지므로 초대 보상과 관련된 처리를 할 때 사용하면 된다.

윈도우 버전에서 기능 테스트가 필요한 경우, PatiPublish\_preference.dat를 열고 아래와 같이 JSON을 수정하면 된다. 지정된 Key로 로그인 했을 때 받아올 친구 데이터를 설정할 수 있다. (PatiPublish\_preference.dat 파일은 프로젝트 경로나 실행 경로에 있다.)

|  |
| --- |
| (카카오 친구 윈도우 테스트 예제)  {  "PseudoKakaoRelation" : {  "%{KakaoUserId}" : {  "app\_friends\_info" : [  {  "game\_user\_id" : %{KakaoAppFriendGameUserID},  "nickname" : "AppFriend1",  "user\_id" : "%{KakaoAppFriendId}"  },  {  "game\_user\_id" : %{KakaoAppFriendGameUserID},,  "nickname" : "AppFriend2",  "user\_id" : "%{KakaoAppFriendId}"  }  ],  "friends\_info" : [  {  "nickname" : "NonAppFriend1",  "user\_id" : "%{KakaoNonAppFriendId}"  },  {  "nickname" : "NonAppFriend2",  "user\_id" : "%{KakaoNonAppFriendId}"  },  ]  }  }  } |

설정된 JSON은 getKakaoFriends를 호출했을 때 그대로 전달되며, 필요에 따라 각 유저의 정보에 profile\_image\_url, nickname, message\_blocked 등을 추가할 수 있다.

## [메시지 보내기]

|  |
| --- |
| (카카오 메시지 보내기 예제)  PatiSDK.KakaoPlatform.LinkMessageMetainfo metaInfo = new PatiSDK.KakaoPlatform.LinkMessageMetainfo();  metaInfo.ExecuteURL = "executeURL";  metaInfo.ImagePath = "imagePath";  metaInfo.Tag = new Dictionary<string,string>();  PatiSDK.Instance.Kakao.SendLinkMessage("receiverId", "templateId", metaInfo, () => {  // success  }, (errorCode, errorMsg, otherInfos) =>  {  // errorCode (int)  // errorMsg (string)  // otherInfos (Dictionary<string, object>)  }); |

카카오 친구에게 지정한 형식의 메시지를 보낸다. 각 인자와 LinkMessageMetainfo에 넣을 수 있는 값의 종류는 다음과 같다.

receiverId : 메시지를 받을 사용자의 카카오톡 ID

templateId : 채팅방 ID (optional, 채팅 플러스 지원용)

LinkMessageMetainfo.ImagePath : 메시지와 함께 전송할 이미지의 경로

LinkMessageMetainfo.ExecuteURL : App 구동시 전달할 parameter.

LinkMessageMetainfo.Tag : 템플릿에 등록한 tag와 tag에 치환될 문자열 전달 (Dictionary<string, string>)

# Facebook

## [친구목록 얻기]

|  |
| --- |
| (페이스북 친구 목록 예제)  PatiSDK.Instance.Facebook.GetFriends((appFriends, nonAppFriends) => {  // appFriends (Dictionary<string, object>)  // nonAppFriends (Dictionary<string, object>)  foreach (KeyValuePair<string, object> v in appFriends)  {  string facebookUserID = v.Key;  Dictionary<string, object> userInfo = (Dictionary<string, object>)v.Value;  int gameUserID = (int)userInfo["game\_user\_id"];  string userName = (string)userInfo["fbname"];  string userPicture = (string)userInfo["fbpic"];  }  foreach (KeyValuePair<string, object> v in nonAppFriends)  {  string facebookUserID = v.Key;  Dictionary<string, object> userInfo = (Dictionary<string, object>)v.Value;  string userName = (string)userInfo["fbname"];  string userPicture = (string)userInfo["fbpic"];  }  }, (errorCode, errorMsg, otherInfos) => {  // errorCode (int)  // errorMsg (string)  // otherInfos (Dictionary<string, object>)  }); |

페이스북 친구목록을 가져온다. appFriends는 게임에 가입된 친구, nonAppFriends는 게임에 가입하지 않은 친구이며, appFriends는 game\_user\_id를 추가로 받을 수 있다.

윈도우 버전에서 기능 테스트가 필요한 경우, PatiPublish\_preference.dat를 열고 아래와 같이 JSON을 수정하면 된다. 지정된 Key로 로그인 했을 때 받아올 친구 데이터를 설정할 수 있다. (PatiPublish\_preference.dat 파일은 프로젝트 경로나 실행 경로에 있다.)

|  |
| --- |
| (페이스북 친구 윈도우 테스트 예제)  {  ...    "PseudoFacebookRelation" : {  "%{FacebookID}" : {  "app\_friends\_info" : {  "1000001001" : {  "fbname" : "AppFriend1",  "game\_user\_id" : 80001  },  "1000001002" : {  "fbname" : "AppFriend2",  "game\_user\_id" : 80002  }  },  "friends\_info" : {  "1000002001" : {  "fbname" : "NonAppFriend1"  },  "1000002002" : {  "fbname" : "NonAppFriend2"  },  "1000002003" : {  "fbname" : "NonAppFriend3"  }  }  }  },    ...  } |

설정된 JSON은 getKakaoFriends를 호출했을 때 그대로 전달되며, 필요에 따라 각 유저의 정보에 fbname, fbpic 등을 추가할 수 있다.

## [요청 보내기]

|  |
| --- |
| (페이스북 요청 예제)  Dictionary<string, object> param = new Dictionary<string, object>();  PatiSDK.Instance.Facebook.SendRequestWithRequestDialog(param, "message", "title", (requestId) => {  // requestId (string)  }, (errorCode, errorMsg, otherInfos) => {  // errorCode (int)  // errorMsg (string)  // otherInfos (Dictionary<string, object>)  }); |

페이스북 요청을 보내기 위한 다이얼로그를 띄운다. param은 JSON으로 변환되며, Response로 받는 request ID가 콜백으로 반환된다. param에 넣을 수 있는 값은 <https://developers.facebook.com/docs/games/requests/v2.0> 를 참고.

## [뉴스피드 게시]

|  |
| --- |
| (페이스북 뉴스피드 게시 예제)  Dictionary<string, object> param = new Dictionary<string, object>();  PatiSDK.Instance.Facebook.PostFeedWithFeedDialog(param, (postId) => {  // postId (string)  }, (errorCode, errorMsg, otherInfos) => {  // errorCode (int)  // errorMsg (string)  // otherInfos (Dictionary<string, object>)  }); |

페이스북 타임라인에 게시하기위한 다이얼로그를 띄운다. Param은 JSON으로 변환되며, Response로 받는 postID가 콜백으로 반환된다. Param에 넣을 수 있는 값은 <https://developers.facebook.com/docs/sharing/reference/feed-dialog/v2.0> 를 참고.

# 계정전환

|  |
| --- |
| (Patifriends 계정으로 전환하는 예제)  PatiSDK.Instance.SwitchingAnotherAccount(new AuthInfo().setPati("email", "password"), (uid, timestamp, authToken, otherInfos) => {  // uid (int)  // timestamp (int)  // authToken (string)  // otherInfos (Dictionary<string, object>)  }, (errorCode, errorMsg, otherInfos) => {  // errorCode (int)  // errorMsg (string)  // otherInfos (Dictionary<string, object>)  }); |

이미 로그인 된 상태에서 다른 계정으로 전환하는 경우에 호출한다. 게스트를 제외한 어떠한 계정으로든 전환이 가능하다. 전환에 성공하게 되면 이전 계정에 대한 정보는 상실한다. 따라서 계정 전환에 성공하게 되면 즉시 게임 데이터가 전환된 계정으로 변경되어야 한다. 또한 게스트의 경우 다른 계정으로 전환한 후에는 다시 계정을 찾아올 방법이 없어지는 점을 유저에게도 충분히 알려주어야 한다.

만약 탈퇴 대기중인 계정을 탈퇴 철회하면서 전환하고자 하면 SwitchingAnotherAccountWithCancelUnregister(AuthInfo, OnLoginSuccessCallback, OnFailureCallback) 메소드를 사용하면 된다.

# 결제

결제를 통해 변경할 수 있는 화폐 단위는 여러 개가 될 수 없다. (플랫폼에서는 cash라 호칭한다.) 만약 게임 내에 충전 가능한 화폐단위가 여럿인 경우에는 결제를 통해 1차 화폐를 통해 2차 화폐를 충전하는 방식으로 변경해야 한다. Cash는 내부적으로 paid, bonus, free로 분리된다.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Paid | Bonus | Free |
| 환불에 포함 | ○ | Ⅹ | Ⅹ |
| 유료캐시용 컨텐츠 | ○ | ○ | Ⅹ |
| 사용 우선 순위 | 1 | 2 | 3 |

## [캐시 조회]

기본적으로 로그인 시에 otherInfos에 포함되어 있다. (자세한 사항은 로그인 항목을 참조) 로그인 후에 결제를 제외한 상황에서 캐시의 변동이 있을 경우 (서버에서 사용 / 지급등의 이벤트 발생) 캐시 잔액을 조회하는 api를 별도로 제공한다. (로그인이 필요함)

|  |
| --- |
| PatiSDK.Instance.GetCashBalance( (paid, bonus, free) =>  {  // paid (int)  // bonus (int)  // free (int)  }, (errorCode, errorMsg, otherInfos) =>  {  // errorCode (int)  // errorMsg (string)  // otherInfos (Dictionary<string, object>)  }); |

## [결제 결과 Callback 등록]

|  |
| --- |
| (결제 결과 Callback 등록 예제)  PatiSDK.Instance.SetPurchaseCashCallback((hasSuccess, result) => {  // hasSuccess (bool)  // result (Dictionary<string, object>)  }); |

결제 결과를 전달받을 콜백을 등록한다. 콜백을 등록하면 이전 결제시 결제는 성공했짐나 캐시를 지급받지 못한 경우에 대한 복구를 시도한다. 따라서 이 함수는 로그인 직후에 바로 호출되어야 하며, 유저에게 결제 결과를 안내할 수 있는 준비가 되어있어야 한다. 만약 비정상적인 결제가 여러 건 쌓여있다면 한 건씩 처리되면서 콜백 함수가 호출된다.

(\* result에 대한 정보는 Android문서를 참고할 것.)

## [결제 상품 조회]

|  |
| --- |
| PatiSDK.Instance.GetCashProducts((products) => {  // products (List<ProductInfo>)  }, (errorCode, errorMsg, otherInfos) => {  // errorCode (int)  // errorMsg (string)  // otherInfos (Dictionary<string, object>)  }); |

결제 상품 목록을 받는다. 결제 상품 목록은 기본적으로 백오피스를 통해 등록된 상품 목록을 가져오게 되며 앱스토어, 구글, 네이버 앱스토어 등 일부 마켓에서는 실제로 마켓에 해당 아이템이 등록되었는지 확인하는 절차를 거쳐서 목록이 필터링된다.

ProductInfo의 리스트로 전달되며, ProductInfo에는 해당하는 product의 구성과 ProductID가 존재한다. localprice의 경우 해당 지역의 가격이 등록되어 있지 않으면 null을 갖는다.

원본 JSON형태를 참조하고 싶은 경우, ProductInfo의 RawData 프로퍼티를 사용하면 Dictionary<string, object>의 형식으로 얻을 수 있다.

## [결제 시도]

|  |
| --- |
| PatiSDK.Instance.PurchaseCashProduct("pid", (errorCode, errorMsg, otherInfos) => {  // errorCode (int)  // errorMsg (string)  // otherInfos (Dictionary<string, object>)  }); |

결제 상품 조회를 통해 받아온 상품아이디로 결제를 시도한다. 이 함수는 결제 결과 Callback 함수가 등록되어 있어야 호출이 가능하다. 실제로 결제 결과는 위에 설명한 결제 결과 Callback으로 반환되지만, 방금과 같이 Callback 자체가 등록되지 않은 경우 등 몇가지 오류상황을 처리하기 위한 Callback이 필요하다.

## [구글/NStore/TStore 결제]

Android 문서를 참고하여 Manifest에 권한 또는 meta-data를 추가/수정한다.

(\*) Android 문서의 결제 오류 코드 항목을 반드시 확인할 것!

# 선물함

|  |
| --- |
| (선물함 조회 예제)  PatiSDK.Instance.GetGifts((gifts) =>  {  // gifts (List<GiftInfo>)  }, (errorCode, errorMsg, otherInfos) =>  {  // errorCode (int)  // errorMsg (string)  // otherInfos (Dictionary<string, object>)  }); |
| (선물 지급 처리 예제)  PatiSDK.Instance.ConsumeGift(1234, () =>  {  // success  }, (errorCode, errorMsg, otherInfos) =>  {  // errorCode (int)  // errorMsg (string)  // otherInfos (Dictionary<string, object>)  }); |

선물함을 조회하고, 선물 지급후 플랫폼에 지급 됬음을 알린다. consumeGift 함수를 사용하기 위해서는 파티게임즈와 사전에 협의해야만 사용 가능하다.

# 배너 호출

|  |
| --- |
| (배너 및 공지 호출 예제)  PatiSDK.Instance.ShowNoticeView("placement"); |

프로모션이나 공지목적으로 사용되는 배너를 호출한다. 배너는 백오피스에서 설정 가능하다.

첫번째 인자에는 미리 협의한 호출 위치를 지정하면 되며, 두번째 인자인 ignoreFrequency(기본값 false)에 true를 넘기면 백오피스에서 설정한 출력 빈도 제한을 무시하고 무조건 출력한다.

배너가 정상적으로 출력되었거나, 설정된 조건에 의해 스킵되었을 경우 true를, 잘못된 값으로 인해 실패할 경우 false를 반환한다.

# 약관 처리 관련

## [약관 가져오기]

|  |
| --- |
| (약관 가져오기 예제)  PatiSDK.Instance.CheckTerms((termsOfService, termsOfServiceLink, privacyPolicy, privacyPolicyLink) => {  // termsOfService (string) 이용약관 화면 표시 문구  // termsOfServiceLink (string) 이용약관 전문 보기 링크  // privacyPolicy (string) 개인정보 취급 방침  // privacyPolicyLink (string) 개인정보 취급 방침 전문 보기 링크  }, (errorCode, errorMsg, otherInfos) => {  // errorCode (int)  // errorMsg (string)  // otherInfos (Dictionary<string, object>)  }); |
| (약관 동의 여부 확인 예제)  if (PatiSDK.Instance.TermsAgreed)  // 동의한 상태  else  // 동의하지 않거나 약관이 만료된 상태 |

파티게임즈 서버에서 약관을 가져온다. 캐시된 약관이 없거나 마지막으로 약관을 서버에 요청한지 3일이 지나면 재요청을 하며, 서버에 요청할때마다 받은 약관을 캐시해둔다.

## [약관 동의/동의 취소]

|  |
| --- |
| (약관 동의/취소 처리 예제)  if (userinputAgree)  {  if (PatiSDK.Instance.AgreeTerms())  {  // 동의 완료  }  else  {  // 약관이 없거나 새로 갱신이 필요함. (CheckTerms 호출이 필요함.)  }  }  else  {  PatiSDK.Instance.DisagreeTerms();  } |

PatiSDK.Instance.AgreeTerms()는 약관에 동의할 때 호출한다. 만약 약관정보가 새로 갱신되었거나 약관이 존재하지 않으면 false를 반환하며, CheckTerms를 호출해야 한다.

PatiSDK.Instance.DisagreeTerms()는 약관이 갱신되었을 때 기본적으로 동의하지 않은 상태이므로 꼭 호출할 필요는 없다.

## [저장된 약관 삭제]

테스트나 기타 다른 이유로 로컬에 캐시되어있는 약관의 삭제가 필요한 경우 PatiSDK.Instance.RemoveTermsPermanently()를 사용하면 캐시된 약관 정보와 동의 여부가 삭제되어 다시 서버에서 받아올 수 있다.

# Push 서비스 관련

유니티 iOS환경에서 푸시 사용은 다음과 같은 설정이 필요하다

유니티에서 빌드를 한 뒤, 출력되는 Xcode 프로젝트가 열리면 UnityAppController.mm 파일을 열고, 아래와 같이 수정한다.

|  |
| --- |
| extern “C” void \_patiunity\_registerPushToken(unsigned char\* deviceToken, int size);  - (void)application:(UIApplication\*)application didRegisterForRemoteNotificationsWithDeviceToken:(NSData\*)deviceToken  {  … (기존 내용)  \_patiunity\_registerPushToken((unsigned char\*)deviceToken.bytes, deviceToken.length);  } |

|  |
| --- |
| (Push 서비스 사용 관련 예제)  if (PatiSDK.Instance.PushServiceEnabled)  PatiSDK.Instance.PushServiceEnabled = false;  else  PatiSDK.Instance.PushServiceEnabled = true; |

Push 서비스의 상태를 조회하거나, 키거나 끌 때는 PatiSDK.Instance.PushServiceEnabled property를 참조한다. 값을 참조하거나 대입함으로써 상태를 조회하고 설정할 수 있다.

# 외부 웹 브라우저/웹뷰 사용

|  |
| --- |
| (지정된 URL로 웹뷰 표시 예제)  PatiSDK.Instance.OpenWebView("http://www.google.com"); |
| (지정된 URL로 외부 웹 브라우저 표시 예제)  PatiSDK.Instance.OpenWebBrowser("http://www.google.com"); |
| (고객센터 웹 페이지 호출 예제)  PatiSDK.Instance.OpenCSWebPage(); |

PatiSDK에 내장된 웹뷰 또는 외부 웹 브라우저를 호출한다. 지정된 URL로 접속하며, 내장 웹뷰를 사용하는 경우, 웹뷰를 조작할 수 있는 자바스크립트 인터페이스를 제공한다.

안드로이드의 경우 AndroidManifest.xml에서 내장 웹뷰 사용 설정을 먼저 진행해야 한다.

(\*) 안드로이드의 경우 페이지에 pati.js를 포함하지 않아도 인터페이스를 사용가능 하지만, iOS에서 문제가 발생하므로 반드시 포함시켜야 한다.