Tech requirements Bambuser Live Video Shopping WebView integration (iOS/Android)

This document should explain briefly how to use the Bambuser Live Video Shopping(LVS) Web player with Picture-in-Picture(PiP) video support inside of a WebView inside of a native app on iOS and Android.

There's certain caveats and possibilities for both iOS and Android, therefore the process to integrate it will differ technically and conceptually.

Prerequisites
Native iOS/Android application(s) with basic Bambuser LVS WebView integration.

iOS
HTML web players presented inside of a `WKWebView` can present the underlying media file using the OSs' PiP functionality. In order to enable a website presented inside of a `WKWebView` to request the media file to be played like this the webview needs to be configured via the `WKWebViewConfiguration` class. The `WKWebViewConfiguration` should have the property `allowsPictureInPictureMediaPlayback` set to true.
https://developer.apple.com/documentation/webkit/wkwebviewconfiguration/1614792-allowspictureinpicturemediaplayb

There might be situations where it's best practice to also enable the background execution mode *"Audio, AirPlay, and Picture in Picture"* for your application as well depending on how your application currently functions.

With these options enabled inside of your application it should now be possible to allow the Bambuser LVS player to display the video inside of the OSs' PiP functionality.

In order to request the player to go into PiP you should call the embed pages' player instance function called `` `requestPictureInPicture` ``.

In this example we will demonstrate how the implementation could look like if you wanted the player to go into the PiP mode if a product is clicked inside of the player:

```
<html>
  // Bambuser LVS embed code HTML page
  ...

  <script>
    window.onBambuserLiveShoppingReady = (player) => {
      player.configure({
        buttons: {
          product: player.BUTTON.NONE, // Signals to the player that
we want to handle any product clicks manually on this embed page.
        },
```

```
      });

      player.on(player.EVENT.SHOW_PRODUCT_VIEW, (event) => {
        // A product is pressed inside of the web player.
        // Request Picture in Picture mode.
        player.requestPictureInPicture();

        // REMARK:
        // The event parameter in this function contains information
about the pressed product.
        // We can then use this information to navigate to signal to
the native app code which can navigate
        // to the PDP inside of this native apps navigation stack.
        //
        // Pseudo code to send an event to the apps native code layer
via WKScriptMessageHandler:
        //
        //
window.webkit.messageHandlers.WKScriptMessageHandler.postMessage({
        //   'eventName': 'present-pdp',
        //   'sku': event.sku,
        // });
        // Important note: In your native code, don't discard this
WebView instance when
        // navigating away from this WKWebView. The PiP requires the
actual website to still be
        // loaded inside of the app in order for the video to continue
playing in most cases.
      });
    };

    player.on(player.EVENT.ENTERED_PICTURE_IN_PICTURE, () => {
      // This callback will be called if the player successfully
entered PiP mode
      // At this point it could be a good idea to hide the currently
presented WebView which
      // might be covering the view of the applications navigation
stack, depending how the app is built.
      //
      // Pseudo code to send an event to the apps native code layer
via WKScriptMessageHandler:
      //
      //
window.webkit.messageHandlers.WKScriptMessageHandler.postMessage({
      //   'eventName': 'hide-lvs-webview',
      // });
```

```
      // Important note: In your native code, don't discard/release
this WebView instance when
      // hiding this WKWebView. The PiP requires the actual website to
still be
      // loaded inside of the app in order for the video to continue
playing in most cases.
    });

    player.on(player.EVENT.EXITED_PICTURE_IN_PICTURE, () => {
      // This callback will be called if the player exited PiP mode
      // The PiP mode will be exited in these cases:
      // a) player.exitPictureInPicture();
      // b) User interacts clicks the "Exit picture in picture" icon
in the PiP UI
      //
      // At this point it could be a good idea to "unhide" the WebView
which contains the LVS player.
      //
      // Pseudo code to send an event to the apps native code layer
via WKScriptMessageHandler:
      //
      //
window.webkit.messageHandlers.WKScriptMessageHandler.postMessage({
      //   'eventName': 'show-lvs-webview',
      // });
    });
  </script>
</html>
```

Android:
The implementation will be different on the Android platform as the WebViews' javascript engine does not support PiP. Instead the goal will be to set up a WebView inside of the app and instead make this entire WebView capable of running in PiP mode. The code example below is written in Kotlin but should conceptually be very similar to an application written in Java.
https://developer.android.com/develop/ui/views/picture-in-picture

Make the App activity support PiP by adding this to the activities XML:
```
  <activity android:name="Activity"
    android:supportsPictureInPicture="true"
    android:configChanges=
        "screenSize|smallestScreenSize|screenLayout|orientation"
    ...
```

Now that the App activity supports PiP we'll have to implement the function call where we, similarly to the iOS code, want to request the player to go into PiP when a product is clicked. To begin with we want to use a similar button setup config and listen to the same `player.EVENT.SHOW_PRODUCT_VIEW` as we did in the iOS example. But this time instead of requesting the Bambuser LVS player to go into PiP mode we'll have to communicate to the native code that it should instead handle the PiP presentation, and we can do that by calling a function on the `Android` object in the window.

```
window.onBambuserLiveShoppingReady = (player) => {
  player.configure({
    buttons: {
      product: player.BUTTON.NONE, // Signals to the player that we
want to handle any product clicks manually on this embed page.
    },
  });

  player.on(player.EVENT.SHOW_PRODUCT_VIEW, (event) => {
    // Call a function in the native app to signal that we want this
activity to go into PiP mode
    Android.enterPictureInPicture();
  });

  ...
};
```

In order for us to receive this function call in our native Android application we have to implement the `enterPictureInPicture` function which we call above. We can accomplish this by adding a javascript interface class named WebViewInterface which could look like this:

```
class WebViewInterface(private val context: Context, private val
activity: WebViewActivity) {
  @JavascriptInterface
  fun enterPictureInPicture() {
    if (android.os.Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
      // Here could be a good idea to determine the aspect ratio of
the PiP we will present and use that inside of the.
      // PictureInPictureParams.
      // By listening to the `player.EVENT.PLAYER_CONTAINER_UPDATE`
you can find the latest player rectangle announced
      // from the web player, the width and height of that rect could
be used to calculate the appropriate PiP aspect
      // ratio.
      val params = PictureInPictureParams.Builder().build();
      // Tell the activity to enter PiP.
```

```
      activity.enterPictureInPictureMode(params);
    }
  }
}
```

We then add this interface to our WebView in our application like this:

```
class WebViewActivity : AppCompatActivity() {
  ...

  override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    ...

    val webView = findViewById<View>(R.id.webview) as WebView

webView.addJavascriptInterface(WebViewInterface(this@WebViewActivity,
this@WebViewActivity), "Android")
  }
}
```

Now when we call `Android.enterPictureInPicture()` in the javascript code on the html embed page that should make the Activity which holds the WebView to go into PiP. However, at this point the Bambuser LVS player still presents the player UI inside of the PiP which looks unpleasant. In order to solve that we can use the LVS players' functions `hideUI()` and `showUI()` to hide and show the player UI respectively

In order to call these functions on the HTML embed page we can store the players' instance pointer in the global window scope add these functions to the embed page:

```
...
<script>
  window.onBambuserLiveShoppingReady = (player) => {
    window.player = player;
    player.configure({
      ...
    });

    ...
  };

  const hidePlayerUI = () => {
    // Calling this function will hide the players UI, exposing only
the underlying video players content
```

```
    window.player.hideUI();
  };

  const showPlayerUI = () => {
    // Calling this function will hide the players UI, exposing only
the underlying video players content
    window.player.showUI();
  };
...
```

Inside of the app activity which we requested to enter PiP mode we can override this callback to determine when the activity entered or left PiP, and hide or show the UI accordingly:

```
class WebViewActivity : AppCompatActivity() {
  ...

  override fun onPictureInPictureModeChanged(
    isInPictureInPictureMode: Boolean,
    newConfig: Configuration?
  ) {
    val webView = findViewById<View>(R.id.webview) as WebView
    if (android.os.Build.VERSION.SDK_INT >=
android.os.Build.VERSION_CODES.KITKAT) {
      Handler(Looper.getMainLooper()).post {
        if (isInPictureInPictureMode) {
          // ... If in PiP mode, hide the player UI
          webView.evaluateJavascript("hidePlayerUI();", null);
        } else {
          // ... If not in PiP mode, hide the player UI
          webView.evaluateJavascript("showPlayerUI();", null);
        }
      }
    }

    super.onPictureInPictureModeChanged(isInPictureInPictureMode,
newConfig)
  }
}
```

Now when we run the app and tap a product inside of the Bambuser LVS player the WebView will go into PiP mode and hide the player UI, and when exiting the PiP mode we present the UI again.