# MyGeotab Add-in Single Sign-On

### **What is SSO?**

* Single sign-on (SSO) is an authentication process that allows a user to access multiple applications with one set of login credentials.
* The user’s MyGeotab credentials are the one set of login credentials that will automatically log the user into the Partner’s solution in the MyGeotab platform.

### **Why is SSO needed?**

* Requiring users to enter credentials after they have already logged into the MyGeotab platform doesn’t provide a seamless experience.

### **Concepts to understand before implementing SSO:**

1. When a user authenticates into MyGeotab with their username and password, a sessionID is created. The Session ID is a token that can be used instead of a password to validate the user in the SDK. This token will be used for SSO.
2. MyGeotab database names are unique. If a MyGeotab database has the name “ABC”, there is no other database with the name “ABC”.
3. Within a database, usernames are unique. In the example above, there can only be one “[joesmith@gmail.com](mailto:joesmith@gmail.com)” in the database named “ABC”.
4. A user can exist in multiple databases with the same username. For example, a “[joesmith@gmail.com](mailto:joesmith@gmail.com)” can exist in the “ABC” database and also the “XYZ” database.
5. Based on points #2, #3, and #4, you can understand that the **combination of a database and a username** in that database are unique across the entire Geotab system. For example, “[joesmith@gmail.com](mailto:joesmith@gmail.com)” from “ABC” database is a unique set across all of Geotab. This unique set will be useful for SSO when Joe Smith uses your solution in the “ABC” database.
6. Geotab database names can change upon customer request, so using the database name as an identifier is NOT RECOMMENDED. Instead, use the database GUID, which is a unique identifier of the database. This is available in the ‘companyGuid’ property of the following API call result:

api.call("Get", { "typeName":"SystemSettings" }, function(result){ console.log(result) })

This is a “Get” method on the “SystemSettings” object.

When doing SSO, your unique set should be the username and database GUID. In the example from point #5, it will now be: “[joesmith@gmail.com](mailto:joesmith@gmail.com)” from “7a9d936b72f9018”. (Where “7a9d936b72f9018” is the GUID)

1. Before needing to perform SSO, your system should have a sync maintained between the users in the MyGeotab database and your backend system, as well as a sync between the MyGeotab database GUID and your backend account for the company.

### **How can I implement Single Sign On within my Add-In?**

1. Retrieve the current session id, username, and database of the active user. Do this with the [getSession](https://github.com/Geotab/mg-api-js#getsession) API in Javascript, or by reading the “api” object in other programming languages. **The getSession API is not available within the SDK Runner.**

api.getSession(function (result) { console.log(result) })

The result variable from the code above will look like this:

{

database: "ABC"

sessionId: "11d8e291672twgv6683c655ea9572"

userName: "joesmith@geotab.com"

}

1. You will also need the server path, which is available in the URL of the add-in page in MyGeotab. For example, in: https://my5.geotab.com/ABC

The server path is **“my5.geotab.com”**. It can be accessed from your add-in HTML page with:

var serverPath = window.location.href

1. Send the details from #1 and #2 to your backend to validate that these credentials are valid. Do this validation by making a small MyGeotab API call using the Username, Session ID, Database name, and server path. For example, do a Get Device call with a resultsLimit set to 1. If the call is successful, then the user credentials are valid. If the call fails with the “Incorrect Login Credentials” error, the user might have spoofed their credentials and is attempting to gain unauthorized access into your system. The next steps will assume the API call was successful.
2. If your solution maintains a sync between users in MyGeotab and your system, this is when your solution should check which user is synced to the user attempting to login, and then allow them into your system.
3. After the user has been authenticated, your add-in page should load the landing page of your add-in.

Notes:

1. This entire process should happen upon the “initialize” lifecycle stage of the Add-In. Check the Toolkit slides for links about lifecycle management.
2. If the API call in step 3# fails, your add-in should NOT authenticate the user, and should instead display an error message to the user. The error message can say something like “You are not authorized to access this page, please contact your administrator.”