Blaze Social SDK. Developer’s Documentation

This document contains both Developer’s Guide and Internal Functions specification. They will be separated in future versions.

## Blaze Social SDK. What is it? and What for?

JavaScript Blaze Social SDK is a server-side JavaScript module (and corresponding server modules) that provides 3rd-party game developers with the following functions:

* Transparent use of Facebook SDK functions
* Auto replacing of game feed URLs to direct users traffic on Oberon distribution channels
* Additional Simplified wrappers for major Facebook SDK functions

(For internal use)

* Automatic URL substitutions of game URLs to Channel game URLs - in all POSTS to FaceBook from the game
* Simplified calls for Omniture Report module. (For some events the logging can be automatic)

Simply include and initialize Blaze module and all the links posted from game will lead to the proper partner page.

Blaze Social SDK consists of:

* Included from server JavaScript Library,
* Developer-customized JavaScript initialization code embedded into page code,
* Server-side code for providing Server connection

## Initializing Blaze Social SDK for game page

**2.1. Include JavaScript Library**

To use Blaze Social SDK HTML code of a game page must contain the following line:

<script type="text/javascript" SRC="http://sdk.oberon-media.com/JS/blaze.js"></script>

**2.2. Prerequisites**

Blaze Social SDK and its components need some constants and code to be defined before the SDK is loaded.

Firstly define Facebook <div> container. It is needed for Facebook SDK. Define it in HTML code before the initialization script. (Described later)

<div id="fb-root"></div>

Also there are some essential constants. Define them inside JavaScript tag.

<script language="JavaScript">

<!—

var GAME\_ID='123456';

//-->

</script>

GAME\_ID is the ID of the game on the Oberon Server.

**2.3. Blaze Social SDK Modules**

Blaze Social SDK consists of some modules that can be loaded in case the functionality is required. Modules use the functionality of each other, i.e. depended, hence every module must be loaded after the modules that it uses/depends on. For example you cannot perform any Server requests until the AJAX proxy container is not loaded yet.

Here is the list of modules and their dependences:

|  |  |  |  |
| --- | --- | --- | --- |
| Module Name | Loading Function | Depends on | Description |
| Blaze Main Module | SURE\_Blaze | -nothing- | Base module |
| Page Content | \_waitLoad() | -nothing- | Function signals with its callback that the essential parts of page content are loaded |
| Facebook SDK | \_loadFacebook() | \_waitLoad() | Provides most of Facebook functions transparently. |
| AJAX Proxy | \_loadProxy() | -nothing- | Needed to make requests to Server |
| Settings | \_getSettings() | \_loadProxy() | Get settings from server. Need not srLogin. |
| Game Info | \_getGameURL() | \_loadProxy() | Get game information from server: Facebook game page and Game Name |
| Facebook Login | login() | \_loadFacebook() | Login to Facebook. Shows Facebook login dialog or login silently if Facebook cookie is set already |
| Facebook status | getLoginStatus() | \_loadFacebook() | Get Facebook status (logged or not) and fills the internal variables with Facebook info |
| Sure Facebook Login | \_sureFBLogin() | \_loadFacebook() | Get Facebook status and login to Facebook if status is "not logged" |
| Social Ring Login | \_srLogin() | \_loadProxy(), \_sureFBLogin() | Login to Server using Facebook ID |
| Omniture Report | \_sureOmniture() | \_getSettings() \_srLogin() | Start Onmiture Report subsystem |
| Report Game Start | \_omnitureLaunch() | \_sureOmniture() | Report Start Game Statistics to the Server |
| Report game Session Start | \_omnitureSessionStart() | \_sureOmniture() |  |
| Redirect | \_redirect() | \_srLogin() | Perform redirect to the channel game page if proper URL parameters present, do nothing if no parameters |

Blaze Social SDK provides two ways of managing modules loading.

The simplest way to load Blaze Social SDK modules properly is to place the next function call into the callback of previous. The second way is to use signals, it discussed later.

<script language=javascript>

<!--

SURE\_Blaze( function(blaze) {

// SET blaze variables here...

...

// Initialization below

window.blaze.\_waitLoad(function() {

window.blaze.\_loadFacebook(APP\_ID,function(blaze) {

window.blaze.\_loadProxy( function() {

window.blaze.\_getSettings(function() {

window.blaze.\_sureFBLogin(function() {

window.blaze.\_srLogin(function() {

window.blaze.\_sureOmniture(function() {

window.blaze.\_redirect();

window.blase.\_onmitureLaunch();

});

});

});

});

});

});

});

});

//-->

</script>

The APP\_ID is the Facebook Application ID of your game page under Facebook.

It was defined before, in Prerequisites section.

Setting Blaze Social SDK variables is discussed later.

**2.4. Using FB and blaze singleton objects**

When initialization of Blaze Social SDK completes, the singleton window.blaze is available.

Please use this variable to access to all Blaze functions.

Also you can if you wish to use FB variable in usual way and even do not change all the Facebook SDK calls! To do it simply use blaze.\_extend() function to grab all the Facebook functions to blaze and then assign blaze variable to FB variable.

window.blaze.\_extend(FB);

FB=window.blaze;

Now you can use FB variable to access to all Facebook function (but modified by Blaze!).

And it is recommended to use window.blaze variable for all Blaze functions.

**2.5. Setting internal variables**

Setting internal Blaze variables required for some Blaze functions and contains the essential information about identification of the game application on server.

Internal Blaze variables must be set when the main Blaze module already completed and the other modules are not yet loaded. (See previous section)

It is good to use constants here was defined in Prerequisites section.

blaze.gameId = GAME\_ID;

blaze.channel = getUrlParameter(window.location.href,'channel');

blaze.channel is concerned to the game placement on a partner’s channel, so developer need to get it from partner and pass it to Blaze. Usually partner place the game into container and provides the channel ID in the URL of the game using the channel parameter. To get the channel ID value from the URL string use getUrlParameter() function.

This performs automatically by Blaze initialization code.

**2.6. Status signals and Error handling**

Every Blaze Social SDK Function signals its state to the main module. So you can use it for tracking of Blaze tasks running, for showing status information to user and to managing module loading. This is the second way to manage the modules loading mentioned in part 2.3.

Blaze Social SDK functions calls blaze.on\_begin() on start and blaze.on\_end() on finishing with the function name in parameter. If errors occurs the blaze.on\_error() function been called.

You can define your own handlers to track the status in the same section of the code that used for definition of Blaze internal variables.

blaze.on\_begin=function(blaze,func) {

if(func=='\_loadProxy') str='Loading AJAX Proxy...';

if(func=='\_sureFBLogin') str='trying login to FB...';

if(func=='getLoginStatus') str='Get FB login status...';

...

// here report this string to user somehow

}

blaze.on\_end=function(blaze,str) {

...

// here report this string to user somehow

}

blaze.on\_error=function(str) {

...

// here report this string value to user somehow

}

Also, developer can track the current loading processes using blaze.status\_pending array.

For example:

blaze.on\_end=function(blaze,str) {

if(blaze.status.pending.length==0) alert('Loading processes are completed!');

}

**2.7. The redirect functions**

All the game web pages and the Facebook Game App page (!) must contain the redirect code. Simply call the window.blaze.\_rerirect() function and be sure that the right redirection will be performed. It your game page is the final result of redirection, the \_redirect() function will do nothing.

**2.8. Automatic Omniture reporting**

(Partly for internal use)

Blaze automatically provides the ‘Start game’ and ‘Start game unique’. Simply call window.blaze.\_omnitureLaunch() function when initialization of Omniture module will be completed (\_sureOmniture() function) and be sure that the right Omniture events will be reported to the server.

**2.9. The complete initialization code 'as is'**

All above is wrapped into one function:

BlazeInit(GAME\_ID);

Summarizing the chapter 2, here is the complete and minimal initialization code. You can use it 'as is' changing the constant and omitting the parts you have got already.

<head>

<script type="text/javascript" SRC="http://sdk.oberon-media.com/JS/blaze.js"></script>

</head>

<body>

<div id="fb-root"></div>

<script language="JavaScript">

<!—

var GAME\_ID='11261382';

BlazeInit(GAME\_ID);

blaze.on\_begin=function(blaze,func) {

// here report this string to user somehow

}

blaze.on\_end=function(blaze,str) {

// here report to user somehow

//(or hide 'loading...' messages) to user somehow

if(blaze.status.pending.length==0) alert('Loading processes are completed!');

}

blaze.on\_error=function(str) {

// here report this string to user somehow

}

//-->

</script>

</body>

## URL substitution

(This part for internal use)

When end user post some wall messages (game gifts / invitation / bookmark) with the links to this game on Facebook — the links will be substituted with the corresponding links to the same game on partner’s site.

Actually if mandatory function (see previous part) are used the URL substitution will be performed automatically. Here is the description of this process.

* User posts the wall message (game gift / invitation / bookmark) using one of Blaze Social SDK offered functions. These functions automatically substitute the URLs with the URL that leads to *Blaze Redirect Page* with parameters added to link

Example:

<http://apps.facebook.com/blaze-sdk/?reason=gift&userId=0011-1234-1313-2224&gameId=myGame&url=http>...

* Another user (Facebook visitor) clicks the link on his wall and go to the Facebook Game Page. The JavaScript code of this page contains the *Blaze Redirect Code,* It get the command line parameters to identify all the info about that user who posted the game (e.g. the Oberon channel he uses to play the game at) and get info about the user who clicked the link (from Facebook).
* *Blaze Redirect Code* performs login to the Server, then get the server’s game info.
* *Blaze Redirect Code* redirect user to the link based on “game page on channel” info from the Server response.

## transparent functions

If you use FB=window.blaze substitution (as described in 2.4) all the URL translations in Facebook API will be performed automatically.

If you do not use it – you obliged to use Blaze functions instead of similar Facebook ones. All the parameters are completely similar to Facebook ones, see Facebook SDK documentation for details.

Blaze.login() instead of FB.login()

Blaze.logout() instead of FB.logout()

Blaze.getLoginStatus() instead of FB.getLoginStatus()

Blaze.getSession() instead of FB.getSession()

Blaze.ui() instead of FB.ui()

Blaze.api() instead of FB.api()

Blaze.parse() instead of FB.XFBML.parse(el) //ATTENTION in red!

Using these functions instead of Facebook ones guarantees the URL substitution and all advantages of using Blaze Social SDK.

## 6. omniture reporting

Most of events reports automatically and developer do not need to call them directly. For example game starting etc.

But some events developer must call from the code, for example purchasing some in-game items, the reaction for some in-game events etc.

**6.1. General reporting function**

Then developer should report the events to Omniture System using universal \_omnitureSend() function of simplified wrappers for special cases (see below):

window.blaze.\_omnitureSend(post\_id,{custom\_parameters});

Most of custom parameters will be set automatically inside function and developer need not specify them. Some parameters are required. See List of Omniture Events (below) for details.

Custom parameters can be any of fillowing:

channel

products

eVar1

eVar2

eVar3

eVar4

eVar5

eVar6

eVar7

eVar8

The exact value depends on certain event.

Example:

window.blaze.\_omnitureSend('Game Event',{ events: 'event7', eVar5:'Discovered', eVar2:'viral', products:’Category;zuma’});

window.blaze.\_omnitureSend('SDK Purchase',{ products : ’Category;zuma;1,1.44’ });

You don’t need to specify, for example, channel:’0011234341’ in custom parameters, it will be inserted automatically. If you forget any required custom parameter for some event, the diagnostic message appears, so developer can correct the code while testing.

**6.2. Simplified wrappers**

(This part for internal use?)

Simplified wrappers perform the proper call to the General function with predefined parameters.

\_omnitureLaunch()

No parameter needed.

Automatically reports some of starting events (or all of them).

‘Game Started’ every time when game started and SDK loaded

‘Game Started Unique’ user started the game for the first time. User added to Database.

‘Game per User Unique Started’ existing user started the new game for him. User’s game data added to database.

\_omnitureSessionStart();

No parameter needed.

Reports the purchase of the product (the Game itself) with specified price.

\_omniturePurchase(price);

Reports the purchase of the product (the Game itself) with specified price.

## list of all functions

(This part for internal use)

Blaze Social SDK provides some wrapping functions for more easy and comfortable use of Facebook SDK, and some additional functions that can be useful for developer.  
They can be accessed through window.blaze variable.  
  
**Transparent Facebook wrappings**

Developer can call them in the same way as Facebook functions with the same result.

(With some additional Blaze work).

See Facebook SDK documentation for details (<http://developers.facebook.com/docs/reference/javascript/>):

login(callback,perms)

logout(callback)

getLoginStatus(callback)

getSession()

ui(obj,callback)

api(path,....,callback)

parse(el)

**Load status and error Events**

Developer can define this functions and Blaze will call them in proper moment to report the functions starts finishes and error events.

on\_begin(blaze,func\_name)

on\_end(blaze,func\_name)

on\_error(error\_text)

**Blaze internal functions**

These functions are used internally, some of them must be called during initialization

(see 2.3 for details).

\_getChannelURL(url,reason,game\_generated)

\_loadFacebook(app\_id,callback)

\_sureFBLogin(callback)

\_srLogin(callback)

\_queryChannelUrl(callback,gameId,referralUserId,referralChannel,

isForceCreated,linkType,currentChannel)

\_getSettings(callback)

\_getOmnitureSuit(callback)

\_loadProxy(callback)

\_loadOmniture(suit,callback)

\_omnitureInit(callback)

\_omnitureSend(evNum,evars\_obj)

\_sureOmniture(callback)

\_omnitureLaunch()

\_omniturePurchase()

\_extend(obj)

\_waitLoad(callback) {

\_redirect(ref)

\_statusChanged(begin,func\_name)

**Useful Blaze functions**

Functions are not used now but can be useful for developer

run : function()

\_getSite : function(url, callback)

\_traverse : function(callback)

\_isAppUser : function(callback)

\_startLoad(func\_name,callback)

\_canLaunch(func\_name)

**Standalone common functions (not blaze methods)**

Functions that are not the methods of Blaze object but defined in blaze.js file.

SURE\_Blaze(callback)

loadScript(url,callback)

getUrlParameter(url, name)

JSON2str(obj)