Table of Contents

Objective

In order to support most of the requested features we need some changes on the East2West solution. So, this document will try to explain what we need and how we need it for each feature so we could work together.

We will order the features in terms of how quick and easy they are to implement so we could deliver them one by one so you could test how they perform in terms of players' feedback and revenue.

Feature: Better Rating

We already support asking for rating in the game in Google Play and iTunes by asking the player to leave a feedback after some time of playing the game and after a specific Stage.

So, in order to make this feature available to you, we need a way to get the feedback URL of each channel. That is something that could be configured on your side completely with a few changes.

Extend the E2W wrapper with methods like:

```
IsRatingAvailable() : bool - Returns true if rating is available, false otherwise (in
case you don't have a url for rating you can return false).
```

```
GetRatingURL() : string - Returns the URL to Open for the rating.
```

Both suggested methods should be configured and implemented on your side the way you want, for example, adding configuration to the AndroidManifest.xml, as you do for the channel name, and then returning that to us.

The main thing here is that we are going to ask you about the URL like we did with other stuff like the login, etc.

[EASY]

Feature: Remote Configuration & Offers

Currently, the only remote configuration interaction we have with your servers is for asking the list of IAPs and their prices.

Inside the game we support more stuff for configuring remotely and we use remote services like Firebase, etc, to configure the game. Some examples of what we configure are: showing ads or not, a list of possible offers, in which stage show the rating request, etc.

In order to allow you to configure all the remote config we have, first we have to explain the format we use, what each value is for, and then you have to extend your server to support this kind of configuration.

Even though we have the list of available IAPs in the remote configuration, what we don't have is the list of prices for each IAP, that is requested to the store itself, like Google Play or iTunes. I believe we will have to keep the same idea, so we would have to ask for the configuration and, separately, the list of iaps with prices to simplify behaviour changes in the game.

So, some proposed methods for you to implement are:

`GetIAPs(Callback<int, string> callback) : void` - Attempts to get the list of IAPs with their description (price, name, etc). It should call the callback with a string with the JSON with the IAPs.

The IAPs list should be the same format we are using right now with you, that is not a problem, the main difference is we are going to delegate the server interaction to the East2West wrapper instead of doing it ourselves. We are still responsible of processing the JSON result. In that way, you could have more control of which server to request info from, and if you want to send extra parameters or headers with the request to be able to do A/B tests.

`GetRemoteValuesWithPrefix(string keyPrefix, Callback<int, List<string>> callback) : void` - We ask for all the remote configuration values identified by keys starting with the specified prefix. The callback should be called with the proper status code if something went wrong.

`GetRemoteValue(string key, Callback<int, string>) : void` - Returns the specific value for the specified key. Used for example to get unique values like `ads_enabled` (explained later).

For example, we could ask for values for each special offer, like:

```
GetRemoteValuesWithPrefix("offer_special_", callback);
```

That should return a list of strings with the JSONs of each value identified by keys starting with "offer_special_" (the formats are explained later).

This is a similar API behaviour to how Firebase works and would make so much easier for us to integrate.

The Remote Configuration format is explained here. Inside that configuration, there are the offers, explained here.

From our side, what we do is try to override the default configuration (shipped with the game) with remote values. If don't find a remote value when overriding, we use the default one. For this reason, we suggest you to store the last remote configuration in a local store (from the android side) and return that one if you failed to read a new one from the server, some kind of cache.

If you implement A/B testing as suggested, different remote configuration values should be returned to user given the corresponding user group.

[HARD]

Feature: Cloud Save

First of all, cloud save is related to an account in a service where we can write and read data. In Google, we are using Google Play Games Services which allow saving data for a user signed in Google Play in the game. For iOS, we are using iCloud and using the iCloud account of the player.

In order to support that in China, we have to find one or more services (for different channels) and provide both sign in and cloud save. I believe the best way is you handle the middle layer in the East2West wrapper like we did with everything else and we depend on that layer like we do with Google Play and iCloud.

So the suggested API for E2W:

```
Login(Callback<int> callback) : void - Starts a sign in attempt, delegating
completely to the wrapper and the current service (this will probably show the
sign in window for that service).
```

```
Logout() : void - Signs out from the service. Cloud save will be disabled
after this until a new sign in.
```

(Note: we already have these two with what you did for WeChat and QQ)

```
Load(string name, Callback<int, string> callback) : void - Starts a read data
attempt at the specified file name. It should call the callback with status ok
or error, and with the file contents (probably a json).
```

```
Save(string name, string data, Callback<int> callback) : void - Starts a write
data attempt at the specified file name, it should try to write the specified
data and then call the callback with the status code ok or error depending
what happened.
```

Since most of the methods require an interaction with a server, all of the would be async, that's why I propose using callbacks for everything.

In that way, you could then create the specific implementation for each service you want to use for each channel and we should be able to support it.

About authentication, with Google Play, what we normally do is to automatically try to sign in the first time the game starts. If the player chooses to not sign in or sings out at some point, we save that and we don't try again an automatic sign in until the player goes to the Settings and signs in manually.

However, what we are going to do for E2W, to simplify, is add an extra button to the Settings screen with a custom login/logout of the cloud service (similar to what we do with iCloud for iOS). If the player was signed in once, then we will try to auto sign in each time the player opens the game until he signs out.

In the case of WeChat/QQ, we are already have login/logout implemented. However, if they provide a way to store data, and you want to use that service, we could disable the extra cloud button from the settings screen for that channel, and you only have to implement the load/save methods using WeChat/QQ store data service and the game will almost not know about that.

[MEDIUM]

Feature: Rewarded Videos

This basic version of video ads provides some credits as reward for watching a video.

Extend the East2West wrapper with these methods:

```
IsVideoAdAvailable() : bool - return whether there is a video available (consider caching this value since we might check it frequently). Also, it should return false if the current channel doesn't support ads.
```

```
ShowVideoAd(Callback<int> callback) : void - Shows the video Ad if available, and call the callback once the video was completed with proper status code (video completed, video failed, video canceled, etc) so we can provide the rewards.
```

In any case, the main thing is you have a consistent API between all the channels and also to allow us in some way to check if ads are available, to show a video ad and to know if the video ad was completed properly in order to give the player the reward.

With the Remote Configuration you could change the rewards for watching a video, it is explained in the last section of the document.

[EASY]

Feature: Custom Unlock screen

First of all, the price was always customizable from your side as it is an IAP like the others.

For the other features, what we have to change is:

- Remove the text from the image and make it customizable by strings.
- Add a value to the Remote Configuration to set the campaign stage number where the locked badge should be.
    - We have to adjust this on our side to support changing the stage with the locked badge.
    - And also to force show the locked badge in the next stage if you change to a stage lesser than the last played one. For example, if the locked stage is the stage 5, and people played until stage 4, and then you change it to stage 3, they will not see the locked badge and will continue playing
- To enable/disable the unlock screen, we could define a special value like -1 or 9999 to disable the unlock screen through the previous config.
- Change the extra rewards values (we reward the player with some credits after unlocking the game, we could change it to be customizable).

Right now, even if we remove texts from the image and change to strings, we still have to change that from our side or you have to change it by modifying the strings file yourselves. Just propose which texts could work better and we would put that in the default strings.

[DEPENDS ON RemoteConfiguration FEATURE]

[EASY]

Feature: A/B Testing

The first way to do A/B testing is, since you have different channels, I believe you could do A/B testing configuring differently each channel and see the differences. That could be one way of doing A/B testing.

The second way is to use the same channel and make changes in time, and then compare, so, try with A for a week and then B for a week, multiply by users during week, etc, compare.

The third way is to do the A/B testing in the same channel. For that case, we need to first make the proposed Remote Configuration changes and after that, it will be almost completely on your side.

What Firebase does to be able to do A/B testing is assigning a user percentile from the app side and then send that when requesting data like the remote configuration. Then the server, depending the percentile assigned returns the proper configuration.

So you could, from the East2West wrapper, create that random percentile the first time the user start the app (if it wasn't created) and then use it for all request to your server. So you could gather the config from a server for us the way you want, using the custom parameters you want.

This is just an idea, we could talk if there is a better API for the remote config. This depends on adding support to Remote Configuration.

[DEPENDS ON RemoteConfiguration FEATURE]

Appendix

Status Codes

There are several methods in the API proposal that expect a status code (int) to specify if the operation was successful or not. I suppose we could assume something like:

- `0 - no error`
- `1 - generic error`

And then we can start expanding as we need to identify specific cases we want to react in a different way. Only if we need.

Remote Configuration Format

We are going to explain the list of keys we could ask for remote config and the possible values.

- `credits_pack_most_popular: string` - the identifier of the credits pack that should have the most popular badge.
- `credits_pack_best_value: string` - the identifier of the credits pack that should have the best value badge.
- `ads_enabled : bool -` if ads are enabled or not (this is to temporary disable the ads from remote config if something is not working).
- `ads_map_button_enabled : bool -` To enable/disable the ads button in the map itself (ads inside the credits store will continue working).
- `ratingSettings : string -` A JSON with the configuration for ratings setting (format explained later).
- `"iaps" : string -` a JSON with a list of available IAP identifiers (format explained later)
- `specialOffer : object -` a JSON with the data of the offer, explained here.
- `heroSale : object -` a JSON with the configuration for showing heroes in sale in the hero room (format explained later).

Rating settings

These are the values to configure the behaviour of the rating dialog.

A JSON with the format:

```
"ratingSettings" : {
    "minStage" : 4,
    "minTimeAfterLaunchInMinutes" : 1,
    "minTimeAfterPostponeInMinutes" : 180
}
```

Where:

- `"minStage" : int -` We starts showing the rating dialog after this stage, never before. Could be 0 to disable this logic.
- `"minTimeAfterLaunchInMinutes" : int -` Time in minutes to avoid showing the dialog after the player open the game.
- `"minTimeAfterPostponeInMinutes" : int -` Time in minutes we avoid showing the dialog again after the player pressed "Later" in the dialog.

IAP identifiers list

This is the list of available IAP identifiers, we use it to then request the Store for product information on each IAP, and also to disable game products, if one IAP is not here, then we don't show the corresponding product in the game. Example of configuration:

```
"iaps" : [
    "com.ironhidegames.ironmarines.hero_mercenary",
    "com.ironhidegames.ironmarines.hero_mecha",
    "com.ironhidegames.ironmarines.hero_shatra",
    "com.ironhidegames.ironmarines.hero_dr_graff",
    "com.ironhidegames.ironmarines.hero_darwin",
    "com.ironhidegames.ironmarines.hero_tank",
    "com.ironhidegames.ironmarines.hero_paifeng",
    "com.ironhidegames.ironmarines.hero_trabuco",
    "com.ironhidegames.ironmarines.credits_pack1",
    "com.ironhidegames.ironmarines.credits_pack2",
    "com.ironhidegames.ironmarines.credits_pack3",
    "com.ironhidegames.ironmarines.credits_pack4",
    "com.ironhidegames.ironmarines.credits_pack5",
    "com.ironhidegames.ironmarines.credits_pack6",
    "com.ironhidegames.ironmarines.unlock_campaign"
 ]
```

Special Offers

This is used to configure optional special offers. This is an example of the special offer for all heroes:

```
{
    "identifier": "offer_special_all_heroes",
    "iapIdentifier": "com.ironhidegames.ironmarines.special_pack1",
    "offerType": 0,
    "offerCost": {
        "currencyType": 0,
        "cost": 0
    }   ,
    "rewards": [{
        "identifier": "all_heroes",
        "rewardType": 4,
        "count": 1
    } ],
    "metadata": {
        "value": "2",
        "offerType": "all_heroes",
        "duration": "0.12:59:59",
        "unique": "true",
        "checkPurchasedHeroes":"false"
    }
}
```

Where:

- `identifier : string -` The identifier of this special offer. All special offers should start with `"offer_special_"` prefix, since that is used by the remote configurator (explained before).
- `iapIdentifier : string -` The IAP identifier, used when start a purchase attempt with the store.
- `offerType : int -` The type of offer, could be:
    - 0 - hard currency (real money, this is the default case)
    - 1 - virtual currency (to buy stuff with credits, we never implemented it) [NOT IMPLEMENTED]
    - 2 - free (we used this on launch to give the mercenary hero for free)
- `offerCost : object -` The cost of the offer, in the proper currency type. It was meant to be used for virtual currency, leave default values [NOT IMPLEMENTED].
- `rewards - object -` a JSON with the list of rewards of the offer, should have at least 1 and max 3 offers. The format is explained later.
- `metadata - object -` a JSON with optional behaviour configuration, like when should the offer start showing and the duration, etc.

    - `value : int` - The value to show in the card, like how good the offer is, we show it as "x2 value" in the corner of the offer card.
    - `startingDate : date` – (optional) the date to start showing the offer.
    - `endingDate : date` – (opcional) the ending date until we show the offer.
    - `duration: time` – the total duration of the offer after it was shown for the first time.
    - `unique: bool` – If we don't allow to buy the offer again or not, true by default. We use the unique special offer identifier to check this. So each new offer you create, you have to create a new identifier if you want them to be unique.
    - `offerType: int` – the type of offer, mainly used for special occasions, like halloween, etc, so the offer shows special visual stuff (not all of them have visual customization. The possible types are:
        - `special`
        - `starter`
        - `gift`
        - `holyday_halloween`
        - `holyday_blackfriday`
        - `holyday_christmas`
        - `holyday_newyear`
        - `holyday_lunarfestival`
        - `holyday_summersale`
        - `holyday_ironhideday`
    - `checkPurchasedHeroes: bool` – Checks if the player already purchased a hero (which is not consumable) in the current offer to show it. Use `true` to avoid showing offers with purchased heroes, `false` if you still want to show the offer (we use this one mainly for the all heroes offer).

Rewards

These are the possible rewards for the special offer, we allow at least one and at most 3. Each reward is configured in the following way:

- `identifier: string` – Specifies the specific reward, for example, if rewardType is a hero, the identifier is the specific hero. Possible identifiers are explained later.
- `rewardType: int` – Specifies the type of reward, here is the list:
    - `0 - hero`
    - `1 - power (item)`
    - `2 - credits`
    - `3 - techpoints`
    - `4 - all heroes`
    - `5 - unlock content (only used in E2W version)`
- `count: int` – the count of rewarded powers (items), credits or techpoints, only used in the case of consumables.

These are the possible identifiers for each rewardType.

For rewardType hero:

- `hero_mercenary`
- `hero_mecha`
- `hero_dr_graff`
- `hero_darwin`
- `hero_shatra`
- `hero_tank`
- `hero_paifeng`
- `hero_trabuco`

For rewardType power:

- `power_titandoom`
- `power_orbital_strike`
- `power_etherium_surge`
- `power_mortarbros`
- `power_proximity_bollard`
- `power_accumulator`
- `power_replicant_mines`
- `power_frostbite_bomb`

For rewardType credits:

- `offer_credits_pack1`
- `offer_credits_pack2`
- `offer_credits_pack3`
- `offer_credits_pack4`
- `offer_credits_pack5`
- `offer_credits_pack6`

For rewardType techpoints:

- `offer_techpoints_pack1`

For rewardType all heroes:

- `all_heroes`

For rewardType unlock content:

- `content_campaign`

Hero Sales

This configuration is used mainly to show discounts in the hero room if you want to let the user know a hero is on sale. You still need to adjust the price of the hero on your side, but with this you can show in the game a discount badge. Here is an example configuration:

```
"heroSale" : {
    "heroes":[
        {
            "heroKey":"hero_shatra",
            "discount":50
        },
        {
            "heroKey":"hero_tank",
            "discount":25
        }
        ],
    "metadata":{
        "startingDate":"07/01/2017 09:05:33",
        "duration":"1.00:00:00"
        }
}
```

- `heroes` : a list of json objects specifying the hero identifier and the discount percentage, in the example above, Sha'Tra is discounted 50% and the Tank is discounted 25%.
    - `heroKey : string` – the identifier of the hero (explained with the rewards).
    - `discount : int` – the number to show in the discount badge.
- `metadata`: optional values to define the discount behaviour.
    - `startingDate : date` – when the discount badges should start showing (before that day will not be visible).
    - `duration: time` – the duration while the discount badges should be visible after startingDate.

Overriding default configurations

You can override other configurations like how much credits each credits pack should provide. In order to do that you have to add configurations for default stuff included in the game but with new values. It uses the same format of special offers but with specific identifiers,

Here are two examples of overriding default config.

Overriding how much the ads credits pack provides:

```
{
    "identifier":"offer_credits_pack_ads",
    "iapIdentifier":null,
    "offerType":1,
    "offerCost":{
        "currencyType":3,
        "cost":0
    },
    "rewards":[
        {
        "identifier":"offer_credits_pack_ads",
        "rewardType":2,
        "count":120
        }
    ],
    "metadata":{

    }
}
```

If you want to override how much the sixth credits pack provide:

```
{
    "identifier":"offer_credits_pack6",
    "iapIdentifier":"com.ironhidegames.ironmarines.credits_pack6",
    "offerType":0,
    "offerCost":{
        "currencyType":0,
        "cost":0
    },
    "rewards":[
        {
        "identifier":"offer_credits_pack6",
        "rewardType":2,
        "count":55212
        }
    ],
    "metadata":{

    }
}
```