TeamSeer API v1_0_1

For Customer Use

CONTENTS

Copyright

Document Control

Version	Date	Author/Reviewer	Comments
1.0	29/06/2012	Karen Fergusson	
1.1	18/2/2014	Lettie Noel	Footer updated
1.2	18/2/2014	Alex Dutton	Added getUsersChangedSinceLastAPIRequest and updated to latest API version

Introducing SOAP API

TeamSeer provides programmatic access to your organization's information using a simple, powerful, and secure application programming interface (API). To use this document, you should have a basic familiarity with software development, Web services, and the TeamSeer user interface.

If your organization has business processes that would benefit from integrated information on leave and absence, you can use the SOAP API to retrieve that data programmatically.

Use SOAP API in any language that supports Web services.

SOAP API support policy

Backward Compatibility

TeamSeer will strive to make backward compatibility easy when using the API.

As we release new APIs, these will have a new WSDL file.

We maintain support for each SOAP API version across releases of the platform software. SOAP API is backward compatible in that an application created to work with a given SOAP API version will continue to work with that same SOAP API version in future platform software releases.

TeamSeer does not guarantee that an application written against one SOAP API version will work with future SOAP API versions: Changes in method signatures and data representations are often required as we continue to enhance SOAP API. However, we strive to keep SOAP API consistent from version to version with minimal if any changes required to port applications to newer SOAP API versions.

SOAP API End-of-Life

TeamSeer is committed to supporting each SOAP API version for as long as practically possible. When a SOAP API version is scheduled to be unsupported, an advance end-of-life notice will be given at least two months before support for SOAP API version is ended. TeamSeer will directly notify customers using SOAP API versions scheduled for end of life.

Use of SOAP API in business-critical applications

TeamSeer is not designed as a business critical application. There is a risk that by using the API that data from TeamSeer and embedding it in a business critical application you can introduce unplanned risk into your organisation. TeamSeer's terms and conditions specify:

7.7 EXCEPT IN RESPECT OF AN INDEMNIFICATION OBLIGATION OR AS A RESULT OF A BREACH OF A CONFIDENTIALITY OBLIGATION: NEITHER PARTY WILL BE LIABLE FOR INCIDENTAL, SPECIAL OR CONSEQUENTIAL DAMAGES.

An example might be a risk management application relating to oil rigs: whether an employee is on or off a particular oil rig. If there was a fire on that oil rig then data from TeamSeer might be used to work out who is on or off the rig. If there was a failure in TeamSeer or in the API at that critical moment, then TeamSeer cannot be liable for this.

API limits

We monitor API usage, but a "fair use" policy applies. We do not charge for API calls at the current time. However may introduce this in the future, so make sure that you are not over-using the API (we did have a client who wrote code that went into an endless loop and was doing 100 API calls a second for an entire weekend!)

API consultancy

If you would like support, consultancy or testing infrastructure services we would be delighted to help with this but we do charge an hourly and daily rate for this. Please contact us for details.

API development

At some point we will probably:

- implement a REST-ful API (and deprecate the SOAP API)
- add a lot more API calls
- restructure the DayActivity array data type

API End-Point

The TeamSeer API end-point is

https://www.teamseer.com/services/soap/coreapi/1_0_1/teamseer_core_api.wsdl

API Calls

API	Parameters	Description
authenticate	companyId, companyKey, apiVersion	Authenticate the session: Check if the company Id is valid; Check if the IP address is registered; Check if the company Key can match; Set the API version NO.; Only if authentication is successful, the other API operations are allowed.
setUserIdentifier	identifierType	Set the identifier type used to locate a user; Valid options include ‘userId’, ‘emailAddress’ and ‘payrollId’.
setDateFormat	dateFormat	Set the date format used to load data; Valid options include ‘dateIx’ (date offset from 2005-01-01) and ‘yyyy-mm-dd’
getChangesSinceLastAPIRequest		Get the changes since the last API request; Result will actually be a day-activity array; Fields of each DayActivity include “userIdentifier”, “date”,”typeStr”, ”statusStr”,”categoryArr”,”hasNotes”,”needsApproval”
getActiveUsers		Get a list of the active users (identifiers);
getRecordsFor	userIdentifier, fromDate, toDate	Get all the records for a given user in a date range: Check if the user identifier is valid; Check if “fromDate” and “toDate” are in valid date format; Result will actually be a DayActivity array; Fields of each day-activity include “userIdentifier”, “date”,”typeStr”,”statusStr”,”categoryArr”,”hasNotes”,”needsApproval”
getUsersChangedSinceLastAPIRequest		Get a list of the users whose DayActivity records have changed since the last API request (identifiers);

Important Note!

For SOAP you need to store the session data (sent as a cookie) so that you remain authenticated from one SOAP request to the next. You may need to do this in C# for example using

```
coreAPI.teamseer_core_apiService s = new coreAPI.teamseer_core_apiService();
s.CookieContainer = new CookieContainer();
```

Interpreting DayActivity record structure

typeStr

A two-digit format, the first digit representing the AM, the second digit representing the PM.

Each digit is in ASCII format, less 48.

e.g. 01 would mean that a leave type of 0 was booked in the AM, and 1 in the PM.
BB would mean that a leave type of 18 was booked in the AM and the PM.

The function we use to determine the type index at a given
typeIx(timeIx) = charCodeAt(typeStr[timeIx])-48;

The type indexes defined map to the Types table in HR Admin > Setup > Types.

The common types are
0: Holiday
1: Sickness
2: Miscellaneous

StatusStr

A two-digit format, the first representing the status of the AM leave, and the second the status of the PM leave.

0 – Nothing booked
1 – Leave Requested
2 – Leave Approved
3 – Cancellation of leave requested (would then go back to 0 when that cancellation request is approved)

CategoryArr

A 2 part string, the first representing the category of the AM leave, and the second the category of the PM leave.

For example 10,10

The function we use to determine the category index at a given timeIx (0=am,1=pm)

```
if(categoryArr!==null)

{

        cArr=explode(",",categoryArr);

        categoryIx(timeIx)=(int)cArr[timeIx];
```

}

Where a leave category has been booked (the level below type), the Category Index as per the HR Admin > Setup > Categories grid will be entered for the relevant AM or PM.

For hourly leave types, the number of minutes in the AM or PM is entered here instead of a Category Index. There are no Category Indexes for hourly leave types.

needsApproval
does some element of the day need approval – yes or no?

Code Sample – C#

You will need a project with simple form Form1 with a button1, a textBox1, and a web reference called coreAPI.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.Net;

namespace TeamSeerCoreAPI
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void addNote(string s)
        {
            textBox1.Text = textBox1.Text + s + "\r\n";
        }

        private void addBlankLine()
        {
            addNote("");
        }

        private void addRecord(coreAPI.DayActivityAPI d)
        {
            addNote(d.userIdentifier + "," + d.date + "," + d.statusStr + "," + d.typeStr + "," +
d.categoryArr);
        }

        private void addRecordCount(int ct)
        {
            addNote(ct + " records");
            addBlankLine();
        }

        private void addRecords(coreAPI.DayActivityAPI[] dayActivities)
        {
            int ct = 0;
            foreach (coreAPI.DayActivityAPI d in dayActivities)
            {
                addRecord(d);
                ct++;
            }
            addRecordCount(ct);
        }

        private void button1_Click(object sender, EventArgs e)
        {
            textBox1.Text = "";

            coreAPI.teamseer_core_apiService s = new coreAPI.teamseer_core_apiService();
            s.CookieContainer = new CookieContainer();
            try
            {
                bool a = s.authenticate(1, "authcode", "1_0_0");

                if (a)
                {
                    s.setDateFormat("yyyy-mm-dd");
                    s.setUserIdentifier("emailAddress");

                    addNote("Records changed since last API request");
                    addNote("======================================");
                    coreAPI.DayActivityAPI[] dayActivities;
                    dayActivities = s.getChangesSinceLastAPIRequest();
                    addRecords(dayActivities);

                    string[] users = s.getActiveUsers();
                    foreach (string u in users)
                    {
                        addNote("Records for " + u);
                        addNote("================================");
                        dayActivities = s.getRecordsFor(u, "2011-01-01", "2011-12-31");
                        addRecords(dayActivities);
                    }
                }
            }
            catch (Exception ex)
            {
                textBox1.Text = textBox1.Text + "Error: " + ex.ToString();
            }
        }
    }
}
```