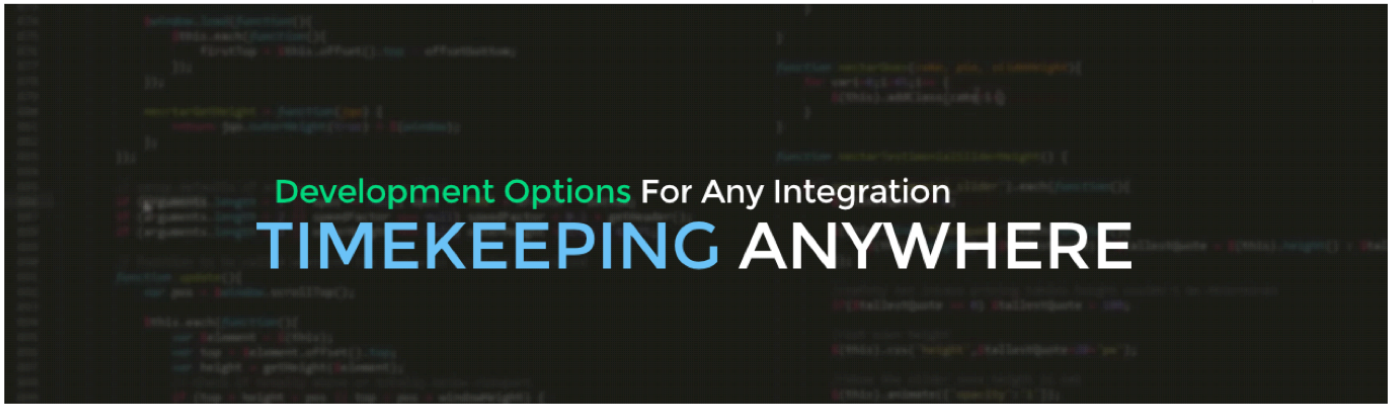


Contents

Summary

This document provides an overview of how to complete a single sign on (SSO) into SwipeClock. SwipeClock's SSO model follows an OAuth 2.0 JWT Bearer Token Flow. JWT stands for JSON Web Token which is JSON-based security token encoding that enables identity and security information to be shared across domains. For more information on JWT visit https://jwt.io.

The standard workflow of the OAuth 2.0 JWT Bearer Token Flow is as follows:

1. The JWT created is posted to SwipeClock's OAuth token endpoint.
2. SwipeClock's OAuth token end point validates the content and signature of the JWT token.
3. Assuming the posted JWT is valid and from an approved integration partner, an access token is issued.
4. The issued access token includes an expiration date and is valid for subsequent calls to other SwipeClock API endpoints (including SSO endpoints).

When completing an SSO into SwipeClock, you can display SwipeClock's system in one of two ways:

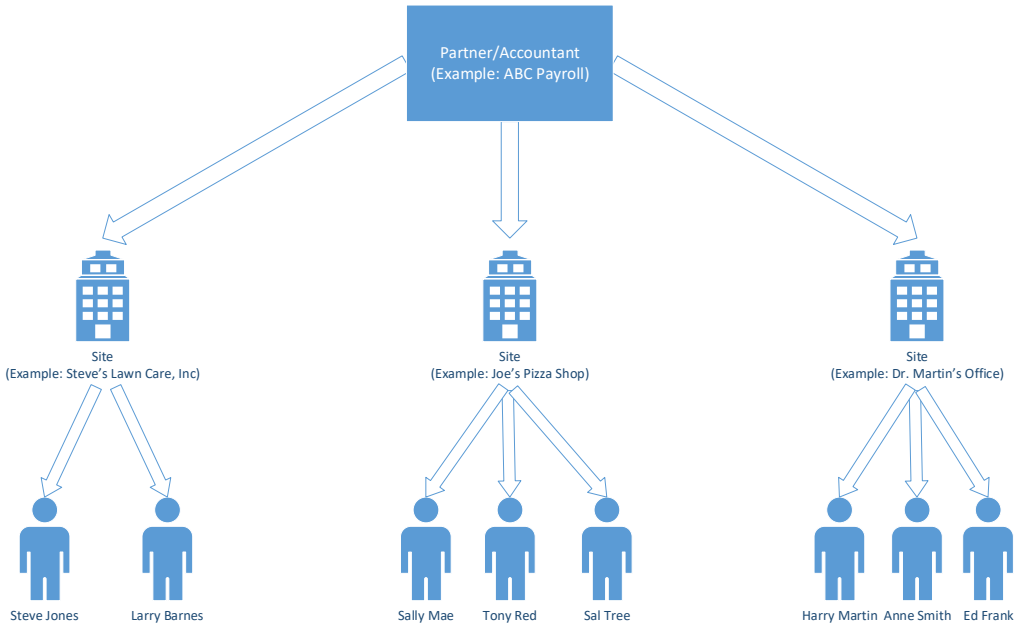
1. Embedded HTML iframe/object
2. Independent browser window/tab

It's your preference and this document will share samples of each.

Pre-Requisites

Before accessing any of the exposed SwipeClock APIs, you must first obtain an API secret. This secret is your "password" to programmatically access SwipeClock's API endpoints.

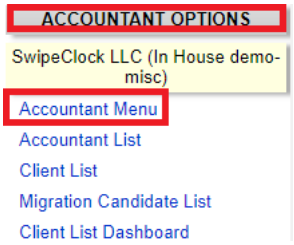
In order to understand the types of API secrets available, it's important to review the hierarchy built into SwipeClock's systems. The image below depicts some important entities and actors in SwipeClock.



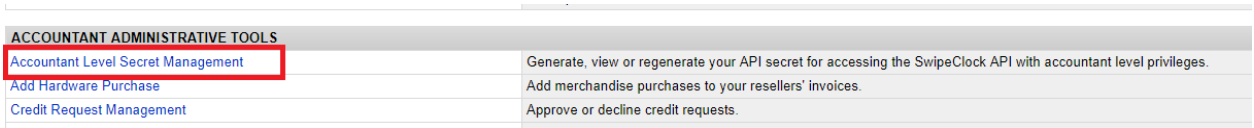
1. **Partner/Accountant** – A partner/accountant administers any number of sites below them. A typical example of a partner is a payroll provider who has setup SwipeClock for a number of employers they manage.
2. **Client/Site** – A client/site is an end employer setup for SwipeClock's services. Each site will have a number of employees managed along with users identified as client/site administrators and supervisors. In our image we have three sites: Steve's Lawn Care, Joe's Pizza Shop and Dr. Martin's Office.
3. **Employee** – an employee using SwipeClock's services.
4. **Client/Site Administrator** – a client/site can have certain user accounts granted administrative privileges to manage aspects of the client/side configuration.
5. **Supervisor** – a client/site can have user accounts with supervisor/management privileges to manage aspects of specific employees within the client/site.

Now that you're familiar with the entities within SwipeClock's landscape, we'll review the types of API secrets available:

1. **Partner/Accountant API Secret** – this secret provides access to all administered sites and employees within a specific partner/accountant in SwipeClock. Only users with partner/accountant level administration rights have the ability to generate and view this level of secret. To do this, follow these steps:
    a. Sign in at https://payrollservers.us/pg
    b. Click on "Accountant Options" on the left hand menu
    c. Within "Accountant Options", click on the Accountant Menu link



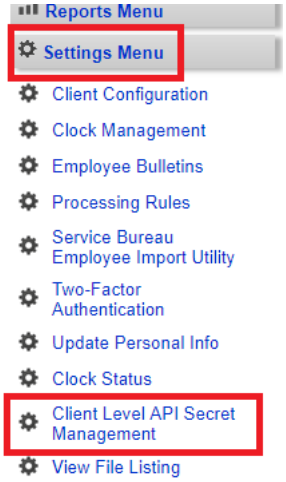
    d. Within "Accountant Options", click on the Accountant Level Secret Management link in the main window frame



    e. If a partner/accountant API secret has already been generated you can view it. If you need to, you can regenerate a new API secret by clicking the "regenerate" button. *Please note that regenerating a new secret will deactivate all prior partner/accountant level secret issued on your account which may affect current SwipeClock integrations.* If a secret has not been generated yet, click the "Generate" button to create a new one.

2. **Client/Site API Secret** – this secret provides access to just the site it's created for and all employees within this site. Like partner API secrets, only users with partner/account level administration rights have the ability to generate client API secrets. Users who are client/site level administrators can view these secrets by following these steps:

a. Sign in at https://payrollservers.us/pg
b. Click on “Settings Menu” on the left hand menu
c. Within the “Settings Menu”, click on Client Level API Secret Management
d. If a client API secret has already been generated you can view it. If one has not been generated, please contact your SwipeClock reseller to generate one for you.



Single Sign On

The single sign on workflow allows you to render SwipeClock with your platform as an embedded HTML iframe/object or into a new browser window/tab. No matter which method you choose, the initial set of steps are the same.

Step 1: Authenticate Against SwipeClock’s OAuth Token Endpoint

URL: https://clock.payrollservers.us/AuthenticationService/oauth2/userToken.

You must first create your JWT (**tip:** visit https://jwt.io/#libraries-io for a number of free libraries/SDKs available for use to create your JWT).

Creating a JWT Header

The first part of your JWT will be the header. This identifies the algorithm and token type you’re using. Included in the JWT header are:

Name	Type	Description
alg	String	Identifies the hashing algorithm used being used. Use “HS256”.
typ	String	Identifies the type of token which is JWT. Use “JWT”.

Below is an example of a header of a JWT:

```
{
 alg: "HS256",
 typ: "JWT"
}
```

Creating a JWT Body/Payload

Included in the body/payload of your JWT should be the following elements in your JSON:

Name	Type	Description
iss	String	SwipeClock partner ID
product	String	twpemp = for employee SSO twplogin – for client/site administrator or supervisor SSO
sub	String	partner = if you have a partner API secret client = if you have a site API secret
exp	String	A date in Unix epoch time. Visit https://www.epochconverter.com/ for more information.
siteInfo	Object	An object of name value pairs: • type: "id" • id: site ID of the employee/supervisor
user	Object	An object of name value pairs: • type: ○ "empcode" if you're identifying the employee by their ID/code from the source payroll platform ○ "id" if you're identifying the employee by their SwipeClock clock number. ○ "login" if you're identifying the login/username you'd like to SSO into

Below is an example of the JSON body/payload for your JWT for partner 1, site 69481 and employee with employee code 1234 if you're using a partner level API secret.

```
{
    iss: 1,
    product: "twpemp",
    sub: "partner",
    exp: 1517004886,
    siteInfo: {
      type: "id",
      id: "69481"
    },
    user: {
      type: "empcode",
      id: "1234"
    }
  }
```

Below is an example of the JSON body/payload for your JWT for partner 1, site 69481 and employee with the employee code 1234 if you're using a client level API secret.

```
{
    iss: 69481,
    product: "twpemp",
    sub: "client",
    exp: 1517004886,
    siteInfo: {
      type: "id",
```

```
      id: “69481”
    },
    user: {
      type: "empcode",
      id: “1234”
    }
  }
```

Below is an example of the JSON body/payload for your JWT for partner 1, site 69481 and supervisor with the login ssosupervisorlogin if you’re using a client level API secret.

```
{
    iss: 69481,
    product: "twplogin",
    sub: "client",
    exp: 1517004886,
    siteInfo: {
      type: "id",
      id: “69481”
    },
    user: {
      type: "login",
      id: “ssosupervisorlogin”
    }
  }
```

The differences between using the partner API secret and client API secret is within the following elements of the JSON body/payload:

Entity Name	Value when using a partner API secret	Value when using a client API secret
Iss	Partner/Accountant ID	Client/Site ID
sub	“partner”	“client”

Creating a JWT Signature

The last part of the JWT will be the signature. This is the result of taking the base64 encoding of the header, base64 encoding of the payload and your API secret and signing it with the selected hashing algorithm.

Below is an example of creating your full JWT within JavaScript (before actually posting the JWT to the SwipeClock OAuth Token Endpoint) using a client API secret:

```
<script src="http://kjur.github.io/jsrsasign/jsrsasign-latest-all-min.js"></script>
<script type=”text/javascript”>
  let header = {alg: "HS256", typ: "JWT"};

  let token = {
    iss: 69481,
    product: "twpemp",
    sub: "client",
    exp: 1517004886,
    siteInfo: {
      type: "id",
      id: “69481”
    },
```

```
    user: {
      type: "empcode",
      id: “1234”
    }
  }

  let jwt = KJUR.jws.JWS.sign("HS256", JSON.stringify(header), JSON.stringify(token),
“yUQsBz2aTWIL5iB2o3p2syXfQhfiZyGX0Zr6UT4l75aHb7NRln09gqAujFyR9fbu”);

</script>
```

Posting a JWT to SwipeClock’s OAuth Token Endpoint

Now that you have your JWT, the next step will be posting this to the SwipeClock OAuth Token Endpoint to receive your access token. The JWT must be included in the HTTP Authorization header as a Bearer token. Here’s a sample of using JavaScript (and jQuery) to add the JWT as a Bearer token and calling the SwipeClock OAuth Token Endpoint:

```
<script src="https://code.jquery.com/jquery-3.1.1.min.js"></script>
<script type=”text/javascript”>
$.ajax({
    url: "https://clock.payrollservers.us/AuthenticationService/oauth2/userToken",
    method: "POST",
    headers: {
      "Authorization": `Bearer [myJwt]`,
      "Content-Type": "application/json"
    },
    success: (result, status) => {
      if (result && result.token) {
          // we’ve received an access token!
      } else {
        // An access token was not issued
      }
    },
    error: (o, err) => {
      // An error occurred calling the token endpoint
    }
  })
</script>
```

Here’s a full JavaScript method to put all the pieces above together. In it, the JWT is created and posted to the SwipeClock OAuth Token Endpoint. A JavaScript callback function is used to act on the result of the post to the OAuth Token Endpoint.

```
<script src="https://code.jquery.com/jquery-3.1.1.min.js"></script>
<script src="http://kjur.github.io/jsrsasign/jsrsasign-latest-all-min.js"></script>
<script type=”text/javascript”>

getJWT(“1”, “69481”, “1234”, “yUQsBz2aTWIL5iB2o3p2syXfQhfiZyGX0Zr6UT4l75aHb7NRln09gqAujFyR9fbu”, function(err, jwt){
  if (err) {
      console.log(`Fail: ${err}`);
    } else {
      // JWT authenticated received.  We have two options:
      //  1. Host SwipeClock as an embedded iframe
      //  2. Open up a new browser window/tab into SwipeClock
      console.log(‘Success: ‘ + jwt);
    }
```

```
});

function getJWT(partnerID, siteID, employeeID, apiSecret, callback){

    let header = { alg: "HS256", typ: "JWT" };

    // body/payload token being created is based on using a client api secret
    let token = {
            iss: siteID,
            product: "twpemp",
            sub: "client",
            exp: Math.floor(Date.now() / 1000) + 60 * 5,
            siteInfo: {
               type: "id",
               id: siteID
            },
            user: {
               type: "empcode",
               id: employeeID
            }
    }

    let jwt = KJUR.jws.JWS.sign("HS256", JSON.stringify(header), JSON.stringify(token), apiSecret);

    console.log(`Calling Authentication Service with JWT' + jwt);

    $.ajax({
      url: "https://clock.payrollservers.us/AuthenticationService/oauth2/userToken",
      method: "POST",
      headers: {
        "Authorization": `Bearer ` + jwt,
        "Content-Type": "application/json"
      },
      success: (result, status) => {
        if (result && result.token) {
              // we received an access token!
              callback(null, result.token);
        } else {
              // An access token was not issued
              callback('Status: ' + status + ', Result: ' + JSON.stringify(result), null);
        }
      },
      error: (o, err) => {
              // An error occurred calling the token endpoint
                console.log(o);
      }
     })
}
</script>
```

Step 2: Complete the Single Sign On into SwipeClock's

All the hard work and heavy lifting of the SSO was done in step 1. Step 2 involves taking the access token JWT returned from the SwipeClock OAuth Token Endpoint and using it to complete a single sign on into the SwipeClock system. This can be done either as an embedded iframe/object or into a new browser window/tab.

Either way you choose, the access token JWT you received will be added as a query string to one of the following URLs:

- Web Clock: https://clock.payrollservers.us
- Employee Self Service (ESS): https://payrollservers.us/pg/ess
- Supervisor/Administrator Portal: https://payrollservers.us/pg/login.aspx

For example:

- Web Clock: https://clock.payrollservers.us/?jwt=[myIssuedJwt]
- Employee Self Service (ESS): https://payrollservers.us/pg/ess?jwt=[myIssuedJwt]
- Supervisor/Administrator Portal: https://payrollservers.us/pg/login.aspx?jwt=[myIssuedJwt]

In the examples below, we'll look at three SSO rendering scenarios:

1. Embedding the Web Clock as an iframe
2. SSO into Employee Self Service as a new browser window/tab
3. Supervisor SSO into SwipeClock as a new browser window/tab

Keep in mind, you could implement either rendering option (iframe or new browser/tab) for any of the scenarios above. Other options for embedding include the timecard, schedule and time-off request.

- Time Off: https://clock.payrollservers.us/ess/tor/?jwt=[myIssuedJwt]
- Time Card: https://payrollservers.us/pg/Ess/TimeCard.aspx?jwt=[myIssuedJwt]
- Scheduling: https://clock.payrollservers.us/ess/employee-schedule/?jwt=[myIssuedJwt]

Embedding the Web Clock as iframe

SwipeClock's Web Clock is built to be stand-alone or as an embeddable widget. Embedding as a widget is easily done using an HTML iframe or object tag. Below is a code example showing authentication against the SwipeClock OAuth Token Endpoint and creating an iframe to display the web clock using a client API secret.

```
<!DOCTYPE html>

<html>
<head>
  <title>Embedded Web Clock Sample</title>
</head>
<body>
  <h2 id="processingText">Please wait while we complete SSO and generate the SwipeClock Web Clock</h2>
  <div id="result"></div>
  <script src="https://code.jquery.com/jquery-3.1.1.min.js"></script>
  <script src="http://kjur.github.io/jsrsasign/jsrsasign-latest-all-min.js"></script>
  <script type="text/javascript">
    $(function () {
      // Invoked when the document is ready
      getJWT("1", "69481", "1234", "yUQsBz2aTWIL5iB2o3p2syXfQhfiZyGX0Zr6UT4l75aHb7NRln09gqAujFyR9fbu", function
(err, jwt) {
        if (err) {
```

```
            console.log('Fail: ' + err);
            $('#result').html('Fail: ' +  err);
        } else {
            // JWT authenticated received.
            // Host SwipeClock as an embedded iframe
            let webClockUrl = 'https://clock.payrollservers.us/?enclosed=1&compact=1&showess=1&jwt=' + jwt;
            $('#result').html('<iframe src="' + webClockUrl + '" style="height: 900px; width: 100%;"></iframe>');
            $('#processingText').text('Please use the web clock below');
        }
    });
});

function getJWT(partnerID, siteID, empcode, apiSecret, callback){

    let header = {alg: "HS256", typ: "JWT"};

    let token = {
      iss: siteID,
      product: "twpemp",
      sub: "client",
      exp: Math.floor(Date.now() / 1000) + 60 * 5,
      siteInfo: {
        type: "id",
        id: siteID
      },
      user: {
          type: "empcode",
          id: empcode
      }
     }

    let jwt = KJUR.jws.JWS.sign("HS256", JSON.stringify(header), JSON.stringify(token), apiSecret);

    console.log('Calling Authentication Service with jwt: ' + jwt);

    $.ajax({
      url: "https://clock.payrollservers.us/AuthenticationService/oauth2/userToken",
      method: "POST",
      headers: {
        "Authorization": 'Bearer ' + jwt,
        "Content-Type": "application/json"
      },
      success: (result, status) => {
        if (result && result.token) {
    // we received an access token!
          callback(null, result.token);
        } else {
          // An access token was not issued
          callback('Status: ' + status + ', Result: ' + JSON.stringify(result), null);
        }
      },
      error: (o, err) => {
          // An error occurred calling the token endpoint
          console.log(o);
          $('#result').html('<h3 style="color: red;">See the console log for error details.</h3>')
      }
     })
  }
```

```
  </script>
</body>
</html>
```

When embedding the web clock, you can also apply a few settings via query strings that help control the layout of the clock. These query string settings include:

Query String Name	Value(s)	Description
enclosed	1 (recommended) 0	If set to 1, the background and other extraneous content is removed from the page when rendering.
compact	1 (recommended) 0	If set to 1, the Web Clock UI elements are rendered smaller.
showess	1 0	If set to 1, a single sign on link is shown in the Web Clock to SwipeClock's ESS system (implemented as opening a new browser window/tab).

Employee SSO into SwipeClock's ESS in a New Browser Window/Tab

Getting employee's access into the full SwipeClock ESS system is often a better experience if completed by opening a new browser window/tab. The example below shows how clicking on a button will invoke the SSO call to SwipeClock and result in a new browser window opening with SwipeClock's ESS.

```
<!DOCTYPE html>

<html>
<head>
  <title>SwipeClock ESS SSO</title>
</head>
<body>
  <h2>Please click the button to sign into SwipeClock's ESS system in a new browser window/tab</h2>
  <button id="btnSso">Sign into SwipeClock's ESS</button>
  <div id="result"></div>
  <script src="https://code.jquery.com/jquery-3.1.1.min.js"></script>
  <script src="http://kjur.github.io/jsrsasign/jsrsasign-latest-all-min.js"></script>
  <script type="text/javascript">
    $(function () {
      $('#btnSso').click(function () {
        getJWT("1", "69481", "1234", "yUQsBz2aTWIL5iB2o3p2syXfQhfiZyGX0Zr6UT4l75aHb7NRln09gqAujFyR9fbu", function
(err, jwt) {
          if (err) {
            console.log('Fail: ' + err);
            $('#result').html('Fail: ' + err);
          } else {
            // JWT authenticated received.
            // Access ESS via a new browser tab
            let webClockUrl = 'https://payrollservers.us/pg/ess?jwt=' + jwt;
            window.open(webClockUrl);              }
        });
      });
    });

    function getJWT(partnerID, siteID, empcode, apiSecret, callback) {
```

```
        let header = { alg: "HS256", typ: "JWT" };

        // body/payload token being created is based on using a client api secret
        let token = {
          iss: siteID,
          product: "twpemp",
          sub: "client",
          exp: Math.floor(Date.now() / 1000) + 60 * 5,
          siteInfo: {
            type: "id",
            id: siteID
          },
          user: {
            type: "empcode",
            id: empcode
          }
        }

        let jwt = KJUR.jws.JWS.sign("HS256", JSON.stringify(header), JSON.stringify(token), apiSecret);

        console.log('Calling Authentication Service with ' + jwt);

        $.ajax({
          url: "https://clock.payrollservers.us/AuthenticationService/oauth2/userToken",
          method: "POST",
          headers: {
            "Authorization": 'Bearer ' + jwt,
            "Content-Type": "application/json"
          },
          success: (result, status) => {
            if (result && result.token) {
              // we received an access token!
              callback(null, result.token);
            } else {
              // An access token was not issued
              callback('Status: ' + status + ', Result: ' + JSON.stringify(result), null);
            }
          },
          error: (o, err) => {
            // An error occurred calling the token endpoint
            console.log(o);
            $('#result').html('<h3 style="color: red;">See the console log for error details.</h3>')
          }
        })
      }

    </script>
</body>
</html>
```

Supervisor SSO into SwipeClock's in a New Browser Window/Tab

Getting supervisor (or client/site level administrator) access into the full SwipeClock system is often a better experience if completed by opening a new browser window/tab. The example below shows how clicking on a button will invoke the SSO call to SwipeClock for a supervisor login and result in a new browser window opening with SwipeClock's.

Keep in mind, to SSO into SwipeClock as either a client/site administrator or supervisor, you need to know the user account's SwipeClock login/username. You do not need to know their password. In the example below, the supervisor's login into SwipeClock is ssosupervisorlogin.

```
<!DOCTYPE html>

<html>
<head>
  <title>SwipeClock ESS SSO</title>
</head>
<body>
  <h2>Please click the button to sign into SwipeClock's ESS system in a new browser window/tab</h2>
  <button id="btnSso">Sign into SwipeClock's ESS</button>
  <div id="result"></div>
  <script src="https://code.jquery.com/jquery-3.1.1.min.js"></script>
  <script src="http://kjur.github.io/jsrsasign/jsrsasign-latest-all-min.js"></script>
  <script type="text/javascript">
    $(function () {
      $('#btnSso').click(function () {
        getJWT("1", "69481", "ssosupervisorlogin",
"yUQsBz2aTWIL5iB2o3p2syXfQhfiZyGX0Zr6UT4l75aHb7NRln09gqAujFyR9fbu", function (err, jwt) {
          if (err) {
            console.log('Fail: ' + err);
            $('#result').html('Fail: ' + err);
          } else {
            // JWT authenticated received.
            // Access ESS via a new browser tab
            let webClockUrl = 'https://payrollservers.us/pg/login.aspx?jwt=' + jwt;
            window.open(webClockUrl);          }
        });
      });
    });

    function getJWT(partnerID, siteID, login, apiSecret, callback) {

      let header = { alg: "HS256", typ: "JWT" };

      // body/payload token being created is based on using a client api secret
      let token = {
        iss: siteID,
        product: "twplogin",
        sub: "client",
        exp: Math.floor(Date.now() / 1000) + 60 * 5,
        siteInfo: {
          type: "id",
          id: siteID
        },
        user: {
          type: "login",
          id: login
        }
      }

      let jwt = KJUR.jws.JWS.sign("HS256", JSON.stringify(header), JSON.stringify(token), apiSecret);

      console.log('Calling Authentication Service with ' + jwt);

      $.ajax({
```

```
        url: "https://clock.payrollservers.us/AuthenticationService/oauth2/userToken",
        method: "POST",
        headers: {
          "Authorization": 'Bearer ' + jwt,
          "Content-Type": "application/json"
        },
        success: (result, status) => {
          if (result && result.token) {
            // we received an access token!
            callback(null, result.token);
          } else {
            // An access token was not issued
            callback('Status: ' + status + ', Result: ' + JSON.stringify(result), null);
          }
        },
        error: (o, err) => {
          // An error occurred calling the token endpoint
          console.log(o);
          $('#result').html('<h3 style="color: red;">See the console log for error details.</h3>')
        }
      })
    }

  </script>
</body>
</html>
```

For more information

For more information on SwipeClock's API capabilities, please visit http://developer.swipeclock.com or email developersupport@swipeclock.com.