The *`window.TJ_API`* object

This documents the standard Javascript API for 3rd party playable content integrations with Tapjoy's product line. The API constitutes the official lines of communication between a Tapjoy ad's webview and any 3rd party Javascript running as part of said ad.

Tapjoy has made this API available in the `` `TJ_API` `` variable on the parent window of playable units.

```
window.TJ_API;
```

Runtime data

`adInfo`

Object. Miscellaneous data provided from Tapjoy to the 3rd party content about the running ad. The contained data is guaranteed to be static, **but may be null.**

`reward`

String. The amount of in-game currency the user will be rewarded after completing the objective of the playable ad.

`currencyName`

String. The display name of the in-game currency in which the user reward is measured.

```
if (window.TJ_API) {
  let adInfo = window.TJ_API.adInfo;
  text = `You’ve won ${adInfo.reward} ${adInfo.currencyName}!`
}
```

`directives`

Object. Directives given to the 3rd party content for configuring supported features. The contained data is guaranteed to be static, and will be made available to the unit when it is initialized, ensuring the maximum amount of time is given for the unit to make use of them.

`showEndCard`

Boolean. If true, indicates that the playable unit is responsible for displaying an end card. If false, the playable should not display its own end card. This information should be used to drive a smooth transition between the gameplay and a Tapjoy end card.

```
if (gameOver) {
  // Signal to Tapjoy that the gameplay is finished.
  window.TJ_API && window.TJ_API.gameplayFinished();
  if (window.TJ_API &&
      window.TJ_API.directives.showEndCard) {
    // render end card
  } else { /* prepare for Tapjoy endcard */ }
}
```

Required API Functions

`setPlayableAPI(interface)`

Function. Defines a set of callbacks acknowledged by Tapjoy and **optionally** implemented by the content provider, allowing Tapjoy's ad client to hook into and control the behavior of the 3rd party content.

Parameters

`interface`	*Object.* Tapjoy recognizes specific functions on this object, but makes no guarantee that they will be called.

Recognized interface functions

`skipAd`
Function. Skips over the remainder of the playable to its internal end card or whatever internal mechanism marks the end of the playable experience. Providing this function allows Tapjoy to use the playable in our interstitial supply, **and '`objectiveComplete`' is implied by its invocation.**

Usage

To be set on initialization of the playable, prior to the beginning of user engagement.

```
(window.TJ_API &&
 window.TJ_API.setPlayableAPI({
   skipAd: function() { /* go to the end card */ }
 }));
```

For more details on how you might implement `` `skipAd` `` in real life, check out this example.

`click()`

Function. Communicates the user has answered a call-to-action (CTA), and **terminates the playable unit**. As such, its occurrence implies that:

- the gameplay objective has been completed, and
- the gameplay has finished.

Only the first signal of this kind will have an effect.

Usage

Hook this up to any event that indicates the user has answered a call-to-action. The exact behavior of this CTA event is defined by the Tapjoy ad setup, but it will always result in the end of the playable unit.

```
if (userAnsweredCTA) window.TJ_API && window.TJ_API.click();
```

For further details on how you might use this in combination with the other lifecycle events, see the examples section.

`objectiveComplete()` *[formerly '`playableFinished`']*

⚠ Units with `playableFinished` will continue to work, but going forward `objectiveComplete` should be used.

Function. Communicates the main objective of the playable has been completed, but not necessarily that gameplay has finished: the user may continue to engage with the content.

Only the first signal of this kind will have an effect.

Usage

Hook this up to any action or event which indicates the main objective of the playable has been completed. For example, you may choose to use it when a built-in end card appears, or after a certain amount of time has passed, or perhaps when the user score reaches a predefined total.

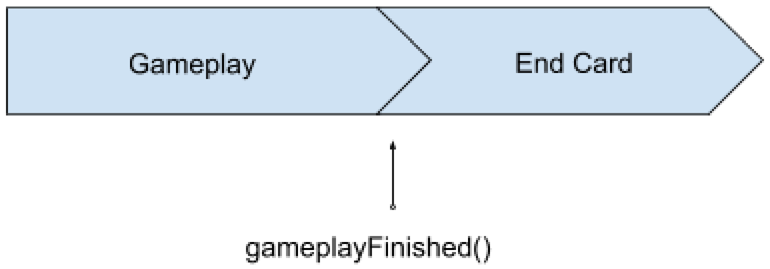
```
if (userReached1000Points) {
  window.TJ_API && window.TJ_API.objectiveComplete();
}
```

For further details on how you might use this in combination with the other lifecycle events, see the examples section.

`gameplayFinished()`

Function. Communicates the gameplay phase of the playable is over, and the user may no longer engage with the playable content.

Usage



We can generalize a playable unit into two distinct phases, the ‘gameplay’ and the ‘end card’; gameplay is finished when it is time to transition to the end card.

For playables that will show a Tapjoy end card (see [`directives#showEndCard`](directives#showEndCard)), `` `gameplayFinished` `` must be signaled to indicate that the end card should be shown.

```
if (gameIsOver) {
  window.TJ_API && window.TJ_API.gameplayFinished();
}
```

Optional API Functions

setPlayableBuild(*buildID*)

Function. Communicates the ID of this particular build/version/release of the playable.

Parameters

`buildID`	*String.* A unique identifier that varies across builds of the playable.

Usage

To be called on initialization of the playable; we will use this information for debugging purposes if users experience problems on certain versions of the playable.

You can use your own naming convention, as long as you guarantee that every unit, and every version of the same unit, has a unique build identifier.

```
(window.TJ_API &&
 window.TJ_API.setPlayableBuild( “abc-123 v4.5” ));
```

`error(`*`cause`*`)`

Function. Communicates a **fatal** error which occurred during the lifetime of the playable, ‘fatal’ meaning one that breaks the experience for the user or makes the ad unusable.

Only the first signal of this kind will have an effect, and its use **will trigger the end of the ad experience.**

Parameters

`cause`	*String.* Any available information regarding the cause of the error.

Usage

```
(window.TJ_API &&
 window.TJ_API.error( “Failed to download important assets” ));
```

Examples

Example: `setPlayableAPI#skipAd`

```javascript
if (window.TJ_API) {
  var AdAPI = {
    skipAd: function() {
      try {
        // Save the configured delay for automatically showing the endcard
        // if the user has been idle long enough.
        var endcardOnIdleDelay = AwesomeConfig.endcardOnIdleDelay;

        // Skip to the endcard by modifying the value being used by some
        // background countdown process.
        AwesomeConfig.endcardOnIdleDelay = 200;
        window.requestAnimationFrame(function() {
          // Restore the original configured value: we were never even here....
          AwesomeConfig.endcardOnIdleDelay = endcardOnIdleDelay;
        })

        // Log the event, if we care.
        AwesomeAnalytics.logAwesomeEvent("ad_skipped");
      } catch (e) {
        console.warn("Could not skip ad! | " + e);
      }
    }
  };

  // Give the `skipAd` function to Tapjoy if we can.
  window.TJ_API.setPlayableAPI
    ? window.TJ_API.setPlayableAPI(AdAPI)
    // This scenario shouldn't happen in real mobile ads, but if you're previewing
    // this playable in, say, a browser, this little hack let's you test out
    // your implementation of `skipAd` by pressing the <ESC> key.
    : window.onkeyup = function(keyEvent) {
      "Escape" === keyEvent.code && AdAPI.skipAd();
    };
}
```

Example: API usage scenarios

Figure 1. Ad experience with internal end card

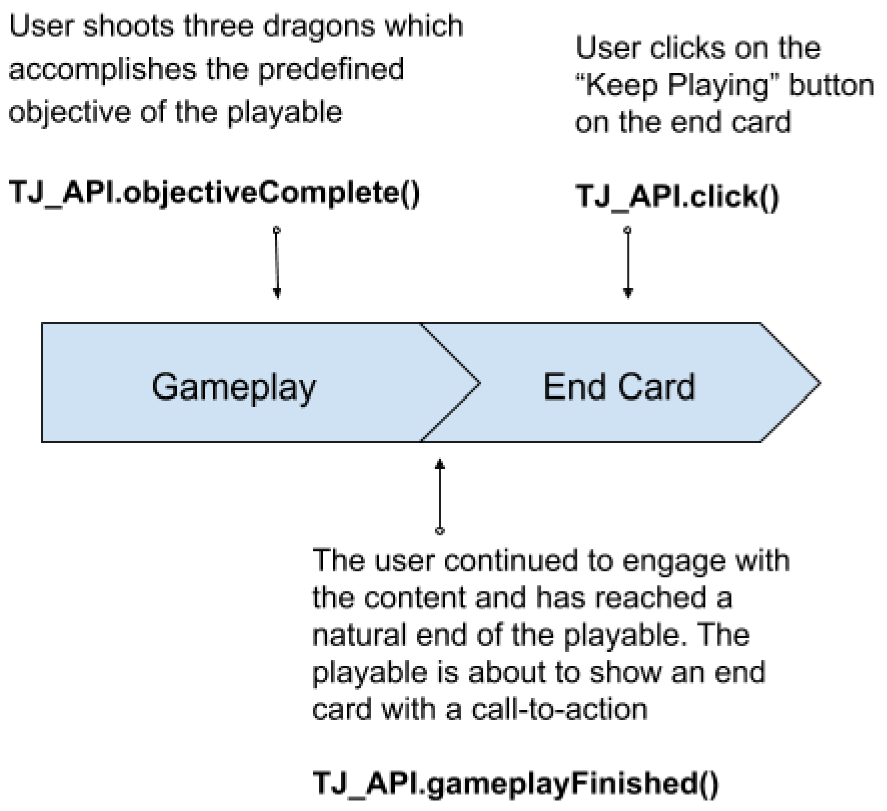
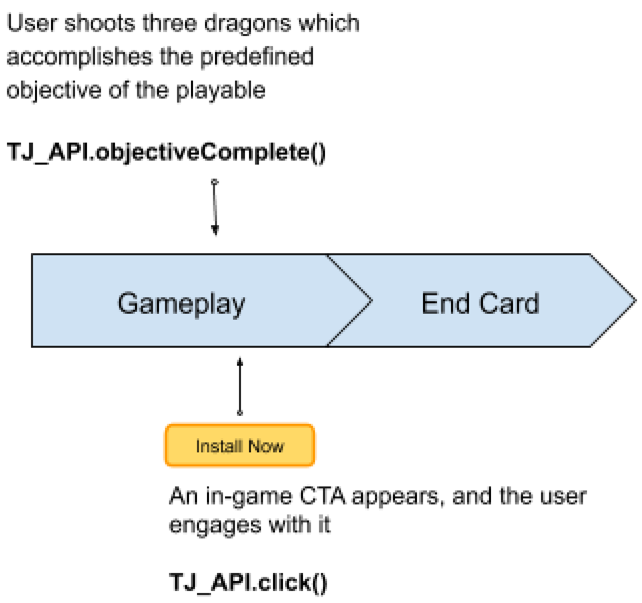

User shoots three dragons which accomplishes the predefined objective of the playable
TJ_API.objectiveComplete()
User clicks on the "Keep Playing" button on the end card
TJ_API.click()
Gameplay
End Card
The user continued to engage with the content and has reached a natural end of the playable. The playable is about to show an end card with a call-to-action
TJ_API.gameplayFinished()

Figure 2. Ad experience with an in-game call-to-action


User shoots three dragons which accomplishes the predefined objective of the playable
TJ_API.objectiveComplete()
Gameplay
End Card
Install Now
An in-game CTA appears, and the user engages with it
TJ_API.click()

In this example we do not expect a `gameplayFinished` signal since the `click` triggers the end of the playable unit; the conclusion of gameplay is implied in this case.

Figure 3. Ad experience where user does not engage with the ad

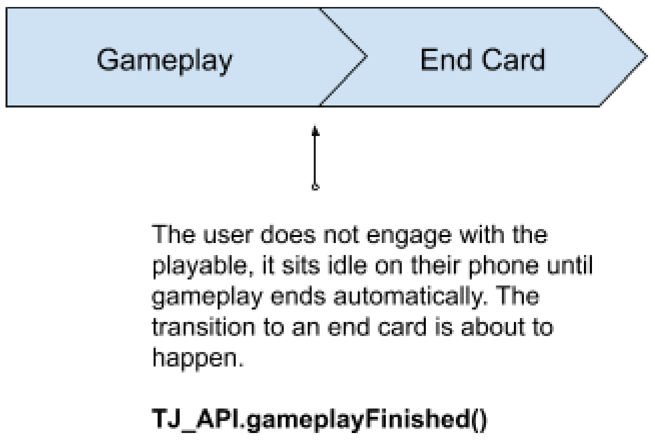
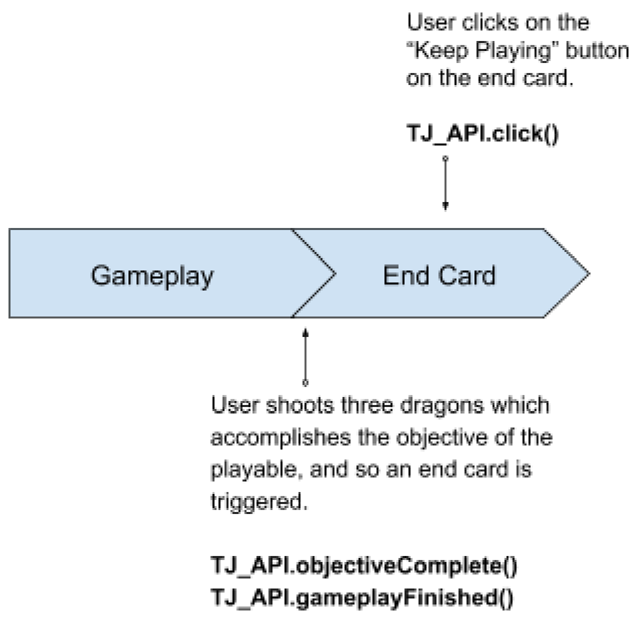
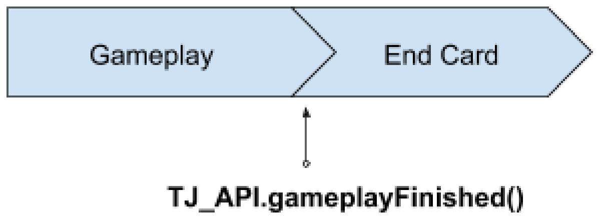

Gameplay
End Card
The user does not engage with the playable, it sits idle on their phone until gameplay ends automatically. The transition to an end card is about to happen.
TJ_API.gameplayFinished()

Figure 4. Ad experience where user completes an end card call-to-action


User clicks on the "Keep Playing" button on the end card.
TJ_API.click()
Gameplay
End Card
User shoots three dragons which accomplishes the objective of the playable, and so an end card is triggered.
TJ_API.objectiveComplete()
TJ_API.gameplayFinished()

In this example, the user completing the objective is what triggers the end card to appear. We would expect `objectiveComplete` and `gameplayFinished` to be signaled one after another.

Figure 5. Ad experience that supports Tapjoy end cards



```
if (window.TJ_API.directives &&
    window.TJ_API.directives.showEndCard) {
  // show end card
} else {
  // pause on last screen of playable
}
```

Playables that support Tapjoy displaying an end card should check `directives#showEndCard` to see if they should display an end card or not.

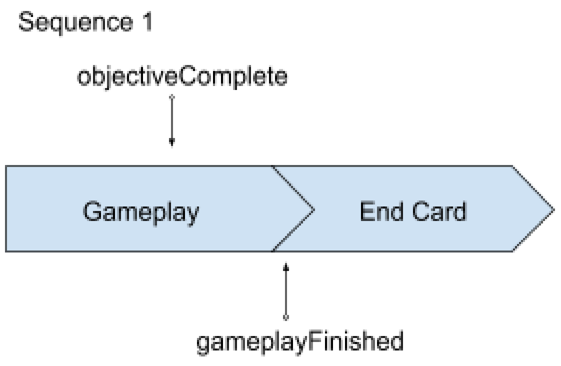
The `gameplayFinished` life cycle event will signal to Tapjoy to display their end card.

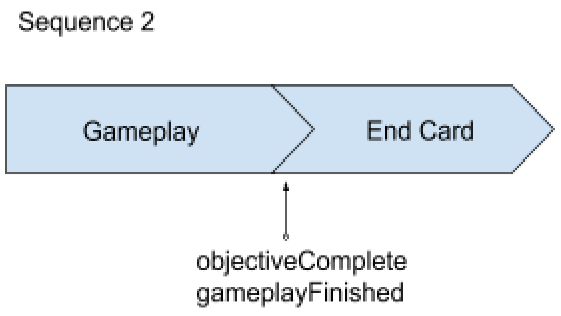
FAQ

What is the difference between `objectiveComplete` and `gameplayFinished`?

These two events are related, but represent fundamentally different things. Consider a playable unit representing an actual level of the game being advertised: the advertiser may consider it "good enough" for the user to accumulate a certain number of points, but only transition to an end card at the end of the level, allowing the user to keep playing through to the end. "Completing the objective", in this case, would be reaching a certain score, while "finishing gameplay" would be reaching the end of the level.

Alternatively, the advertiser may think finishing the game should be the main way to complete the objective, and so in this case `objectiveComplete` and `gameplayFinished` are triggered by the same game event. Both sequences are valid, and can be visualized like this:





Deprecated Items

- `playableFinished()`, as of API v3.0
  - This has been renamed to [`objectiveComplete`]() to better reflect its meaning. Old playables will continue to work, but going forward `objectiveComplete` should be used.