PATIGAMES Publish SDK v1.4.1

(for Android)

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://admin.patigames.com(라이브) / https://admintest.patigames.com (스테이지))

Library Project 추가

먼저 PatiSDK의 라이브러리를 추가해야 한다. PatiSDK는 기본적으로 네이티브 라이브러리로 되어 있으며, prebuilt/android경로에 있다. armeabi, armeabi-v7a폴더와 Android.mk를 복사해 적당한 경로에 두고, Native 빌드를 진행할 때 그 경로를 링크하면 된다. 또한 SDK가 안드로이드의 기능을 사용하기 위해 patisdk.jar도 게임 프로젝트의 libs에 포함해야 한다.

그 다음으로, 프로젝트에 SDK가 요구하는 라이브러리 및 추가로 연동할 플랫폼들의 라이브러리를 넣어야 한다. 프로젝트 형태로 된 라이브러리는 Workspace에 추가한뒤 Project Properties > Android > Library에도 추가하면 되며, jar형태로 묶여있는 라이브러리나 so같은 네이티브 라이브러리는 게임 프로젝트의 libs에 추가하면 된다. (네이티브 라이브러리의 경우 armeabi, armeabi-v7a가 따로 있는 경우 둘다 넣는다.)

SDK가 필수로 요구하는 라이브러리 및 필요에 따라 추가해야할 라이브러리들은 아래와 같다. (내려받은 SDK 폴더 내의 external 경로에 있다.)

▣ 필수 라이브러리

1. android-support-v4.jar
2. gcm.jar

▣ 선택 라이브러리

1. kakao-android-sdk (카카오 플랫폼 연동시)
2. facebook (페이스북 플랫폼 연동시)
3. google-play-services (Google+ 플랫폼 연동시)
4. naveroauthlib-4.1.3 (네이버 플랫폼 연동시)
5. simple-crop-image-lib (프로필 이미지 기능 사용시)
6. GooglePlayBilling (구글 결제 사용시, 상세한 내용은 해당 항목 참조)
7. naver-iap-sdk.jar (네이버 결제 사용시, 상세한 내용은 해당 항목 참조)
8. tstore-iap_plugin.jar, tstore-libdodo.so (T스토어 결제 사용시)

(google-play-services는 ADID수집 이외의 기능을 사용하지 않을 경우, 프로젝트 내부의 libs/google-play-services.jar과 res/values/version.xml만 추가해도 된다.)*

AndroidManifest.xml 설정

▣ Android SDK버전

```
<uses-sdk android:minSdkVersion="9"/>
```

▣ 권한

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.GET_ACCOUNTS" />
<uses-permission android:name="android.permission.WAKE_LOCK"/>
<uses-permission android:name="android.permission.VIBRATE" />
<uses-permission android:name="android.permission.READ_PHONE_STATE" />
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />

<permission android:name="%{PackageName}.permission.C2D_MESSAGE"
android:protectionLevel="signature" />
<uses-permission android:name="%{PackageName}.permission.C2D_MESSAGE" />
<uses-permission android:name="com.google.android.c2dm.permission.RECEIVE"
/>
```

▣ application meta-data (<application> 태그 내에 삽입)

```
<meta-data android:name="com.patigames.api.encryptedClientData"
android:value="%{EncryptedClientData}" />
<meta-data android:name="com.patigames.api.marketData"
android:value="%{Market}"/>
<meta-data android:name="com.patigames.api.usingPrivacyInfo"
android:value="%{UsingPrivacyInfo}"/>
<meta-data android:name="com.google.android.gms.version"
android:value="@integer/google_play_services_version" />
```

%{EncryptedClientData}에는 발급받은 해당 값을 넣는다.

%{Market}에는 GOOG, NStore, SK, KT, LG 중 실제로 서비스될 마켓값을 입력한다. 문서에 존재하지 않는 마켓 값이 필요한 경우에는 SDK 업데이트가 필요하다.

%{UsingPrivacyInfo}에는 게임의 개인 정보 수집 여부에 따라 true나 false를 넣는다.

(없으면 기본으로 false값 사용)

초기화

Activity의 onCreate(Bundle)에 아래 내용 추가

```
(Java)
Pati.createInstance(this);
if (Pati.getInstance().initSDK())
{
        //Pati.getInstance().setTestMode();
        Pati.getInstance().gameLaunched();
}
```

Facebook을 사용하기 위해서는 별도의 설정 작업이 필요하다. (페이스북 항목 참조)

initSDK()에는 게임에 필요한 값들이 제대로 세팅되었는지 확인한다.

initSDK()에서는 크게 네 개의 작업이 진행된다.

1. Permission 확인 작업. 필수 Permission이 AndroidManifest.xml에 없을 경우 에러 로그를 출력하고, 선택 Permission이 없을 경우 경고 로그를 출력한다.
2. Manifest에서 meta-data로 넣은 항목들을 점검한다. encryptedClientData에서는 게임고유의 식별번호, 통신에 사용할 보안키등이 포함되어 있다. 해석에 실패할 경우 에러로그를 출력한다. marketData에는 어플리케이션을 서비스하는 마켓정보를 읽어온다. 정해진 마켓에 해당하지 않는 값이 있는 경우 에러로그를 출력한다.
3. deviceid 또는 WIFI의 Mac Address(WIFI 전용 기기의 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 태더링으로 인터넷이 가능한 경우가 있는데 극소수라 판단한다.
4. GCM Token을 발급 받게 되는데 기본적으로 파티게임즈 공통 GCM Project Number를 사용하게 된다. 기존에 GCM Project Number를 별도로 발급 받았던 게임들은 추가 인자로 넣으면 된다.

Activity의 onPause, onResume, onSaveInstanceState, onActivityResult, onStop, onDestroy에도 각각에 대응하는 Pati.getInstance().onXXX 함수를 추가해야한다.

JNI 설정

(유니티는 이 설정이 자동으로 진행되므로 확인할 필요 없다.)

PatiSDK native library를 안드로이드와 연결하려면 JNI가 필요하다. PatiSDK는 기본적으로 shared library를 지원하나 lua를 사용하는 경우 lua-binding은 static library이다. static library를 사용하지 않는 경우 so파일(shared library 파일)들을 프로젝트의 libs 디렉터리로 옮기면 된다. static library를 사용하는 경우에는 아래의 설명을 참고할 것.

1. Application.mk를 열고, APP_CPPFLAGS에 -std=c++11을 추가한다. **(없을 경우에만)**
2. Android.mk에 추가.

```
LOCAL_STATIC_LIBRARIES += patisdk_lua_static
LOCAL_SHARED_LIBRARIES += patisdk_shared
include $(LOCAL_PATH)/{patisdk Android.mk PATH} (파일 최하단에 위치하도록 추가)
```

3. 작업한 shared library를 빌드하고 안드로이드 프로젝트에 반영한다.
4. Java코드 MainActivity에 static으로 다음 코드를 추가

```
static
{
    System.loadLibrary("gnustl_shared");
    System.loadLibrary("patisdk");
}
```

트래픽 유입 추적

[BroadcastReceiver 설정]

```
<receiver android:name="com.patigames.api.InstallTrackingReceiver"
android:exported="true">
       <intent-filter>
              <action android:name="com.android.vending.INSTALL_REFERRER"
/>
       </intent-filter>
       <meta-data android:name="forward.%{ExternalBroadcastReceiverName}"
android:value="%{ExternalBroadcastReceiverName}"/>
</receiver>
<service android:name="com.patigames.api.InstallTrackingService"></service>
```

위의 태그를 AndroidManifest.xml 파일의 <application> 태그 내에 넣는다.

혹시 다른 트래픽 유입 추적용 API를 사용하는 경우, 두 리시버를 등록하면 안 된다. 액션 하나에는 하나의 BroadcastReceiver만 등록할 수 있게 되어있기 때문이다. 따라서 전달 기능이 마련되어 있다.

<meta-data>에 name은 "forward."으로 시작하는 이름을 정하고, value는 다른 BroadcastReceiver 클래스의 전체 경로(예: com.patigames.apitest.InstallReferrerReceiver)를 넣고, 해당 receiver에는 intent-filter 및 action 설정을 제거한다. 그러면 com.patigames.api.InstallTrackingReceiver가 메시지를 받아 전달한다.

주의할 것은 보통 다른 트래픽 추적 API도 forward 설정을 하게 되어 있을텐데, com.patigames.api.InstallTrackingReceiver를 receiver로 등록하고, 다른 트래픽 유입 추적용 API로 forward하는 방식으로 해야 한다. 당연하게도 다른 트래픽 추적 API의 forward에는 아무 것도 넣을 필요가 없다.

<meta-data>의 name은 키로 사용되므로 <receiver> 태그 내에서는 유일해야 한다.

[테스트 방법]

선행 작업: BroadcastReceiver까지 모두 설정된 앱을 디버깅할 장치에 설치한다.

```
adb shell am broadcast -a com.android.vending.INSTALL_REFERRER -n
<PackageName>/com.patigames.api.InstallTrackingReceiver --es
"referrer" "campaign name"
```

위와 같이 adb shell을 이용해서 Intent를 전송하는 방식으로 테스트 가능하다. 이 부분은 테스트 모드 설정이 반영되지 않는다. 따라서 라이브 서버에서 확인이 가능하다.

푸시 수신 설정

[NotificationReceiver 설정]

```
<receiver android:name="com.patigames.api.NotificationReceiver"
android:permission="com.google.android.c2dm.permission.SEND">
    <intent-filter>
        <action android:name="com.google.android.c2dm.intent.REGISTRATION"
/>
        <action android:name="com.google.android.c2dm.intent.RECEIVE" />
        <category android:name="%{AppPackageName}" />
    </intent-filter>
</receiver>
```

위의 <receiver> 태그를 AndroidManifest.xml 파일의 <application> 태그 내에 넣는다. ***%{AppPackageName}***에는 앱의 패키지 이름을 적는다.

기본적으로 푸시 아이콘은 게임 아이콘을 그대로 사용하며, 만약 푸시 아이콘과 게임 아이콘을 따로 설정해야할 경우**(구글 피쳐드에 필요함)** 아래와 같은 태그를 <receiver>안에 넣는다.

```
<meta-data android:name="notificationicon" android:value="%{Name}" />
```

위의 ***{Name}***에는 푸시 아이콘으로 사용할 drawable 리소스의 이름을 넣는다. (ex: 푸시 이미지가 res/drawable/noti_app_icon.png 인 경우, noti_app_icon만 입력.)

Push를 키거나 끌 때는 아래의 함수를 사용한다. 설정한 상태는 해당 기기에 저장되며, 유저가 다시 값을 재설정하기 전까지는 로그인/로그아웃에 관계없이 유지된다. (단, 로그인하지 않은 상태에서 푸시를 받을수는 없다.)

Push 서비스 on/off 예제
(C++) PatiSDK::enablePushService(); PatiSDK::disablePushService();
(Lua) PatiSDK.enablePushService() PatiSDK.disablePushService()

위에서 설정한 Push가 켜져있는지를 확인할때는 아래의 함수를 사용한다.

Push 설정 확인
(C++) PatiSDK::isEnablePushService()
(Lua) PatiSDK.isEnablePushService()

(registerRemoteNotification, unregisterRemoteNotification API는 deprecated 되었으나 여전히 사용은 가능하다.)*

게임 버전 정보

버전 정보 받아오기 예시

(C++)

```
PatiSDK::getVersionInfo([](bool isMaintenance, std::string maintenanceMsg,
JsonString versionInfo) {
        // success
}, [](int errorCode, std::string errorMsg, JsonString otherInfos) {
        // failure
});
```

(Lua)

```
PatiSDK.getVersionInfo(function(isMaintenance, maintenanceMsg, versionInfo)
        -- isMaintenance            (boolean)
        -- maintenanceMsg    (string)
        -- versionInfo(table)
end, function(errorCode, errorMsg, otherInfos)
        -- errorCode  (number)
        -- errorMsg   (string)
        -- otherInfos (table)
end)
```

Success Callback 인자 설명

isMaintenance : 점검 중이면 True, 아니면 False

maintenanceMsg : 백오피스에서 입력한 점검 메시지. 점검 중이 아니면 빈문자열

versionInfo : 항상 내려오는 데이터로 구조는 다음과 같다.

```
{
   "dataVer" : 최신 데이터 버전 (string)
   "appVer" : 최신 앱 버전 (string)
   "appupdate" : 백오피스에서 입력한 업데이트 설정. 상관없음(NONE), 업데이트 권장
(RECOMMEND), 업데이트 필수(FORCE) 셋 중 하나의 값이 들어온다. 만약 현재 앱 버전과 최신 앱 버전
사이에 업데이트 필수인 버전이 있다면 무조건 'FORCE'가 들어온다. (string)
   "marketurl" : 현재 앱에 설정된 마켓 주소 (string)
}
```

점검은 백오피스의 버전관리에서 시작 할 수 있다.

업데이트 설정이 검수용버전(INREVIEW)인 앱버전은 버전 체크에서 제외된다.

appupdate에 FORCE 값이 들어온다면 콜백 호출 후에 SDK에서 deprecated된 앱으로 판단하여 모든 API request를 차단한다.

게임 로그인

- 필요 권한: READ_PHONE_STATE, GET_ACCOUNTS

자동 로그인 시도 및 기기 기반 계정 목록을 이용한 로그인

자동 로그인 예시

(C++)

```
PatiSDK::tryAutoLogin([](int uid, int timestamp, std::string authToken,
JsonString otherInfos) {
        // login success
}, [](int errorCode, std::string errorMsg, JsonString otherInfos) {
        // login failure
});
```

(Lua)

```
PatiSDK.tryAutoLogin(function(uid, timestamp, authToken, otherInfos)
        -- uid         (number)
        -- timestamp   (number)
        -- authToken   (string)
        -- otherInfos  (table)
end, function(errorCode, errorMsg, otherInfos)
        -- errorCode   (number)
        -- errorMsg    (string)
        -- otherInfos  (table)
end)
```

기기 기반 계정 목록을 이용한 로그인 예시

(C++)

```
PatiSDK::loginWithUserOnDeviceIdx("idx", PatiAuthInfo().setFacebook(), [](int
uid, int timestamp, std::string authToken, JsonString otherInfos) {
        // login success
}, [](int errorCode, std::string errorMsg, JsonString otherInfos) {
        // login failure
});
```

(Lua)

```
PatiSDK.loginWithUserOnDeviceIdx("idx", PatiAuthInfo.setFacebook(),
function(uid, timestamp, authToken, otherInfos)
        -- uid         (number)
        -- timestamp   (number)
        -- authToken   (string)
        -- otherInfos  (table)
end, function(errorCode, errorMsg, otherInfos)
        -- errorCode   (number)
        -- errorMsg    (string)
        -- otherInfos  (table)
end)
```

게임 초기화면에서 로그인을 구현하기 위한 API. tryAutoLogin을 호출하여 로그인을 시도하고 자동으로 로그인할 수 없다고 판단되는 경우 기기 기반으로 검색된 계정 목록을 반환한다. 계정 정보에는 lastlogin(최종 로그인 시각), joindate(가입 시각), patifriends(파티프렌즈 인증 정보, 인증한 경우 파티프렌즈 id, 아닌 경우에는 null), facebook(페이스북 인증 정보), googleplus(구글 플러스 인증 정보), kakaotalk(카카오톡 인증 정보) 등이 있다.

로그인 시나리오

1. 신규 기기에서 최초 실행 시
   tryAutoLogin을 호출하면 신규 계정을 발급하여 로그인.
2. 클라이언트 로컬 데이터에 로그인 기록이 존재하는 경우
   tryAutoLogin을 호출하면 마지막으로 로그인 했던 계정으로 로그인.
3. 로컬 인증기록 없음 / 마지막 로그인 계정에 연동한 인증수단이 없음
   tryAutoLogin을 호출하면 마지막으로 로그인 했던 계정으로 로그인.
4. 로컬 인증기록 없음 / 마지막 로그인 계정에 연동한 인증 수단이 있음
   tryAutoLogin을 호출하면 -728 에러코드와 함께 인증 가능한 계정 목록이 반환된다. 로그인 UI를 통해 유저가 인증할 계정을 선택하여 인증을 시도한다. 인증에 성공하는 경우 로그인. 선택한 계정의 인증정보와 일치하지 않는 인증정보를 제공한 경우에는 -730 혹은 -149 에러가 발생한다.

커스텀 로그인

수동 게스트 로그인 예시

(C++)
```
PatiSDK::login(false, PatiAuthInfo().setGuest(), [](int uid, int timestamp,
std::string authToken, JsonString otherInfos) {
        // login success
}, [](int errorCode, std::string errorMsg, JsonString otherInfos) {
        // login failure
});
```

(Lua)
```
PatiSDK.login(false, PatiAuthInfo.setGuest(), function(uid, timestamp,
authToken, otherInfos)
        -- uid         (number)
        -- timestamp   (number)
        -- authToken   (string)
        -- otherInfos  (table)
end, function(errorCode, errorMsg, otherInfos)
        -- errorCode   (number)
        -- errorMsg    (string)
        -- otherInfos  (table)
end)
```

자동 로그인 예시

(C++)
```
PatiSDK::login(true, PatiAuthInfo(), [](int uid, int timestamp, std::string
authToken, JsonString otherInfos) {
        // login success
}, [](int errorCode, std::string errorMsg, JsonString otherInfos) {
        // login failure
});
```

(Lua)
```
PatiSDK.login(true, nil, function(uid, timestamp, authToken, otherInfos)
        -- uid         (number)
        -- timestamp   (number)
        -- authToken   (string)
        -- otherInfos  (table)
end, function(errorCode, errorMsg, otherInfos)
        -- errorCode   (number)
        -- errorMsg    (string)
        -- otherInfos  (table)
end)
```

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

로그인에 성공하면, 로그인 정보가 현재 앱의 SharedPreferences에 저장되어 이후 자동 로그인에 사용된다. (자동 로그인에는 따라서 AuthInfo가 필요없다. 예시 참고.) 페이스북이나 카카오 로그인의 경우, otherInfos로 해당 페이스북/카카오 계정과 관련된 정보가 내려온다.

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.

```
(C++) bool enableAutoLogin = PatiSDK::isEnableAutoLogin();
```

```
(Lua) local enableAutoLogin = PatiSDK.isEnableAutoLogin()
```

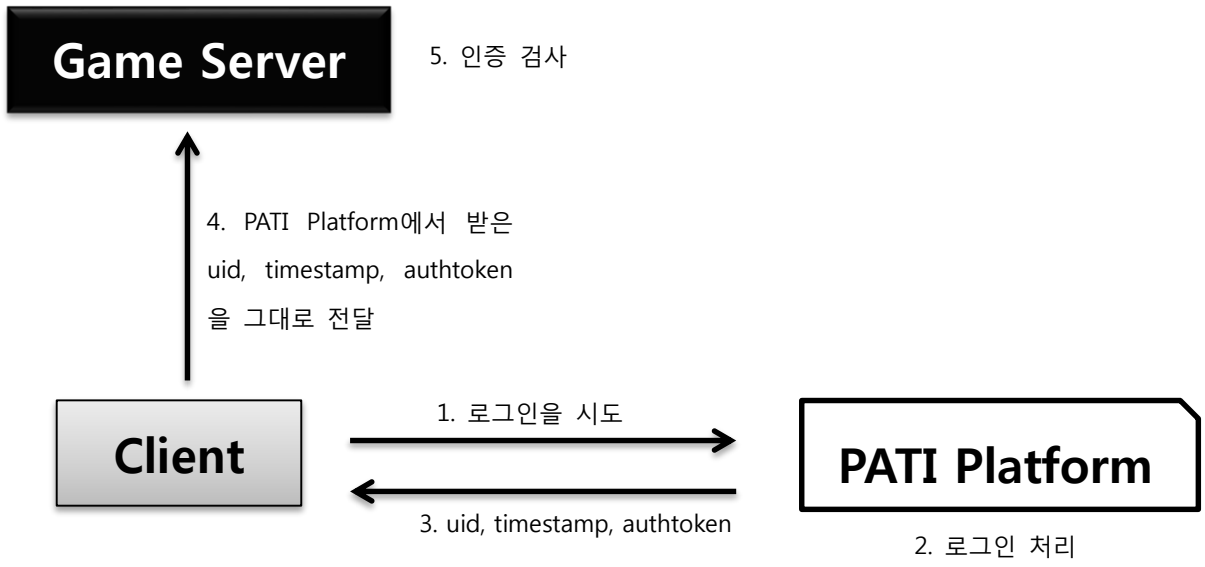
위 API를 사용하면, 저장된 로그인 정보가 있는지 확인할 수 있다. 게임 초기화면에서 로그인 선택을 생략하고 싶을 때 호출하면 된다. 하지만 로그인 정보가 있어도 여러 가지 이유로 인해 로그인에 실패할 수도 있고, 게스트 로그인의 경우 신규 가입이 될 수도 있음에 주의하라.

로그인을 시도할 때, Android 버전, 휴대폰 제조사 및 기종, deviceid(또는 Mac Address), 휴대전화 번호, 구글 계정, GCM Token 등의 정보를 수집하여 프로모션, 트래픽 분석, 고객문의에 활용한다.

로그인에 성공하면 onLoginSuccess가 호출되는데, uid가 회원번호이다. timestamp와 authToken은 보안과 관련된 부분으로 게임서버로 두 값을 함께 전송하여 검증한다. 클라이언트가 보낸 uid, timestamp와 미리 발급된 serverKey의 조합으로 계산한 값이 클라이언트에서 보낸 authToken과 같아야 하고(다르면 해킹), timestamp가 서버시간과 차이가 크다면 에러로 간주해야 한다.

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.

로그인 보안 프로세스



(*) serverkey는 client에 절대 노출하면 안되며, 게임 서버만 알고 있어야 한다.

이런 프로세스는 악의적인 유저가 Client를 해킹하거나, 가짜 클라이언트를 만들어 다른 사람의 계정을 해킹하는 문제를 방지하기 위해 꼭 필요하다. (이런 절차를 거치지 않으면 게임 서버는 해당 uid가 PATI Platform에서 정상적인 프로세스로 온 것이 맞는지 확인할 수가 없다.)

[인증 검사 방법]

1. 클라이언트가 보낸 uid, timestamp와 미리 발급된 serverkey의 조합으로 계산한 값이 클라이언트가 보낸 authtoken과 일치하는지 확인한다. 채널을 사용하는 경우 채널 선택 후 channelnum을 공식에 포함해서 확인한다.

```
EX) if (authTokenFromClient != sha256( str(uid) + str(timestamp) + str(serverKey))
       throw Exception("잘못된 로그인: 해킹시도 가능성 있음");

EX_channel) if (authTokenFromClient != sha256( '%d%03d%d%s' % (uid, channelnum, timestamp, server_key) )
```

2. 클라이언트가 보낸 timestamp를 현재 서버시간과 비교해 차이가 크다면 조작되었을 가능성이 있으므로 에러로 간주해야 한다. (timestamp는 GMT 기준 unix_timestamp이다.)

[* 언어별 sha256 사용법]

(sha256('hello world!') = 7509e5bda0c762d2bac7f90d758b5b2263fa01ccbc542ab5e3df163be08e6ca9)

```
1. PHP : echo hash('sha256', 'hello world!');
2. Python : import hashlib
            print hashlib.sha256("hello world!").hexdigest()
```

채널 기능

한 게임에서 완전히 분리된 유저 풀을 여럿 필요로 할 때 지원한다. 이 기능의 활성화를 위해서는 사전에 협의가 필요하다.

로그인 성공 콜백 변경

- 제외 항목
  - paid, bonus, free, total_payments : 결제 정보는 채널단위로 제공
  - servertoken, timestamp : 게임 서버에서 보안 검증에 채널정보를 로그인 콜백에서 제외
- 추가 항목
  - channels_summary: 채널 별로 게임에서

```
{
  "1": {
    "level": 레벨 값. 게임에서 api를 통해 제공한 값. int
    "summary": 레벨과 마찬가지로 api를 통해 게임에서 제공한 값을 그대로 내려보내 준다. json 형식. ex} 캐릭터 정보,닉네임 등등
    "numpeople": 채널에 존재하는 개략적인 유저 수
  }
  "2": {
    "summary":"{}","level":0,"numpeople":0
  }
  "3": {
    "numpeople":0
  }
}
```

  - last_selected_channel: 마지막으로 선택한 채널 넘버

채널 선택

채널 선택 예시

(C++)

```
PatiSDK::selectChannel(int channelnum, [](JsonString infos) {
        // success
}, [](int errorCode, std::string errorMsg, JsonString otherInfos) {
        // failure
});
```

(Lua)

```
PatiSDK.selectChannel(function(info)
        -- isMaintenance            (boolean)
        -- maintenanceMsg    (string)
        -- versionInfo(table)
end, function(errorCode, errorMsg, otherInfos)
        -- errorCode  (number)
        -- errorMsg   (string)
        -- otherInfos (table)
end)
```

Success Callback 인자 설명

paid, bonus, free, total_payments : 채널을 사용하는 경우 결제는 채널단위로 이루어진다. 따라서 결제관련 정보들이 로그인이 아닌 채널 선택 콜백에 포함된다

servertoken, timestamp : 채널을 사용하는 경우 게임서버에서 로그인 검증을 할 때 선택한 채널넘버도 포함해서 계산하게 된다. 자세한 사항은 로그인 보안 프로세스 항목을 참조

채널 퇴장

채널 퇴장 예시

(C++)

```
PatiSDK::exitChannel( []() {
        // success
}, [](int errorCode, std::string errorMsg, JsonString otherInfos) {
        // failure
});
```

(Lua)

```
PatiSDK.exitChannel(function()
        -- success
end, function(errorCode, errorMsg, otherInfos)
        -- errorCode  (number)
        -- errorMsg   (string)
        -- otherInfos (table)
end)
```

채널 선택은 로그인이 되어 있는 상황에서만 가능하다. 한번 채널을 선택하면 해당 채널에서 퇴장할 때까지 채널을 변경할 수 없다.

게임의 채널 정보 플랫폼으로 업데이트

업데이트 예시

```
(C++)
PatiSDK::updateChannelSummary(int level, JsonString jsonSummary, bool
forceOverwrite, []() {
        // success
}, [](int errorCode, std::string errorMsg, JsonString otherInfos) {
        // failure
});
```

```
(Lua)
PatiSDK.updateChannelSummary(level, jsonSummary, forceOverwrite, function()
        -- success
end, function(errorCode, errorMsg, otherInfos)
        -- errorCode   (number)
        -- errorMsg    (string)
        -- otherInfos  (table)
end)
```

로그인 시 계정의 각 채널별 정보를 제공하기 위한 API. 게임에서 UI에 표현할 최소한의 유저 정보를 플랫폼으로 전송한다.

level 값은 항상 제공해야 하며 양의 정수만 가능하다. 만약 게임에 level이란 개념이 없다고 해도 게임의 전체적인 달성도를 기준으로 적당한 값을 제공할 것을 권장한다.

jsonSummary는 level을 제외하고 채널선택 단계에서 유저에게 보여주거나 필요한 간단한 데이터(예를 들어 캐릭터 코스춤)를 JSON형식의 문자열로 전달해야 한다. (루아의 경우 JSON형식의 테이블을 그대로 넘겨도 상관없다) 전송할 데이터에 플랫폼에 이미 존재하는 key가 있다면 value가 갱신이 되고 없다면 기존 value가 유지된다.

- EX)

기존 플랫폼 summary : '{"nick":"파티게임즈", "class":"beginner"}'

전송한 summary : '{"nick":"플랫폼개발팀", "gender":"male"}'

다음 로그인 시 : '{"nick":"플랫폼개발팀", "class":"beginner", "gender":"male"}'

이 summary는 게임 데이터를 저장하기 위한 기능이 아니므로 로그인 화면에서 꼭 필요한 정보만을 업로드해야 하며 저장할 summary의 전체 길이가 JSON 문자열을 기준으로 1024를 넘어갈 수 없다.

forceOverwrite은 기존 summary를 삭제하고 현재 제공할 summary 완전히 덮어쓸 때 사용한다.

계정전환

계정 전환 예제

```
(C++)
PatiSDK::switchingAnotherAccount(PatiAuthInfo().setKakao(), [](int uid, int
timestamp, std::string authToken, JsonString otherInfos) {
        // success
}, [](int errorCode, std::string errorMsg, JsonString otherInfos) {
        // failure
});
```

```
(Lua)
PatiSDK.switchingAnotherAccount(PatiAuthInfo.setKakao(), function()
        -- uid         (number)
        -- timestamp   (number)
        -- authToken   (string)
        -- otherInfos  (table)
end, function(errorCode, errorMsg, otherInfos)
        -- errorCode   (number)
        -- errorMsg    (string)
        -- otherInfos  (table)
end)
```

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.

게임 로그아웃

로그아웃 예제

```
(C++)
PatiSDK::logout([]() {
        // success
}, [](int errorCode, std::string errorMsg, JsonString otherInfos) {
        // failure
});
```

```
(Lua)
PatiSDK.logout(function()
        -- success
end, function(errorCode, errorMsg, otherInfos)
        -- errorCode   (number)
        -- errorMsg    (string)
        -- otherInfos  (table)
end)
```

게임에서 로그아웃 할 때 호출한다. 게스트에서는 호출이 불가능하다. 성공하면 자동 로그인, 푸시 토큰정보가 삭제되며, 카카오나 페이스북 로그인이 되있는 경우 해당 플랫폼의 로그아웃이 호출된다.

로컬 로그인 정보 제거

로컬 로그인 정보 제거 예제

```
(C++) PatiSDK::clearLocalLoginInfo();
```

```
(Lua) PatiSDK.clearLocalLoginInfo()
```

로그인 상태에선 실행되지 않으며 이 때는 false가 리턴된다.

로그인 정보가 꼬이는 문제가 발생하거나 잘못된 정보가 남아있을 때 제거하기 위한 함수이다. 게스트를 포함한 로컬에 저장되어있는 로그인 정보가 제거되므로 주의해서 사용해야만 한다.

탈퇴

탈퇴 예제

```
(C++)
PatiSDK::unregister("passwd", []() {
        // success
}, [](int errorCode, std::string errorMsg, JsonString otherInfos) {
        // failure
});
```

```
(Lua)
PatiSDK.unregister('passwd', function()
        -- success
end, function(errorCode, errorMsg, otherInfos)
        -- errorCode   (number)
        -- errorMsg    (string)
        -- otherInfos  (table)
end)
```

파티 회원을 탈퇴할 때 호출하며, 탈퇴시 7일의 유예기간이 생긴다. PatiFriends 회원으로 로그인한 경우에는 패스워드 입력을 필요하며, 그 외의 플랫폼은 비워두어도 상관없다. 또, 게스트일때는 불가능하다.

탈퇴 철회 및 로그인

탈퇴 철회/로그인 예제

(C++)

```
PatiSDK::loginCancelUnregister(PatiAuthInfo().setKakao(), [](int uid, int
timestamp, std::string authToken, JsonString otherInfos) {
        // login success
}, [](int errorCode, std::string errorMsg, JsonString otherInfos) {
        // login failure
});
```

(Lua)

```
PatiSDK.loginCancelUnregister(PatiAuthInfo.setKakao(), function(uid,
timestamp, authToken, otherInfos)
        -- uid         (number)
        -- timestamp   (number)
        -- authToken   (string)
        -- otherInfos  (table)
end, function(errorCode, errorMsg, otherInfos)
        -- errorCode   (number)
        -- errorMsg    (string)
        -- otherInfos  (table)
end)
```

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 로그인을 해야한다.)

게임 친구 - 목록 얻기

게임 친구 목록 얻기 예제

(C++)

```
PatiSDK::getGameFriends([](JsonString friends) {
        // success
}, [](int errorCode, std::string errorMsg, JsonString otherInfos) {
        // failure
});
```

(Lua)

```
PatiSDK.getGameFriends(function(friends)
        -- friends    (table)
end, function(errorCode, errorMsg, otherInfos)
        -- errorCode  (number)
        -- errorMsg   (string)
        -- otherInfos (table)
end)
```

게임 친구 목록을 받아 온다. 콜백으로 반환되는 friends는 JSONArray형식의 string이며, 그 형식은 아래와 같다.

friends 예시

(JSON)

```
[
        { "uid" : 10000001, "rel" : 1, "profile_url" : "http://xxxxx" },
        { "uid" : 10000002, "rel" : 2, "profile_url" : "http://yyyyy" },
        { "uid" : 10000003, "rel" : 3, "profile_url" : "http://zzzzz" },
]
```

(배열 안의 각 오브젝트의 Key/Value는 각각 게임 uid/해당 유저와의 관계이다.)

관계는 3가지 종류가 있다. 1은 현재 로그인한 유저가 다른 유저에게 친구 요청을 보내온 상태(팔로잉) 2는 반대로 다른 유저가 현재 유저에게 친구 요청을 보내온 상태(팔로워). 3은 서로 요청을 보낸 상태이다.(서로 친구)

profile_url의 경우 플랫폼 프로필 사진을 사용하는 경우에만 포함되는 값이며 url형식이다.

게임 친구 - 요청

게임 친구 요청 예제

```
(C++)
PatiSDK::requestGameFriend(friendUserID, [](int relation) {
        // success
}, [](int errorCode, std::string errorMsg, JsonString otherInfos) {
        // failure
});
```

```
(Lua)
PatiSDK.requestGameFriend(friendUserID, function(relation)
        -- relation    (number)
end, function(errorCode, errorMsg, otherInfos)
        -- errorCode   (number)
        -- errorMsg    (string)
        -- otherInfos  (table)
end)
```

특정 게임 유저에게 친구 요청을 한다. 상대방이 이미 친구 요청을 보내온 상태라면 서로 친구가 되어 3이 그렇지 않다면 1이 callback을 통해 반환되게 된다.

게임 친구 - 취소

게임 친구 취소 예제

```
(C++)
PatiSDK::cancelGameFriend(friendUserID, [](int relation) {
        // success
}, [](int errorCode, std::string errorMsg, JsonString otherInfos) {
        // failure
});
```

```
(Lua)
PatiSDK.cancelGameFriend(friendUserID, function(relation)
        -- relation    (number)
end, function(errorCode, errorMsg, otherInfos)
        -- errorCode   (number)
        -- errorMsg    (string)
        -- otherInfos  (table)
end)
```

특정 게임 유저와 친구 관계를 취소한다. 상대방과 서로 친구인 경우 상대방의 친구 관계만 남게 되어 2가 아닐 경우 관계가 완전히 소멸하여 0이 반환된다.

게스트 - 회원 연동

게스트 회원 연동 예제

```
(C++)
PatiSDK::addAuth(PatiAuthInfo().setKakao(), []() {
	// success
}, [](int errorCode, std::string errorMsg, JsonString otherInfos) {
	// failure
});
PatiSDK::addAuthWithLogoutOption(PatiAuthInfo().setKakao(), true, []() {
	// success
}, [](int errorCode, std::string errorMsg, JsonString otherInfos) {
	// failure
});
```

```
(Lua)
PatiSDK.addAuth(PatiAuthInfo.setKakao(), function()
	-- success
end, function(errorCode, errorMsg, otherInfos)
	-- errorCode  (number)
	-- errorMsg   (string)
	-- otherInfos (table)
end)
PatiSDK.addAuthWithLogoutOption(PatiAuthInfo.setKakao(), true, function()
	-- success
end, function(errorCode, errorMsg, otherInfos)
	-- errorCode  (number)
	-- errorMsg   (string)
	-- otherInfos (table)
end)
```

로그인한 게임 계정에 플랫폼 회원정보를 연동한다. 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. addAuth가 실패하는 경우 기본적으로 로그인한 플랫폼 계정은 그대로 유지되나 addAuthWithLogoutOption을 이용하는 경우 실패할 때 로그아웃을 자동으로 수행할 수 있다.

게스트 - 삭제

```
(C++) bool removeOK = PatiSDK::removeGuestInfoPermanently();
```

```
(Lua) local removeOK = PatiSDK.removeGuestInfoPermanently()
```

게스트 정보를 삭제한다. 게스트로 로그인한 상태에서만 호출이 가능하며, 그 외의 경우에는 false를 반환하고 실패한다. 이 함수가 호출된 후에는 자동으로 로그아웃된다.

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.

게스트 – 계정 이전/복구

```
(C++) const char* guestAuthData = PatiSDK::getLoggedInGuestAuthData();
```

```
(Lua) local guestAuthData = PatiSDK.getLoggedInGuestAuthData()
```

로그인된 게스트 계정을 이전하기 위한 인증 데이터를 가져온다. 인증 데이터는 19 자리의 숫자/영문 알파벳(대소문자를 구분하지 않지만, 위 함수에서는 대문자로 반환한다.)의 조합으로 주어지며, 게스트 로그인에 필요한 데이터를 포함하고 있다. 계정 이전의 경우 이 값을 그대로 유저에게 제공하면 되며, 이를 위한 UI 가 필요하다. **또한, 폰트에 따라 영문자와 숫자가 구별하기 어려울수 있으므로 해당 UI 에 한해 글자가 쉽게 구별가능한 폰트를 사용할것을 권장한다.**

(게스트 인증 데이터는 백오피스 고객센터 페이지에서 해당 계정을 조회해서 얻을 수도 있다.)*

게스트 이전/복구 예시

```
(C++)
PatiSDK::login(false, PatiAuthInfo().setGuest("guestauthdata"), [](int uid, int timestamp, std::string authToken, JsonString otherInfos) {
        // login success
}, [](int errorCode, std::string errorMsg, JsonString otherInfos) {
        // login failure
});
```

```
(Lua)
PatiSDK.login(false, PatiAuthInfo.setGuest('guestauthdata'), function(uid, timestamp, authToken, otherInfos)
        -- uid         (number)
        -- timestamp   (number)
        -- authToken   (string)
        -- otherInfos  (table)
end, function(errorCode, errorMsg, otherInfos)
        -- errorCode   (number)
        -- errorMsg    (string)
        -- otherInfos  (table)
end)
```

유저에게 인증 데이터를 입력받아 게스트 로그인을 시도하려면 PatiAuthInfo.setGuest 를 호출할 때 입력받은 인증 데이터를 넘겨준뒤 수동 로그인을 시도하면 되며, 이를 위한 게스트 인증 데이터를 받아 로그인하기 위한 UI 가 별도로 필요하다.

게스트 인증 데이터가 있으면 해당 인증 데이터로만 접속을 시도하며, 문제가 발생할 경우 일반 게스트 로그인과는 다르게 계정을 새로 만들지 않고 오류를 반환한다. 만약 입력받은 인증 데이터의 길이가 잘못되면 -722 오류가, 인증 데이터에 해당되는 계정이 없으면 -142 오류가 발생한다. 자동 로그인시에는 입력받은 게스트 인증 데이터를 사용하지 않는다.

계정 복구의 경우 문의를 통해 게스트 인증 데이터 문자열을 유저에게 알려주면 해당 유저가 게스트 인증 데이터를 사용해 로그인할 수 있도록 유도하기 위한 안내가 필요하다.

파티 회원 가입

파티 회원 가입 예제

(C++)

```
PatiSDK::signupPati(PatiAuthInfo().setPati("email", "passwd", "nick"),
[](int uid, int timestamp, std::string authToken, JsonString otherInfos) {
        // success
}, [](int errorCode, std::string errorMsg, JsonString otherInfos) {
        // failure
});
```

(Lua)

```
PatiSDK.signupPati(PatiAuthInfo.setPati('email', 'passwd', 'nick'),
function(uid, timestamp, authToken, otherInfos)
        -- uid         (number)
        -- timestamp   (number)
        -- authToken   (string)
        -- otherInfos  (table)
end, function(errorCode, errorMsg, otherInfos)
        -- errorCode   (number)
        -- errorMsg    (string)
        -- otherInfos  (table)
end)
```

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).

파티 회원 비밀번호 찾기

파티 회원 비밀번호 찾기 예제

(C++)
```
PatiSDK::forgotPassword("email", []() {
        // success
}, [](int errorCode, std::string errorMsg, JsonString otherInfos) {
        // failure
});
```

(Lua)
```
PatiSDK.forgotPassword('email', function()
        -- success
end, function(errorCode, errorMsg, otherInfos)
        -- errorCode   (number)
        -- errorMsg           (string)
        -- otherInfos  (table)
end)
```

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. 안내에 메일이 없을 경우 스팸메일을 확인하도록 추가하면 좋다.

파티 회원 비밀번호 변경

파티 회원 비밀번호 변경 예제

(C++)
```
PatiSDK::changePassword("curPassword", "newPassword", []() {
        // success
}, [](int errorCode, std::string errorMsg, JsonString otherInfos) {
        // failure
});
```

(Lua)
```
PatiSDK.changePassword('curPassword', 'newPassword', function()
        -- success
end, function(errorCode, errorMsg, otherInfos)
        -- errorCode   (number)
        -- errorMsg           (string)
        -- otherInfos  (table)
end)
```

파티 회원의 비밀번호를 변경한다. 파티 회원으로 로그인한 상태에서만 가능하다.

Kakao

SDK 설정

AndroidManifest.xml에 아래의 퍼미션을 등록한다.

```
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission    android:name="android.permission.WRITE_EXTERNAL_STORAGE"
/>
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
```

그리고 카카오톡에서 게임의 설치 여부를 확인할 수 있도록 Custom Scheme을 intent-filter로 설정한다. Custom Scheme을 추가하지 않으면 카카오톡 게임하기와, 게임 메시지의 앱으로 연결을 눌러도 게임이 실행되지 않는다. intent-filter는 <application> 태그 안에 아래 태그를 추가한다.

```
<intent-filter>
    <data android:scheme="kakao%{KakaoClientID}" />
    <action android:name="android.intent.action.VIEW" />
    <category android:name="android.intent.category.BROWSABLE" />
    <category android:name="android.intent.category.DEFAULT" />
</intent-filter>
```

또한, Activity의 onCreate내의 PatiSDK 초기화 코드에서 initSDK가 호출된 후,

Pati.getInstance().initKakao(clientID, clientSecret);를 실행해야 한다.

initKakao 예제

```
(Java)
Pati.createInstance(this);
if (Pati.getInstance().initSDK() == true)
{
        Pati.getInstance().initKakao("%{KakaoClientID}",
"%{KakaoClientSecret}");
        Pati.getInstance().setTestMode();
        Pati.getInstance().gameLaunched();
}
```

(* ClientID, ClientSecret은 카카오에서 지급받은 값을 사용한다.)

친구 목록 얻기

카카오 친구 목록 가져오기 예제

(C++)

```
PatiSDK::getKakaoFriends([](JsonString appFriends, JsonString
nonAppFriends) {
        // success
}, [](int errorCode, std::string errorMsg, JsonString otherInfos) {
        // failure
});
```

(Lua)

```
PatiSDK.getKakaoFriends(function(appFriends, nonAppFriends)
        -- appFriends          (table)
        -- nonAppFriends       (table)
end, function(errorCode, errorMsg, otherInfos)
        -- errorCode           (number)
        -- errorMsg            (string)
        -- otherInfos          (table)
end)
```

카카오 친구목록을 가져온다. appFriends는 다음과 같은 JSON형식이다. (Lua에서는 JSON을 자동으로 테이블로 적절히 변환해서 제공하지만 구조와 값의 타입은 똑같다.) nonAppFriends는 appFriends와 포맷이 비슷하지만 게임에 가입되어있지 않기 때문에 game_user_id가 비어있다.

appFriends 예시

(JSON)

```
[
    {
        “user_id”: 88000000000000000,
        “nickname”:””,
        “friend_nickname”:””,
        “profile_image_url”:””,
        “message_blocked”:true/false,
        “hashed_talk_user_id”:””,
        “game_user_id”:0,
        “pati_profile_url”:””
    }
]
```

user_id : 카카오톡 UID

nickname / friend_nickname : 카카오톡 닉네임 / 설정된 친구 닉네임

profile_image_url / pati_profile_url : 카카오톡 프로필 URL / 파티 프로필 URL

message_blocked : 카카오 메시지 블럭 여부

game_user_id : 해당 유저의 게임 UID

메세지 보내기 API

카카오 이미지 포함된 메시지 보내기 예제

(C++)

```
PatiSDK::sendKakaoLinkMessage("receiver", "templateId", metainfo []() {
        // success
}, [](int errorCode, std::string errorMsg, JsonString otherInfos) {
        // failure
});
```

(Lua)

```
PatiSDK.sendKakaoLinkMessage("receiver", "templateId", metainfo, function()
        -- no parameter
end, function(errorCode, errorMsg, otherInfos)
        -- errorCode   (number)
        -- errorMsg    (string)
        -- otherInfos  (table)
end)
```

receiver : 메시지를 받을 유저의 카카오톡 id

templateId : 미리 설정된 메시지 템플릿 id

metainfo : 메시지 내용을 구성하는데 필요한 데이터. JSON-Object형식이 되야하며, 내부 형식은 아래와 같아야 한다.

ex)

```
{
    "imagePath" : "",
    "executeurl" : "item=0001&count=1",
    "tag" : {
        "message1" : "Hello!",
        "message2" : "Test",
        ...
     }
}
```

imagePath : 이 값이 있으면 해당 Local경로에 있는 이미지를 decode해 image 태그로 전달한다.

executeurl : 이 값이 있으면 executeurl 태그에 해당 값을 전달한다.

tag : 그 외에 템플릿에서 사용할 태그 값을 등록한다.

친구에게 카카오톡 메시지를 보낼 때 사용한다. 카카오 어드민에서 미리 템플릿을 생성하고 그 id를 사용해야 한다. metainfo의 key-value는 예약된 태그(image, imageUrl, imageWidth, imageHeight, executeurl)에도 동일하게 적용가능하다.

Lua에서 metainfo에는 JSON형태와 동일하게 테이블을 만든뒤 전달하면 된다. JSON 문자열을 직접 생성해 전달해도 된다.

Facebook

SDK 설정

페이스 북 개발자 페이지 (https://developers.facebook.com/apps)

1. 앱 생성 (Create a New App)
2. Android Platform을 추가하고 Package Name, Class Name, Key Hashes를 등록.
   Key Hashes는 keystore파일을 이용하여 java keytool로 구할 수 있음.

```
keytool -exportcert -alias androiddebugkey -keystore {KEYSTORE_PATH} |
openssl sha1 -binary | openssl base64
```

Android Project

res/values/strings.xml에 아래 값을 추가한다.

```
<string name="Facebook_App_ID">%{FacebookAppID}</string>
```

그리고 AndroidManifest.xml의 <application>태그 안에 아래 태그를 추가한다.

```
<activity android:name="com.facebook.LoginActivity" />
<meta-data android:name="com.facebook.sdk.ApplicationId"
android:value="@string/Facebook_App_ID" />
```

또한, Activity의 onCreate내의 PatiSDK 초기화 코드에서 initSDK가 호출된 후,

Pati.getInstance().initFacebook(savedInstanceState);를 실행해야 한다.

initFacebook 예제

```
(Java)
Pati.createInstance(this);
if (Pati.getInstance().initSDK() == true)
{
        Pati.getInstance().initFacebook(savedInstanceState);
        //Pati.getInstance().setTestMode();
        Pati.getInstance().gameLaunched();
}
```

(*) APP_ID는 페이스 북 개발자 페이지에서 확인 가능하다.

(https://developers.facebook.com/apps)

친구 목록 얻기

페이스북 친구 목록 가져오기 예제

(C++)

```
PatiSDK::getFacebookFriends([](JsonString appFriends, JsonString
nonAppFriends) {
        // success
}, [](int errorCode, std::string errorMsg, JsonString otherInfos) {
        // failure
});
```

(Lua)

```
PatiSDK.getFacebookFriends(function(appFriends, nonAppFriends)
        -- appFriends          (table)
        -- nonAppFriends       (table)
end, function(errorCode, errorMsg, otherInfos)
        -- errorCode           (number)
        -- errorMsg            (string)
        -- otherInfos          (table)
end)
```

페이스북 친구목록을 가져온다. appFriends는 다음과 같은 JSON형식이다. (Lua에서는 JSON을 자동으로 테이블로 적절히 변환해서 제공하지만 구조와 값의 타입은 똑같다.)

appFriends 예시

(JSON)

```
{
        "friend_facebook_id_1" : {
                "fbname" : "가나다"
                "fbpic" : "http://aaaa"
                "game_user_id" : 10000001
        },
        "friend_facebook_id_2" : {
                "fbname" : "라마바"
                "fbpic" : "http://ffff"
                "game_user_id" : 10000002
        },
        …
}
```

(friend_facebook_id_1, friend_facebook_id_2는 친구의 페이스북 UID이다.)

fbname은 친구의 페이스북 이름, fbpic은 프로필 사진 경로, game_user_id는 친구의 게임 uid이다. nonAppFriends는 appFriends와 포맷을 따르지만 게임에 가입되어있지 않기 때문에 game_user_id가 비어있다.

요청 보내기

페이스북 요청 보내기 다이얼로그 띄우기 예제

(C++)
```
PatiSDK::sendFacebookRequestWithRequestDialog(params, "message", "title",
[](std::string requestId) {
        // success
}, [](int errorCode, std::string errorMsg, JsonString otherInfos) {
        // failure
});
```

(Lua)
```
PatiSDK.sendFacebookRequestWithRequestDialog(params, "message", "title",
function(request_id)
        -- request_id (string)
end, function(errorCode, errorMsg, otherInfos)
        -- errorCode  (number)
        -- errorMsg   (string)
        -- otherInfos (table)
end)
```

페이스북 친구에게 요청을 보내는 다이얼로그를 띄운다. params에 JSON string형태로 관련 정보를 추가할 수 있다. (Lua의 경우, params를 직접 JSON으로 인코딩해서 string 형태로 전달하거나, 그냥 테이블채로 전달하는 것 둘 다 가능하며, 후자의 경우는 SDK가 알아서 JSON으로 인코딩해준다.)

params에 들어갈 수 있는 값은 공식 페이스북 SDK 문서 (https://developers.facebook.com/docs/games/requests/v2.0)를 참고할 것.

뉴스피드 게시

페이스북 요청 보내기 다이얼로그 띄우기 예제

(C++)
```
PatiSDK::postFacebookFeedWithFeedDialog(params, [](std::string postId) {
        // success
}, [](int errorCode, std::string errorMsg, JsonString otherInfos) {
        // failure
});
```

(Lua)
```
PatiSDK.postFacebookFeedWithFeedDialog(params, function(post_id)
        -- post_id    (string)
end, function(errorCode, errorMsg, otherInfos)
        -- errorCode  (number)
        -- errorMsg   (string)
        -- otherInfos (table)
end)
```

자신의 페이스북 담벼락에 글을 게시한다. 요청 보내기와 마찬가지로 params는 JSON형태로 관련 정보를 추가할 수 있으며, 전달하는 방법도 동일하다.

Params에 들어갈 수 있는 값은 공식 페이스북 SDK 문서 (https://developers.facebook.com/docs/sharing/reference/feed-dialog/v2.0)를 참고할 것.

권한 확인 및 재요청

권한 확인 예제

(C++)

```
PatiSDK::isFacebookGrantedPermission(permissionName) // return boolean
PatiSDK::getFacebookPermissions( [](JsonString permissions) {
        // success
}, [](int errorCode, std::string errorMsg, JsonString otherInfos) {
        // failure
});
```

(Lua)

```
PatiSDK.isFacebookGrantedPermission(permssionName) -- return boolean
PatiSDK.getFacebookPermissions(function(permissions)
        -- permissions(table)
end, function(errorCode, errorMsg, otherInfos)
        -- errorCode   (number)
        -- errorMsg    (string)
        -- otherInfos  (table)
end)
```

isFacebookGrantedPermission으로 개별 권한의 허락 여부를 getFacebookPermissions으로는 앱이 가입 시 요구하는 권한 전체의 허락 여부를 확인할 수 있다.

권한 재요청 예제

(C++)

```
PatiSDK::reAskFacebookDeclinedPermission(permssionName, []() {
        // success
}, [](int errorCode, std::string errorMsg, JsonString otherInfos) {
        // failure
});
```

(Lua)

```
PatiSDK.reAskFacebookDeclinedPermission(permssionName, function()

end, function(errorCode, errorMsg, otherInfos)
        -- errorCode   (number)
        -- errorMsg    (string)
        -- otherInfos  (table)
end)
```

가입 시 거부했던 권한이나 가입시점에는 approved가 아니었던 권한을 다시 요청할 수 있다. 페이스북 앱 대시보드에서 허락되지 않은 권한을 요청할 수는 없다.

Google+

SDK 설정

1. 작업을 하기 전에, 아래 명령어를 터미널에서 입력한뒤, 출력되는 SHA1 fingerprint 해시 값을 파티게임즈에 전달해야 한다.

```
keytool -exportcert -alias androiddebugkey -keystore <path-to-debug-or-production-keystore> -list -v
```

(path-to-debug-or-production-keystore에는 개발용 혹은 배포용 keystore 정보를 입력하면 되며, 개발(디버그)용 keystore는 윈도우의 경우 "Users/%사용자%/.android/debug.keystore", Mac이나 리눅스의 경우 "~/.android/debug.keystore"에 있는 것을 사용하면 된다. debug.keystore의 패스워드는 기본적으로 'android'이다.)

2. 자바 코드에서 initSDK가 호출된 후에, 다음 코드를 호출한다.

```
(Java)
Pati.getInstance().initGooglePlus();

// GoogleApiClient Build시 추가로 설정해야할 API나 Scope가 있다면, 아래와 같은 방법을 사용할 수 있다.
Pati.getInstance().initGooglePlus(new BuildGoogleApiClient() {
        @Override
        public void onBuild(Builder builder) {
                // example
                // builder.addApi(Games.API);
                // builder.addScope(Games.SCOPE_GAMES);
        }
});
```

(그 외 연동관련 상세한 내용은 https://developers.google.com/+/mobile/android/getting-started 참조)*

GoogleApiClient

SDK기능을 사용하면 자동으로 내부에 GoogleApiClient가 생성되며, 이 객체를 가지고 로그인을 시도하도록 되어있다.

PatiSDK의 인증정보를 가지고 다른 구글 서비스 (구글 게임 등)을 이용하려면 SDK의 GoogleApiClient를 이용해야 하는데, 이 객체를 가져오는 방법은 다음과 같다.

GoogleApiClient 가져오기

```
(Java)
Pati.getInstance().getGoogleApiClient();
```

(현재는 SDK에서 구글 관련 서비스(게임 업적 등)를 직접적으로 지원하지 않으며, 필요하다면 게임쪽에서 직접 구현해야한다.)*

Google Play가 설치되지 않은 기기에서의 처리

일부 기기의 경우 Google play가 설치되어 있지 않거나 아예 설치할 수 없는 경우가 있다. (대표적으로 중국 내수용 기기) SDK에서 해당 기기가 아예 Google Play가 설치가 불가능한지 알 수 있는 방법이 없으므로 Google Play가 설치되어 있지 않은 경우에는 에러가 반환된다. (errorCode:-723 OtherInfos["errorCode"]:PATI-1) 따라서 게임에서 해당 에러가 발생하는 경우 유저에게 Google Play를 설치해보길 권장하거나 다른 인증수단으로 로그인할 수 있도록 유도해야 한다.

Naver

SDK 설정

참고 문서 : https://nid.naver.com/devcenter/docs.nhn?menu=Android

AndroidManifest.xml에 아래의 퍼미션을 등록한다.

```
<uses-permission android:name="android.permission.INTERNET"/>
```

그리고 네이버 로그인에서 사용하는 아래의 Activity들을 <application> 태그 안에 추가한다.

```
<activity android:name="com.nhn.android.naverlogin.ui.OAuthLoginActivity"
        android:theme="@android:style/Theme.Translucent.NoTitleBar"/>
<activity
android:name="com.nhn.android.naverlogin.ui.OAuthLoginInAppBrowserActivity"
        android:label="OAuth2.0 In-app"/>
```

또한, Activity의 onCreate내의 PatiSDK 초기화 코드에서 initSDK가 호출된 후, Pati.getInstance().initNaver(clientID, clientSecret, appName, callbackIntent);를 실행해야 한다.

initNaver 예제

```
(Java)
Pati.createInstance(this);
if (Pati.getInstance().initSDK() == true)
{
        Pati.getInstance().initNaver("%{NaverClientID}",
"%{NaverClientSecret}",          "%{NaverAppName}",
"%{NaverCallbackIntent}");
        Pati.getInstance().setTestMode();
        Pati.getInstance().gameLaunched();
}
```

%{NaverAppName}: 네이버 앱의 로그인 화면에 표시될 이름. 모바일 웹으로 로그인 하는 경우에는 서버에 저장된 앱 이름이 표시된다.

%{CallbackIntent}: 네이버 로그인 이용 신청시 입력한 intent값. 실제 intent가 호출되지는 않는다.

로그인된 계정의 플랫폼 ID 조회

로그인된 계정의 플랫폼 ID 조회 예제

(C++)

```
auto platformInfo = PatiSDK::getLoggedInPlatformId();
std::string pati = platformInfo.patifriendsId;
std::string kakao = platformInfo.kakaoId;
std::string facebook = platformInfo.facebookId;
std::string googleplus = platformInfo.googleplusId;
std::string naver = platformInfo.naverId;
```

(Lua)

```
local platformId = PatiSDK.getLoggedInPlatformId()
local pati = platformId.patifriendsId            -- (string)
local kakao = platformId.kakaoId          -- (string)
local facebook = platformId.facebookId           -- (string)
local googleplus = platformId.googleplusId-- (string)
local naver = platformId.naverId          -- (string)
```

현재 로그인된 계정에 연동되어있는 각 플랫폼들의 ID를 조회한다. 이 함수를 사용해 유저에게 연동되어있는 플랫폼 ID를 노출시키거나, AddAuth사용시 추가로 연동가능한 플랫폼들의 정보를 알 수 있다.

만약 연동되어있지 않은 플랫폼인 경우, 빈 문자열이나 "0" 문자열이 반환되므로 이를 통해 연동 여부를 알아낼 수도 있다.

한 계정에 여러 플랫폼을 연동할 때 처리

한 계정에 여러 플랫폼을 동시에 연동하는 경우, 맨 처음 로그인할때에는 마지막으로 사용한 플랫폼에 대해서만 로그인이 진행된다.

즉, 해당 계정에 여러 플랫폼이 연동되어있더라도 자동 로그인을 실행할 때 사용한 플랫폼만 로그인이 진행되므로, 그 외 플랫폼들의 서비스를 사용하기 위해서는 별도의 절차가 필요하다.

파티SDK에서 지원하는 플랫폼 API들(페이스북 친구 조회 등)은 해당 플랫폼이 로그인되지 않은 경우 해당 플랫폼으로 로그인을 진행한뒤, 원래 로그인되어있던 계정의 연동정보와 비교해 다를 경우 해당 플랫폼의 로그아웃 처리와 함께 오류를 발생하도록 구현되어 있다.

외부에서 구글 플레이 서비스 관련 기능을 사용하거나, 파티SDK에서 지원하지 않는 플랫폼의 API를 사용하고자 하는 경우, 반드시 API를 사용해서 파티SDK 내부에서 처리하는것과 동일한 검증을 진행한 뒤에 사용해야 한다.

현재 로그인된 계정에 연동된 플랫폼 로그인

(C++)

```
PatiSDK::loginPlatformWithLoggedInAuth(PatiSDK::FACEBOOK, [](){
    // success
}, [](int errorCode, std::string errorMsg, PatiSDK::JsonString jsonStr){
    // error
});
```

(Lua)

```
PatiSDK.loginPlatformWithLoggedInAuth(PatiAuthType.FACEBOOK, function()
        -- success
end, function(errorCode, errorMsg, otherInfos)
        -- failure
end)
```

이 API는 파티SDK 내부에서 검증을 위해 사용하는 API와 동일하며, 이미 로그인이 성공해 사용가능한 플랫폼에 대해서는 항상 success를 호출하도록 되어있다. 계정에 연동된 플랫폼ID와 로그인한 플랫폼ID가 다를경우 -725 오류를 발생하며, 다른 이유로 로그인에 실패할 경우 일반 로그인과 동일한 오류를 발생시킨다.

또한, 로그아웃, 탈퇴 처리시 여러 플랫폼이 연동되어있는 경우, 현재 활성화 되어있는 플랫폼 서비스들에 대해서만 로그아웃 및 탈퇴가 진행된다. 이 때문에 탈퇴의 경우 유저가 탈퇴를 진행해 성공하더라도 연동은 되어있지만 탈퇴 시점에 비활성화되어있던 (아직 로그인이 되지 않은) 서비스들에 대해서는 탈퇴가 진행되지 않는다.

이 부분은 해당 플랫폼에서 비활성화하는 기능이 있으므로 그것에 맡기고 그냥 무시하거나, 탈퇴를 진행하려면 연동되어있는 모든 서비스를 활성화 해야 하도록 처리할 수도 있다.

이외에 파티SDK에서는 디버깅 및 각종 로직 처리를 위해 현재 어떤 플랫폼 서비스가 활성화되어있는지 조회할 수 있는 API를 제공한다.

현재 사용가능한 플랫폼 조회
(C++) `PatiSDK::isEnablePlatformService(PatiSDK::GOOGLEPLUS);`
(Lua) `PatiSDK.isEnablePlatformService(PatiAuthType.KAKAO) -- boolean`

인자로 조회를 원하는 인증타입을 입력하면 현재 해당 플랫폼 서비스가 활성화 되어있을 경우 true가 반환된다. 로그인된 계정에 어떤 플랫폼이 연동되어있는지는 **"로그인된 계정의 플랫폼 ID 조회"** 항목을 통해 확인할 수 있다.

약관 가져오기 / 처리

약관 가져오기 예제

```
(C++)
PatiSDK::checkTerms([](std::string termsOfService, std::string
termsOfServiceLink, std::string privacyPolicy, std::string
privacyPolicyLink) {
        // success
}, [](int errorCode, std::string errorMsg, JsonString otherInfos) {
        // failure
});
```

```
(Lua)
PatiSDK.checkTerms(function(termsOfService, termsOfServiceLink,
privacyPolicy, privacyPolicyLink)
        -- termsOfService     (string)
        -- termsOfServiceLink(string)
        -- privacyPolicy      (string)
        -- privacyPolicyLink  (string)
end, function(errorCode, errorMsg, otherInfos)
        -- errorCode   (number)
        -- errorMsg    (string)
        -- otherInfos  (table)
end)
```

* termsOfService : 이용약관 화면 표시 문구
* privacyPolicy : 개인정보 취급 방침
* termsOfServiceLink, privacyPolicy: 해당하는 내용 전문 보기 링크

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

checkTerms는 캐시되어있는 약관을 가져오거나, 서버에 약관 정보를 새로 요청해 받아와서 지정한 콜백에 제공해준다. 서버에 마지막으로 요청한지 3일이 지났거나, 캐시된 약관이 없을때만 서버에서 새로운 약관을 받아온다.

```
(C++) bool termsAgreed = PatiSDK::isTermsAgreed ();
```

```
(Lua) local termsAgreed = PatiSDK.isTermsAgreed()
```

isTermsAgreed()함수로 약관에 동의했는지의 여부를 알 수 있다. 만약 약관 정보가 새로 갱신되어 동의를 다시 받아야 하거나 약관에 동의하지 않은 경우는 false가 된다.

```
(C++) bool needCheckTerms = !PatiSDK::agreeTerms();
```

```
(Lua) local needCheckTerms = !PatiSDK.agreeTerms()
```

약관에 동의할 때는 agreeTerms 함수를 호출한다. 이미 동의를 했거나, 현재 로컬에 캐시된 약관이 없거나, 오래되어 새로 갱신이 필요한 경우에는 false를 반환하며, CheckTerms를 호출할 필요가 있다.

(C++) PatiSDK::removeTermsPermanently();
(Lua) PatiSDK.removeTermsPermanently()

테스트나 기타 목적을 위해 로컬에 캐시된 약관을 삭제해야되는 경우, `removeTermsPermanently()`를 사용해 캐시된 약관 정보와 동의 여부를 삭제해 서버에서 약관을 다시 내려받을 수 있다.

+ 로그아웃/탈퇴시에는 자동으로 약관 동의가 해제되므로 다시 동의를 받는 과정이 필요하다.

프로필 이미지

SDK 를 통해 자신의 프로필 이미지나 페이스북/카카오톡등에서 제공하는 프로필 이미지 주소를 플랫폼에 등록한다. 외부 플랫폼에서 제공하는 프로필 이미지 주소는 로그인 함수를 호출할 때 SDK 내부에서 수집해서 등록한다. 등록한 프로필 이미지는 url 의 형태로 로그인 / 친구 목록 얻기 등을 통해 제공된다. 외부 플랫폼의 이미지 주소가 아니고 플랫폼에서 관리하는 이미지 주소의 경우 이미지 주소에는 아래와 같은 규칙을 가지고 있다.

http://xxxxxx/gameid/uid_size_timestamp.jpg

기본적으로 size 는 업로드시에 클라이언트에서 정한 오리지널 size 를 내려주지만 플랫폼에서 몇 몇 크기의 thumbnail 을 생성해두기에 이 size 를 변경하면 오리지널 size 보다 작은 size 의 이미지를 가져오는게 가능하다. (thumbnail 제공 사이즈 : 1024, 256, 128)

프로필 이미지 업로드 (uploadProfileInGallery, uploadProfileByTakeshot)

(C++)

```
PatiSDK::uploadProfileInGallery
PatiSDK::uploadProfileByTakeshot (int width, int height, [](std::string url,
std::string filePath) {
        // success
},[](int errorCode, std::string errorMsg, JsonString otherInfos) {
        // failure
})
```

(Lua)

```
PatiSDK.uploadProfileInGallery
PatiSDK.uploadProfileByTakeshot (width--[[number]], height--[[number]],
function(url, filePath)
        -- url          (string)
        -- filePath     (string)
end, function(errorCode, errorMsg, otherInfos)
        -- errorCode    (number)
        -- errorMsg     (string)
        -- ohterInfos   (table)
end)
```

자신의 갤러리 혹은 카메라로 촬영한 image를 함수 호출 시에 지정한 width, height으로 편집하여 플랫폼에 등록한다. 플랫폼 등록에 성공하면 앞으로 이미지를 불러올 수 있는 url과 편집한 이미지가 저장된 filepath가 반환된다. 기본적으로 편집한 이미지 파일은 항상 정해진 이름(profile.jpg)으로 고정되어 있고 cache directory에 저장하므로 이 파일의 용량에 대해 크게 걱정할 필요는 없다.

프로필 이미지 삭제

(C++)
```
PatiSDK::rollbackProfileImage([](std::string url) {
        // success
}, [](int errorCode, std::string errorMsg, JsonString otherInfos) {
        // failure
});
```

(Lua)
```
PatiSDK.rollbackProfileImage(function(url)
        -- url (string)
end, function(errorCode, errorMsg, otherInfos)
        -- errorCode  (number)
        -- errorMsg   (string)
        -- ohterInfos (table)
end)
```

직접 등록한 프로필 이미지를 삭제하고 default 프로필 이미지로 복구한다. (카카오톡/페이스북등의 외부 플랫폼인 경우 외부 플랫폼의 이미지 주소)

결제 공통

SDK를 통해 여러 마켓의 결제를 공통으로 관리할 수 있다. 현재 지원하는 마켓은 구글 플레이, 애플, SK T Store, Naver Store이다.

현재는 결제 시 플랫폼 화폐(cash) 1종류만 구매 가능하다. 아이템을 직접 구매하거나, 캐시와 골드를 둘 다 직접 구매하는 것은 불가능하다. 캐시(1차 화폐)를 구매한 뒤 캐시로 아이템이나 골드(2차 화폐)를 구매해야 한다.

cash는 내부적으로 paid, bonus, free로 분리된다. Free는 게임 시스템에서 지급하는 경우이고, paid, bonus는 결제를 통해서만 지급된다.

	Paid	Bonus	Free
환불에 포함	o	X	X
유료캐시 전용 컨텐츠	o	o	X
사용 우선 순위	1	2	3

paid/bonus는 큰 금액을 구매할 경우 보너스를 얹어주는 것과 이후 환불 문제 때문에 분리되어 있다. 1천원에 10캐시이고, 1만원에는 130캐시라고 가정하자. 기본 단가는 1캐시에 100원이다. 1만원 구매의 경우는 paid=100, bonus=30인 셈이다. 어떤 유저가 1만원 구매하고 30캐시를 사용하면 잔액은 paid=70, bonus=30이 된다. 이 때 환불을 요청하면 1만원 x 100 / 130이 아니라, paid=70 x 100원 = 7천원만 환불이 가능하다. bonus는 환불 시 같이 회수된다.

캐시 조회

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

(C++)
```
PatiSDK::getCashBalance([](int paid, int bonus, int free) {
});
```

(Lua)
```
PatiSDK.getCashBalance(function(paid, bonus, free)
        -- paid         (int)
        -- bonus        (int)
        -- free         (int)
end)
```

결제 결과 callback함수 등록

```
(C++)
PatiSDK::setPurchaseCashCallback([](bool hasSuccess, std::string result) {
	// result	(JsonString)
});
```

```
(Lua)
PatiSDK.setPurchaseCashCallback(function(hasSuccess, result)
	-- hasSuccess (bool)
	-- result	(table)
end)
```

결제 결과를 전달받을 콜백을 등록한다. 콜백을 등록하면 이전 결제시 결제 자체는 성공했지만 캐시를 지급하지 못한 케이스에 대한 복구시도를 한다. 따라서 이 콜백의 등록 시점은 반드시 로그인 후여야 하며 유저에게 결제 결과를 안내할 수 있는 준비가 되어있어야 한다. 혹여 비정상적으로 복구해야 할 결제가 여러 건 쌓여있는 상황에는 한 건씩 처리되어 콜백 함수가 호출된다. 결제 관련 정보가 담기는 result의 경우에는 cpp에서는 json형태의 string으로 전달되지만 루아에서는 테이블 형태로 전달된다.

[* 결제 성공 시 Result의 구성 (cpp의 경우 json으로 파싱할 것. lua에서는 이미 테이블 형태) *]

- paid, bonus, free: 유저의 현재 paid, bonus, free cash량
- paid_diff, bonus_diff, free_diff : 결제 결과 충전된 paid, bonus, free cash량
- product_name: 결제한 상품 이름
- product_id : 결제한 상품 아이디

[* 결제 실패 시 *]

- code : 결제 실패 에러코드
- msg : 결제 실패 메시지

결제 상품 조회

```
(C++)
PatiSDK::getCashProducts([](std::string products) {
	// products	(JsonString)
}, [](int errorCode, std::string errorMsg, std::string otherInfos) {
	// otherInfos (JsonString)
});
```

```
(Lua)
PatiSDK.getCashProducts(function(products)
       -- products   (table)
end, function(errorCode, errorMsg, otherInfos)
       -- errorCode  (int)
       -- errorMsg   (string)
       -- otherInfos (table)
end)
```

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

[* 목록 조회 성공 시 products 구성 *]

products 예시

```
(JSON)
{
       "SN_1" : {
              "product_id" : "cash_1",
              "name" : "22 캐시",
              "paid" : 22,
              "bonus" : 0,
              "price" : "$1.99",
              "priceInt" : 1990,
              "localprice" : "2200원"
       },
       "SN_2" : {
              ……
       }
}
```

- SN : 플랫폼에 등록된 상품 번호. 상품 구매 시 넘겨줘야 하는 key 값이다.
- product_id : 결제 상품의 마켓에 등록된 실제 상품 ID(위의 예시에서는 cash_1)
- name : 상품의 이름
- paid : 결제로 충전될 유료 cash량
- bonus : 결제로 충전될 보너스 cash량
- price : 결제 상품의 가격. 문자열. 기본값 (ex: $1.99, 2200원)
- priceInt : 결제 상품의 가격을 숫자로 표현. 달러의 경우 1000을 곱한 값
- localprice : 게임의 언어설정(지정되어 있지 않다면 기기 설정에 맞춤)에 맞춘 결제 상품 가격. 해당 지역의 가격이 등록된 경우에만 포함되어 있다.
- 그 외 : 월정액, 패키지, 기간한정이벤트 등의 정보가 함께 포함되어 내려온다.

결제 시도

```
(C++)
PatiSDK::purchaseCashProduct(productid, [](int errorCode, std::string errorMsg,
std::string otherInfos) {
	// otherInfos (JsonString)
});
```

```
(Lua)
PatiSDK.purchaseCashProduct(productid, function(errorCode, errorMsg,
otherInfos)
	-- errorCode  (int)
	-- errorMsg   (string)
	-- otherInfos (table)
end)
```

결제 상품 조회를 통해 받아온 상품아이디로 결제를 시도한다. 이 함수는 결제 결과 callback이 등록되어 있어야만 호출이 가능하다. 실제로 결제 결과는 대부분 위에 설명한 결제 결과 callback 함수로 돌아오나 방금 전과 같이 callback자체가 등록 안되어 있는 등의 몇몇 케이스를 위해 결제 시도 함수에도 callback이 필요하다.

결제 공통 에러코드

- -705 : login required. 로그인 되어 있지 않음
- -709 : user cancel purchase item. 결제를 시도 중에 취소하는 경우
- -712 : purchase callback must be settled. 결제 결과 callback이 등록되지 않음
- -713 : can't found market. payment unavailable. market이름이 잘못되었음

구글 결제

```
<uses-permission android:name="com.android.vending.BILLING"/>
```

manifest에 위의 권한을 추가하고 함께 포함된 library project Googlebilling을 library로 추가한다. 또는 IInAppBillingService.aidl을 패키지 이름이 com.android.vending.billing.IInAppBillingService이 되도록 추가한다. 마켓의 이름은 GOOG가 되어야 initSDK에서 구글결제 모듈이 초기화되고 결제 시도시 자동으로 구글 결제가 호출된다. 마켓 이름의 세팅은 AndroidManifest.xml의 com.patigames.api.marketData 부분을 참고하면된다.

구글 결제 에러 코드

- 3 : BILLING_RESPONSE_RESULT_BILLING_UNAVAILABLE
- 4 : BILLING_RESPONSE_RESULT_ITEM_UNAVAILABLE
- 5 : BILLING_RESPONSE_RESULT_DEVELOPER_ERROR
- 6 : BILLING_RESPONSE_RESULT_ERROR
- 7 : BILLING_RESPONSE_RESULT_ITEM_ALREADY_OWNED
- 8 : BILLING_RESPONSE_RESULT_ITEM_NOT_OWNED

이 중 3, 7의 케이스는 유저에게 적극적으로 에러를 알려줘야 할 필요성이 있다. 3의 케이스는 실제로 구글 결제 서비스 자체가 연결 불가일 수 도 있지만 유저가 기기에서 구글 계정을 삭제하는 경우에도 발생한다. 유저가 제대로 구글 계정을 등록했는지 확인시켜 줄 필요가 있다. 만약 계정이 등록되어 있지 않아서 생긴 에러라면 계정을 등록한 후에 게임을 재 시작해야 결제를 진행할 수 있다. 7의 경우는 실제로는 발생하지 않아야 한다. 결제 콜백을 등록하는 시점에 이전에 진행한 결제 중 미 처리된 결제들을 복구하는 과정이 들어있기 때문이다. 어떠한 이유로 해당 에러가 발생하는 경우에는 계속해서 결제 복구가 실패하고 있다는 것이므로 유저로 하여금 cs로 상황을 접수 할 수 있도록 유도해주는 것이 좋다.

네이버 결제

```
<meta-data android:name="com.patigames.api.nstorepublickey"
android:value="{YOUR_PUBLIC_KEY}"/>
```

네이버 앱스토어에 등록된 앱정보에서 확인할 수 있는 publicKey(서명 인증키)를 Manifest에 위와 같이 추가하고 마켓의 이름을 NStore로 지정한다. 또한 함께 포함된 naip-v2-sdk.jar파일을 프로젝트 라이브러리로 추가한다.

네이버 결제 에러 코드 (유저에게 알려줘야 하거나 알려주길 권장하는 에러코드)

네이버 결제중에 발생한 에러는 네이버의 에러코드를 동일하게 사용하며, NST***와 같이 앞에NST가 붙은 세자리의 숫자 형식으로 주어진다. 네이버 결제 에러코드 전체는 다음 링크를 참조할것.

(http://gamedev.naver.com/index.php/%EB%84%A4%EC%9D%B4%EB%B2%84_%EC%95%B1%EC%8A%A4%ED%86%A0%EC%96%B4_%EC%9D%B8%EC%95%B1%EA%B2%B0%EC%A0%9C_%EA%B0%80%EC%9D%B4%EB%93%9C/%EB%84%A4%EC%9D%B4%EB%B2%84_%EC%95%B1%EC%8A%A4%ED%86%A0%EC%96%B4_%EC%9D%B8%EC%95%B1%EA%B2%B0%EC%A0%9C_%EC%A0%81%EC%9A%A9_%EB%B0%A9%EB%B2%95#.EA.B5.AC.EB.A7.A4_.EA.B2.B0.EA.B3.BC_.EC.B2.98.EB.A6.AC_.EC.8B.9C_.EC.A3.BC.EC.9D.98.ED.95.A0_.EC.A0.90)

- NST002 : 구매자가 결제 데이터를 잘못 입력했거나 외부 결제 CP사에서 오류가 발생하여 결제가 실패한 경우. 앱스토어 앱에서 직접 사용자가 인지하도록 오류 메시지를 노출해 줍니다. 클라이언트에서는 간단하게 결제 실패를 알리는 팝업을 발생시키면 좋음

- NST008 : 네트워크 오류 발생.

- NST011 : 네이버 앱스토어에서 로그아웃하는 경우 발생. 유저에게 다시 로그인을 할 수 있도록 적절한 메시지를 표시해주길 권장.

티스토어 결제

```
<uses-permission android:name="android.permission.RECEIVE_SMS" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE"/>
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="android.permission.READ_PHONE_STATE"/>
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
```

```
<meta-data android:name="iap:api_version" android:value="2" />
<activity
       android:name="com.skplanet.dodo.IapWeb"
       android:configChanges="orientation|screenSize|locale|keyboardHidden|
layoutDirection"
       android:excludeFromRecents="true"
       android:windowSoftInputMode="stateHidden" >
</activity>
<meta-data android:name="com.patigames.api.tstoreappid"
android:value="{YOUR_APP_ID}"/>
<meta-data android:name="com.patigames.api.tstorepluginmode"
android:value="{DEVELOPMENT_OR_RELEASE}"/>
```

티스토어에 등록된 앱정보에서 확인 가능한 appid와 테스트 혹은 실제 결제를 결정한 pluginmode를 manifest에 추가하고 마켓의 이름을 TStore로 지정한다. 또한 함께 포함된 tstore-iap-plugin.jar / tstore-libdodo.so 파일을 프로젝트 라이브러리로 추가한다.

선물함

쿠폰이나 프로모션으로 받은 선물이 저장된 선물함 정보를 SDK에서 직접 받아올수 있다.

선물 지급기능은 서버에서 처리할것을 권장하지만 그게 불가능한 경우 API 설정을 변경해 SDK에서 사용할 수 있다. SDK에서 지원이 필요한 경우 파티게임즈에 문의한다. (서버 구현 관련 내용은 PATIGAMES Promotion System 문서 참조)

선물 목록 조회

선물 목록 조회 예제

```
(C++)
PatiSDK::getGifts([](JsonString gifts) {
        // getGifts success
}, [](int errorCode, std::string errorMsg, JsonString otherInfos) {
        // getGifts failure
});
```

```
(Lua)
PatiSDK.getGifts(function(gifts)
        -- gifts       (table)
end, function(errorCode, errorMsg, otherInfos)
        -- errorCode   (number)
        -- errorMsg    (string)
        -- otherInfos  (table)
end)
```

플랫폼에 저장된 선물 목록을 가져온다. 목록은 JSON Array형태로 전달되며, Array에는 보상 정보를 나타내는 JSON Object가 들어있다. 보상 정보를 나타내는 JSON Object는 아래와 같이 구성되어 있다.

sn(int) : 보상의 serial number. useGift나 서버 API의 /giftbox/use에 사용됨.
Expire(int) : 만료 시각. (UNIX TIMESTAMP)
t(string) : 지급되야할 아이템의 이름. (cash 등) 아이템은 백오피스에서 설정할 수 있다.
n(int) : 지급되야할 아이템의 수량.
m(string) : 보상 메시지

선물 지급 처리

선물 지급 처리 예제

(C++)

```
PatiSDK::consumeGift(giftSN, []() {
        // consumeGift success
}, [](int errorCode, std::string errorMsg, JsonString otherInfos) {
        // consumeGift failure
});
```

(Lua)

```
PatiSDK.consumeGift(giftSN, function()
        -- consumeGift success
end, function(errorCode, errorMsg, otherInfos)
        -- errorCode   (number)
        -- errorMsg    (string)
        -- otherInfos  (table)
end)
```

게임쪽에서 선물 지급을 완료한 뒤, 플랫폼에 선물이 지급됐음을 알리기 위한 함수이다. 이 함수는 서버 API의 /giftbox/use와 동일한 동작을 하지만, 서버에서 이 함수를 호출할 수 없는 특정한 경우를 위해 제공된다.

이 함수를 사용하려면 파티게임즈와 사전에 협의해야 사용 가능하며, 그 이외의 경우에서는 오류가 발생하므로 서버쪽에서 구현해야 한다.

쿠폰 사용

쿠폰 사용 예제

```
(C++)
PatiSDK::useCoupon(const char* coupon, []() {
        // useCoupon success
}, [](int errorCode, std::string errorMsg, JsonString otherInfos) {
        // useCoupon failure
});
```

```
(Lua)
PatiSDK.useCoupon(coupon, function()
        -- useCoupon success
end, function(errorCode, errorMsg, otherInfos)
        -- errorCode   (number)
        -- errorMsg    (string)
        -- otherInfos  (table)
end)
```

백오피스를 통해 미리 발급한 쿠폰을 사용한다. 쿠폰 문자열의 길이는 항상 12자로 고정이다. 대소문자를 구분하지 않으며, '-'가 포함된 경우에는 자동 삭제 처리한다. 쿠폰사용에 성공한 경우 설정한 아이템은 선물함으로 지급된다. 따라서 쿠폰 사용이 성공한 경우, 선물함을 다시 읽어올 필요가 있다.

게임 언어 설정 수집

```
(C++) PatiSDK::setGameLanguage(lang)
```

```
(Lua) PatiSDK.setGameLanguage(lang)
```

게임 언어설정을 플랫폼에서 수집해야 할 필요가 있는 경우, 이 함수를 호출한다. 이 함수는 로그인된 유저가 선택한 언어를 플랫폼으로 전송하며, 조건에 맞지 않으면 false를 리턴한다.

lang에는 영문자(대소문자는 무조건 대문자로 변환된다) 2글자로 이루어진 언어 코드를 입력해야 한다. **언어 코드는 반드시 ISO 639-1을 사용하며, 이외의 형식을 사용하는 것은 문제가 발생할 수 있다.** ISO 639-1에 대한 정보는 (http://en.wikipedia.org/wiki/List_of_ISO_639-1_codes) 참조.

이 함수는 로그인할 때, 언어 설정을 변경했을때마다 호출하는 것을 권장한다. SDK 내부에서 마지막으로 설정한 언어를 저장하며(로그아웃하면 저장된 값이 사라진다), 설정한 언어가 변하지 않았을 경우에는 따로 플랫폼으로 전송하지 않도록 되어있다.

외부 웹 브라우저 호출

(C++) PatiSDK::openWebBrowser(url)
(Lua) PatiSDK.openWebBrowser(url)

웹 브라우저를 호출해야할 필요가 생기면 이 함수를 호출한다.

마켓으로 redirect하는 등 외부 링크가 필요한 경우 이 함수를 사용한다

내장 웹뷰 호출

```
<activity android:name="com.patigames.api.WebViewActivity"
android:theme="@android:style/Theme.NoTitleBar" />
```

위의 activity 태그를 AndroidManifest.xml 의 <application> 태그 안에 추가해야 웹뷰를 사용할 수 있다. (고객센터 웹 페이지 호출 또한 웹뷰를 사용하므로 이 태그를 추가해야 한다.)

웹뷰를 생성하려면 아래의 함수를 호출한다.

(C++) PatiSDK::openWebView(url);
(Lua) PatiSDK.openWebView(url)

웹뷰를 조작할수 있는 자바스크립트 인터페이스를 제공하며, 사용하기 위해서는 pati.js 스크립트를 웹페이지에 포함시켜야 한다.

자바스크립트 인터페이스	기능
patijs.close()	웹뷰를 닫는다.
patijs.back()	이전 페이지로 이동한다. 이동할 페이지가 더 이상 없으면 창을 닫는다.
patijs.closeDontOpenForDay(day)	웹뷰를 닫고, 다음번에 해당 URL 로 다시 접속할경우, 지정한 기간만큼 웹뷰를 다시 열지 않는다.
patijs.link(url)	해당 URL 로 이동한다. 마켓의 URL 인경우 마켓으로 이동한다.
patijs.openWebBrowser(url)	웹브라우저를 연다.

(*) 안드로이드의 경우 페이지에 pati.js 를 포함하지 않아도 인터페이스를 사용가능하지만, iOS 에서 문제가 발생하므로 반드시 포함시켜야 한다.

고객센터 웹 페이지 호출

(C++) PatiSDK::openCSWebPage();
(Lua) PatiSDK.openCSWebPage()

내장된 웹뷰를 통해 파티게임즈 고객센터 페이지 화면을 띄운다.

배너 및 공지 호출

프로모션이나 공지 목적으로 사용되는 배너를 호출한다. 배너는 백오피스에서 설정 가능하며, SDK에서는 백오피스에서 설정한 '호출 위치(placement)'를 통해 설정한 배너를 표시할 수 있다. 게임 실행시 자동으로 호출가능한 리스트를 캐싱하게 되며, 주기적으로 갱신하게 된다. 테스트 모모드일 때는 원활한 테스트를 위해 10초주기로 갱신되게 되어있다.

플랫폼 서버에서도 배너 리스트를 주기적으로 캐싱해서 사용하므로, 백오피스에서 변경한 사항이 바로 적용되지 않을수도 있다.

```
(C++) PatiSDK::showNoticeView("placement", ignoreFrequency);
```

```
(Lua) PatiSDK.showNoticeView("placement", ignoreFrequency)
```

placement에는 미리 협의한 호출 위치를 지정하면 된다.
ignoreFrequency는 bool값으로, true를 넘기면 백오피스에서 설정한 출력 빈도 제한 (항상, 앱 실행시 한번 등)을 무시하고 무조건 출력한다.

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

에러 코드 목록

아래 에러 코드는 모든 API 공통이다. 일부 오류는 디버깅에 도움되는 정보를 포함하고 있다.

플랫폼 에러

-100 로그인이 필요한 API인데, 로그인하지 않은 상태

-101 잘못된 닉네임

-102 너무 짧은 닉네임

-103 너무 긴 닉네임

-104 회원 가입 email주소 중복 (파티자체회원)

-105 자동 로그인 실패

-106 존재하지 않는 유저

-107 로그인 패스워드 오류

-108 잘못된 email 형식

-109 잘못된 패스워드 형식

-110 잘못된 게스트 ID

-111 패스워드 변경 시, 새로 입력한 패스워드가 잘못된 형식

-112 잘못된 생년월일 형식

-122 이미 파티 회원 연동이 되어 있음

-123 다른 게임 계정에 연동된 회원

-124 세션 정보가 잘못 됨

-125 알 수 없는 인증 정보

-130 로그인 불가능한 계정. 로그인 불가능한 이유를 알 수 있는 정보가 onLoginFailure의 otherInfos 인자로 주어진다. 알 수 있는 정보는 아래와 같다.

- status : "unregistered" / "blocked" (탈퇴중인지 / 블록되었는지)
- message : "블록시 사유 메시지" (탈퇴한 경우엔 빈 문자열)
- timeleft : 3600 (sec) (접속 불가능한 이유가 얼마나 남았는지 시간.)

-133 쿠폰번호가 맞지 않음

-134 이미 사용한 쿠폰

-135 계정 사용 횟 수 제한

-136 만료된 쿠폰

-137 가입일자 설정에 맞지 않음

-138 쿠폰 보상 설정에 문제가 있음

-139 플랫폼 프로필 이미지를 사용하고 있지 않음. 플랫폼에서 해당 게임이 프로필 이미지를 사용하지 않는 게임으로 간주하고 있음

-140 프로필 이미지를 찾을 수 없음

-141 프로필 이미지를 아직 교체할 수 없음

- timeleft : 3600 (sec) (프로필 이미 교체가 가능해지기 위해 남은 시간)

-142 게스트 인증 데이터가 유효하지 않음.

-149 로그인하려는 계정의 인증정보와 일치하지 않음.

-153 게임의 버전 정보를 찾을 수 없음

-900 잘못된 HttpRequest

-901 HMAC-MD5 값 불일치

-902 필요 Parameter가 없음. 에러 메시지에 필요한 항목 노출

-999 서버에서 알 수 없는 오류 발생.

SDK 에러

-700 SDK 미 지원 기능

-701 게스트로 허용하지 않는 기능

-702 게스트 정보가 존재함 (게스트 계정을 삭제하거나 게스트 자동로그인을 해야함)

-703 API에 허락되지 않은 인증 타입

-704 이미 로그인 되어 있음

-705 로그인이 필요함

-706 수동 로그인시 로그인 정보가 null

-707 계정전환 시에 불러오는 계정이 게스트임

-708 create instance에서 페이스북을 사용하지 않도록 설정.

-709 유저가 결제 시도를 취소

-711 카카오SDK 내부 에러. otherInfos로 카카오에서 내려온 에러가 제공됨.

-712 결제 결과를 받을 callback이 등록되지 않음

-713 market이름이 잘 못 됨. 결제를 실행할 적절한 모듈을 찾지 못함.

-714 페이스북SDK 내부 에러. otherInfos로 페이스북에서 내려온 에러가 제공됨.

-715 페이스북SDK 내부 에러. RequestWithRequestDialog가 실패했을 때 반환됨.

-716 페이스북SDK 내부 에러. FeedWithFeedDialog가 실패했을 때 반환됨.

-718 같은 api가 이미 호출 중.

-719 잘못된 coupon번호.

-720 프로필 이미지 교체 중에 유저가 취소함.

-721 initSDK가 실패함.

-722 입력한 게스트 인증 데이터의 길이가 21글자가 아님.

-723 구글+ SDK 오류. OtherInfos에 좀 더 자세한 정보가 담겨있다.(code가 PATI로 시작하는 경우는 SDK에서 판단한 에러, GOOG로 시작하는 경우는 구글에서 반환하는 에러). 아래는 SDK에서 발생하는 에러 목록.

PATI-1 구글플레이가 설치되어 있지 않음.
PATI-2 구글플레이 로그인 시도 중
PATI-3 구글 플레이 로그인이 필요함
PATI-4 유저가 로그인을 취소 함
PATI-5 JSON error
PATI-6 인증 토큰을 가져오는데 실패함
PATI-7 구글 플레이와 연결을 끊는데 실패함.
PATI-999 알 수 없는 에러

-724 이미 해당 인증방식으로 연동되어있는 회원.

-725 현재 로그인된 계정에 연동된 정보와 다른 플랫폼 계정으로 로그인을 시도함

-726 계정 연동(addAuth)이 필요함

-728 자동 로그인 실패, 기기기반으로 찾은 인증 가능한 계정이 1 개 이상 있으니 계정을 선택하여 로그인해야 함

-729 잘못된 기기 기반 유저목록 idx 값

-730 로그인하려는 계정의 인증정보와 일치하지 않음.

-731 파티프렌즈로 인증한 계정이 아님.

-732 네이버 SDK 오류. (code가 PATI로 시작하는 경우는 SDK에서 판단한 에러, NAVR로 시작하는 경우는 네이버에서 반환하는 에러).

PATI-1 네이버 인스턴스가 초기화 되지 않음
PATI-2 현재 네이버 로그인이 이미 진행중
PATI-4 유저가 로그인을 취소함
PATI-5 JSON error
PATI-6 XML error
PATI-999 알 수 없는 에러

-733 해당 앱 버전을 더 이상 사용할 수 없음

-734 채널을 이미 선택함

-735 채널 선택이 필요함

-736 채널 선택이 실패함 (플랫폼 서버에서는 성공했는데 SDK내부에서 실패하는 경우. SDK 버그일 가능성이 높음. 플랫폼 팀으로 문의)

-800 JSON Exception

-801 IO Exception

-802 IllegalState Exception

-804 HttpRequestFail Exception

-805 InvalidKey Exception

-806 NoSuchAlgorithm Exception

-807 SSL 인증 실패 오류

개정 이력

1.4.1

- 안드로이드 SDK 안정성 강화
- iOS에서 메인 파티SDK 라이브러리와 각 플랫폼(카카오, 페이스북 등)별 라이브러리로 분리됨. (업데이트시 기존에 설치된 iOS 라이브러리는 제거할 것. iOS 연동에 변경사항이 있으므로 확인할 것.)

1.4.0

- requestGameFriend, cancelGameFriend의 uid를 long long으로 변경
- 루아 error handler를 따로 설정하지 않아도 PatiSDK 로그에서 간단히 출력되도록 수정.
- 구글 결제 시 상품 정보가 캐싱되던 버그 수정
- 구글 결제 팝업을 띄워놓고 백그라운드로 내려가면 다시 포그라운드로 올라올 때 앱이 종료되던 버그 수정

1.3.23

- 푸시 아이콘을 AndroidManifest에서 별도로 설정할수 있는 기능 추가.
- 카카오 탈퇴 버그 수정

1.3.22

- 간편 로그인 삭제.
- 자동 로그인 시도 및 기기 기반 계정 목록을 이용한 로그인 추가.

1.3.21

- 페이스북 피드 다이얼로그에서 에러 발생하는 버그 수정.
- 구글플러스 로그인 다이얼로그에서 입력 취소를 하는 경우 에러코드가 일관되지 않은 버그 수정
- loginPlatformWithLoggedInAuth 호출 시 항상 로그인을 시도하도록 변경
- addAuth에서 실패할 때 로그아웃을 할지 옵션을 줄 수 있는 addAuthWithLogoutOption를 추가.

1.3.20

- 간편 로그인 추가.

1.3.19

- 안드로이드에서 ADID 수집시 google_play_services 라이브러리가 필요없도록 수정.
- 페이스북 게임 친구가 최대 25명까지만 출력되던 버그 수정
- setTestMode 호출시 토스트 알람 출력되도록 수정.
- iOS의 경우, iCloud에 게스트 계정 정보를 불러올 수 있도록 수정.

1.3.18

- 여러 개의 플랫폼이 한 계정에 연동되있는 경우, 로그인한 플랫폼 타입과 다른 연동된 플랫폼의 서비스를 사용하기위한 기능 추가.
- 현재 사용할수 있는 플랫폼 서비스를 조회하는 기능 추가.
- 로그아웃/탈퇴시 현재 사용가능한 모든 플랫폼에서 동일하게 로그아웃/탈퇴를 처리하도록 수

정.
- 안드로이드 구글+ 로그아웃시 마지막으로 로그인한 계정 정보가 로컬에서 완전히 삭제되지 않던 버그 수정
- 안드로이드 웹뷰 전체화면으로 출력할 때 orientation이 정상적이지 않던 문제 수정

1.3.17
- 1.3.12 이후로 푸시 설정 disable할 경우 로그인이 풀리던 문제 수정
- enablePushService를 로그인 하기 전에 임의로 호출할 경우 이후에 로그인하더라도 푸시 토큰이 전송되지 않는 버그 수정
- 1.1.6 이후로 iOS에서 상품조회가 실패하는 버그 수정

1.3.16
- 1.1.6 이후로 특정환경에서 구글플러스 로그인에 생긴 버그 수정
- initGooglePlus시 custom api/scope를 위한 BuildGoogleApiClient가 노출되지 않던 버그 수정

1.3.15
- iOS에서 웹뷰가 너무 작게 보이던 버그 수정
- 1.3.12 이후로 계정전환 시 -702 에러가 리턴되는 버그 수정
- 트래픽 유입 추적관련 Install Referrer 로직 개선

1.3.14
- InstallTrackingReceiver가 제대로 동작하지 않던 버그 수정
- 구글 플러스로 연동하거나 로그인한 경우 isEnableAutoLogin이 항상 false를 반환하던 버그 수정

1.3.13
- 웹뷰가 안드로이드 lollipop(5.0)이상에서 정상적으로 출력되지 않던 버그 수정.

1.3.12
(* 내부의 버전 관리와 배포 버전을 동일하게 맞추기 위해 버전 체계가 바뀌었습니다. 착오없으시길 바랍니다.)
- 공지 기능 개선 및 버그 수정
- 세션이 종종 잘못된 값을 가져 -127 오류가 발생하던 버그 수정
- 게스트 로그인 정보가 남아 있는 상태에서 다른 플랫폼으로 수동 로그인할 때 발생하던 버그 수정
- 유니티 Desktop 플러그인이 제대로 갱신되지 않았던 버그 수정
- 유니티 안드로이드/iOS에서 선물함 조회기능이 제대로 동작하지 않았던 버그 수정

1.1.6
- 개인 정보 수집 여부 설정(UsingPrivacyInfo) 추가
- 일부 윈도우 환경에서 SDK 오류 수정

- 결제 다양화 지원을 위해 캐쉬관련 API 변경.

1.1.5

- 구글 플러스 연동 기능 추가
- 구글 ADID 수집 (별도의 API 적용 필요 없음)
- SDK 버전 정보 수집 (별도의 API 적용 필요 없음)

1.1.4

- 게스트 이전 기능 추가.
- 선물함 기능 추가.
- 배너 호출 기능 추가.

1.1.3

- 프로필 이미지 기능 제공 (기존 SDK를 사용하는 경우 활성화를 위한 협의가 필요)
- 푸시 수신 설정 함수 변경
- 페이스북 API에 현재 유저의 권한 상태 확인 및 재요청 기능 추가

1.1.2

- PatiFriends 안드로이드 자동 로그인이 불가능한 문제 수정
- 유니티 Desktop 플러그인의 위치가 x86에서 Plugins로 변경됨.

1.1.1

- 안드로이드 플레이 스토어 결제 lolipop(Android L) 대응
- 네이버 결제 API v2로 교체
- Unity Desktop 환경에서 실행시 patisdk.dll 오류 수정