Developer Guide

GameStick SDK

---

19 April 2013 Version 1.0.3

Table of Contents

Section 1 Introduction

The GameStick SDK provides a set of APIs which allow your game to make use of the PlayJam Online Services.

The following three steps can get you publishing games on the GameStick:

1. You can develop games for GameStick using various different development environments that can compile to Android, such as Unity, C++ (via Marmalade or the Android NDK) or Java.
2. Register and download the GameStick SDK from the Developer Site. Versions of the SDK are available for Unity, C++ and Java. Use the API calls in the SDK to add a range of services and functionality like controller support, billing, and social aspects such as leader-boards and achievements.
3. Compile your application to Android and submit it through the PlayJam Publishing Portal for distribution to GameStick.

About the PlayJam Platform

The PlayJam Platform actually consists of two things:

- **The PlayJam Publishing Portal**

  This is a web-based management application used by developers to register themselves with PlayJam, upload games onto the servers, and manage and configure the PlayJam on-line services used by their games.

  Note: at the time of publication, you can't create your own accounts and data. In the meantime, you can either contact us to ask us to add something, or use the following:

  Java — name your package `com.playjam.servicesamples` and your main activity `com.playjam.servicesamples.MainActivity`

  C++ — name your package `com.playjam.cppsample` and your main activity `com.playjam.cppsample.Main`

  Unity — name your package `com.playjam.gamestick` and your main activity `com.unity3d.player.UnityPlayerProxyActivity`

  You should then be able to make API calls and at least get some test data back for achievements, etc.

- **The GameStick SDK**

  The SDK is the core component of the PlayJam platform. It provides a simple set of APIs which you can call from Unity, Marmalade, C++ or Java in order to access the on-line services provided by PlayJam.

What to read next

We recommend that you tackle the information provided here in the following order:

1. Review the SDK Overview for a summary of what is provided and how it is structured.
2. Make sure you understand the Standardization and UI Guidelines which you need to be aware of when designing and coding your game.
3. Check the relevant parts of the Developer Setup Guide section for how to set up before you can start building against the SDK with various platforms.
4. Refer to the relevant API Reference section for details of the APIs you can call to make requests of the services provided by the SDK, and what you need to implement in order to handle the responses.

The rest of this document is concerned with the GameStick SDK. For more details of the the PlayJam Publishing Portal and what you need to do to get your game submitted and accepted, see the separate documentation. You'll need to understand some details of what items (achievements, purchasable items, etc.) you set up via the PlayJam Publishing Portal when writing your code to use the SDK.

Note that this site is a work in progress and we expect to add further documentation as the SDK develops, so check back here regularly.

1.1 SDK Overview

The GameStick SDK provides access to the on-line services provided by the PlayJam Online Services.

This is actually implemented on the GameStick by a service running on the stick that responds to the API calls you make, and connects to the PlayJam web server and handles all the requests. As a developer, you do not need to know anything about how the web services actually work, or how to establish a web connection - you simply make calls to the API and everything is handled for you.

Supported on-line services

The SDK currently supports the following back-end PlayJam Online Services:

Leader boards module:

Enables players to register high scores by game or game bundle. Customizable by country, or region, by device or by time.

Save state module:

Allows a player to save the score and level of a game and resume playing later. Data is stored on the PlayJam Cloud Storage servers.

Awards and achievements module:

Allows the developer to give any awards or achievements associated with the game. These accomplishments are stored in the player's profile and can be viewed in game. Awards and achievements are defined and managed in the PlayJam Developer Application.

Analytics module:

Automatically save out data to the PlayJam servers about any feature or event from your games which you wish to analyze. Events can be defined and managed in the PlayJam Developer Application.

Billing / purchasing module:

Transaction engine for all payments. We support credit and debit card payments globally as well as mobile payment solutions in many markets. Provides for almost any type of game or in-game transaction and links to virtual currency system. Used for in-game micro transactions, pay per play, subscription, etc.

The structure of the GameStick SDK components

The diagrams below (Illustration 1 and Illustration 2) shows the structure of the GameStick SDK components. Illustration 1 on the left shows the structure for Java applications. Illustration 2 on the right shows how C++ applications are structured:

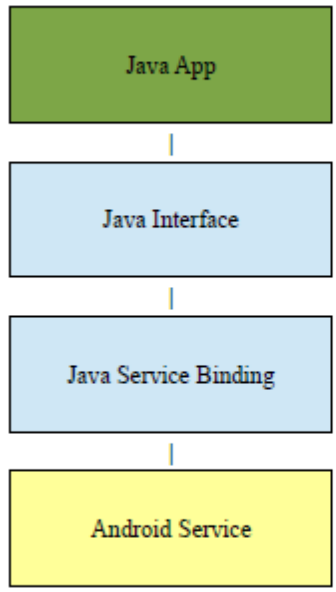


Illustration 1: SDK for Java Apps

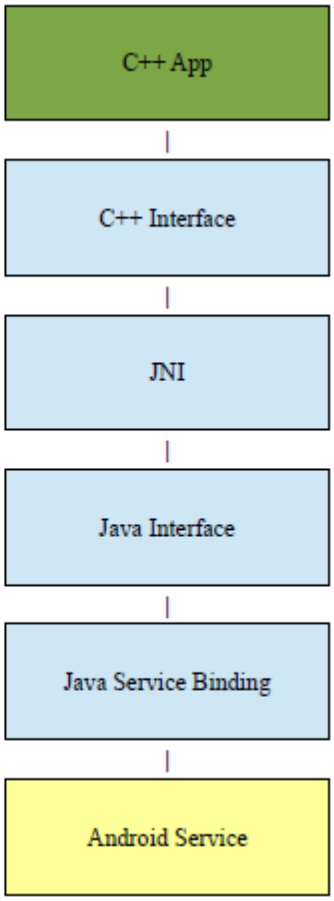
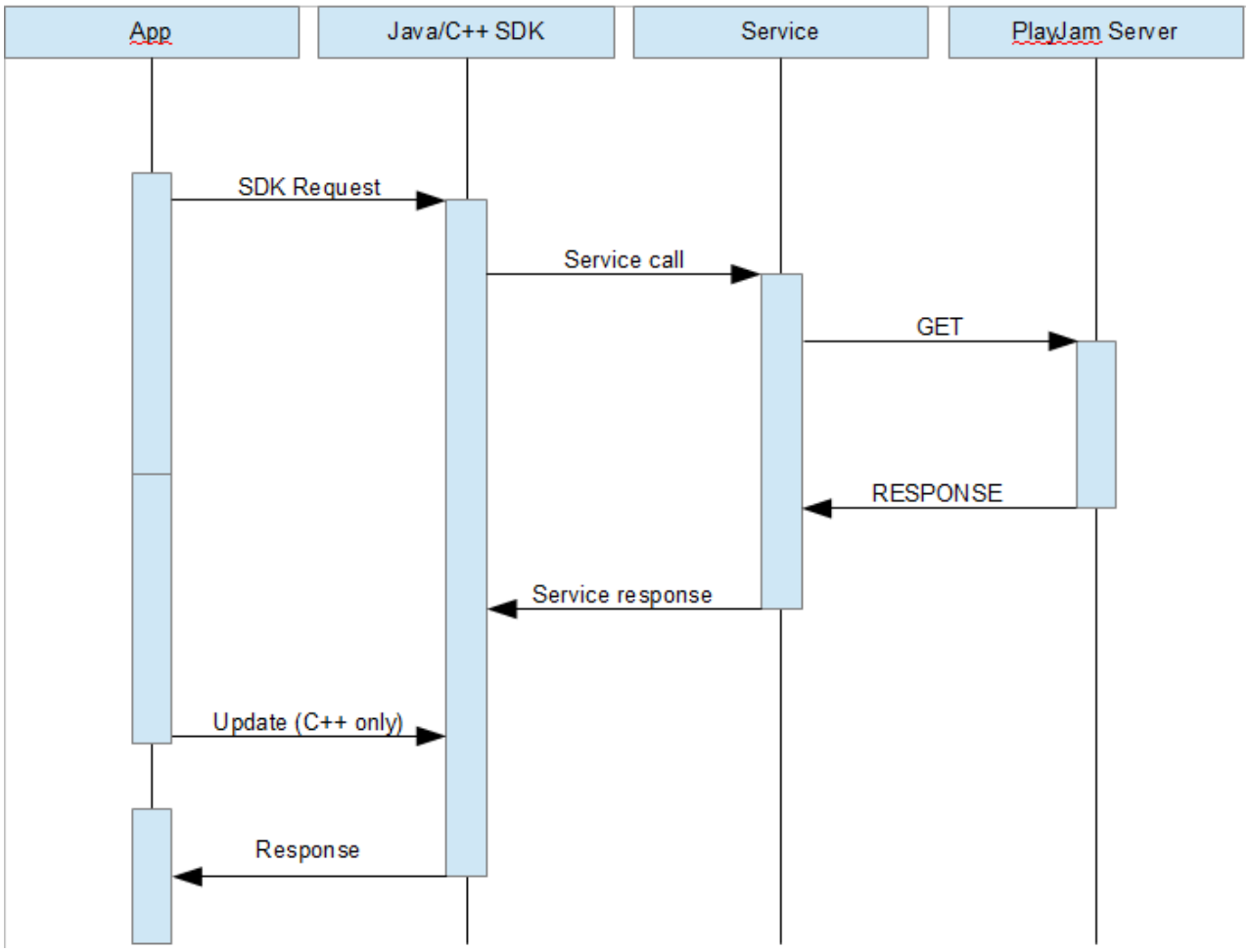


Illustration 2: SDK for C++ / Marmalade Apps

The items in green represent your application code. The items in blue represent PlayJam libraries that will be built into the final application, and the items in yellow represent Android services that are either part of the OS or provided by PlayJam. The C++ set-up involves a C++ wrapper which we provide and use of JNI (the Java Native Interface) to allow this to talk to Java. The C++ API we provide simply maps onto the Java version of the API without you having to perform this wrapping yourself. The Java version of the GameStick API will then make the relevant calls to the service running on the GameStick, which listens for these calls and accesses the PlayJam Online Services via the internet.

Note: the situation for Unity is slightly different in that the Unity version of the SDK download provides C# scripts which wrap around the Java interface, so from your Unity project you only need to call functions from these scripts, but the principle is similar.

The general procedure for calling a service is shown in the diagram below.



For each request, there is always a status result indicating the result of the request dispatch. There is also an optional response containing the requested data. This is in the form of a Messenger object which the service uses to post data back to the calling application. The Java Service Binding object passes this data to the Java Interface object which then parses the data for consumption by either the Java App or a C++ application accessed via the JNI interface. The only slight difference for C++ users is that they must issue an Update request to the SDK at intervals to prompt the service to respond (this is automatic with Java).

For those readers who don't find such diagrams helpful, don't worry. The main thing to remember is that you will make an API call to the service running on the stick which will generate a response — your code will receive these responses via some response handler class that you need to implement (the response methods called will either be one which returns the data you requested, or if there was an error, an error handler will be called). All you need to do is implement these response handlers as described in this document and then add your code to do whatever you wish to do with the data. You may like to think of the response handlers as delegates or call-backs that you need to provide and which the service will call.

Everything else is hidden from you, so your code doesn't need to worry about getting an internet connection to the server, or anything of that nature — this is taken care of by the SDK service running on the stick.

What you need to provide

As well as making calls to the API functions themselves, you will have to register delegate methods or call-back functions with the API that will handle the data returned from the web service. Classes to do this are provided in the SDK, so you just need to implement these classes in your code. Example code is provided to

show how this works. C++ users will also need to call an Update() API to ask the service to send any pending messages back to your application at regular intervals.

Note that currently, only the Java and C++ implementations of the SDK provide any ability to parse the data returned by the service. If you are using Unity, you will need to parse the JSON data yourself (we've provided some information about the format returned in the Unity reference).

Section 2
GameStick Standardization and UI Guidelines

Before submitting your game, you need to ensure you have complied with the following guidelines:

- General guidelines
- Controller guidelines
- On-line services SDK usage guidelines
- Adapting from mobile / tablet

We've also provided the following collateral for you:

- GameStick controller image files (for use in your UI)

2.1 General Guidelines

Please observe the following guidelines in your game:

Game

- The game must support full screen mode at 16:9 aspect ratio. Game resolution must be at least 960x540 and at most 1920x1080. Other resolutions supported in the 16:9 aspect ratio are 1024×576, 1280×720, 1366×768, 1600×900.
- Graphics Quality – If the game supports multiple graphics mode, the game must default to the best quality mode that suits the GameStick v1.
- Controller support – If the game supports multiple control schemes, it *must* default to support the GameStick controller.
- The game should maintain a frame rate of 25 FPS on average.

Menus

- Roll over states are required in menus to highlight selected option.

- Menu items should be browsable using the D-pad or Joystick.
- When a menu that contains options is loaded, a reticle/border should surround one option by default.
- Volume Controls are required in the settings or options menu.
- An Exit confirmation menu is required before quitting the game .
- Menus should include instructions on navigating using the controller, e.g.: “A - Select” “B - Exit”.

Links

- Web views to render web pages or linking to a browser are not supported. All links will be disabled hence should be visually removed from the game.
- Advertisements – Only ads that link to other games in the GameStick Store will be supported.
- More Games Links – Links to games on non GameStick stores must be replaced with links to the GameStick store instead.
- Social Media Links (Facebook / Google+ / Twitter) – URLs to open Social Media pages on a browser will not open a browser window. However, Social APIs including Facebook Connect can be used.

Tutorials

- Tutorials for game controls should reflect the use of the controller. For example, use 'A TO CONTINUE'.

2.2 On-line Services SDK Usage Guidelines

To access the on-line services you need to set up game data (purchasable items, achievements, and so on) using the PlayJam Publishing Portal, and then use the SDK to access the services in your game.

We recommend that you use the following services via the SDK:

- **In-game purchase**

  All in-game purchase is facilitated by our in-app billing API. All billing must be through this API. See the In-App Purchasing Guidelines topic or contact us for details.

- **Achievements**

  It is recommended to use the GameStick Achievements API to leverage the GameStick Achievements UI system. Please refer to SDK documentation or contact us for details.

- **Save State Module**

  Allows a player to save the score and level of a game and resume playing later. Data is stored on the PlayJam Cloud Storage servers.

See the SDK Overview (and the API reference if necessary) for more details.

2.2.1 In-App Purchasing Guidelines

The GameStick SDK provides you with the facility to provide in-app purchasing, so that players can buy additional levels, upgrades and so on, and generate additional revenue for you.

You will need to use the The PlayJam Publishing Portal to create the purchasable items for your game (each item will be assigned an id number that you can use), specify the cost in each supported territory / currency, and add a name and description. Once those details are on our system, you can make calls to the SDK from your game to query the items available, what the player has already purchased, etc.

The SDK provides the following request APIs:

- GetItemsForPurchase — Get a list of all available purchasable items for this game.
- GetPurchasedItems — Get a list of the items that the player has purchased.
- PurchaseItem — Purchase the specified item.
- GetPurchasedItemURL — Obtain the URL relating to the specified item.

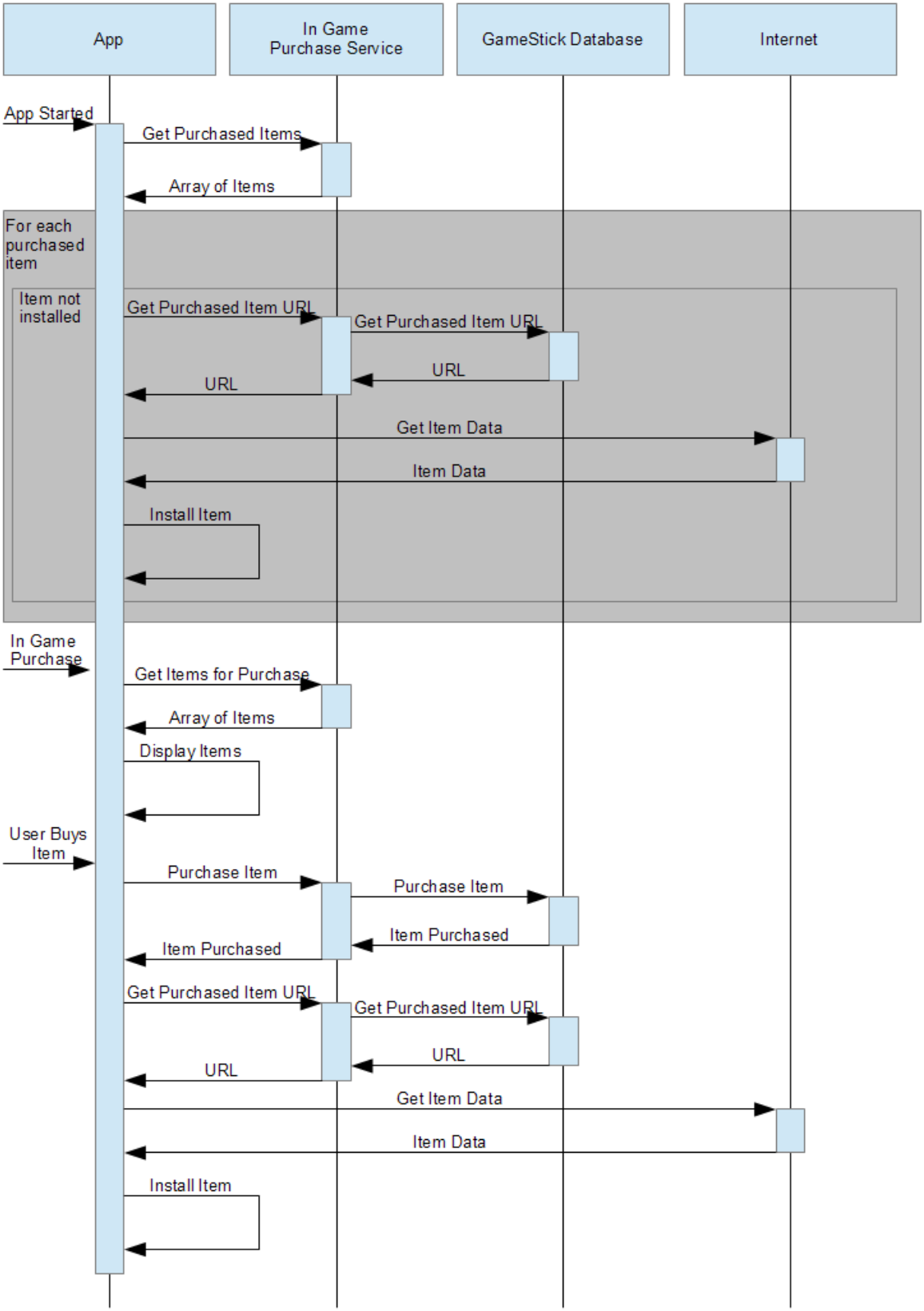
The responses from the PlayJam service via the SDK will involve the following data:

- *id* — A string representing the application *purchase item id*. This unique id is set up when you create the purchasable item on the Publishing Portal.
- *name* — The string name of the application purchase item.
- *description* — A string description of the application purchase item.
- *cost* — The cost of the purchase item, to the player (i.e., the amount in the local currency of the territory where the player is located — we currently support Euros, US Dollars, and UK Pounds).

The data returned will be appropriate to the location of the player, so for example a player in the US will have a wallet on the GameStick system which they can put funds into in US Dollars, and payments are then in US Dollars — so the cost value returned by the SDK will be in Dollars, since the service knows where the player is located, you don't have to do any currency conversion in-game.

There are a few complexities to handle, to do with what happens in certain situations such as, if the player downloads a game, purchases some in-game items, removes the game, then later re-installs the game — in this situation we need to ensure that the items the user previously purchased are also re-installed. The PlayJam on-line service will remember the purchased items, so you don't have to keep track yourself, just remember to query the service (via the SDK) and handle any that have been purchased but are not currently installed.

The following flow diagram illustrates how the process should work:



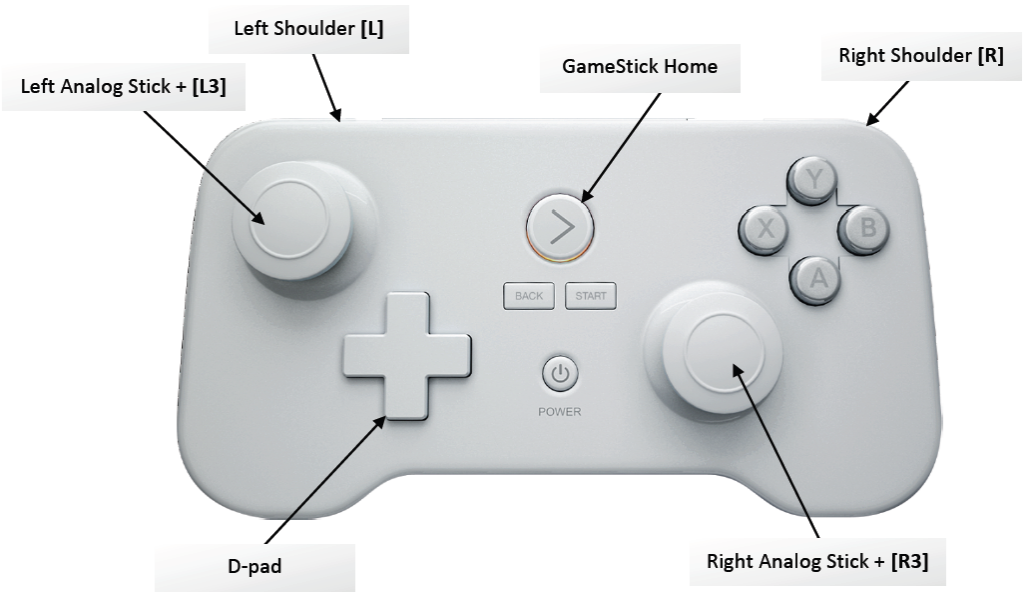
So, each time your game starts, it should call the SDK to obtain the list of items that the player has already purchased, then go through that list, and if any are not currently installed into your game, call the SDK again to get a URL to the item data, then download that data and install it (this part is shown in gray in the diagram above).

The lower part of the diagram illustrates the regular flow for purchasing an item in the game, and is fairly straightforward. First call the SDK to obtain a list of purchasable items, display them to the player. If the player buys an item, call the SDK to make the purchase, and if that worked without error, call the SDK again to obtain the URL of the item data, then download (again, calling the appropriate download API with the URL supplied) and then install the data. Your code should of course check that the purchase request was successful (be sure to check for the service sending you a request failure response message — this could happen for example if the player doesn't have sufficient funds in their wallet).

It is also possible that you could arrange for some or all purchasable items to be pre-installed as part of your game and just need unlocking — in this situation the basic approach will be identical except that you will not need to call the SDK to get a URL or to download the data, instead you would just need to do whatever is necessary in your code to unlock the item.

2.3 Controller Guidelines

The GameStick controller is a HID compatible Bluetooth controller, with the following controls:



The GameStick controller

Below we list some simple rules about how we recommend you use these controls in your game, so that players will see consistent behavior in different games and not become confused. We've listed the actual codes that the controller will report back to the system in the Button Mappings topic which follows; you'll need to review the Android documentation for details of how to implement these mappings in your code.

Menus

- The A buttons should carry out select functions.
- The B and BACK buttons should carry out back functions.
- The START button is used to bring up menus. When pressed inside a game, the pause menu is presented. When pressed anywhere else, it should bring up the main menu.
- Analog sticks and D-pad can be used to control scrollable lists and menus.
- Menu options should have roll over states, or button images graphically associated with them that reflect which controller button is pressed to access the option.
- When a menu that contains options has loaded, a selection border should surround one option by default.
- The Back button will bring up the exit confirmation menu when pressed in the main menu.

In-game

- START button should be used to PAUSE game and bring up in-game menu or display items like help/ tutorial/control/maps

2.3.1 Button Mappings

The following table gives the mappings for the buttons on the standard GameStick controller and also the Nyko Pro controller should you wish to support that. We've listed the Nvidia recommended codes also.

You may well find it worth your while to review the NVidia documentation on supporting Android game controllers.

Buttons	GameStick Controller reports	Nvidia recommended (Scancode)	Nyko Pro Controller
Dpad Up	Axis_Hat_Y(-1.0)	Axis_Hat_Y(-1.0)	Axis_Hat_Y(-1.0)
Dpad Down	Axis_Hat_Y(1.0)	Axis_Hat_Y(1.0)	Axis_Hat_Y(1.0)
Dpad Left	Axis_Hat_X(-1.0)	Axis_Hat_X(-1.0)	Axis_Hat_X(-1.0)
Dpad Right	Axis_Hat_X(1.0)	Axis_Hat_X(1.0)	Axis_Hat_X(1.0)
Button A	Button A(0x130h)(304)	Button A(0x130h)(304)	Button A(0x130h)(304)
Button B	Button B(0x131h)(305)	Button B(0x131h)(305)	Button B(0x131h)(305)
Button X	Button X(0x133h)(307)	Button X(0x133h)(307)	Button X(0x133h)(307)

Button Y	Button Y(0x134h)(308)	Button Y(0x134h)(308)	Button Y(0x134h)(308)
Left Shoulder (L1)	Button L1(0x136h)(310)	Button L1(0x136h)(310)	Button L1(0x136h)(310)
Right Shoulder (R1)	Button R1(0x137h)(311)	Button R1(0x137h)(311)	Button R1(0x137h)(311)
Button Back	Button Back(0x9Eh)(158)	Button Back(0x9Eh)(158)	Button Back(0x9Eh)(158)
Button Home	Button Home(0xACh)(Can't see Scancode))	Button Home(0xACh)(Can't see Scancode))	Button Home(0xACh)(Can't see Scancode))
Button Start	Button Start(0x13Bh)(315)	Button Start(0x13Bh)(315)	Button Start(0x13Bh)(315)
Left joystick button (L3)	Button Left Thumbl(0x13Dh)(317)	Button Left Thumbl(0x13Dh)(317)	Button Left Thumbl(0x13Dh)(317)
Right joystick button (R3)	Button Right Thumbl (0x13Eh)(318)	Button Right Thumbl (0x13Eh)(318)	Button Right Thumbl (0x13Eh)(318)
Left joystick	Axis X(-1.0(L) to 1.0(R))	Axis X(-1.0(L) to 1.0(R))	Axis X(-1.0(L) to 1.0(R))
	Axis Y(-1.0(U) to 1.0(D))	Axis Y(-1.0(U) to 1.0(D))	Axis Y(-1.0(U) to 1.0(D))
Right joystick	Axis Z(-1.0(L) to 1.0(R))	Axis Z(-1.0(L) to 1.0(R))	Axis Z(-1.0(L) to 1.0(R))
	Axis RZ(-1.0(U) to 1.0(D))	Axis RZ(-1.0(U) to 1.0(D))	Axis RZ(-1.0(U) to 1.0(D))
Button L2(Brake))	No buttons on GameStick controller	Axis Brake(0.0 to 1.0)	Axis Brake(0.0 to 1.0)
Button LR2(Gas)		Axis Gas(0.0 to 1.0)	Axis Gas(0.0 to 1.0)

2.4 Adapting from Mobile or Tablet

If you are adapting your game from mobile or tablet, you'll have the following issues to address:

Things to remove:

- Vibration
- Accelerometer / gyroscope
- Sensor based calibration
- Touch based buttons that don't need to be on-screen due to the lack of touch controls
- On the platform, for example the Pause icon in-game.

Things to replace:

- Tutorials for the control mechanism
- Links to the Google Play store must be removed OR changed to point to the GameStick store instead.
- Select optimal resolution in 16:9 aspect ratio in proportion to average viewing distances for TV content and performance of your game on the hardware (minimum resolution of 960x540)

Things to add:

- Landscape mode support with full screen rendering at 16:9 aspect ratio. All portions of the screen must be appropriately utilized.
- Instructional text to use menus with the GameStick controller.

Note that touch and drag support can be retained by players using GameStick Companion App on their phone or tablet to provide extra control to the game, but as not all players will with to do this, you should support the standard game controller wherever possible.

2.5 GameStick Images

This page contains images of the GameStick controller for use by developers. These can be used within games for example to display game controls.

If you have any queries about using these images please contact us.

Top view

Angle view

Slant view

Back view

Back slant view

Section 3
Developer Setup Guide

Before you can get started, you'll first need to download the relevant SDK download pack for your project.

The following SDK downloads are available via the GameStick website:

- Java
- C++ / Marmalade
- Unity

Note that everything you need to use the SDK from Unity is also provided via the Unity Asset Store.

The download pack will consist of a set of libraries (JAR files in the case of Java) which define the SDK classes, and an Android Application Package (.apk) file containing the PlayJam service which needs to be running on your system for you to make API calls.

You'll need to make sure your project includes the PlayJam libraries, and do some other simple project set-up depending on your development environment, such as ensuring that the .apk containing the SDK is included in your output project.

When you've built your project for Android, you should end up with an Android Application Package (.apk) file. This will enable you to test install the application on a test GameStick device.

See the following articles for more details about getting set up for development on these platforms:

- Java setup
- Marmalade setup
- Unity setup

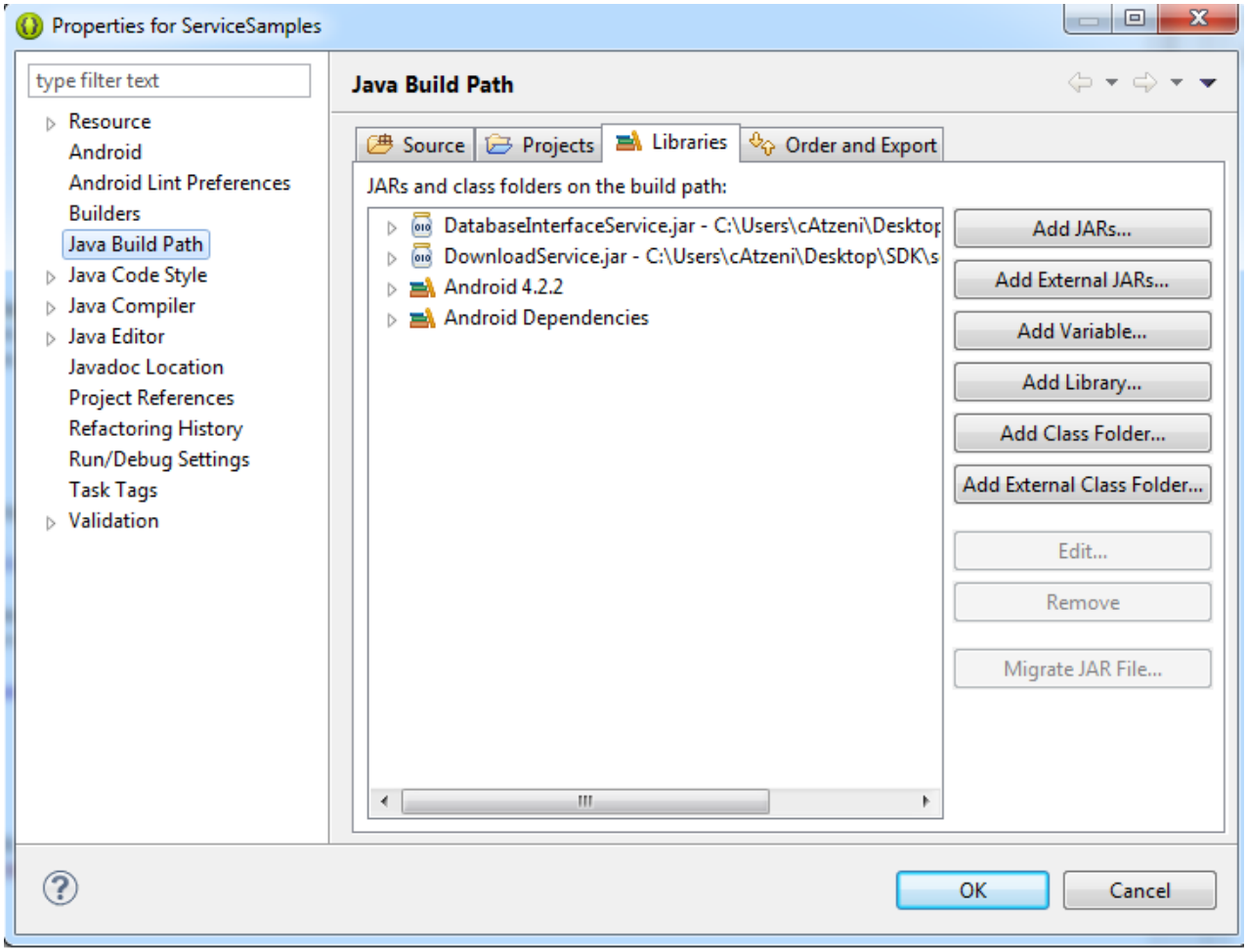
3.1 Java

When you unpack the download you will see:

- a folder `sdk/services/java` containing two `.jar` files which define two classes for you to use,
- a folder `debug/services` containing two `.apk` files which you will need.

In Eclipse's project properties, click on the Add External JARs... button and point the project to the location of the `.jar` file(s).

You should see the new libraries from the .jar files appear in your list of Libraries, like so:



Next, on the Order and Export tab, tick the new .jars to make sure they are included in your exported .apk (it's also a good idea to move them to the top of the list, just in case). You need to do this so that your exported application package contains the SDK libraries it needs to run.

In your code, you need to make sure the following libraries are imported so that you can use the GameStick SDK classes:

```
import com.playjam.gamestick.databaseinterfaceservice.*;
import com.playjam.gamestick.downloadservice.*;
```

Then in your code, implement the `IDownloadResponse` and `IJavaDatabaseInterfaceResponse` classes:

```
class DownloadResponse implement IDownloadResponse
{
    ...
}
```

Eclipse can automatically produce blank functions calls for all the things you need to implement (you must have something for every method in the interface, even if empty / trivial, or you will get run time errors - use the autocomplete function, or cut and paste from the example).

You'll also need to declare instances of the `DownloadService` class (which you don't need to implement as it's defined in the .jar) and an instance of your newly implemented `DownloadResponse` class:


```
DownloadService m_download_service;
DownloadResponse m_download_response;
```

Use the DownloadService class to call, for example:

```
m_download_service.DownloadPackage( url );
m_download_service.DownloadResource( url );
```

These are in fact the only two APIs in the `DownloadService jar` — the `DataBaseInterfaceService.jar` contains many more APIs, but the principle is the same. See the API Reference for details of each of the APIs.

Important Note:

in order to be able to be run your code on the device, you will also need to add the following line(s) to the `AndroidManifest.xml` of your project:

```
<uses-permission android:name="com.playjam.gamestick.permission.DATABASE_SERVICE"/>
<uses-permission android:name="com.playjam.gamestick.permission.DOWNLOAD_SERVICE"/>
```

...depending on whether you are using the Database or Download service APIs (or both, which will be most common).

For example:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
   package="com.playjam.servicesamples"
   android:versionCode="1"
   android:versionName="1.0" >
<uses-sdk
   android:minSdkVersion="8"
   android:targetSdkVersion="17" />
<uses-permission android:name="com.playjam.gamestick.permission.DOWNLOAD_SERVICE"/>
   <application
      android:allowBackup="true"
      android:icon="@drawable/ic_launcher"
      android:label="@string/app_name"
      android:theme="@style/AppTheme" >
      <activity
         android:name="com.playjam.servicesamples.MainActivity"
         android:label="@string/app_name" >
         <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
         </intent-filter>
      </activity>
   </application>
</manifest>
```

3.2 Marmalade

In the cpp folder you will see:

- `/lib/` - the C++ libraries needed (built for ARM)
- `/h/` - the C++ header files needed
- `DownloadService.mkf` - Marmalade subproject for the DownloadService APIs

- `DownloadService.jar` - the Java classes needed for the DownloadService APIs
- `DatabaseInterfaceService.mkf` - Marmalade subproject for the DatabaseInterfaceService APIs
- `DatabaseInterfaceService.jar` - the java classes needed for the DatabaseInterfaceService APIs

To build with Marmalade and C++ we will assume you are using Visual Studio. Obviously in your project you need to make sure it picks up the C++ libs and headers so you can build, so copy the /cpp folder hierarchy from the download pack into somewhere handy for your project.

Then, make sure you add some lines to your application project's .mkb file:

```
subproject "C:/pathname/cpp/DownloadService"
subproject "C:/pathname/cpp/DatabaseInterfaceService"
```

before the following section:

```
subprojects
{
 ...
}
```

If you try and build at this point, Visual Studio will detect changes and display a dialog asking if you want to reload your project; say yes. You should now see the subproject(s) as part of your Visual Studio project.

You will need to add the following lines to the top of your code to include the necessary C++ header files and set the namespace:

```
#include "DatabaseInterface.h"
#include "DownloadServices.h"
using namespace PlayJam;
```

... and of course you will need to alter the additional include directories setting in your project to pick up the header files.

You will then be able to specify:

```
DownloadResponse *response;
DownloadServiceInterface *service_interface;
DatabaseInterfaceResponse *db_response;
DatabaseInterfaceServiceInterface *db_interface;
```

you'll then need to implement a `DownloadResponse` class, based on the `IDownloadResponse` class:

DownloadResponse.h:

```
include <iostream>
include "DownloadServiceInterface.h"
using namespace PlayJam

class DownloadResponse ; public DownloadServiceInterface::IDownloadResponse
{
   public:
      DownloadResponse(void);
      ~DownloadResponse(void);
      virtual void DownloadProgress ( const std::string &url, int progress );
      virtual void DownloadSuccess ( const std::string &url, const std::string &destination );
      virtual void DownloadFailed ( const std::string &url, const std::string &message);
};
```


These are the functions which will be called by the service to let your application know how the downloads are going:

- `DownloadProgress()` - is called (several times) after you ask for a download, indicating current progress (so you can implement a progress bar using this info).
- `DownloadSuccess()` - is called at the end of the download if all is well - the string destination is a message saying where the file went (place on the SD card on the stick, for example, e.g., `/storage/sdcard0/Download/dowbload-645679000.tmp` — use this location to go grab the file in your application.
- `DownloadFailed()` - is called when the download fails (along with a suitable error message hopefully telling you what went wrong): e.g., "Exception during download" - these messages are pretty standard fare from java - what you do with this message is up to you.

Sending update requests:

It's important if you are using the download services to send an update message from time to time via:

```
service_interface->Update();
```

Probably the easiest thing to do is to call this from your main game loop so it updates once per frame.

Integrating support for DatabaseServices is largely the same process, though there are more APIs there—see the API reference for details.

Lastly, remember also to ensure that the SDK libraries are included in your .apk build target so that the finished project will be able to make API calls when running on the GameStick.

3.3 Unity

The Unity package contains everything you need including scripts that provide functions you can call from your Unity project, and Unity-specific versions of the necessary JAR files.

Android Manifest Permissions

In one or both of Unity's AndroidManifest.xml templates found at...

```
\Unity\Editor\Data\PlaybackEngines\androiddevelopmentplayer\AndroidManifest.xml
\Unity\Editor\Data\PlaybackEngines\androidplayer\AndroidManifest.xml
```

...place the following permissions (you can put them just after the opening <manifest> tag:

```
<uses-permission android:name="com.playjam.gamestick.permission.DOWNLOAD_SERVICE" />
<uses-permission android:name="com.playjam.gamestick.permission.DATABASE_INTERFACE_SERVICE" />
```

...without these, service calls with fail (leaving a suitable message in the ADB log).

Do not forget to remove these for other projects. Probably the easiest way is to make copies of the original template manifests before making modifications. That way you can easily switch back and forth between these if you are working on several different Unity projects, GameStick and non-GameStick. You will likely need to set the permissions on these folders in order to make the above modifications, at least under

Windows.

Runtime Dependencies

Certain JAR files are required to supply Unity with the ability to make service calls. These should be placed in your project under /Assets/Plugins/Android.

Mandatory (SDK service providers):

```
unitydatabaseinterfaceservice.jar
unitydownloadservice.jar
```

Optional (for troubleshooting basic Unity<->JVM interop):

```
unitytestjni.jar
```

Code

The project consists of the scene containing `GameObjects`, finalised service classes and sample implementation classes.

Scene:

Open the scene and examine the `ExampleServiceRequests` and `ExampleServiceResponseHandler` objects. Their names don't matter; nor do the names of the scripts attached; however the scripts themselves do.

Services:

- `PlayJamServices` encapsulates Unity->Java service calls, giving access to all SDK services.
- The abstract class `ServiceResponseHandler` must be implemented to allow callbacks to the appropriate method, Java->Unity. This should then be attached to a `GameObject` (see also below).
- `PlayJamServices.SetResponseHandler(name)` allows service callbacks to be made to the chosen object, Java->Unity.
- The above two points offer flexibility in placing the service response handlers anywhere in your implementation.

Example implementations:

- `ExampleServiceResponseHandler.cs` implements the aforementioned interfaces. At this point there is no parsing of the data returned from the service (which will be some JSON).
- `ExampleServiceRequests.cs` implementats a screen full of buttons, each of which makes a different service call via PlayJamServices, and response text fields (one for download services, one for database services) to see the results of each button press. Where arguments are required for service calls, dummy arguments have been hardcoded for a simple demonstration.
- TestJNI method is used to ascertain whether the simplest of the three JARs, `unitytestjni.jar`, is

communicating using the basic mechanism used by the service JARs.

To ensure everything works, be sure you have called `PlayJamServices.SetResponseHandler ("<YourGameObjectName>")`, where `<YourGameObjectName>` is the name of the object on which your response handler script is placed; in the example provided, this name is `"ExampleServiceResponseHandler"` which refers to the object with that name in the scene.

Cleanup should always be done at app exit by calling `PlayJamServices.Dispose()`, or the Java objects / classes will likely not be cleared from the heap.

Section 4
API Reference

Reference information for the classes in the SDK is provided for Java, C++ and Unity bindings.

- Java reference
- C++ reference
- Unity reference

4.1 Java API

The following document outlines all the API functions in the Java version of the SDK.

Bear in mind the overall structure of the SDK is that you send messages, via API calls, to a service running on the stick that will use the internet to make a request to the PlayJam server. The responses are then sent by the server and picked up by the service running on the stick. In order to receive the response, you need to implement listening objects in your code, and to do this you use additional classes provided in the SDK.

The data returned via the internet is actually in a JSON form, but as a Java developer you don't need to worry about this — we've provided classes to encapsulate and parse the relevant information. For example, the Achievements class provides methods to get the name or ID of an achievement, or to iterate to the next achievement, and so on. So you really don't have to concern yourself with the actual structure of the data, or how it is sent and received by the on-line service — all you need to do is implement the provided classes, make API calls to send messages, and use the data returned.

The following packages are of interest:

- DatabaseInterfaceService

  This package provides access to the on-line database via the following classes:

DatabaseInterfaceService	Defines the API calls you use to send messages to the service.
IJavaDatabaseInterfaceResponse	Interface which you need to implement in order to receive responses back from the database service.
Achievements	Class which encapsulates achievement data passed back to you.
AppItems	Class which encapsulates the app items data passed back to you.
LeaderboardData	Class which encapsulates leader board data passed back to you.

- DownloadServiceInterface

  This package provides access to the download services via the following classes:

DownloadServiceInterface	Defines the API calls you use to send messages to the download service.
IDownloadResponse	Interface which you need to implement in order to receive responses back from the download service.

4.1.1 DatabaseInterfaceService

The `DatabaseInterfaceService` class provides APIs enabling you to make requests relating to each of the following parts of the PlayJam On-line Services:

- **Leader board APIs**

```
DatabaseInterfaceService.LeaderBoard_GetTop50();
DatabaseInterfaceService.LeaderBoard_GetRange( int from, int to );
DatabaseInterfaceService.LeaderBoard_SaveScore( int score );
DatabaseInterfaceService.LeaderBoard_GetNearest( unsigned range,
                                                 bool sort_ascending  );
```

- **Save / load game state APIs**

```
DatabaseInterfaceService.Game_SaveState( byte [] data  );
DatabaseInterfaceService.Game_LoadState();
```

- **Achievements APIs**

```
DatabaseInterfaceService.Achievement_GetAllAchievements();
DatabaseInterfaceService.Achievement_SetAchievmentComplete( String id );
```

- **Analytics APIs**

```
DatabaseInterfaceService.Analytics_GameEvent(  String hashmap  );
```

- **In app purchasing APIs**

```
DatabaseInterfaceService.InAppPurchase_GetPurchasedItems();
DatabaseInterfaceService.InAppPurchase_GetItemsForPurchase();
DatabaseInterfaceService.InAppPurchase_PurchaseItem( String item_id );
DatabaseInterfaceService.InAppPurchase_GetPurchasedItemURL( String item_id );
```

Note that in all cases, the actual API functions do not have a return value - the service will issue responses which may contain returned data, and you need to implement the IJavaDatabaseInterfaceResponse interface to handle these responses. For each call, the service call either

the appropriate response handler (which we indicate below for each call) or if something went wrong, `DatabaseRequestFailed`.

Before you can make any SDK calls you will need to create your `DatabaseInterfaceService` object and associate it with your response handler, which you do by passing your handler to the `DatabaseInterfaceService` constructor like so:

```
m_database_response = new JavaDatabaseInterfaceResponse();
m_database_service = new DatabaseInterfaceService(this, m_database_response);
```

4.1.1.1 Leaderboard APIs

These APIs issue requests to the leader board service. Remember you will have to implement the `IJavaDatabaseInterfaceResponse` interface to handle the responses from the service. The name of the response call-back corresponding to each call is listed below.

DatabaseInterfaceService.LeaderBoard_GetTop50()

Description

Create a request to get top 50 leader-board entries.

Parameters

None.

Response

```
LeaderBoard_GetTop50_Response (LeaderboardData data);
```

The data returned to your response handler will be a list of leader board entries which you can iterate through and access information about — see the `LeaderboardData` class reference page.

DatabaseInterfaceService.LeaderBoard_GetRange(int from, int to)

Description

Create a request to get a set of leader-board entries.

Parameters

int	from	The highest ranking entry on the leader-board you want to get (1

		being the top ranked entry).

Response

```
LeaderBoard_GetRange_Response (LeaderboardData data);
```

The data returned to your response handler will be a list of leader board entries which you can iterate through and access information about — see the `LeaderboardData` class reference page.

DatabaseInterfaceService.LeaderBoard_SaveScore(int score)

Description

Create a request to set a leader-board entry.

Parameters

int	score	The score to save into the leader-board.

Response

```
LeaderBoard_SaveScore_Response ();
```

The response will not include any data, the response call simply confirms that your request to save the score was successful.

DatabaseInterfaceService.LeaderBoard_GetNearest(int range, boolean sort_ascending)

Description

Create a request to get a set of leader-board entries near to the current score (i.e., the last score saved with `DatabaseInterfaceService.LeaderBoard_SaveScore`).

Parameters

int	range	The size of the range of nearby values - for example if you want the 5 leaders above and 5 below the last score saved, specify 5 here.
boolean	sort_ascending	Specify the order of the sorted entries to be returned.

Response

```
LeaderBoard_GetNearest_Response (LeaderboardData data);
```

The data returned to your response handler will be a list of leader board entries which you can iterate through and access information about — see the `LeaderboardData` class reference page.

4.1.1.2 Save State APIs

These APIs issue requests to the Save State module of the service. Remember you will have to implement the `IJavaDatabaseInterfaceResponse` interface to handle any responses from the service.

DatabaseInterfaceService.Game_SaveState(byte [] data)

Description

Create a request to save the current game state.

Parameters

byte []	data	Your data (which should be stored as an array of byte/char data).

Response

```
Game_SaveState_Response ();
```

DatabaseInterfaceService.Game_LoadState()

Description

Create a request to load the previously saved game state.

Parameters

None.

Response

```
Game_LoadState_Response (InputStream data);
```

4.1.1.3 Achievements APIs

These APIs issue requests to the Achievements module of the service. Remember you will have to implement the `IJavaDatabaseInterfaceResponse` interface to handle any responses from the service.

DatabaseInterfaceService.Achievement_GetAllAchievements()

Description

Create a request to get all game achievements. The data passed back to you via your `IJavaDatabaseInterfaceService` response handler will contain a list of achievements which you can easily access using the `Achievements` class.

Parameters

None.

Response

```
Achievement_GetAllAchievements_Response (Achievements data);
```

DatabaseInterfaceService.Achievement_SetAchievmentComplete(String id)

Description

Create a request to set an achievement as completed.

Parameters

String	id	String representing the achievement id.

Response

```
Achievement_SetAchievementComplete_Response ();
```

4.1.1.4 Analytics APIs

These APIs relate to storing data to do with game events.

The service will save the data on the PlayJam server, so you can use this API to store any arbitrary data to do with events during the game which you want to track - for example, you can store something every time a player performs a certain action (reads the instructions, quits a game, or whatever) so you can use this data later to help analyse how people interact with your game.

Remember you will have to implement the `IJavaDatabaseInterfaceResponse` interface to handle any responses from the service.

DatabaseInterfaceService.Analytics_GameEvent(Map <String, String> hashmap)

Description

Create a request to set a game event.

Parameters

Map <String, String>	hashmap	A set of key-value paired strings representing the game event data.

Response

```
Analytics_GameEvent_Response ();
```

No data is returned to the response handler for this request, the response just indicates that the request was received.

4.1.1.5 In-app purchasing APIs

These APIs issue requests to the in-app purchasing module of the service. Remember you will have to implement the `IJavaDatabaseInterfaceResponse` interface to handle any responses from the service. Data passed back to your response handler is parsed for you by the `AppItems` class.

The In-App Purchasing Guidelines topic explains what you need to do to support purchasing in more detail.

DatabaseInterfaceService.InAppPurchase_GetPurchasedItems()

Description

Create a request to get all purchased items.

Parameters

None.

Response

```
InAppPurchase_GetPurchasedItems_Response (AppItems data);
```

The data returned will be a list of all the purchased items, which you can iterate through and access information about — see the `AppItems` class reference page.

DatabaseInterfaceService.InAppPurchase_GetItemsForPurchase()

Description

Create a request to get all items available for purchase.

Parameters

None.

Response

```
InAppPurchase_GetItemsForPurchase_Response (AppItems data);
```

The data returned will be a list of all the available items, which you can iterate through and access information about — see the `AppItems` class reference page.

DatabaseInterfaceService.InAppPurchase_PurchaseItem(String item_id)

Description

Create a request to purchase an item.

Parameters

String	item_id	String value representing the item id.

Response

```
InAppPurchase_PurchaseItem_Response (boolean bought);
```

The boolean value passed back to your response handler will indicate whether the purchase request was successful.

DatabaseInterfaceService.InAppPurchase_GetPurchasedItemURL(String item_id)

Description

Request the URL of the purchased item.

Parameters

String	item_id	String value representing the item id.

Response

```
InAppPurchase_GetPurchasedItemURL_Response (String url);
```

4.1.2 IJavaDatabaseInterfaceResponse

This interface class provides the methods needed to receive messages back from the service. You need to implement your own object that uses this interface and provides something for each of the following methods:

```
void LeaderBoard_GetTop50_Response (LeaderboardData data);
void LeaderBoard_GetRange_Response (LeaderboardData data);
void LeaderBoard_GetNearest_Response (LeaderboardData data);
```

```
    void LeaderBoard_SaveScore_Response ();

    void Game_SaveState_Response ();
    void Game_LoadState_Response (InputStream data);

    void Achievement_GetAllAchievements_Response (Achievements data);
    void Achievement_SetAchievementComplete_Response ();

    void Analytics_GameEvent_Response ();

    void InAppPurchase_GetPurchasedItems_Response (AppItems data);
    void InAppPurchase_GetItemsForPurchase_Response (AppItems data);
    void InAppPurchase_PurchaseItem_Response (boolean bought);
    void InAppPurchase_GetPurchasedItemURL_Response (String url);

    void DatabaseRequestFailed (Request id, String message);
```

The service will then call your implementations of these methods in order to send information back to your application. Note that the data is often passed via objects such as `Achievements`, `AppItems` and `LeaderboardData` which we have provided to help with parsing the information.

Error handling

You will need to implement `DatabaseRequestFailed()` in order to listen out for messages back from the service telling you which request did not work (a string is also returned containing an error message). The id of the request is returned via the Request type which is an enum defined as follows:

```
enum Request
{
   LeaderBoard_GetTop50,
   LeaderBoard_GetRange,
   LeaderBoard_GetNearest,
   LeaderBoard_SaveScore,
   Game_SaveState,
   Game_LoadState,
   Achievement_GetAllAchievements,
   Achievement_SetAchievementComplete,
   Analytics_GameEvent,
   InAppPurchase_GetPurchasedItems,
   InAppPurchase_GetItemsForPurchase,
   InAppPurchase_PurchaseItem,
   InAppPurchase_GetPurchasedItemURL,
   NumberOfRequests
};
```

Example

The following very basic code example shows how you might implement the `IJavaDatabaseInterfaceResponse` interface and provide handlers for all the call-backs. Naturally in your own code you will eventually need to do something more complex...

```
import com.playjam.gamestick.databaseinterfaceservice.*;
```

```
public class JavaDatabaseInterfaceResponse implements IJavaDatabaseInterfaceResponse
{

  @Override
  public void Achievement_GetAllAchievements_Response(Achievements achievements)
  {
    for (Iterator iterator = achievements.GetItems().iterator(); iterator.hasNext();)
    {
      Item item = (Item) iterator.next();

      Log.d("GameStick", "ID :" + Integer.toString( item.ID() ));
      Log.d("GameStick", "Name : " + item.Name());
      Log.d("GameStick", "Description : " + item.Description());
      Log.d("GameStick", "Type : " + item.Type());
      Log.d("GameStick", "fileName : " + item.FileName());
      Log.d("GameStick", "file URL : " + item.FileURL());
      Log.d("GameStick", "Completed : " + Boolean.toString( item.isCurrentUserOwner() ));
      Log.d("GameStick", "XP Value : " + Integer.toString( item.XPValue() ));
    }
  }

  @Override
  public void Achievement_SetAchievementComplete_Response()
  {
    Log.d("GameStick", "Set achievement complete");
  }

  @Override
  public void Analytics_GameEvent_Response()
  {
    Log.d("GameStick", "Analytics response");
  }

  @Override
  public void DatabaseRequestFailed(DatabaseInterfaceService.Request request, String message)
  {
    Log.d("GameStick", "Error : " + message);
  }

  @Override
  public void Game_LoadState_Response(InputStream input)
  {
    Log.d("GameStick", "Loading data");
    java.util.Scanner s = new java.util.Scanner(input).useDelimiter("\\A");
    String result = s.next();
    Log.d("GameStick","Converted data : " + result);
  }

  @Override
  public void Game_SaveState_Response()
  {
    Log.d("GameStick", "Saving");
  }

  @Override
  public void InAppPurchase_GetItemsForPurchase_Response(AppItems items)
  {
    for (Iterator iterator = items.GetItems().iterator(); iterator.hasNext();)
    {
      AppItems.Item item = (AppItems.Item) iterator.next();
      Log.d("GameStick", items.toString());
    }

  }
```

```
@Override
public void InAppPurchase_GetPurchasedItemURL_Response(String url)
{
  Log.d("GameStick", "Purchased item url : " + url);
}

@Override
public void InAppPurchase_GetPurchasedItems_Response(AppItems items)
{
  for (Iterator iterator = items.GetItems().iterator(); iterator.hasNext();)
  {
    AppItems.Item item = (AppItems.Item) iterator.next();
    Log.d("GameStick", items.toString());
  }
}

@Override
public void InAppPurchase_PurchaseItem_Response(boolean response)
{
  Log.d("GameStick", "Purchase response : " + response);
}

@Override
public void LeaderBoard_GetNearest_Response(LeaderboardData data)
{
  for (Iterator iterator = data.GetEntries().iterator(); iterator.hasNext();)
  {
    Entry item = (Entry) iterator.next();
    Log.d("GameStick", Integer.toString( item.AvatarID() ));
    Log.d("GameStick", item.Name());
    Log.d("GameStick", Integer.toString( item.Score() ));
    Log.d("GameStick", Integer.toString( item.Position() ));
  }
}

@Override
public void LeaderBoard_GetRange_Response(LeaderboardData data)
{
  for (Iterator iterator = data.GetEntries().iterator(); iterator.hasNext();)
  {
    Entry item = (Entry) iterator.next();
    Log.d("GameStick", Integer.toString( item.AvatarID() ));
    Log.d("GameStick", item.Name());
    Log.d("GameStick", Integer.toString( item.Score() ));
    Log.d("GameStick", Integer.toString( item.Position() ));
  }
}

@Override
public void LeaderBoard_GetTop50_Response(LeaderboardData data)
{
  for (Iterator iterator = data.GetEntries().iterator(); iterator.hasNext();)
  {
    Entry item = (Entry) iterator.next();
    Log.d("GameStick", Integer.toString( item.AvatarID() ));
    Log.d("GameStick", item.Name());
    Log.d("GameStick", Integer.toString( item.Score() ));
    Log.d("GameStick", Integer.toString( item.Position() ));
  }
}

@Override
public void LeaderBoard_SaveScore_Response()
{
  Log.d("GameStick", "Save Score Response");
```

```
    }
}
```

4.1.3 Achievements

This object wraps the data for achievements that will be returned by the service via the `IJavaDatabaseInterfaceClass` interface you implemented for:

```
Achievement_GetAllAchievements_Response (Achievements data);
```

Note that an `Achievements` object is returned as a result of the `DatabaseInterfaceService.Achievement_GetAllAchievements()` API call. It won't come as a surprise to you then, that the object encodes a list of achievements (actually returned via the service as some JSON, but parsed for your convenience by this class).

`GetItems` — The `GetItems` method is used to access the items in the object.

`Item` — this represents a particular item in the Achievement data (i.e., a single element from the list of achievements).

- `ID` — (int) The achievement id number (allocated when the achievement was defined via the PlayJam Publishing Portal).
- `Name` — (String) Name of the achievement.
- `Description` — (String) Description of the achievement.
- `Type` — (String) The achievement type.
- `FileName` — (String) The filename of some data associated with the achievement, passed back by the service.
- `FileURL` — (String) The URL of some data associated with the achievement.
- `isCurrentUserOwner` — (boolean) Indicates whether the achievement was completed by the current user. This will therefore be true for those achievements that you have set to be complete using `DatabaseInterfaceService.Achievement_SetAchievmentComplete`.
- `XPValue` — (int) The number of experience points to be granted to the player for completing the achievement.

Example

The following very simple example shows how to implement `IJavaDatabaseInterfaceResponse` to provide your response handling class, override the *get all achievements* response, and iterate through the data returned (here we just write the data out to the log, of course you will want to do something more intelligent

and useful...).

```
public class JavaDatabaseInterfaceResponse implements IJavaDatabaseInterfaceResponse
{

   ...

   @Override
   public void Achievement_GetAllAchievements_Response(Achievements achievements)
   {
      for (Iterator iterator = achievements.GetItems().iterator(); iterator.hasNext();)
      {
         Item item = (Item) iterator.next();

         Log.d("GameStick", "ID :" + Integer.toString( item.ID() ));
         Log.d("GameStick", "Name : " + item.Name());
         Log.d("GameStick", "Description : " + item.Description());
         Log.d("GameStick", "Type : " + item.Type());
         Log.d("GameStick", "fileName : " + item.FileName());
         Log.d("GameStick", "file URL : " + item.FileURL());
         Log.d("GameStick", "Completed : " + Boolean.toString( item.isCurrentUserOwner() ));
         Log.d("GameStick", "XP Value : " + Integer.toString( item.XPValue() ));
      }
   }

   ...

}
```

4.1.4 AppItems

This object wraps the data for application items that will be returned by the service via the `IJavaDatabaseInterfaceClass` interface you implemented for:

```
    void InAppPurchase_GetPurchasedItems_Response (AppItems data);
    void InAppPurchase_GetItemsForPurchase_Response (AppItems data);
```

In other words this class is used to represent in-app purchase items you might have defined and which can be returned via the service in response to API calls.

`GetItems` — The `GetItems` method is used to access the items in the object.

`Item` — this represents a particular item in the `AppItems` data (i.e., a single element from the list of application purchase items).

- `ID` — (String) The application application purchase item id.
- `Name` — (String) Name of the application purchase item.
- `Description` — (String) Description of the application purchase item.
- `Cost` — (String) The cost of the purchase item.

Example

The following very simple example shows how to implement `IJavaDatabaseInterfaceResponse` to provide your response handling class, override the achievements response call, and iterate through the achievements data returned (here we just write the data out to the log, of course you will want to do something more interesting).

```
public class JavaDatabaseInterfaceResponse implements IJavaDatabaseInterfaceResponse
{

   ...

   @Override
   public void InAppPurchase_GetItemsForPurchase_Response(AppItems items)
   {
      for (Iterator iterator = items.GetItems().iterator(); iterator.hasNext();)
      {
         AppItems.Item item = (AppItems.Item) iterator.next();
         Log.d("GameStick", items.toString());
      }

   }

   ...

}
```

4.1.5 LeaderboardData

This object wraps the data for leader boards that will be returned by the service via the `IJavaDatabaseInterfaceClass` interface you implemented for:

```
    void LeaderBoard_GetTop50_Response (LeaderboardData data);
    void LeaderBoard_GetRange_Response (LeaderboardData data);
    void LeaderBoard_GetNearest_Response (LeaderboardData data);
```

In each case the value returned as `LeaderboardData` represents a list of items from the leader board (i.e., a list of leader board entries). We provide accessors for parsing the entries to get the name, score, avatar id, and position of each list entry without you having to parse the data yourself (which as always is actually returned as some JSON behind the scenes).

`GetItems` — The `GetItems` method is used to access the items in the object.

`Item` — this represents a particular item in the leader board data (i.e., a single element from the list of leader board entries returned).

- `Name` — (String) The player's name.
- `Score` — (int) The score.

- `AvatarID` — (int) The id of the player's avatar.
- `Position` — (int) The current position on the leader board of that score.

Example

The following very simple example shows how to implement `IJavaDatabaseInterfaceResponse` to provide your response handling class, override the *leader board get top 50* response, and iterate through the data returned (here we just write the data out to the log, of course you will want to do something more intelligent and useful...).

```
public class JavaDatabaseInterfaceResponse implements IJavaDatabaseInterfaceResponse
{

   ...

   @Override
   public void LeaderBoard_GetTop50_Response(LeaderboardData data)
   {
      for (Iterator iterator = data.GetEntries().iterator(); iterator.hasNext();)
      {
         Log.d("GameStick", "item");
         Entry item = (Entry) iterator.next();

         Log.d("GameStick", item.Name());
         Log.d("GameStick", Integer.toString( item.Score() ));
         Log.d("GameStick", Integer.toString( item.AvatarID() ));
         Log.d("GameStick", Integer.toString( item.Position() ));

      }
   }

   ...

}
```

4.1.6 DownloadServiceInterface

The download services group of APIs issue requests to the downloads module of the service. Remember you will have to implement the `IDownloadResponse` class to handle the responses.

```
DownloadServiceInterface.DownloadPackage( String url );
DownloadServiceInterface.DownloadResource( String url );
```

Before you can make any SDK calls you will need to create your `DownloadServiceInterface` object and associate it with your response handler, which you do by passing your handler to the `DownloadServiceInterface` constructor like so:

```
m_download_response = new DownloadResponse();
```

```
m_download_service = new DownloadService(this, m_download_response);
```

DownloadServiceInterface.DownloadPackage(String url)

Description

Requests a package (or large file) download from a server. Progress messages are sent during download. This can be used for larger files such as game levels and has higher priority.

Note that if you make multiple requests to `DownloadPackage()`, the packages will be returned from the service in FIFO (first-in first-out) order, i.e., the packages are returned in the order you requested them.

Parameters

String	url	The file to be downloaded.

Response

One of:

```
DownloadSuccess (String url, String destination);
DownloadFailed (String url, String message);
```

DownloadServiceInterface.DownloadResource(String url)

Description

Create a request to download a file. This is intended for smaller, lower priority files and doesn't issue progress report messages.

Note that if you make multiple requests to `DownloadResource()`, the files will be returned from the service in FILO (first-in last-out) order, i.e., the most recent request will be downloaded first. This means for example that when you draw a screen full of graphics the last graphic requested will be sent first.

Parameters

String	url	The file to be downloaded.

Response

One of:

```
DownloadSuccess (String url, String destination);
DownloadFailed (String url, String message);
```

4.1.7 IDownloadResponse

This interface class provides the methods needed to receive messages back from the download service. You need to implement your own object that uses this interface and provides something for each of the following methods:

```
void DownloadProgress (String url, int progress);
void DownloadSuccess (String url, String destination);
void DownloadFailed (String url, String message);
```

DownloadProgress(String url, int progress)

Description

Will be called by the service when a download progress callback occurs.

Parameters

String	url	The file being downloaded.
int	progress	The amount downloaded so far as a percentage.

Return

None.

DownloadSuccess(String url, String destination)

Description

Will be called when a download success callback occurs. The information passed to you in the parameters tell you the name of the file (a file you requested earlier via `DownloadServiceInterface.DownloadPackage` or `DownloadServiceInterface.Resource`) and the location on the stick in which the file is (temporarily) located.

Note that the file *will be removed* from the stick when the call to this method ends, so you will need to read the contents or copy it to where you need it in your `DownloadSuccess` method.

Parameters

String	url	The file being downloaded.
String	destination	The temporary local location of the

		downloaded file.

Return

None.

DownloadFailed(String url, String message)

Description

Will be called when a download failed callback occurs. The information passed to you in the parameters tell you the name of the file (a file you requested earlier via `DownloadServiceInterface.DownloadPackage` or `DownloadServiceInterface.Resource`) and an error message indicating what went wrong.

Parameters

String	url	The file being downloaded.
String	message	The error message.

Return

None.

Example

The following very basic code example shows how you might implement the `IDownloadResponse` interface and provide handlers for `DownloadFailed`, `DownloadSuccess`, and `DownloadProgress`. Naturally in your own code you will eventually need to do something more complex.

```
import com.playjam.gamestick.downloadservice.DownloadService;
import com.playjam.gamestick.downloadservice.DownloadService.IDownloadResponse;

class DownloadResponse implements IDownloadResponse
{

   @Override
   public void DownloadFailed(String url, String success) {
      Log.d("PLAYJAM", "Failed : " + url + " value " + success);
      // you could display an appropriate message to the user here
   }

   @Override
   public void DownloadProgress(String url, int progress)
   {
      Log.d("PLAYJAM", "Progress : " + url + " value " + progress);
```

```
        // you could add code to provide a progress bar here
    }

    @Override
    public void DownloadSuccess(String url, String destination)
    {
        Log.d("PLAYJAM", "Success : " + url + " value " + destination);
        // insert code here to copy the file somewhere safe if needed later
    }

}
```

4.2 C++ API

The following document outlines all the API functions in the C++ version of the SDK.

Bear in mind the overall structure of the SDK is that you send messages, via API calls, to a service running on the stick that will use the internet to make a request to the PlayJam server. The responses are then sent by the server and picked up by the service running on the stick. In order to receive the response, you need to implement listening objects in your code, and to do this you use additional classes provided in the SDK.

The data returned via the internet is actually in a JSON form, but as a C++ developer you don't need to worry about this — we've provided classes to encapsulate and parse the relevant information. For example, the Achievements class provides methods to get the name or ID of an achievement, or to iterate to the next achievement, and so on. So you really don't have to concern yourself with the actual structure of the data, or how it is sent and received by the on-line service — all you need to do is implement the provided classes, make API calls to send messages, and use the data returned.

The following .h files are of interest:

- DatabaseInterfaceService.h

  Classes defined here provide access to the on-line database:

DatabaseInterfaceServiceInterface	Defines the API calls you use to send messages to the service.
IRequestResponse	Interface which you need to implement in order to receive responses back from the database service.
Achievements	Class which encapsulates achievement data passed back to you.
AppItems	Class which encapsulates the app items data passed back to you.
LeaderboardData	Class which encapsulates leader board data passed back to you.

- DownloadService.h

  Classes defined here provide access to the download services:

DownloadServiceInterface	Defines the API calls you use to send messages to the download service.
IDownloadResponse	Interface which you need to implement in order to receive responses back from the download service.

4.2.1 DatabaseInterfaceServiceInterface

The Database services group actually provides APIs relating to each of the following parts of the PlayJam On-line Services:

- **Leader board APIs**

```
DatabaseInterfaceServiceInterface::LeaderBoard_GetTop50();

DatabaseInterfaceServiceInterface::LeaderBoard_GetRange ( unsigned from,
                                                          unsigned to );

DatabaseInterfaceServiceInterface::LeaderBoard_SaveScore( unsigned score );

DatabaseInterfaceServiceInterface::LeaderBoard_GetNearest ( unsigned range,
                                                            bool sort_ascending );
```

- **Save / load game state APIs**

```
DatabaseInterfaceServiceInterface::Game_SaveState( unsigned char *data,
                                                   size_t size   );

DatabaseInterfaceServiceInterface::Game_LoadState();
```

- **Achievements APIs**

```
DatabaseInterfaceServiceInterface::Achievement_GetAllAchievements();

DatabaseInterfaceServiceInterface::Achievement_SetAchievementComplete(
                                                            const std::string
                                                            &achievement_id );
```

- **Analytics APIs**

```
DatabaseInterfaceServiceInterface::Analytics_GameEvent(  const EventData
                                                         &event_data );
```

- **In app purchasing APIs**

```
DatabaseInterfaceServiceInterface::InAppPurchase_GetPurchasedItems();

DatabaseInterfaceServiceInterface::InAppPurchase_GetItemsForPurchase();
```

```
DatabaseInterfaceServiceInterface::InAppPurchase_PurchaseItem(
                                                const std::string &item_id  );

DatabaseInterfaceServiceInterface::InAppPurchase_GetPurchasedItemURL(
                                                const std::string &item_id  );
```

- **General APIs**

```
DatabaseInterfaceServiceInterface::Update ();
```

Before you can make any database service SDK calls you will need to create your `DatabaseInterfaceServiceInterface` object and associate it with your response handler object, which you do by passing your handler to the `DatabaseInterfaceServiceInterface` constructor like so:

```
database_response = new RequestResponse();
database_interface = DatabaseInterfaceServiceInterface::Create(database_response);
```

General APIs

These APIs are to do with making sure your application and the service on the stick are communicating properly.

DatabaseInterfaceServiceInterface::Update( )

Description

C++ users of the SDK need to call this update function periodically, in order to prompt the service to flush and send any pending messages back to you. Technically you only need to do this when you might be expecting a message (in response to a request you have made of the service via an API call); in practice, it is simplest and safest to simply call `DatabaseInterfaceService::Update()` regularly, for example once a frame from your main game loop.

Note that it is very important that you call this update function from the same thread that you created your `DatabaseInterfaceServiceInterface` object via the class constructor.

Parameters

None.

Response

None.

4.2.1.1 Leaderboard APIs

These APIs issue requests to the leader board service. Remember you will have to override the `IRequestResponse` class to handle the responses from the service.

DatabaseInterfaceServiceInterface::LeaderBoard_GetTop50()

Description

Create a request to get top 50 leader-board entries.

Parameters

None.

Response

```
LeaderBoard_GetTop50_Response (const LeaderboardData &data);
```

The data returned to your response handler will be a list of leader board entries which you can iterate through and access information about — see the `LeaderboardData` class reference page.

DatabaseInterfaceServiceInterface::LeaderBoard_GetRange(unsigned from, unsigned to)

Description

Create a request to get a set of leader-board entries.

Parameters

unsigned	from	The highest ranking entry on the leader-board you want to get (1 being the top ranked entry).

Response

```
LeaderBoard_GetRange_Response (const LeaderboardData &data);
```

The data returned to your response handler will be a list of leader board entries which you can iterate through and access information about — see the `LeaderboardData` class reference page.

DatabaseInterfaceServiceInterface::LeaderBoard_SaveScore(unsigned score)

Description

Create a request to set a leader-board entry.

Parameters

unsigned	score	The score to save into the leader-board.

Response

```
LeaderBoard_SaveScore_Response ();
```

The response will not include any data, the response call simply confirms that your request to save the score was successful.

DatabaseInterfaceServiceInterface::LeaderBoard_GetNearest(unsigned range, bool sort_ascending)

Description

Create a request to get a set of leader-board entries near to the current score (i.e., the last score saved with `LeaderBoard_SaveScore`).

Parameters

unsigned	range	The size of the range of nearby values - for example if you want the 5 leaders above and 5 below the last score saved, specify 5 here.
bool	sort_ascending	Specify the order of the sorted entries to be returned.

Response

```
LeaderBoard_GetNearest_Response (const LeaderboardData &data);
```

The data returned to your response handler will be a list of leader board entries which you can iterate through and access information about — see the `LeaderboardData` class reference page.

4.2.1.2 Save State APIs

These APIs issue requests to the Save State module of the service. Remember you will have to override the `IRequestResponse` class to handle any responses from the service.

DatabaseInterfaceServiceInterface::Game_SaveState(unsigned char *data, size_t size)

Description

Create a request to save the current game state data.

Parameters

unsigned char	*data	Pointer to your data (which should be stored as an array of byte/char data).

Response

```
Game_SaveState_Response ();
```

DatabaseInterfaceServiceInterface::Game_LoadState()

Description

Create a request to load the previously saved game state.

Parameters

None.

Response

```
Game_LoadState_Response (const unsigned char *data, size_t size);
```

4.2.1.3 Achievements APIs

These APIs issue requests to the Achievements module of the service. Remember you will have to override the `IRequestResponse` class to handle any responses from the service.

DatabaseInterfaceServiceInterface::AchievementGetAllAchievements()

Description

Create a request to get all game achievements.

Parameters

None.

Response

```
Achievement_GetAllAchievements_Response (const Achievements &data);
```

DatabaseInterfaceServiceInterface::SetAchievementComplete(const std::string &achievement_id)

Description

Create a request to set an achievement as completed.

Parameters

const std::string	&achievement_id	String representing the achievement id.

Response

```
Achievement_SetAchievementComplete_Response ();
```

4.2.1.4 Analytics APIs

These APIs relate to storing data to do with game events.

DatabaseInterfaceServiceInterface::Analytics_GameEvent(const EventData &event_data)

Description

Create a request to set a game event. The service will save the data on the PlayJam server, so you can use this API to store any arbitrary data to do with events during the game which you want to track — for example, you can store something every time a player performs a certain action (reads the instructions, quits a game, or whatever) so you can use this data later to help analyse how people interact with your game.

Parameters

const EventData	&event_data	A set of key-value paired strings representing the game event data.

Note that const EventData is defined in DatabaseInterfaceServiceInterface.h as follows:

```
typedef std::pair <std::string, std::string> EventDataItem;
typedef std::vector <EventDataItem> EventData;
```

This type is used to construct key-value paired strings (i.e., a hash table or hashmap). It is up to you to think of suitable keys for each event and what values to store against each key.

Response

```
Analytics_GameEvent_Response ();
```

No data is returned to the response handler for this request, the response just indicates that the request was received.

4.2.1.5 In-app purchasing APIs

These APIs issue requests to the in-app purchasing module of the service. Remember you will have to override override the `IRequestResponse` class to handle the responses.

The In-App Purchasing Guidelines topic explains what you need to do to support purchasing in more detail.

DatabaseInterfaceServiceInterface::InAppPurchase_GetPurchasedItems()

Description

Create a request to get all purchased items.

Parameters

None.

Response

```
InAppPurchase_GetPurchasedItems_Response (const AppItems &data);
```

The data returned will be a list of all the purchased items, which you can iterate through — see the `AppItems` class reference page.

DatabaseInterfaceServiceInterface::InAppPurchase_GetItemsForPurchase()

Description

Create a request to get all items available for purchase.

Parameters

None.

Response

```
InAppPurchase_GetItemsForPurchase_Response (const AppItems &data);
```

The data returned will be a list of the items available for purchase, which you can iterate through — see the `AppItems` class reference page.

DatabaseInterfaceServiceInterface::InAppPurchase_PurchaseItem(const std::string &item_id)

Description

Create a request to purchase an item.

Parameters

const std::string	item_id	String value representing the item id.

Response

```
InAppPurchase_PurchaseItem_Response (bool bought);
```

The boolean value passed back to your response handler will indicate whether the purchase request was successful.

DatabaseInterfaceServiceInterface::InAppPurchase_GetPurchasedItemURL(const std::string &item_id)

Description

Request the URL of the purchased item.

Parameters

const std::string	item_id	String value representing the item id.

Response

```
InAppPurchase_GetPurchasedItemURL_Response (const std::string &url);
```

4.2.2 IRequestResponse

This interface class provides the methods needed to receive messages back from the service. You need to implement your own object that uses this interface and provides something for each of the following methods:

```
virtual void LeaderBoard_GetTop50_Response (const LeaderboardData &data) = 0;
virtual void LeaderBoard_GetRange_Response (const LeaderboardData &data) = 0;
virtual void LeaderBoard_GetNearest_Response (const LeaderboardData &data) = 0;
virtual void LeaderBoard_SaveScore_Response () = 0;

virtual void Game_SaveState_Response () = 0;
virtual void Game_LoadState_Response (const unsigned char *data, size_t size) = 0;

virtual void Achievement_GetAllAchievements_Response (const Achievements &data) = 0;
virtual void Achievement_SetAchievementComplete_Response () = 0;

virtual void Analytics_GameEvent_Response () = 0;

virtual void InAppPurchase_GetPurchasedItems_Response (const AppItems &data) = 0;
virtual void InAppPurchase_GetItemsForPurchase_Response (const AppItems &data) = 0;
virtual void InAppPurchase_PurchaseItem_Response (bool bought) = 0;
virtual void InAppPurchase_GetPurchasedItemURL_Response (const std::string &url) = 0;
```

```
virtual void RequestFailed_Response (DatabaseInterfaceCalls::Request request,
                                     const std::string &message) = 0;
```

The service will then call your implementations of these methods in order to send information back to your application. Note that the data is often passed via objects such as `Achievements`, `AppItems` and `LeaderboardData` which we have provided to help with parsing the information.

Note that all objects passed as parameters to your response handler methods will be deleted after your method completes, so you should not hold any reference or pointers to these parameters. If you want the data to persist, you will need to copy it to your own data structures somehow.

Error handling

You will need to implement `RequestFailed_Response()` in order to listen out for messages back from the service telling you which request did not work (a string is also returned containing an error message).

The id of the SDK request which failed is passed back via the Request type which is an enum:

```
enum Request
{
   LeaderBoard_GetTop50,
   LeaderBoard_GetRange,
   LeaderBoard_GetNearest,
   LeaderBoard_SaveScore,
   Game_SaveState,
   Game_LoadState,
   Achievement_GetAllAchievements,
   Achievement_SetAchievementComplete,
   Analytics_GameEvent,
   InAppPurchase_GetPurchasedItems,
   InAppPurchase_GetItemsForPurchase,
   InAppPurchase_PurchaseItem,
   InAppPurchase_GetPurchasedItemURL,
   NumberOfRequests
};
```

Example

The following very basic code example shows how you might implement the `IRequestResponse` interface and provide handlers for all the call-backs. Naturally in your own code you will eventually need to do something more complex...

MyRequestResponse.h:

```
#include <iostream>
#include <vector>
#include "databaseinterfaceserviceinterface.h"

using namespace PlayJam;

class RequestResponse :  public DatabaseInterfaceServiceInterface::IRequestResponse
{
```

```
public:
   RequestResponse(void);
   ~RequestResponse(void);

   virtual void LeaderBoard_GetTop50_Response (const LeaderboardData &data);
   virtual void LeaderBoard_GetRange_Response (const LeaderboardData &data);
   virtual void LeaderBoard_GetNearest_Response (const LeaderboardData &data);
   virtual void LeaderBoard_SaveScore_Response ();
   virtual void Game_SaveState_Response ();
   virtual void Game_LoadState_Response (const unsigned char *data, size_t size);
   virtual void Achievement_GetAllAchievements_Response (const Achievements &data);
   virtual void Achievement_SetAchievementComplete_Response ();
   virtual void Analytics_GameEvent_Response ();
   virtual void InAppPurchase_GetPurchasedItems_Response (const AppItems &data);
   virtual void InAppPurchase_GetItemsForPurchase_Response (const AppItems &data);
   virtual void InAppPurchase_PurchaseItem_Response (bool bought);
   virtual void InAppPurchase_GetPurchasedItemURL_Response (const std::string &url);

   virtual void RequestFailed_Response (DatabaseInterfaceCalls::Request request,
                                        const std::string &message);
};
```

MyRequestResponse.cpp:

```
#include "MyRequestResponse.h"


RequestResponse::RequestResponse(void)
{
}


RequestResponse::~RequestResponse(void)
{
}

void RequestResponse::LeaderBoard_GetTop50_Response (const LeaderboardData &data)
{
       std::vector<LeaderboardData::Entry> items = data.GetEntries();

       std::cout << "CallBack LeaderBoard_GetTop50_Response" << std::endl;
       for ( int i = 0; i < items.size(); i++ )
       {
              std::cout << "Item : " << items[i].Name() << std::endl;
              std::cout << "Item : " << items[i].Score() << std::endl;
              std::cout << "Item : " << items[i].AvatarID() << std::endl;
              std::cout << "Item : " << items[i].Position() << std::endl;
       }
}

void RequestResponse::LeaderBoard_GetRange_Response (const LeaderboardData &data)
{
       std::cout << "CallBack LeaderBoard_GetRange_Response" << std::endl;

       std::vector<LeaderboardData::Entry> items = data.GetEntries();

       for ( int i = 0; i < items.size(); i++ )
       {
              std::cout << "Item : " << items[i].Name() << std::endl;
              std::cout << "Item : " << items[i].Score() << std::endl;
              std::cout << "Item : " << items[i].AvatarID() << std::endl;
```

```
                std::cout << "Item : " << items[i].Position() << std::endl;
        }
}

void RequestResponse::LeaderBoard_GetNearest_Response (const LeaderboardData &data)
{
        std::cout << "CallBack LeaderBoard_GetNearest_Response" << std::endl;

        std::vector<LeaderboardData::Entry> items = data.GetEntries();

        for ( int i = 0; i < items.size(); i++ )
        {
                std::cout << "Item : " << items[i].Name() << std::endl;
                std::cout << "Item : " << items[i].Score() << std::endl;
                std::cout << "Item : " << items[i].AvatarID() << std::endl;
                std::cout << "Item : " << items[i].Position() << std::endl;
        }
}

void RequestResponse::LeaderBoard_SaveScore_Response ()
{
        std::cout << "CallBack LeaderBoard_SaveScore_Response" << std::endl;
}

void RequestResponse::Game_SaveState_Response ()
{
        std::cout << "CallBack Game_SaveState_Response" << std::endl;
}

void RequestResponse::Game_LoadState_Response (const unsigned char *data,
                                               size_t size)
{
        std::cout << "CallBack Game_LoadState_Response" << std::endl;
}

void RequestResponse::Achievement_GetAllAchievements_Response (const Achievements &data)
{
        std::cout << "CallBack Achievement_GetAllAchievements_Response" << std::endl;

        std::vector<Achievements::Item> items = data.GetItems();

        for ( int i = 0; i < items.size(); i++ )
        {
                std::cout << "Item : " << items[i].ID() << std::endl;
                std::cout << "Item : " << items[i].Name() << std::endl;
                std::cout << "Item : " << items[i].Description() << std::endl;
                std::cout << "Item : " << items[i].Type() << std::endl;
        }
}

void RequestResponse::Achievement_SetAchievementComplete_Response ()
{
        std::cout << "CallBack Achievement_SetAchievementComplete_Response" << std::endl;
}

void RequestResponse::Analytics_GameEvent_Response ()
{
        std::cout << "CallBack Analytics_GameEvent_Response" << std::endl;
}

void RequestResponse::InAppPurchase_GetPurchasedItems_Response (const AppItems &data)
{
        std::vector<AppItems::Item> items = data.GetItems();

        for ( int i = 0; i < items.size(); i++ )
```

```
	{
		std::cout << "Item : " << std::endl;
	}
	std::cout << "CallBack InAppPurchase_GetPurchasedItems_Response" << std::endl;
}

void RequestResponse::InAppPurchase_GetItemsForPurchase_Response (const AppItems &data)
{
	std::vector<AppItems::Item> items = data.GetItems();

	for ( int i = 0; i < items.size(); i++ )
	{
		std::cout << "Item : " << std::endl;
	}

	std::cout << "CallBack InAppPurchase_GetItemsForPurchase_Response" << std::endl;
}

void RequestResponse::InAppPurchase_PurchaseItem_Response (bool bought)
{
	std::cout << "CallBack InAppPurchase_PurchaseItem_Response" << std::endl;
}

void RequestResponse::InAppPurchase_GetPurchasedItemURL_Response (const std::string &url)
{
	std::cout << "CallBack InAppPurchase_GetPurchasedItemURL_Response" << std::endl;
}

void RequestResponse::RequestFailed_Response (DatabaseInterfaceCalls::Request request,
                                              const std::string &message)
{
	std::cout << "Response : " << request << "Message : " << message << std::endl;
}
```

4.2.3 Achievements

This object wraps the data for achievements that will be returned by the service via the `IRequestResponse` interface you implemented for:

```
virtual void Achievement_GetAllAchievements_Response (const Achievements &data) = 0;
```

Note that an `Achievements` object is returned as a result of the `DatabaseInterfaceServiceInterface::Achievement_GetAllAchievements()` API call. It won't come as a surprise to you then, that the object encodes a list of achievements (actually returned via the service as some JSON, but parsed for your convenience by this class).

`GetItems` — The `GetItems` method is used to access the items in the object.

`Item` — this represents a particular item in the Achievement data (i.e., a single element from the list of achievements).

- `ID` — (int) The achievement id number (allocated when the achievement was defined via the PlayJam Publishing Portal).

- `Name` — (string) Name of the achievement.
- `Description` — (string) Description of the achievement.
- `Type` — (string) The achievement type.
- `FileName` — (string) Filename of some data passed back, associated with the achievement.
- `FileURL` — (string) The URL of some data associated with the achievement.
- `isCurrentUserOwner` — (bool) Indicates whether the achievement was completed by the current user. This will therefore be true for those achievements that you have set to be complete using `DatabaseInterfaceServiceInterface::Achievement_SetAchievementComplete`.
- `XPValue` — (int) The number of experience points to be granted to the player for completing the achievement.

Example

The following very simple example shows how to implement `IRequestResponse` to provide your response handling class, override the *get all achievements* response, and iterate through the data returned (here we just write the data out to the log, of course you will want to do something more intelligent and useful...).

```
#include "databaseinterfaceserviceinterface.h"

using namespace PlayJam;

class RequestResponse :  public DatabaseInterfaceServiceInterface::IRequestResponse
{
public:
       RequestResponse(void);
       ~RequestResponse(void);

   ...

   virtual void Achievement_GetAllAchievements_Response (const Achievements &data);

   ...

   virtual void RequestFailed_Response (DatabaseInterfaceCalls::Request request,
                                        const std::string &message);
};


void RequestResponse::Achievement_GetAllAchievements_Response (const Achievements &data)
{
   std::cout << "CallBack Achievement_GetAllAchievements_Response" << std::endl;

   std::vector<Achievements::Item> items = data.GetItems();

   for ( int i = 0; i < items.size(); i++ )
   {
```

```
        std::cout << "Item : " << items[i].ID() << std::endl;
        std::cout << "Item : " << items[i].Name() << std::endl;
        std::cout << "Item : " << items[i].Description() << std::endl;
        std::cout << "Item : " << items[i].Type() << std::endl;
    }
}
```

4.2.4 AppItems

This object wraps the data for application items that will be returned by the service via the `IRequestResponse` interface you implemented for:

```
virtual void InAppPurchase_GetPurchasedItems_Response (const AppItems &data) = 0;
virtual void InAppPurchase_GetItemsForPurchase_Response (const AppItems &data) = 0;
```

In other words this class is used to represent in-app purchase items you might have defined and which can be returned via the service in response to API calls.

`GetItems` — The `GetItems` method is used to access the items in the object.

`Item` — this represents a particular item in the `AppItems` data (i.e., a single element from the list of application items).

- `ID` — (string) The application application purchase item id.
- `Name` — (string) Name of the application purchase item.
- `Description` — (string) Description of the application purchase item.
- `Cost` — (string) Value representing the cost of the purchase item.

Example

The following very simple example shows how to implement `IRequestResponse` to provide your response handling class, override the achievements response call, and iterate through the achievements data returned (here we just write the data out to the log, of course you will want to do something more interesting).

```
#include "databaseinterfaceserviceinterface.h"

using namespace PlayJam;

class RequestResponse :  public DatabaseInterfaceServiceInterface::IRequestResponse
{
public:
       RequestResponse(void);
       ~RequestResponse(void);
```



```
   ...

   virtual void InAppPurchase_GetPurchasedItems_Response (const AppItems &data);
   virtual void InAppPurchase_GetItemsForPurchase_Response (const AppItems &data);

   ...

   virtual void RequestFailed_Response (DatabaseInterfaceCalls::Request request,
                                        const std::string &message);
};


void RequestResponse::InAppPurchase_GetPurchasedItems_Response (const AppItems &data)
{
   std::vector<AppItems::Item> items = data.GetItems();

   for ( int i = 0; i < items.size(); i++ )
   {
      std::cout << "Item : " << std::endl;
   }
   std::cout << "CallBack InAppPurchase_GetPurchasedItems_Response" << std::endl;
}

void RequestResponse::InAppPurchase_GetItemsForPurchase_Response (const AppItems &data)
{
   std::vector<AppItems::Item> items = data.GetItems();

   for ( int i = 0; i < items.size(); i++ )
   {
      std::cout << "Item : " << std::endl;
   }
   std::cout << "CallBack InAppPurchase_GetItemsForPurchase_Response" << std::endl;
}
```

4.2.5 LeaderboardData

This object wraps the data for leader boards that will be returned by the service via the `IJavaDatabaseInterfaceClass` interface you implemented for:

```
   virtual void LeaderBoard_GetTop50_Response (const LeaderboardData &data) = 0;
   virtual void LeaderBoard_GetRange_Response (const LeaderboardData &data) = 0;
   virtual void LeaderBoard_GetNearest_Response (const LeaderboardData &data) = 0;
```

In each case the value returned as `LeaderboardData` represents a list of items from the leader board (i.e., a list of leader board entries). We provide accessors for parsing the entries to get the name, score, avatar id, and position of each list entry without you having to parse the data yourself (which as always is actually returned as some JSON behind the scenes).

`GetItems` — The `GetItems` method is used to access the items in the object.

`Item` — this represents a particular item in the leader board data (i.e., a single element from the list of leader board entries returned).

- `Name` — (string) The player's name.

- `Score` — (int) the score.
- `AvatarID` — (int) The id of the player's avatar.
- `Position` — (int) The current position on the leader board of that score.

Example

The following very simple example shows how to implement `IRequestResponse` to provide your response handling class, override the *leader board get top 50* response, and iterate through the data returned (here we just write the data out to the log, of course you will want to do something more intelligent and useful...).

```
#include "databaseinterfaceserviceinterface.h"

using namespace PlayJam;

class RequestResponse :  public DatabaseInterfaceServiceInterface::IRequestResponse
{
public:
   RequestResponse(void);
   ~RequestResponse(void);

   ...

   virtual void LeaderBoard_GetTop50_Response (const LeaderboardData &data);
   virtual void LeaderBoard_GetRange_Response (const LeaderboardData &data);
   virtual void LeaderBoard_GetNearest_Response (const LeaderboardData &data);

   ...

   virtual void RequestFailed_Response (DatabaseInterfaceCalls::Request request,
                                        const std::string &message);
};


void RequestResponse::LeaderBoard_GetTop50_Response (const LeaderboardData &data)
{
   std::vector<LeaderboardData::Entry> items = data.GetEntries();

   std::cout << "CallBack LeaderBoard_GetTop50_Response" << std::endl;
   for ( int i = 0; i < items.size(); i++ )
   {
      std::cout << "Item : " << items[i].AvatarID() << std::endl;
      std::cout << "Item : " << items[i].Name() << std::endl;
      std::cout << "Item : " << items[i].Score() << std::endl;
      std::cout << "Item : " << items[i].Position() << std::endl;
   }
}

void RequestResponse::LeaderBoard_GetRange_Response (const LeaderboardData &data)
{
   std::cout << "CallBack LeaderBoard_GetRange_Response" << std::endl;

   std::vector<LeaderboardData::Entry> items = data.GetEntries();

   for ( int i = 0; i < items.size(); i++ )
```

```
    {
        std::cout << "Item : " << items[i].AvatarID() << std::endl;
        std::cout << "Item : " << items[i].Name() << std::endl;
        std::cout << "Item : " << items[i].Score() << std::endl;
        std::cout << "Item : " << items[i].Position() << std::endl;
    }
}

void RequestResponse::LeaderBoard_GetNearest_Response (const LeaderboardData &data)
{
    std::cout << "CallBack LeaderBoard_GetNearest_Response" << std::endl;

    std::vector<LeaderboardData::Entry> items = data.GetEntries();

    for ( int i = 0; i < items.size(); i++ )
    {
        std::cout << "Item : " << items[i].AvatarID() << std::endl;
        std::cout << "Item : " << items[i].Name() << std::endl;
        std::cout << "Item : " << items[i].Score() << std::endl;
        std::cout << "Item : " << items[i].Position() << std::endl;
    }
}
```

4.2.6 DownloadServiceInterface

The download services group of APIs issue requests to the downloads module of the service. Remember you will have to override the `IDownloadResponse` class to handle the responses.

```
DownloadServiceInterface::DownloadPackage( const std::string url );
DownloadServiceInterface::DownloadResource( const std::string url );
```

Before you can make any download service SDK calls you will need to create your `DownloadServiceInterface` object and associate it with your response handler object, which you do by passing your handler to the `DownloadResponse` constructor like so:

```
response = new DownloadResponse();
service_interface = new DownloadServiceInterface(response);
```

DownloadServiceInterface::DownloadPackage(const std::string url)

Description

Requests a package (or large file) download from a server. Progress messages are sent during download. This can be used for larger files such as game levels and has higher priority.

Note that if you make multiple requests to `DownloadPackage()`, the packages will be returned from the service in FIFO (first-in first-out) order, i.e., the packages are returned in the order you requested them.

Parameters

const std::string	url	The file to be downloaded.

Response

One of:

```
DownloadSuccess (const std::string &url, const std::string &destination);
DownloadFailed (const std::string &url, const std::string &message);
```

DownloadServiceInterface::DownloadResource(const std::string url)

Description

Create a request to download a file. This is intended for smaller, lower priority files and doesn't issue progress report messages.

Note that if you make multiple requests to `DownloadResource()`, the files will be returned from the service in FILO (first-in last-out) order, i.e., the most recent request will be downloaded first. This means for example that when you draw a screen full of graphics the last graphic requested will be sent first.

Parameters

const std::string	url	The file to be downloaded.

Response

One of:

```
DownloadSuccess (const std::string &url, const std::string &destination);
DownloadFailed (const std::string &url, const std::string &message);
```

4.2.7 IDownloadResponse

This interface class provides the methods needed to receive messages back from the download service. You need to implement your own object that uses this interface and provides something for each of the following methods:

```
class IDownloadResponse
{
public:
  virtual void DownloadProgress (const std::string &url, int progress) = 0;
  virtual void DownloadSuccess (const std::string &url, const std::string &destination) = 0;
  virtual void DownloadFailed (const std::string &url, const std::string &message) = 0;
};
```

DownloadProgress(String url, int progress)

Description

Will be called by the service when a download progress callback occurs.

Parameters

String	url	The file being downloaded.
int	progress	The amount downloaded so far as a percentage.

Return

None.

DownloadSuccess(String url, String destination)

Description

Will be called when a download success callback occurs.

The information passed to you in the parameters tell you the name of the file (a file you requested earlier via `DownloadServiceInterface::DownloadPackage` or `DownloadServiceInterface::Resource`) and the location on the stick in which the file is (temporarily) located.

Note that the file *will be removed* from the stick when the call to this method ends, so you will need to read the contents or copy it to where you need it in your `DownloadSuccess` method.

Parameters

String	url	The file being downloaded.
String	destination	The temporary local location of the downloaded file.

Return

None.

DownloadFailed(String url, String message)

Description

Will be called when a download failed callback occurs.

The information passed to you in the parameters tell you the name of the file (a file you requested earlier via `DownloadServiceInterface::DownloadPackage` or `DownloadServiceInterface::Resource`) and an error message indicating what went wrong.

Parameters

String	url	The file being downloaded.
String	message	The error message.

Return

None.

4.3 Unity API

Unity developers will access the API via a C# wrapper, so rather than consult the Java or C++ class reference, you should look here as there are some slight differences.

Bear in mind the overall structure of the SDK is that you send messages, via API calls, to a service running on the stick that will use the internet to make a request to the PlayJam server. The responses are then sent by the server and picked up by the service running on the stick. In order to receive the response, you need to implement listening objects in your code, and to do this you use additional classes provided in the SDK.

The data is returned in JSON form — unlike the Java and C++ versions of the SDK, we haven't (yet) included any classes to parse and extract this data for you, so you will have to search the JSON form string that is returned by the service and look for the suitable tags yourself. See the JSON Response Data section for details of the data returned in this way for each request.

The API calls are defined in the following class:

- PlayJamServices

The reponses you need to implement are defined in the following class:

- ServiceResponseHandler

Overview

Before making a request of the service you need to instantiate a `PlayJamServices` object (this is defined in the `PlayJamServices.cs` script which is included in your download package).

`PlayJamServices` provides a set of methods representing service calls / requests (in fact it's a wrapper around the Java API). It's important to remember that `PlayJamServices` is used to make requests — the responses to those requests are managed via the `ServiceResponseHandler` class.

In order to make PlayJamServices requests, you need to instantiate `PlayJamServices` somewhere in your code and configure it via the `PlayJamServices.SetResponseHandler()` method (more below). We can now make service calls from any of Unity's `Update` or `OnGUI` methods in that implementation script.

Next, you will need to extend the `ServiceResponseHandler` class, implementing each of its methods (so, you need an `OnLeaderBoardGetTop50Success()` method to handle the successful results of a call to the `LeaderBoardGetTop50` request, for example). This will handle responses and must be placed on a `GameObject` whose name must be indicated to PlayJamServices via its `SetResponseHandler(string name)` method before any service calls are made (or else responses will not be received).

You will also need to implement a `OnDatabaseFailed()` method to handle a failure request (whether or not you intend to do anything). Make sure you implement each method listed in `ServiceResponseHandler.cs` in your object (even if your methods don't actually do anything yet). Follow the example in `ExampleServiceResponseHandler.cs` (which just shows a simple message for each request).

Override and implement each method listed in `ServiceResponseHandler.cs` in your derived implementation class (any unimplemented responses will be handled by the base class method, which do nothing). The example in `ExampleServiceResponseHandler.cs`, which shows a simple message for each request, is a good place to start.

Provided scripts

In the `Assets/Scripts` folder there are 4 scripts:

- `PlayJamServices.cs`

  This provides the GameStick API functions you can call from Unity via a sealed class `PlayJamServices`. You should not edit this file. Note that you will need the supplied JAR files to be in place in order to work (see the Set Up Guide).

- `ExampleServiceRequests.cs`

  This is a simple example showing the use of the `PlayJamServices` class in your application. This example just creates a `PlayJamServices` object, then passes the object a suitable response handler (which is defined in the `ExampleServiceResponseHandler.cs` script...) and makes some calls to the API in response to standard Unity button-press events.

- `ServiceResponseHandler.cs`

  This provides the response handler class which you must implement to receive and handle messages from the PlayJam services on the GameStick. You will need to use this class in your own code to handle messenges (note that you must implement something for every method in your code, even if your methods do not do anything at first, or there is the potential for run-time errors). Again, do not edit this file.

  Note: `ServiceResponseHandler` extends `MonoBehaviour` and as such, your implementation of it must be dropped onto a `GameObject` in order to receive responses (from Java). This object must be specified to the PlayJamServices instance via `SetResponseHandler` or the constructor (which enables Java to direct the responses to the correct place).

- `ExampleServiceResponseHandler.cs`

  This is an example showing how you might implement a minimal version of the `ServiceResponseHandler` class in your code. Feel free to copy this and use it as a starting point.

In the `ExampleServicesRequests.cs` script, we create a class `ExampleServiceRequests`, based for convenience on the standard `MonoBehaviour` class in Unity.

In this class, we create a new `PlayJamServices` object, and tell it about our response handler:

PlayJamServices services;

```
void Start()
{
     services = new PlayJamServices();
     services.SetResponseHandler("ExampleServiceResponseHandler");
}
```

Our example response handler function will be defined in the `ExampleServiceResponseHandler.cs` file (based on the `ServiceResponseHandler` class).

In the rest of this `ExampleServicesRequests.cs` script we can then make calls to the PlayJam services in response to Unity events (the script contains some simple examples of this).

In the `ExampleServiceResponseHandler.cs` script, we create a `ExampleServiceResponseHandler` class, based on the `ServiceResponseHandler` class defined in `ServiceResponseHandler.cs` script.

```
public class ExampleServiceResponseHandler : ServiceResponseHandler
{
   ...
}
```

Make sure you implement something here for every response, even if it is trivial / empty.

4.3.1 PlayJamServices

The `PlayJamServices` class provides APIs relating to each of the following parts of the PlayJam On-line Services:

- **Leader board APIs**

```
LeaderBoard_GetTop50()
LeaderBoard_GetRange(  int from, int to );
LeaderBoard_SaveScore( int score );
LeaderBoard_GetNearest(  int range, bool sortAscending );
```

- **Save / load game state APIs**

```
Game_SaveState( byte[] data  );
Game_LoadState()
```

- **Achievements APIs**

```
Achievement_GetAllAchievements();
Achievement_SetAchievmentComplete( string id );
```

- **Analytics APIs**

```
Analytics_GameEvent(string hashmap);
```

- **In app purchasing APIs**

```
InAppPurchase_GetPurchasedItems()
InAppPurchase_GetItemsForPurchase()
InAppPurchase_PurchaseItem( string item_id )
InAppPurchase_GetPurchasedItemURL( string item_id )
```

- **Download APIs**

```
DownloadPackage( string url );
DownloadResource( string url );
```

4.3.1.1 Leaderboard APIs

These APIs issue requests to the leader board service. Remember you will have to override the `ServiceResponseHandler` class to handle the responses from the service.

LeaderBoard_GetTop50()

Description

Create a request to get top 50 leader-board entries.

Parameters

None.

Response

```
OnLeaderBoardGetTop50Success(string response)
```

LeaderBoard_GetRange(int from, int to)

Description

Create a request to get a set of leader-board entries.

Parameters

int	from	The highest ranking entry on the leader-board you want to get (1 being the top ranked entry).

Response

```
OnLeaderBoardGetRangeSuccess(string response)
```

LeaderBoard_SaveScore(int score)

Description

Create a request to set a leader-board entry.

Parameters

int	score	The score to save into the leader-board.

Response

```
OnLeaderBoardSaveScoreSuccess(string response)
```

LeaderBoard_GetNearest(int range, boolean sort_ascending)

Description

Create a request to get a set of leader-board entries near to the current score (i.e., the last score saved with `LeaderBoard_SaveScore`).

Parameters

int	range	The size of the range of nearby values - for example if you want the 5 leaders above and 5 below the last score saved, specify 5 here.

Response

```
OnLeaderBoardGetNearestSuccess( string response )
```

4.3.1.2 Save State APIs

These APIs issue requests to the Save State module of the service. Remember you will have to override the ServiceResponseHandler class to handle any responses from the service.

Game_SaveState(byte[] data)

Description

Create a request to save the current game state.

Parameters

byte []	data	Your data (which should be stored as an array of byte/char data).

Response

```
OnGameSaveStateSuccess(string response)
```

Game_LoadState()

Description

Create a request to load the previously saved game state.

Parameters

None.

Response

```
OnGameLoadStateSuccess(string response)
```

4.3.1.3 Achievements APIs

These APIs issue requests to the Achievements module of the service. Remember you will have to override the `ServiceResponseHandler` class to handle the responses.

Achievement_GetAllAchievements()

Description

Create a request to get all game achievements.

Parameters

None.

Response

```
OnAchievementGetAllAchievementsSuccess(string response)
```

Achievement_SetAchievmentComplete(string id)

Description

Create a request to set an achievement as completed.

Parameters

string	id	String representing the achievement id.

Response

```
OnAchievementSetAchievementCompleteSuccess(string response)
```

4.3.1.4 In-app purchasing APIs

These APIs issue requests to the in-app purchasing module of the service. Remember you will have to override the `ServiceResponseHandler` class to handle the responses.

The In-App Purchasing topic explains what you need to do to support purchasing in more detail.

InAppPurchase_GetPurchasedItems()

Description

Create a request to get all purchased items.

Parameters

None.

Response

```
OnInAppPurchaseGetPurchasedItemsSuccess(string response)
```

InAppPurchase_GetItemsForPurchase()

Description

Create a request to get all items available for purchase.

Parameters

None.

Response

```
OnInAppPurchaseGetItemsForPurchaseSuccess(string response)
```

InAppPurchase_PurchaseItem(string item_id)

Description

Create a request to purchase an item.

Parameters

string	item_id	String value representing the item id.

Response

```
OnInAppPurchasePurchaseItemSuccess(string response)
```

InAppPurchase_GetPurchasedItemURL(string item_id)

Description

Request the URL of the purchased item.

Parameters

string	item_id	String value representing the item id.

Response

```
OnInAppPurchaseGetPurchasedItemURLSuccess(string response)
```

4.3.1.5 Analytics APIs

These APIs relate to storing data to do with game events.

The service will save the data on the PlayJam server, so you can use this API to store any arbitrary data to do with events during the game which you want to track - for example, you can store something every time a player performs a certain action (reads the instructions, quits a game, or whatever) so you can use this data later to help analyse how people interact with your game.

Remember you will have to override the `ServiceResponseHandler` class to handle the responses from the service.

Analytics_GameEvent(string hashmap);

Description

Create a request to set a game event.

Parameters

string	hashmap	A set of key-value paired strings representing the game event data.

Response

```
OnAnalyticsGameEventSuccess (string response);
```

No important data is returned to the response handler for this request, the response just indicates that the request was received.

4.3.1.6 Download service APIs

The download services group of APIs issue requests to the downloads module of the service. Remember you will have to provide the appropriate methods in your implementation of the `ServiceResponseHandler` class to handle the responses.

DownloadPackage(string url)

Description

Requests a package (or large file) download from a server. Progress messages are sent during download. This can be used for larger files such as game levels and has higher priority.

Note that if you make multiple requests to `DownloadPackage()`, the packages will be returned from the service in FIFO (first-in first-out) order, i.e., the packages are returned in the order you requested them.

Parameters

string	url	The file to be downloaded.

Response

`OnDownloadSuccess(string response)` or `OnDownloadFailed(string response)`.

DownloadResource(string url)

Description

Create a request to download a file.This is intended for smaller, lower priority files and doesn't issue progress report messages.

Note that if you make multiple requests to `DownloadResource()`, the files will be returned from the service in FILO (first-in last-out) order, i.e., the most recent request will be downloaded first. This means for example that when you draw a screen full of graphics the last graphic requested will be sent first.

Parameters

string	url	The file to be downloaded.

Response

`OnDownloadSuccess(string response)` or `OnDownloadFailed(string response)`.

4.3.2 ServiceResponseHandler

The `ServiceResponseHandler` interface class provides the methods needed to receive messages back from the service. You need to implement your own object that uses this interface and provides something for each of the following methods:

```
public abstract class ServiceResponseHandler : MonoBehaviour
{
   // not an interface, as it must extend MonoBehaviour.
   // members are virtual, not abstract, as user may not wish to override all.

   public virtual void OnLeaderBoardGetTop50Success(string response) {return;}
   public virtual void OnLeaderBoardGetRangeSuccess(string response) {return;}
   public virtual void OnLeaderBoardGetNearestSuccess(string response) {return;}
   public virtual void OnLeaderBoardSaveScoreSuccess(string response) {return;}
   public virtual void OnGameSaveStateSuccess(string response) {return;}
   public virtual void OnGameLoadStateSuccess(string response) {return;}
   public virtual void OnAchievementGetAllAchievementsSuccess(string response) {return;}
   public virtual void OnAchievementSetAchievementCompleteSuccess(string response) {return;}
   public virtual void OnAnalyticsGameEventSuccess(string response) {return;}
   public virtual void OnInAppPurchaseGetPurchasedItemsSuccess(string response) {return;}
   public virtual void OnInAppPurchaseGetItemsForPurchaseSuccess(string response) {return;}
   public virtual void OnInAppPurchasePurchaseItemSuccess(string response) {return;}
   public virtual void OnInAppPurchaseGetPurchasedItemURLSuccess(string response) {return;}
```

```
    public virtual void OnDatabaseFailed(string response) {return;}

    public virtual void OnDownloadProgress(string response) {return;}
    public virtual void OnDownloadSuccess(string response) {return;}
    public virtual void OnDownloadFailed(string response) {return;}
}
```

You can use the code in `ExampleServiceResponseHandler.cs` as a starting point.

In the `ExampleServicesRequests.cs` script, we create a class `ExampleServiceRequests`, based for convenience on the standard `MonoBehaviour` class in Unity.

In this class, we create a new PlayJamServices object, and tell it about our response handler:

PlayJamServices services;

```
void Start()
{
     services = new PlayJamServices();
     services.SetResponseHandler("ExampleServiceResponseHandler");
}
```

Our example response handler function will be defined in the `ExampleServiceResponseHandler.cs` file (based on the `ServiceResponseHandler` class).

In the rest of this `ExampleServicesRequests.cs` script we can then make calls to the PlayJam services in response to Unity events (the script contains some simple examples of this).

In the `ExampleServiceResponseHandler.cs` script, we create a `ExampleServiceResponseHandler` class, based on the `ServiceResponseHandler` class defined in `ServiceResponseHandler.cs` script.

```
public class ExampleServiceResponseHandler : ServiceResponseHandler
{
   ...
}
```

Make sure you implement something here for every response, even if it is trivial / empty.

Note that in all cases, the data returned from the service via the handler call-back functions will be a string containing data in JSON format. Currently the Unity implementation of the GameStick SDK does not contain any helper classes to parse this JSON for you, so you will have to extract the relevant information in your own code.

4.3.3 JSON Response Data

The data passed back from the service to your response handler functions will, for Unity users, be in JSON form which you will have to parse yourself (when time allows we will add classes to help with this).

It shouldn't be too hard to extract the relevant information for each response, you will just need to match the appropriate tags and get the corresponding value. We've listed the tags used in different types of response data below.

Achievement responses

The responses to request involving in-game achievements will contain the following tags:

- `"id"` — Integer representing the achievement id number (allocated when the achievement was defined via the PlayJam Developer App).
- `"achievementName"` — String name of the achievement.
- `"description"` — String description of the achievement.
- `"achievementType"` — String representing the achievement type.
- `"fileName"` — String representing the location of some data associated with the achievement.
- `"fileUrl"` — String representing the URL of some data associated with the achievement.
- `"isCurrentUserOwner"` — Boolean representing whether the achievement was completed by the current user. This will therefore be true for those achievements that you have set to be complete using `PlayJamServices.Achievement_SetAchievmentComplete()`.
- `"xpValue"` — Integer representing the number of experience points to be granted to the player for completing the achievement.

Leaderboard responses

The responses to request involving on line leader boards will contain the following tags:

- `"userPlayTag"` — String representing the player's name.
- `"score"` — Integer value giving the score.
- `"avatarId"` — Integer value representing the id of the player's avatar.
- `"position"` — Integer value representing the current position on the leader board of that score.

AppItems (purchase items) responses

The responses to request involving in-game purchases will contain the following tags:

- `"id"` — String representing the application application purchase item id.
- `"name"` — String name of the application purchase item.
- `"description"` — String description of the application purchase item.

- `"cost"` — Float value representing the cost of the purchase item.