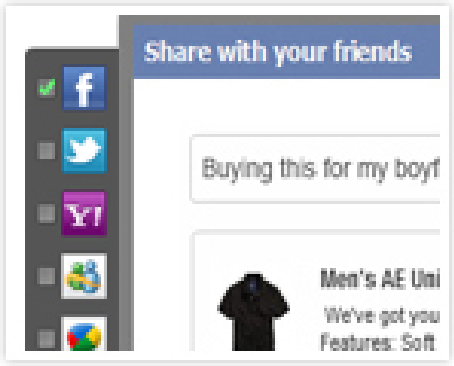
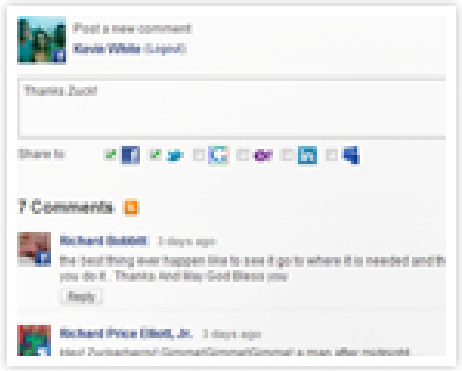
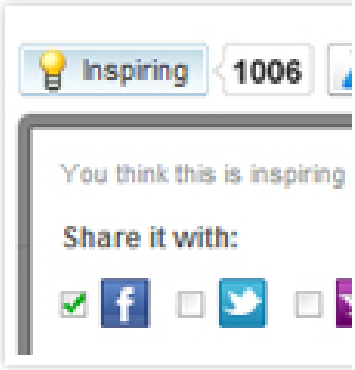
Using Plugins to Initiate Site Login

This guide explains how to leverage Gigya's plugins to acquire new site users, and log in returning site users. We will guide you on how to integrate these plugins with Social Login and Customer Identity.

Gigya's sharing and social interaction plugins embed an inner 'Add Connection' plugin:

Share Plugin	Comments Plugin	Reactions Plugin
		

This guide offers an alternative behavior for the case of a **non logged-in** user. We offer to **login** the user, and in case he is a new user, you may suggest registration.

Follow the steps below to implement this use case.

> **Note:** if you are using Gigya's Customer Identity (Raas) package, the implementation is slightly different and simpler:
> a. You can skip **Step 1** mentioned below (this is the default behaviour in the RaaS environment).
> b. Instead of socialize.addEventHandlers API method, use the accounts.addEventHandlers API method. The usage is the same, but the onLogin event data is different.

Best Practice Implementation Steps

Step 1: Register an Event Handler to the Global onLogin Event

An effect of the previous step is that each time a non-logged-in user clicks on a social network button within one of the plugins, the global onLogin event will be fired.
Make sure to listen to this event, if you want to handle the user's login in your site.
Registering a handler for a global event is done using the socialize.addEventHandlers API method (if you implement RaaS, then use accounts.addEventHandlers). For example:

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

// onLogin Event handler
function onLoginHandler(eventObj) {
   ...
}
```

Calling the addEventHandler method should be done on page load (before the plugins are loaded). The onLoginHandler method in the example, handles the onLogin event. The implementation is explained in step 4 ahead.

In the Login demo page you may find a complete working example that handles the onLogin event.

Step 2: Define a Context for Each Plugin

All of the Gigya plugins accept a ***context*** object (as member of the 'params' object). The ***context*** object is a developer custom object that will be passed back unchanged to your application through the plugin's event handlers as one of the fields of the ***eventObj*** . For example, in the code example above, ***eventObj.context*** contains the *context* that was passed to the plugin that initiated the onLogin event.
In our use case, we are interested in the onLogin event handler. Since we may have several sources (plugins) that initiate the *onLogin* event, we will use the ***context*** object to distinguish between the different sources.

Add to the params of each plugin a *context* parameter. As a value of the *context* parameter you can use different strings, identifying the different plugins (e.g. "CommentsPlugin", "SharePlugin", etc).
For example:

```
var shareParams =
{
    userAction:act
    ,context: "SharePlugin" // string to identify the Share plugin
};

// Show the Share plugin
gigya.socialize.showShareUI(shareParams);
```

Step 3: Handle Global onLogin Event only if Initiated by Relevant Plugins

Now we can handle the onLogin event, and decide per plugin source whether or not to push registration. For example:

```
// onLogin Event handler
function onLoginHandler(eventObj) {
   switch(eventObj.context)
   {
   case "LoginPlugin", "AddConnectionPlugin":
       // Execute the login flow
       // ...
   break;
   case "CommentsPlugin", "SharePlugin":
       // Do nothing
   break;
   default:
   }
}
```

Our recommendation:

- In case of Login Plugin, Add Connections Plugin: execute the login flow right away.
- In case of Share Plugin, Reactions Plugin: wait for the onSendDone event and only then execute the login flow. The motivation for postponing the login flow is that both plugins use a pop-up dialog. We want the user to first finish his sharing process. Only after the dialog closes, we will proceed to the login flow that may include redirecting or popping-up a registration form.
- In case of Comments Plugin: based on your preference; either execute the login flow right away or wait for the *onCommentSubmited* event.

In all cases, implement the login flow as described in the Login Implementation guide, starting with ***Step 3*** *(verifying signature)* of the flow.

Step 4: Handle Relevant Event per Plugin

As mentioned in the previous step, for some plugins we need to initiate the login flow as a result of an alternative event (not the *onLogin* event).

Thus we need to register and handle the relevant event. For example, for the Share Plugin, we need to register and handle the onSendDone even t.
We would like to initiate the login flow from the onSendDone event handler. There is a small catch here - in the *onLogin* event data, we receive the User Object and use it in the login flow; we don't receive the User Object in the onSendDone event. The solution is to call the socialize.getUse rInfo method, as shown in the following code sample:

```
var shareParams =
{
    userAction:act
    ,context: "SharePlugin" // string to identify the Share Plugin
    ,onSendDone: onSendDoneHandler // register to the onSendDone event of
the Share Plugin
};

// Show the Share Plugin
gigya.socialize.showShareUI(shareParams);

// onSendDone Event handler
function onSendDoneHandler(eventObj) {

    // call getUserInfo
    gigya.socialize.getUserInfo({
       callback: userInfoCallback,
       signIDs: true  // sign the user object in the response
    }); }
function userInfoCallback(response) {

    // Do Login flow
}
```

Notes:

- **Important** - please note that we have passed the signIDs parameter to the socialize.getUserInfo method. This will make sure that the Us er object passed to the callback method will be signed by Gigya.
- In the userInfoCallback method, *response.user* holds the User object. Use this object as part of the implementation of the login flow as described in the Login Implementation guide, starting with ***Step 3*** *(verifying signature)* of the flow.
- The implementation for the Reactions Plugin is exactly the same.

Putting it all Together - Running Example

The following code example puts all the pieces together:

Sign in using Plugins Demo

Please click the button below to open the "Share" dialog:

Click the button below to sign out from Gigya platform:

The Example Code

```
<html>
<head>
 <SCRIPT type="text/javascript" lang="javascript"
    src="http://cdn.gigya.com/js/gigya.js?apikey=<YOUR-API-KEY>">
 </SCRIPT>
 <script>

        // This method is activated when page is loaded
        function onLoad() {

            // register for login event
            gigya.socialize.addEventHandlers({
     onLogin: onLoginHandler
     ,onLogout: onLogoutHandler
     });
        }

        // onLogin Event handler
        function onLoginHandler(eventObj) {

     // in this example, we postpone the SignOn flow only in-case of the
Share Plugin
     // in all other cases we do the SignOn flow right away
     if (eventObj.context != "SharePlugin")
     {
         doSignOnFlow(eventObj.user)
     }
  }
  function doSignOnFlow(userObject) {
      alert("logged in");
   document.getElementById('status').style.color = "green";
   document.getElementById('status').innerHTML = "Status: You are now
signed in";
        }
        function showShareUI() {

   // Constructing a UserAction Object
   var act = new gigya.socialize.UserAction();
      act.setUserMessage("Your comment here...");
   act.setTitle("This is my title");
   // ...
   var shareParams =
   {
       userAction:act
       ,context: "SharePlugin" // string to identify the Share Plugin
       ,onSendDone: onSendDoneHandler // register to the onSendDone event
of the Share Plugin
    ,cid:''
```

```
   };

   // Show the Share Plugin
   gigya.socialize.showShareUI(shareParams);
  }

  // onSendDone Event handler
  function onSendDoneHandler(eventObj) {

      // call getUserInfo
      gigya.socialize.getUserInfo({
         callback: userInfoCallback,
         signIDs: true  // sign the user object in the response
      });
  }
  function userInfoCallback(response) {
   doSignOnFlow(response.user);
  }

        // Logout from Gigya platform. This method is activated when
"Logout" button is clicked
  function logoutFromGS() {
            gigya.socialize.logout(); // logout from Gigya platform
        }

  // onLogout Event handler
        function onLogoutHandler(eventObj) {
      alert("logged out");
   document.getElementById('status').style.color = "red";
   document.getElementById('status').innerHTML = "Status: You are now
signed out";
  }
    </script>
</head>
<body onload="onLoad()">
 <h3>Sign in using Plugins Demo</h3>
    <br />
 <h4>Please click the button below to open the "Share" dialog:</h4>
    <input type=button id="btnShare" onclick="javascript:showShareUI()"
value="Share" />
 <br /><br /><br />
    <br /><br /><br />
 <h4>Click the button below to sign out from Gigya platform:</h4>
 <br />
    <input id="btnLogout" type="button" value="Sign Out"
            onclick="logoutFromGS()"/>
    <br />
    <br />
```

```
    <div id="status"></div>
</body>
</html>
```

Notes:

In order to make the above code work in your environment, please note:

- The API key in the sample will only work on *http://localhost/...*
- To load the page from your domain, modify the value of the "APIKey" field in the code to your own Gigya API Key. A Gigya API Key can be obtained from the Site Dashboard page on Gigya's website. Please make sure that the domain from which you are loading the page is the same domain name that you used for generating the API Key.
- If you are using https, be sure to further adjust the JS API url to: **https**://**cdns**.gigya.com/js/gigya.js?apikey=<Your_API_Key>.