New Getting Started Documentation
Getting Started with SwipeClock API12

Step 1. Submit SwipeClock's Technology Alliance Program registration form
Step 2. Get demo account / system access instructions from SwipeClock
Step 3. Start work on the integration

Overview

SwipeClock provides timekeeping and scheduling solutions as part of our Workforce Management technology. Our solutions are offered through a diverse channel primarily consisting of payroll bureaus, accounting firms, staffing agencies and Professional Employer Organizations.

This guide will help you easily develop an integration with our TimeWorksPlus timekeeping product. We require third parties to build a minimum set of core integrated functionality to accommodate the exchange of data and provide an improved user experience.

The minimum set of core integrated functionality means, if the two systems have common data, share it. If the two systems require logging in, develop single sign on. Pretty simple.

Demo account / System Access

If a SwipeClock partner has requested this integration, we suggest working with them directly to get access to a test account. If you are not working with a partner, we can provide you with an Integration Partner and a Client account for testing.

The Integration Partner account is the same type of account a SwipeClock Partner uses to manage their timekeeping client accounts.

- Can activate or modify settings on the client account
- Setup additional data mapping for gathering varying values. (e.g. multiple departments, locations, piece counts, tips)

The Client account is used to setup a company in timekeeping. It has settings and employees unique to that account. We identify the Client account by Site ID that can be found in the Integration Partner account's Client List.

- Sync employee data

- Add timekeeping data to get for running payroll
- Verify single sign on between the two systems for a client level login and employee portal single sign on access points.

Refer to our SwipeClock Terminology for more information about TimeWorksPlus account types or login levels.

Getting Started with SwipeClock API12

Step 1. Common integration process:

i. Create a Session
ii. Add single sign on to TimeWorksPlus and Employee Self Service portal
iii. Add New Employee / Update Employee
iv. Get Timekeeping / Payroll Data
v. Get Accrual Data

Step 2. Add single sign on to TimeWorksPlus and Employee Self Service portal

Example of single sign-on

Single Sign On for TimeWorksPlus and Employee Self Service Portal - Create a session for their user by getting a One Time Credential and then post a form of the login handler containing the user name and one time credential. When automatically directing a user to SwipeClock, we recommend configuring custom URL's to use to redirect the user back to your system when their session ends.

When getting a One Time Credential two calls to the TimeWorksPlus API Web Service are required:

1. Get and authenticate yourself through the CreateSessionSelectClient method.
2. Get an OTC using either the GetOneTimeCredential (for supervisors and clients) or GetOneTimeCredentialESS (for employees).

The Login Handler page will validate the user by their password or OTC. The URL is https://payrollservers.us/pg/LoginHandler.ashx. The Login Handler page will redirect the user to the correct SwipeClock page based on the

login's access level. The Login Handler page expects two form fields, a "Login" and "Password". Here is an example of a form that automatically posts to our handler page:

```
<html>  <head></head>
  <body onload="document.forms[0].submit();" >
    <form id="login" action="https://payrollservers.us/pg/LoginHandler.ashx"
method="post">
      <input name="login" type="text" value="UserName"></input>
      <input name="password" type="text" value="Password"></input>
    </form>
</body></html>
```

The following code snippet shows how to direct a user from a partner's web site to SwipeClock with a single sign on.

Key API calls: CreateSessionSelectClient, GetOneTimeCredential

```
    private const string _accountantUsr = "myAccountant";
    private const string _accountantPwd = "myPassword";
    private const string _clientUsr = "myClient";
    private const string _site = "99999";

    protected void btnGotoSwipeClock_Click(object sender, EventArgs e)
    {
        API12SoapClient client = new  API12SoapClient("API12Soap");
        var session = client.CreateSessionSelectClient(_accountantUsr, _accountantPwd, null, "M
C2ID", _site);
        var oneTimeCredential = client.GetOneTimeCredential(session.SessionID, _clientUsr);

        var sb = new StringBuilder();
        sb.AppendLine(@"<html>");
        sb.AppendLine(@"<head></head>");
        sb.AppendLine(@"<body onload = 'document.forms[0].submit();' >");
        sb.AppendLine(@"<form id = ""login"" action = ""https://payrollservers.us/pg/LoginHandle
r.ashx"" method = ""post"" >");
        sb.AppendLine(@"<input name = ""login"" type = ""text"" value = """+ _clientUsr + @""" />");
        sb.AppendLine(@"<input name = ""password"" type = ""text"" value = """+  oneTimeCrede
ntial.StringResult + @""" />");
        sb.AppendLine(@"</form>");
        sb.AppendLine(@"</body>");
        sb.AppendLine(@"</html>");

        HttpContext.Current.Response.Clear();
        HttpContext.Current.Response.ContentType = "text/html";
```

```
        HttpContext.Current.Response.Write(sb.ToString());
        HttpContext.Current.Response.End();
    }

/*
    Example:
    <html>
        <head></head>
        <body onload = 'document.forms[0].submit();' >
        <form id = "login" action = "https://payrollservers.us/pg/LoginHandler.ashx" method = "po
st" >
            <input name = "login" type = "text" value = "clientNameHere" />
            <input name = "password" type = "text" value = "oneTimeCredentialHere" />
        </form>
        </body>
    </html>
*/
```

Step 3. Add New Employee / Update Employee

(See below for Employee Schema information)

Add New Employee

The following code snippet shows how to add two new employee:

> "John Doe" whose employee code is "H" in department "Accounting" and at Location "Corporate".
>
> "Jane Smith" has an employee code of "I" works in "Accounting" whose location is "Remote"

Key API calls: CreateSessionSelectClient, UpdateEmployeeFields

```
private void AddEmployees()
{
    API12SoapClient client = new API12SoapClient("API12Soap");
    var session = client.CreateSessionSelectClient("user", "password", null, "ClientName", "MyCo
mpany");

    List<EmployeeFieldsToUpdate> employees = new List<EmployeeFieldsToUpdate>();
    EmployeeFieldsToUpdate employee;
    List<FieldToUpdate> fieldsToUpdate;
    FieldToUpdate field;
    var effectiveDate = DateTime.UtcNow.ToString();

    /* Create John Doe */
    employee = new EmployeeFieldsToUpdate();
    employee.Identifier = "H";
    employees.Add(employee);
```

```
    fieldsToUpdate = new List<FieldToUpdate>();

    field = new FieldToUpdate() { FieldName = "EmployeeCode", Value = "H" };
    fieldsToUpdate.Add(field);

    field = new FieldToUpdate() { FieldName = "LastName", Value = "Doe" };
    fieldsToUpdate.Add(field);

    field = new FieldToUpdate() { FieldName = "FirstName", Value = "John" };
    fieldsToUpdate.Add(field);

    field = new FieldToUpdate() { FieldName = "Department", Value = "Accounting", EffectiveDa
te = effectiveDate };
    fieldsToUpdate.Add(field);

    field = new FieldToUpdate() { FieldName = "Location", Value = "Corporate", EffectiveDate =
effectiveDate };
    fieldsToUpdate.Add(field);

    employee.FieldsToUpdate = fieldsToUpdate.ToArray();


    /* Create Jane Smith */
    employee = new EmployeeFieldsToUpdate();
    employee.Identifier = "I";
    employees.Add(employee);

    fieldsToUpdate = new List<FieldToUpdate>();

    field = new FieldToUpdate() { FieldName = "EmployeeCode", Value = "I" };
    fieldsToUpdate.Add(field);

    field = new FieldToUpdate() { FieldName = "LastName", Value = "Smith", EffectiveDate = eff
ectiveDate };
    fieldsToUpdate.Add(field);

    field = new FieldToUpdate() { FieldName = "FirstName", Value = "Jane" };
    fieldsToUpdate.Add(field);

    field = new FieldToUpdate() { FieldName = "Department", Value = "Accounting", EffectiveDa
te = effectiveDate };
    fieldsToUpdate.Add(field);

    field = new FieldToUpdate() { FieldName = "Location", Value = "Remote", EffectiveDate = ef
fectiveDate };
    fieldsToUpdate.Add(field);
```

```
    employee.FieldsToUpdate = fieldsToUpdate.ToArray();

    var result = client.UpdateEmployeeFields(session.SessionID, string.Empty, true, EmployeeId
entifierField.EMPLOYEECODE, employees.ToArray(), SecondaryIdentifierField.NOTUSED);
}
```

Update Employee Field

The following code snippet shows how to change the department of an existing employee whose employer code is "A" to "Development".

Key API calls: CreateSessionSelectClient, UpdateEmployeeFields

```
private void UpdateEmployees()
{
    API12SoapClient client = new API12SoapClient("API12Soap");
    var session = client.CreateSessionSelectClient("user", "password", null, "ClientName", "MyCo
mpany");

    List<EmployeeFieldsToUpdate> employees = new List<EmployeeFieldsToUpdate>();
    EmployeeFieldsToUpdate employee;
    List<FieldToUpdate> fieldsToUpdate;
    var effectiveDate = DateTime.UtcNow.ToString();

    employee = new EmployeeFieldsToUpdate();
    employee.Identifier = "A";
    employees.Add(employee);

    fieldsToUpdate = new List<FieldToUpdate>();

    fieldsToUpdate.Add(
        new FieldToUpdate() { FieldName = "Department", Value = "Development", EffectiveD
ate = effectiveDate });

    employee.FieldsToUpdate = fieldsToUpdate.ToArray();

    var result = client.UpdateEmployeeFields(session.SessionID, string.Empty, true,
            EmployeeIdentifierField.EMPLOYEECODE, employees.ToArray(), SecondaryIdenti
fierField.NOTUSED);
}
```

Step 4. Get Timekeeping / Payroll Data

Getting Payroll Data

The following code snippet gets payroll data for the specified period. The format of the returned data is dependent on the "FormatName" parameter. In the following example the format is "xml".

```
private void GetPayrollData()
 {
   API12SoapClient client = new API12SoapClient("API12Soap");
    var session = client.CreateSessionSelectClient("user", "password", null, "MC2ID", "some SiteNumber");
    var beginDate = new DateTime(2015, 11, 26);
    var endDate = new DateTime(2015, 12, 2);
    var results = client.GetActivityFile(session.SessionID, beginDate.ToString(), endDate.ToString(), "xml" , null);

    /*
   -            Sample Output

      <CardReport>
            <BeginDate>2015-11-26</BeginDate>
            <EndDate>2015-12-02</EndDate>
            <Site>
                  <SiteNumber>999999</SiteNumber>
                  <ClientTag/>
                  <SiteName>RayCo</SiteName>
                  <UseMinuteRounding>0</UseMinuteRounding>
                  <Idefinition />
                  <Jdefinition />
                  <Kdefinition />
                  <Xdefinition />
                  <Ydefinition />
                  <Zdefinition />
                  <PayCodes />
                  <PayCodeCategories />
                  <DateTimeUTC>2015-12-08T15:05:30</DateTimeUTC>
                  <DateTimeSiteLocal>2015-12-08T08:05:30</DateTimeSiteLocal>
            </Site>
            <Cards>
                  <Card>
                        <Employee>
                              <UniqueID>59300457</UniqueID>
                              <FullName>John, Doe</FullName>
                              <FirstName>John</FirstName>
                              <MiddleName/>
                              <LastName>Doe</LastName>
                              <EmployeeCode>A</EmployeeCode>
                              <HomeDepartment>Development1</HomeDepartment>
                              <HomeLocation>Good Bye</HomeLocation>
```

```
                    <HomeSupervisor />
                    <Options />
                    <Home1 />
                    <Home2 />
                    <Home3 />
                    <PayRate0 />
                    <PayRate1 />
                    <PayRate2 />
                    <PayRate3 />
                    <Export>Export</Export>
                    <StartDate />
                    <EndDate />
                </Employee>
                <Punches>
                    <Punch>
                        <id>8362825138824627548</id>
                        <PunchDate>2015-11-26</PunchDate>
                        <Type>Times</Type>
                        <InDT>2015-11-26T09:00:00</InDT>
                        <InUnrounded />
                        <InUTC>2015-11-26T16:00:00</InUTC>
                        <OutDT>2015-11-26T17:00:00</OutDT>
                        <OutUnrounded />
                        <OutUTC>2015-11-27T00:00:00</OutUTC>
                        <LunchMinutes>0</LunchMinutes>
                        <Hours>8</Hours>
                        <NonOTHours>8</NonOTHours>
                        <OT1Hours/>
                        <OT1Category>Overtime</OT1Category>
                        <OT1PayRate>0</OT1PayRate>
                        <OT2Hours/>
                        <OT2Category>Doubletime</OT2Category>
                        <OT2PayRate>0</OT2PayRate>
                        <PayRate>0</PayRate>
                        <ApplicablePayRate>0</ApplicablePayRate>
                        <Category/>
                        <Department>Development1</Department>
                        <Location>Good Bye</Location>
                        <Supervisor />
                        <I>0</I><J>0</J><K>0</K><X />
                        <Y />
                        <Z />
                    </Punch>
            </Punches>
            ...
        </Card>

    </CardReport>

*/

}
```

Get Finalized Payroll for a Specific Date

The following code snippet gets the finalized payroll data for the specified period beginning November 26, 2015.

```
private void GetFinalizedPayroll()
{
   /* Get the Finialized Payroll Record Number for the period beginning  November 26, 2015 *
/
   API12SoapClient client = new API12SoapClient("API12Soap");
   var session = client.CreateSessionSelectClient("user", "password", null, "MC2ID", "someSite
Number");
   var listOfFinalizedPayrolls = client.GetListOfFinalizedPayrolls(session.SessionID);
   var beginDate = "2015-11-26";
   var recordNumberStr = listOfFinalizedPayrolls.PayPeriods.SelectSingleNode(@"./FinalizedP
ayPeriod[./BeginDate = '" + beginDate + "']").SelectSingleNode("RecordNumber").InnerText;
   long recordNumber = long.Parse(recordNumberStr);
   var payroll = client.GetFinalizedPayroll(session.SessionID, recordNumber);
}
```

Get Payroll Data Based on Clock Prompts using OptionalLaborMapping for TimeWorks Client

The following code snippet gets the payroll data for the specific employees who have X clock prompt with a value of "REMOTE". Scenario assumes an X clock prompt is setup that tracks the employee's location for particular types of tasks. Note in this example the method GetActivityFile ("xml") is used; GetFinalizedPayroll and GetUnFinalizedPayroll could also be used for a complete set of data which includes all activity (i.e. edits…).

```
private void GetPayrollData()
{
   /* Get the payroll data for the given time period, where there is X clock prompt
      data and the value is "REMOTE" */

   API12SoapClient client = new API12SoapClient("API12Soap");
   var session = client.CreateSessionSelectClient("user", "password", null, "MC2ID", "99999");
   var beginDate = new DateTime(2015, 12, 17);
   var endDate = new DateTime(2015, 12, 23);
   var results = client.GetActivityFile(session.SessionID, beginDate.ToString(), endDate.ToStrin
g(), "xml", null);
   XElement root = XElement.Parse(results.FormatText);

   var punches = (from card in root.Descendants("Card")
                  from employee in card.Descendants("Employee")
```

```
                from punch in card.Descendants("Punch")
                where punch.Element("X").Value == "REMOTE"
                select new
                {
                    Name = employee.Element("FullName").Value ,
                    Hours = punch.Element("Hours").Value,
                                InUTC =
                                                punch.Descendants("InUTC").Count() >
                                                0 ? punch.Element("InUTC").Value : null,
                    OutUTC = punch.Descendants("OutUTC").Count() > 0 ? punch.Element("OutUT
C").Value : null,
                }).ToList();
}
```

Get Payroll Data with Labor Mapping for TimeWorksPlus Client

Steps to Setup:

1. Log in with your Integration Partner credentials and select client (this will need to be set up for EACH client, as needed)
2. Select Accountant Menu
3. Click File Format Maintenance
4. Enter format name you are using to Get Timekeeping Data in the Format Name box, click Check
5. Select the field(s) you would like mapped. Note: If you see two Department options this means you have a clock prompt called Department and the Department field in the Employee Setup screen. By default, the system will always pull from the Employee Setup.
6. Save

Step 5. Get Accrual Data

Get Accrual Data for a Finalized Payroll

The following code snippet gets the accrual data for each specific employee for a finalized pay period beginning on a specific date.
Methods that can be used GetFinalizedPayroll, GetUnFinalizedPayroll and GetActivityFile with format swipeclockpto (in order of recommended use) to get accruals.

```
    private void GetAccrualsForPayPeriod()
    {
        API12SoapClient client = new API12SoapClient("API12Soap");
        var session = client.CreateSessionSelectClient("user", "password", null, "MC2ID", "siteNum
ber");
        var listOfFinalizedPayrolls = client.GetListOfFinalizedPayrolls(session.SessionID);
        var beginDate = "2015-08-01";
```

```
        var payPeriods = XElement.Parse(listOfFinalizedPayrolls.PayPeriods.OuterXml);
        var number = (from period in payPeriods.Descendants("FinalizedPayPeriod")
                      from date in period.Descendants("BeginDate")
                      from record in period.Descendants("RecordNumber")
                      where date.Value == beginDate
                      select record.Value).FirstOrDefault();

        long recordNumber = long.Parse(number);

        var payroll = client.GetFinalizedPayroll(session.SessionID, recordNumber);

        if(payroll.Success != true) {
           return;
        }

        XElement root = XElement.Parse(payroll.FinalizedPayrollDocument.OuterXml);

        var items = (from item in (
                    (from card in root.Descendants("TimeCard")
                     from employee in card.Descendants("Employee")
                     from timeCardDate in (from c in card.Descendants("Dates") select c.LastNode)
                     select new
                     {
                        LastName = employee.Attribute("LastName").Value,
                        FirstName = employee.Attribute("FirstName").Value,
                        MiddleName = employee.Attribute("MiddleName").Value,
                        EndingAccrualBalances = ((XElement)timeCardDate).Descendants("EndingAcc
rualBalances").Descendants().ToList()
                     }))
                     select new { item.FirstName, item.MiddleName, item.LastName, item.EndingAccr
ualBalances }).ToList();

    }
}
```

The following code snippet gets accruals for a given time period in a simple flat file format.

```
    private void GetAccrualsForPayPeriodSimpleFormat()
    {
        API12SoapClient client = new API12SoapClient("API12Soap");
        var session = client.CreateSessionSelectClient("user", "password", null, "MC2ID", "siteNum
ber");
        var accruals = client.GetActivityFile(session.SessionID, "2015-08-01",
                    "2015-08-07","swipeclockpto", null);
```

```
    if(accruals.Success != true) {
        return;
    }
}
```

Working with Effective Dates

The following code snippet is to explain how effective dates work. Below shows setting an employee's "PayRate1" to a value of 100, effective December 1st, 2015 and then setting "PayRate1" to 200, January 1st, 2016. If the system requires calculations using PayRate1 during the month of December 2015 it will use 100. After December 2015 it will use 200.

IMPORTANT: An error occurs when calling the method UpdateEmployeeFields on fields that require an effective date and NO effective is supplied. These are some of the fields that require an effective date when being EMPLOYEE TYPE, TITLE, DEPARTMENT, LOCATION, SUPERVISOR, HOME1, HOME2, HOME3, PAYRATE0, PAYRATE1, PAYRATE2, PAYRATE3, AUTOLUNCHHOURS, LUNCHMINUTES… If unsure if the "EffectiveDate" property needs to be set, set it. This property will be ignored if it is not need.

```
private void WorkingWithEffectiveDates()
{
    API12SoapClient client = new API12SoapClient("API12Soap");
    var session = client.CreateSessionSelectClient("user", "password", null, "ClientName", "Some
Company");

    List<EmployeeFieldsToUpdate> employees = new List<EmployeeFieldsToUpdate>();
    EmployeeFieldsToUpdate employee;
    List<FieldToUpdate> fieldsToUpdate;


    string effectiveDate =  new DateTime(2015, 12, 1).ToString();

    employee = new EmployeeFieldsToUpdate();
    employee.Identifier = "A";
    employees.Add(employee);

    fieldsToUpdate = new List<FieldToUpdate>();

    fieldsToUpdate.Add(
        new FieldToUpdate()
        {
            FieldName = "PayRate1",
            Value = "100",
            EffectiveDate = effectiveDate
        });
```

```
    employee.FieldsToUpdate = fieldsToUpdate.ToArray();

    var result = client.UpdateEmployeeFields(session.SessionID, string.Empty, true,
                EmployeeIdentifierField.EMPLOYEECODE,
                employees.ToArray(), SecondaryIdentifierField.NOTUSED);

    employees = new List<EmployeeFieldsToUpdate>();
    effectiveDate = new DateTime(2016, 1, 1).ToString();

    employee = new EmployeeFieldsToUpdate();
    employee.Identifier = "A";
    employees.Add(employee);

    fieldsToUpdate = new List<FieldToUpdate>();

    fieldsToUpdate.Add(
          new FieldToUpdate()
          {
             FieldName = "PayRate1",
             Value = "200",
             EffectiveDate = effectiveDate
          });


    employee.FieldsToUpdate = fieldsToUpdate.ToArray();

    result = client.UpdateEmployeeFields(session.SessionID, string.Empty, true,
                EmployeeIdentifierField.EMPLOYEECODE,
                employees.ToArray(), SecondaryIdentifierField.NOTUSED);
}
```

Additional Information

- Employee schema information

  Identity Fields (date-less, these fields do not have effective dates)

  - EMPLOYEECODE
  - FIRSTNAME
  - MIDDLENAME
  - LASTNAME
  - DESIGNATION
  - PHONE
  - EMAIL
  - STARTDATE
  - ENDDATE
  - EXPORTBLOCK

- WEBCLOCKENABLED
- OPTIONS

Below Optional Fields (must be activated with a Rule):

- MOBILEENABLED
- MOBILEPUNCHENABLED
- GEODATAENABLED

Employee Data (Effective Dated, these fields have dates)

Standard fields (always available):

- EMPLOYEE TYPE
- TITLE
- DEPARTMENT
- LOCATION
- SUPERVISOR
- HOME1
- HOME2
- HOME3

Below Optional Fields that are created and updated the same as other Employee Data fields (must be activated with a Rule):

- PAYRATE0 - Payrates Rule
- PAYRATE1 - Payrates Rule
- PAYRATE2 - Payrates Rule
- PAYRATE3 - Payrates Rule
- AUTOLUNCHHOURS - AutoLunch Rule
- LUNCHMINUTES - AutoLunch Rule
- DEPARTMENT1 to DEPARTMENT9 - Departments Rule
- LOCATION1 to LOCATION9 - Locations Rule
- HOME4 to HOME9 - HomeFields Rule
- BIRTHDAY - Birthday Rule

Card/Clock numbers and password

In addition to those fields you can update credentials with:

- ClockNumberAdd – this adds a card number. (limit 15 characters)
- ClockNumberReplace – this replaces all card numbers and adds the one you send. (limit 15 characters)
- PASSWORD – this sets the Web Password. (8 characters min, 15 max)

Best Practice – send the password one-time. Why? Employees are asked to change their password until first login and sending the password again will wipe out the password that has been set by the employee.