Events

The Gigya service generates many different types of events driven by user interactions such as user login, button clicks, etc. Applications may register event handlers that listen for particular events and execute code when these events are received.

This page details the events the Gigya service generates, and specifies how to define events handlers and register for events.

Events generated by the Gigya service are divided into two categories:

- **Global Application Events** - Events that are initiated by the application, such as when a user logs in.
- **Plugin Events** - Events that are initiated by a plugin, such as when a user clicks the "OK" button of a plugin.

Event handlers are defined the same way for both types of events, but registering those event handlers for events is performed differently. Read the sections below for instructions on defining and registering event handlers for each type of event.

See Overriding the Default Event Map for an alternative way to register event handlers.

Global Application Events

Global application events are generated by the Gigya service whenever the event to which they refer occurs, regardless of the action that triggered the event. This is in contrast to plugin events, which are only fired by the specific plugin on which they were configured.

These are the available global application events:

- **onLogin** - Fired whenever a user successfully logs in to Gigya. This may happen, for example, as a result of calling one of the following methods: accounts.socialLogin, socialize.login, socialize.showLoginUI, socialize.notifyLogin or when using Screen-Sets.
- **onLogout** - Fired whenever a user logs out from Gigya, using the accounts.logout or socialize.logout call.
- **onConnectionAdded** - Fired whenever a user is connected to a provider . This may happen, for example, as a result of calling one of the following methods: socialize.addConnection, socialize.showAddConnectionsUI, and socialize.showEditConnectionsUI.
- **onConnectionRemoved** - Fired whenever a user is disconnected from a provider. As above, this event will be fired regardless of the action that triggered it.
- **onLinkBack** - Fired whenever a linkback is detected. Read more about the onLinkback event.

onLogin Event Data

Field	Type	Description
eventName	string	The name of the event: 'login'.
source	string	The source plugin that generated this event. The value of this field is the name of the plugin's API method, in this case '*showLoginUI*'. Note: this field will not be available if the source of this event is not a plugin (e.g. if the source is a socialize.login API call).
context	object	The context object passed as a parameter to the plugin/method that triggered this event, or null if no object was passed.
loginMode	string	The type of login: • *standard* - the user is logging into an existing account. • *reAuth* - the user is proving ownership of an account by logging into it.
provider	string	The name of the provider that the user used in order to login (e.g. "*Facebook*"). Note: If this event is fired as a result of a socialize.notifyLogin call, i.e., the user was authenticated by your site, the *provider* field will be set to "*site*".
UID	string	The User ID that should be used for login verification*. **Note:** The UID string must be encoded using the encodeURIComponent() function before sending it from your client to your server.
UIDSignature	string	The signature that should be used for login verification, as described under the Validate the UID Signature in the Social Sign-On Process section.

user	User object	A User object with updated information for the current user.
signatureTimestamp	string	The GMT time of the response in UNIX time format (i.e. the number of seconds since Jan. 1st 1970). The time stamp should be used for login verification, as described under the Validate the UID Signature in the Social Sign-On Process section.
fullEventName	string	This is occasionally returned as an internally used parameter.

* To learn more about login verification, please refer to the Validate the UID Signature in the Social Sign-On Process section in the Security page of the Developer's Guide.

Defining an Event Handler

An event handler is a function with the following signature:

```
functionName(eventObj)
```

Where eventObj is an object that includes the following members:

- **eventName** property - the name of the event fired. May be: 'login', 'logout', 'connectionAdded' or 'connectionRemoved' respectively.
- **context** property - will be available only if passed as parameter to the plugin/method that triggered the specific event.

The following code defines a simple event handler:

```
function DisplayEventMessage(eventObj) {
    alert("congrats on your " + eventObj.eventName);
}
```

Registering a Handler for an Event

Having defined an event handler, you can register it for an event using the accounts.addEventHandlers or socialize.addEventHandlers Gigya API methods. Like any other Gigya API method, addEventHandlers receives the params object (refer to the JavaScript API Basics page for more information).

The params object may include the following members (all optional):

- **onLogin** - The value assigned to this parameter is a name of an Event Handler function that will be called when the user is successfully authenticated by Gigya.
- **onLogout** - The value assigned to this parameter is a name of an Event Handler function that will be called when the user logs out.
- **onConnectionAdded** - The value assigned to this parameter is a name of an Event Handler function that will be called when the user is successfully connected to a provider.
- **onConnectionRemoved** - The value assigned to this parameter is a name of an Event Handler function that will be called when the user disconnects from a provider.
- **onLinkback** - The value assigned to this parameter is a name of an Event Handler function that will be called when a linkback is detected.
- **callback** - A reference to a callback function that will be called, along with the results of the operation when the operation completes.
- **context** - A developer-created object that will be passed back unchanged to the application as one of the fields in the response object.

The following code registers the DisplayEventMessage function, as defined above, for the user login event:

```
gigya.socialize.addEventHandlers({
 onLogin:DisplayEventMessage
});
```

This means that whenever the user logs in, the DisplayEventMessage function is called. It receives the eventObj parameter which includes the context object (passed to the accounts.socialLogin or socialize.login/socialize.showLoginUI methods) and the eventName (which, in this case is 'login').
The output of this example is:

Combined Event Handlers

You may register one event handler (one function) for several events, and you may register several events within one addEventHandlers method call .

Example:

```
function displayEventMessage(eventObj) {
    alert(eventObj.context+' '+eventObj.eventName);
}

function confirmEventMessage(eventObj) {
    var r = confirm('Have you '+eventObj.eventName+ '?');
    if (r==true) {
        alert(eventObj.context);
 }
}

gigya.socialize.addEventHandlers({
    onLogin:displayEventMessage,
    onConnectionAdded:confirmEventMessage
});

gigya.socialize.addEventHandlers({
    onConnectionRemoved: displayEventMessage
});
```

The onLinkback Event

The onLinkback event is initiated whenever a linkback is detected.

Linkbacks are detected in the following cases:

- A linkback from shortened URLs. These are reported by the server, and an extra parameter (hash or querystring) is added to the URL to let the Gigya Web SDK know that it was reported.
- Facebook like or send buttons referral traffic. In these cases Facebook adds a context parameter to the URL that was passed by our like

button.
- When the referrer is from a list of domains the client detects as linkbacks.
- From AddressBarShares - when ***trackAddressBarShares*** is set to 'true' (see the global Conf object).

The event includes the following fields (if available):

Field	Type	Description
eventName	string	The name of the event.
source	string	The source plugin that shared the link that led to the page.
context	object	The context object passed as a parameter to the method, or null if no object has been passed.
provider	string	The provider from which the linkback came.
cid	string	The context ID used for reports. Read more here.
shortCode	string	The original short URL code.
fullEventName	string	This is occasionally returned as an internally used parameter.

Working Example

Please explore the following pages, where you may activate full working examples and then grab the code:

- The "Login" Example - the example uses the **onLogin** event.
- The "Get User Information" Example - the example uses the **onConnectionAdded** and **onConnectionRemoved** events.

Screen-Set Events

The following events are triggered by RaaS Screen-Sets.

- **onError**: called when an error occurs.
- **onBeforeSubmit**: called before a form is submitted. This event gives you an opportunity to perform certain actions before the form is submitted, or cancel the submission by returning "false".
- **onSubmit**: called when a form is submitted, can return a value or a promise. This event gives you the option of modifying the form data when it is submitted.
- **onAfterSubmit**: called after a form is submitted.
- **onBeforeScreenLoad**: called before a screen is rendered. This event gives you an opportunity to cancel the navigation by returning "false".
- **onAfterScreenLoad**: called after a new screen is rendered.
- **onFieldChanged**: called when a field is changed in a managed form.
- **onHide**: called when a user clicks the "X" (close) button, or when the screen is hidden at the end of the flow.
- **onBeforeValidation**: called after a user clicks **Submit** on a Screen-Set, and before Gigya's built-in field and form validations (onBeforeSubmit). Can be used to add custom validations to the form (e.g., ZIP code validation).

For full details on these events, see accounts.showScreenSet.

You can add custom code to handle these events directly to the UI Builder. For more information, see UI Builder - JavaScript Parameters.

Plugin Events

Each plugin defines events that it generates for specified situations.

The following events are generated by all plugins:

- **onLoad** - When the plugin has finished drawing itself.
- **onError** - If an error occurs.
- **onClose** - When the plugin is closed. Plugins may be "closed" in either of the following ways:

- If the plugin is opened as a popup dialog, an "X" button appears. Clicking the "X" button closes the plugin and an onClose event is generated.
- In case the plugin has an "OK" and/or "Cancel" button, clicking either buttons closes the plugin and an onClose event is generated.

In addition to the events generated by all plugins, the Share plugin generates the following event:

- **onSendDone** - This event is triggered when the process of publishing the newsfeed to all selected social networks has finished. The event data includes the list of the social networks to which the newsfeed was successfully published.

Defining an Event Handler

An event handler is a function with the following signature:

```
functionName(eventObj)
```

Where eventObj is an object whose members change depending on the specific event that was fired.

For example, in an onSelectionDone event, eventObj will include the following members:

Field	Type	Description
eventName	string	The name of the event.
context	object	The context object passed as a parameter to the method, or null if no object has been passed.
friends	Collection	A collection of basic ? Friend objects, representing the list of the selected friends.
fullEventName	string	This is occasionally returned as an internally used parameter.

The following properties of eventObj are available for all events:

- **eventName:** the name of the event fired.
- **context:** available only if passed as a parameter to the plugin/method that triggered the specific event (see the next section for more details).

Registering a Handler for an Event

Registration for plugin events is performed at the creation of the plugin.

The creation method for each plugin (e.g. socialize.showLoginUI) receives the following optional parameters (as fields of the **params** object):

- **onLoad**
- **onError**
- **onClose**

Each of the above parameters may be assigned a value, which is the name of the event handler function to be called when the corresponding event occurs.

Overriding the Default Event Map

An Event Map is an object that maps Gigya events to methods, and it constitutes another way of registering for Gigya's events.

Gigya defines a **default Event Map**, which is used for Google Analytics integration and specifies which Gigya events will be tracked and how (for more information, see Google Analytics - The Default Tracking Behavior).

You may override or add to the default Event Map by setting the **customEventMap** parameter of the Global Conf object.

customEventMap Object Specification

Required	Field Name	Type	Description
Required	**eventMap**	Array of event mapping objects	An array of JSON objects. Each object specifies a certain event mapping. Each event mapping object may have the following fields: • **events** (required) - a comma separated list of Gigya event names. E.g. "onLogin, onLogout, onConnectionAdded". **Note:** ◦ You may remove the "on" from the event names, e.g. "login, logout, connectionAdded". ◦ You may use an asterisk "*" to specify all events. • **sources** (optional) - a comma separated list of Gigya event sources. • **method** (optional) - the method that will be invoked when one of the events (specified in the *events* fiel d) is fired. If not specified then the *defaultMethod* (see below) is used. • **args** (required) - an array of parameters to pass to the method. Each argument can be: ◦ value ◦ Function reference - the function signature should always be: *string function(event)* i.e. receive the event object as a single parameter and return a string. For example: `function (e) { return (e.mode == 'reviews' ? 'Review Published' : 'Comment Published') }` ◦ String template - where fields from the event object may be embedded by preceding their name with $. For example: *'Followed via: $button.provider'* • **override** (optional) - the default value is *false*. Setting this parameter to *true* will cause Gigya to stop processing additional maps that match the current event. You may use this to override the default mapping of certain events.
Optional	defaultMethod	method name	The default method that will be invoked when one of the events (specified in the *eventMap* field) is fired.

Example: Defining a Custom Event Map

```
<script type="text/javascript"
src="https://cdns.gigya.com/js/gigya.js?apiKey=INSERT-YOUR-APIKEY-HERE">
{
    customEventMap : {
        defaultMethod: defaultEventHandler,
        eventMap : [
            {
                events:"sendDone",
                sources:"showShareUI,showShareBarUI,showFollowBarUI",
                args:["provider", "Share Published",
document.location.href]
            },
            {
                events:"followClicked",
                method:followHandler,
                args:["provider", "Gigya Follow - button clicked",
document.location.href]
            },
            {
                events:"reactionClicked,reactionUnclicked",
                args:["Gigya Reaction Bar", "Button Clicked",
"$reaction.text"]
            },
            {
                events:"commentSubmitted",
                method:commentHandler,
                args:[
                    function (e) {return (e.mode == 'reviews' ? 'Gigya
Reviews' : 'Gigya Comments') },
                    function (e) { return (e.mode == 'reviews' ? 'Review
Published' : 'Comment Published') },
                    "stream: $streamID"
                ]
            }
        ]
    }
}
</script>
```

Available Events

Event Name	Triggered By
onReactionClicked	socialize.showReactionsBarUI
onReactionUnClicked	socialize.showReactionsBarUI
onCommentEdited	comments.showCommentsUI
onCommentSubmitted	comments.showCommentsUI
onCommentVoted	comments.showCommentsUI

onLogin	socialize.login, socialize.showLoginUI and various plugins
onLogout	socialize.logout and various plugins
onConnectionAdded	socialize.addConnection, socialize.showAddConnectionsUI and various plugins
onConnectionRemoved	socialize.removeConnection, socialize.showAddConnectionsUI and various plugins

Note: You may remove the "on" prefix from the event names, e.g., "login, logout, connectionAdded".