Scilab Cloud API

User Guide

Scilab Cloud API gives access to your engineering and simulation knowledge through web services which are accessible by any network-connected machine.

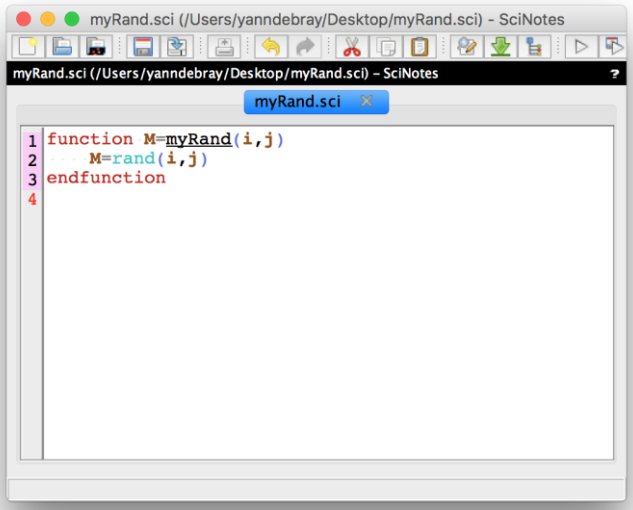
Table of contents

Already selected by clients such as Sanofi, Scilab Cloud allows you to:
- Easily collaborate with colleagues and partners who don't need to master Scilab
- Centralize your data and your algorithms
- Protect the intellectual property of your simulation and post-processing codes
- Simplify and control your application or API deployment

Before deployment

Model your algorithms with Scilab function

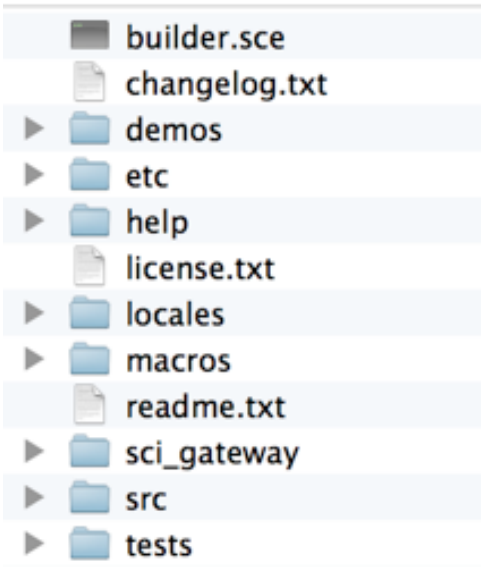
Write each function that you want to expose as a web service, in a separate script file with the extension *.sci*. This script has to start with the command *function* and ends with the command *endfunction.* For more details on how to write functions in Scilab:



https://help.scilab.org/doc/5.5.2/en_US/functions.html

Package your functions in a toolbox

In order to distribute the Scilab functions called from Scilab Cloud API, the developer of the functions will need to follow a formalism in the development of its code. The functions written in Scilab will need to be packaged in our ATOMS format (AuTomatic mOdules Management for Scilab), in order to expose the functions as web services.

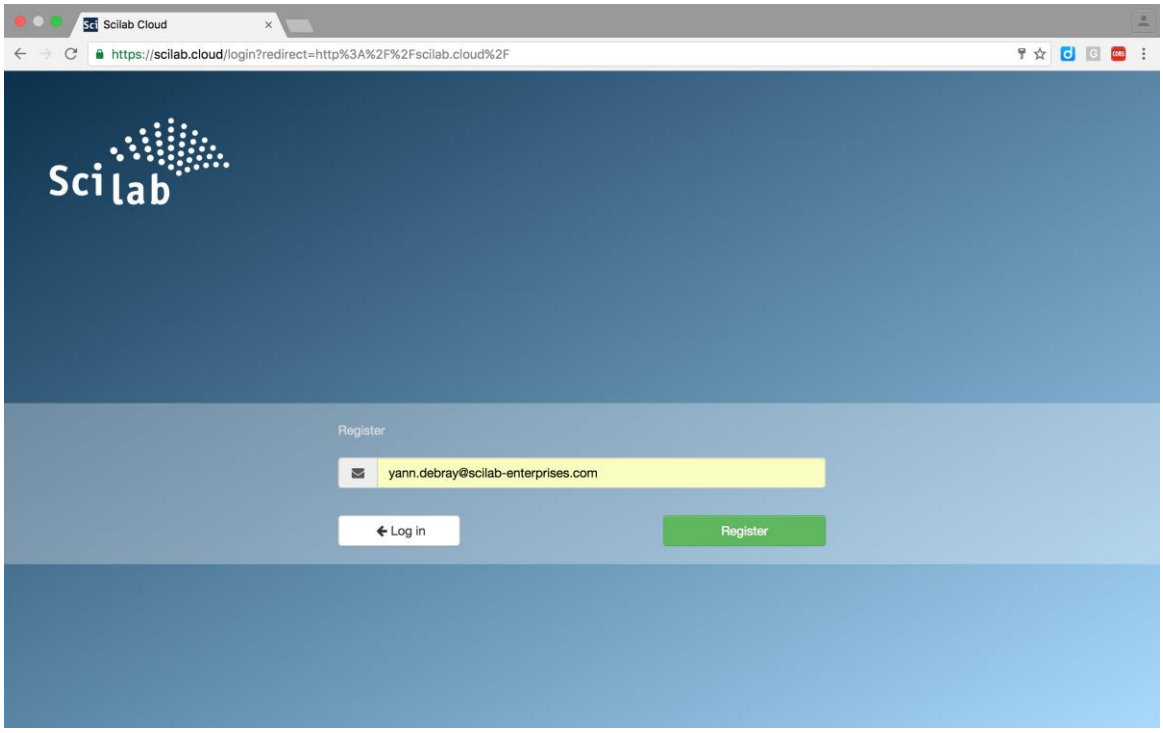


This view shows how the code is being structured on the side of the Scilab ATOMS toolbox. All the function need to be saved individually in .sci files in the macros folder.
For more information on how to package your application in an ATOMS toolbox, please refer to this link:
https://wiki.scilab.org/howto/Create%20a%20toolbox

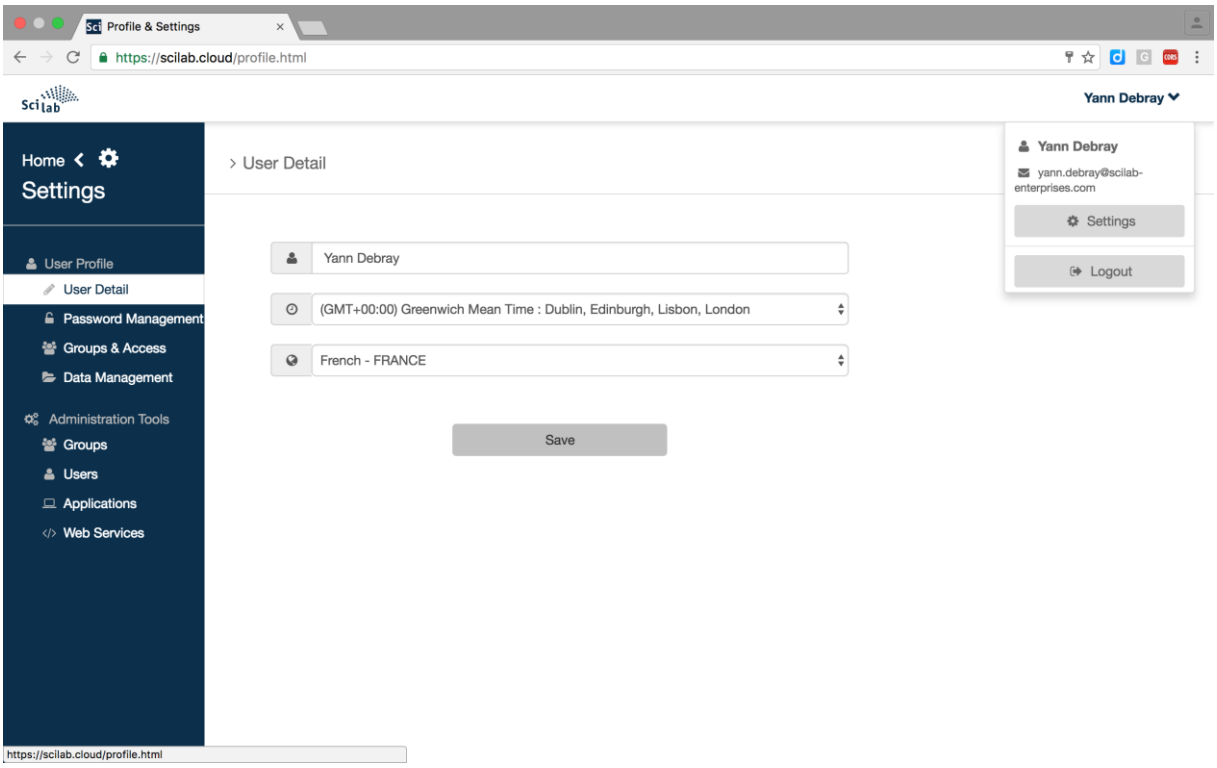
Deployment on Scilab Cloud

Upload your functions through Scilab Cloud

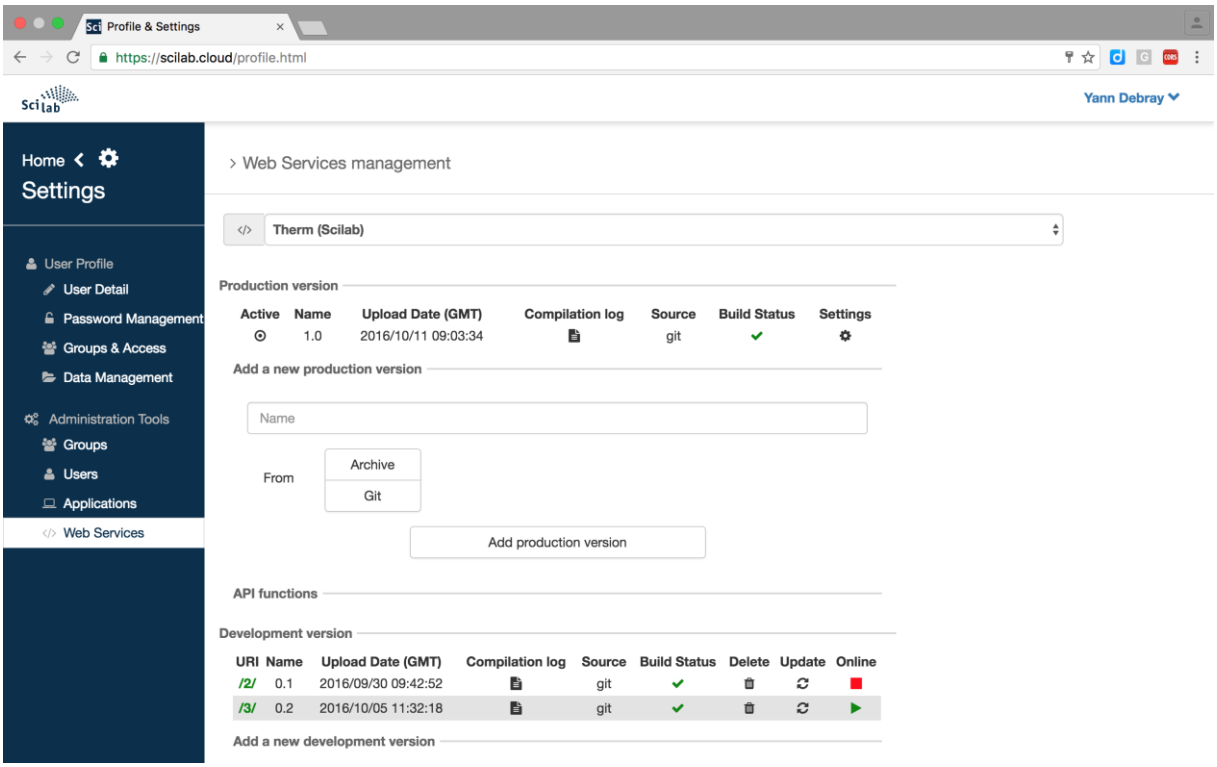
To access the Scilab Cloud administration interface, you have to contact the Scilab team and subscribe to a Scilab Cloud account: team@scilab.io



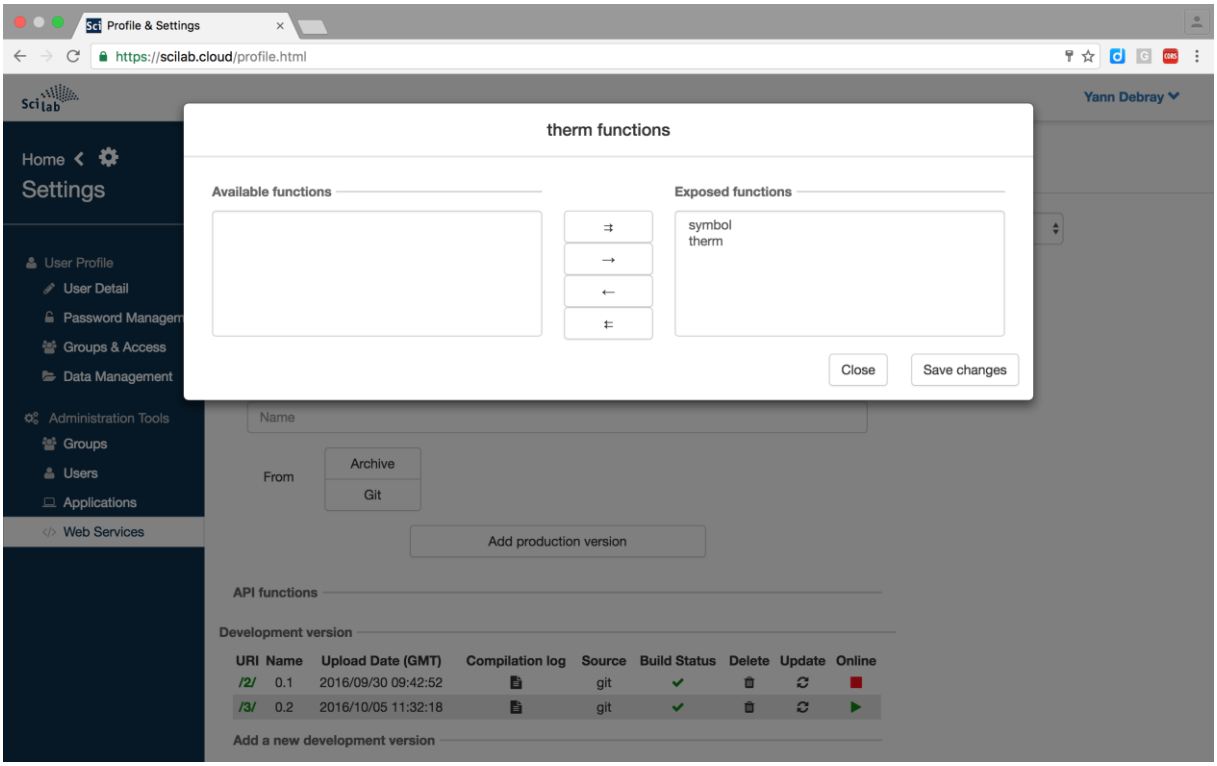
This will give you access to the following administration interface:



In the Web services menu, you will have the ability to upload new versions of your web service, in the format of a toolbox described previously.



By clicking on settings on the production version, you can choose the functions to expose as a web service:

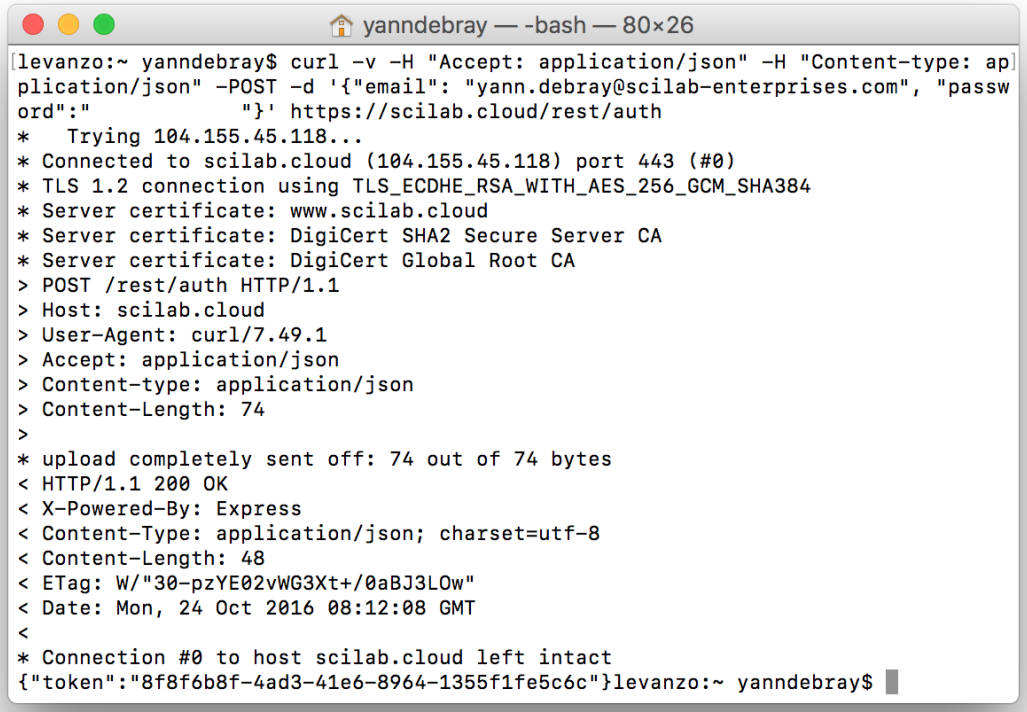


Integration of the API in a third part code

Any third part code can call your web service as a classic REST API, with a HTTP request at https://scilab.cloud/rest/auth

Here is an example how to authenticate yourself with a shell command in your terminal:

```
curl -v -H "Accept: application/json" -H "Content-type: application/json" -
POST -d '{"email": "email", "password":"password"}'
https://scilab.cloud/rest/auth
```

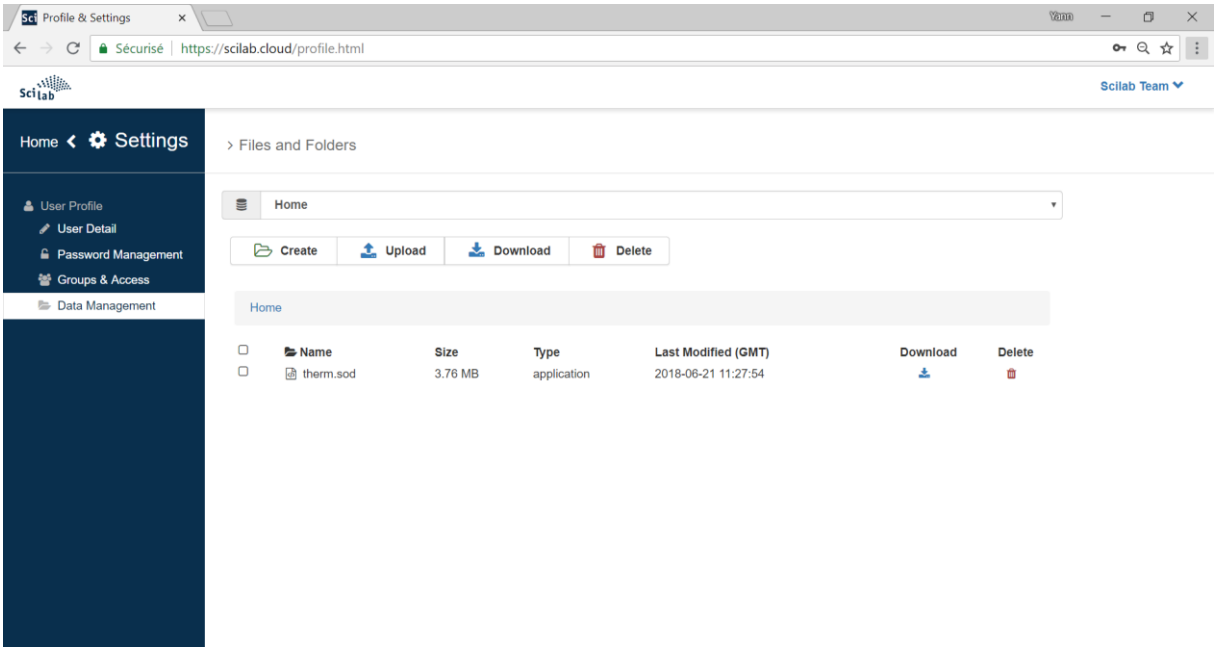


After this phase of authentication, you have to include the token that you've been delivered in every request that you make to the web service. This token is active for 1 day. The data send via the HTTP request should be provided as JSON (in the current version of Scilab Cloud API). Here is an example:

```
data = {
        inputs: [
            ["therm.sod"],
            [symbol],
            [Tstart, Tstep, Tend]
        ],
        files: ['/home/therm.sod'],
        token: token
    };
```

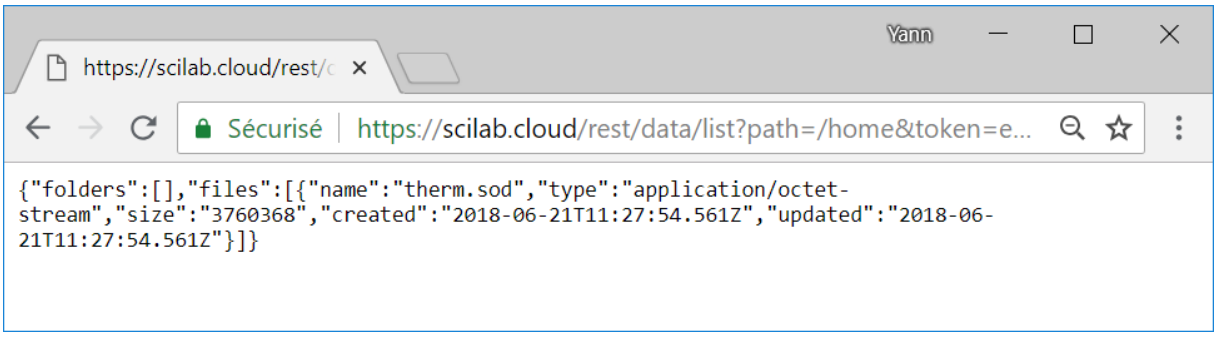
Data management on Scilab Cloud

You can attach files to your HTTP request, that are located on your Scilab Cloud storage, to perform computation on those files.
This is a view of your data management interface in your User profile:



You can access the data located in your home with a HTTP call using the method GET on the following URL:

https://scilab.cloud/rest/data/list?path=/home&token=<token>



Thanks to the function *http_get* (coming soon in Scilab) you can navigate in the data management of Scilab Cloud directly from your Scilab environment:

```
--> url = "https://scilab.cloud";

--> res_group_ls = http_get(url +
"/rest/data/list?path=/home&token=" + token)
 res_group_ls  =

  folders: [0x0 constant]
  files: [1x1 struct]
```

```
--> res_group_ls.files
 ans  =

  name: [1x1 string]
  type: [1x1 string]
  size: [1x1 string]
  created: [1x1 string]
  updated: [1x1 string]


--> file_name = res_group_ls.files.name
 file_name  =

 therm.sod
```

In order to download locally the files from the data management of Scilab Cloud, you can perform a HTTP call using the method GET on the following URL:

https://scilab.cloud/rest/data/files?path=/home/<file_name>&token=<token>

This is an example in Scilab:

```
--> http_get(url + "/rest/data/files?path=/home/" + file_name
+"&token=" + token, TMPDIR + "/" + file_name);
```

In order to upload your local files to the data management of Scilab Cloud like this:

```
--> http_upload(url + "/rest/data/files", "file_path", "file_name",
struct("path", "/home", "token", token))
```

URL to call the web service

There are two scenarios of use of your web service, depending on the stage of its life:

- Development version, to test your web service before setting it in production
- Production version, for your final end user, may they be humans or machines

For the development version, the URL to provide is structure as followed:

https://scilab.cloud/rest/<entity_name>/<toolbox_name>/<URI_version>/<function>

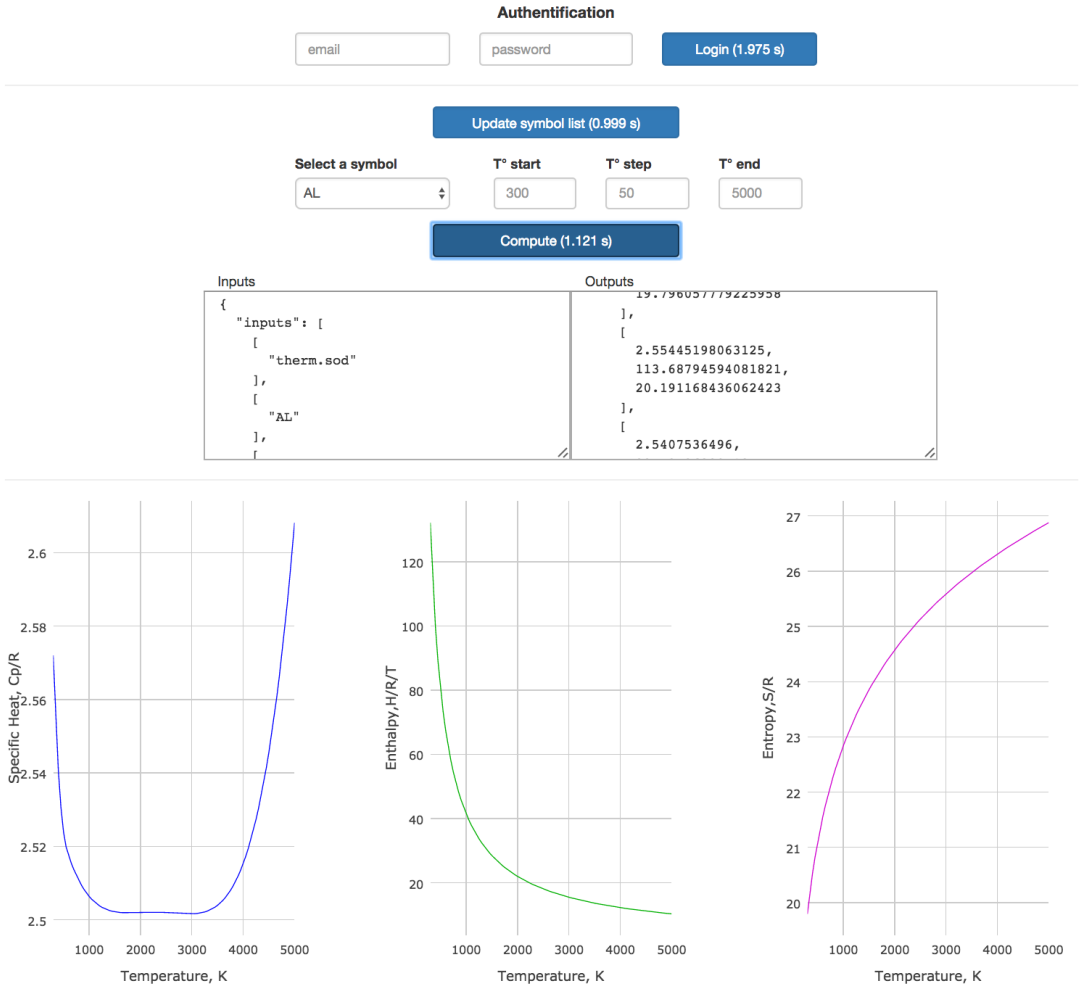
In our example, it gives the following URL:

https://scilab.cloud/rest/scilab/therm/2/symbol

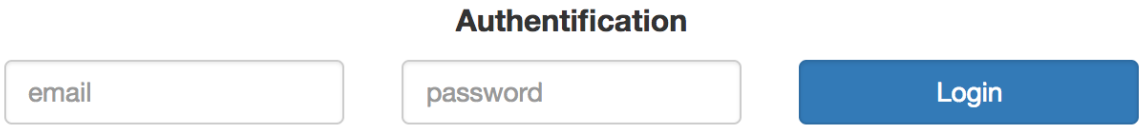
In the production version, the only different is that there are no <URI version>

Integration in a Web Application

In this example, we will detail how to integrate Scilab Cloud API with a web application written in html and javascript.



Authentication



- Implementation in HTML:

```
<div class="col-md-offset-3 col-md-6 title">Authentification</div>
        <div class="col-md-offset-3 col-md-2">
```

```
            <input class="form-control" type="text" placeholder="email"
id="inputUser"></input>
        </div>
        <div class="col-md-2">
            <input class="form-control" type="password"
placeholder="password" id="inputPassword"></input>
        </div>
        <div class="col-md-2">
            <button class="form-control col-md-12 btn btn-primary"
id="btnLogin">Login</button>
        </div>
```

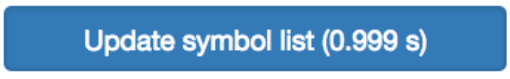
- Implementation in Javascript:

```
var token;

$('#btnLogin').click(function(e) {
    $('#btnLogin').text('Login');
    var addr = $("input:checked").val();

    $('#inputs').text('');
    $('#outputs').text('');

    var data = {
        email: $('#inputUser').val(),
        password: $('#inputPassword').val()
    };

    var start = new Date().getTime();
    $.ajax({
        url: addr + "/rest/auth",
        contentType: 'application/json',
        method: "POST",
        data: JSON.stringify(data)
    }).done(function(v) {
        var end = new Date().getTime();
        var time = end - start;
        $('#btnLogin').text('Login (' + (time/1000) + ' s)');
        token = v.token;

        $('#outputs').text(token);
    }).fail(function(v) {
        token = '';
        $('#outputs').text('Login failed');
    }).always(function() {
    });
});
```

Function call

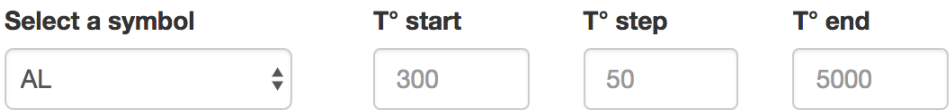
Then you can add a simple button updating a list of items contained in the file *therm.sod* located in the data repository of the user:

```
<button type="button" class="form-control btn btn-primary"
id="btnUpdate">Update symbol list</button>
```



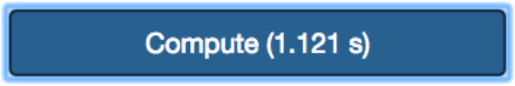
```
$('#btnUpdate').click(function(e) {
    $('#btnUpdate').text('Update symbol list');
    var addr = $("input:checked").val();

    var data = {
        inputs: [
            ["therm.sod"]
        ],
        files: ['/home/therm.sod'],
        token: token
    };

    $('#outputs').text('');
    $('#inputs').text(JSON.stringify(data, null, 2));
    var start = new Date().getTime();
    $.ajax({
        url: addr + "/rest/scilab/therm/symbol",
        contentType: 'application/json',
        method: "POST",
        data: JSON.stringify(data)
    }).done(function(v) {
        var end = new Date().getTime();
        var time = end - start;
        $('#outputs').text(JSON.stringify(v.outputs, null, 2));
        $('#btnUpdate').text('Update symbol list (' + (time/1000) + ' s)');

        $('#selectsymbol option').remove();

        var opt = v.outputs[0];
        for(var i = 0 ; i < opt.length ; ++i) {
            $('#selectsymbol').append('<option value="' + opt[i] + '">' +
opt[i] + '</option>');
        }
    }).fail(function(v) {
        $('#outputs').text(v.responseText);
    }).always(function() {
    });
});
```

The result of this function is to feed the following fields with the data found in the file *therm.sod*

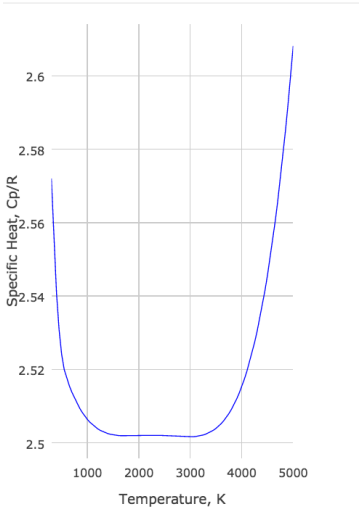


Plots

The final callback triggered by the button Compute will display the following plots:



```
<div class="row col-md-12"><hr></div>

<div class="row">
    <div class="col-md-4 plot" id="heat">
    </div>
    <div class="col-md-4 plot" id="enthalpy">
    </div>
    <div class="col-md-4 plot" id="entropy">
    </div>
    </div>
```



In this javascript code sample, we display only one of the three plots:

```
$('#btnCompute').click(function(e) {
    $('#btnCompute').text('Compute');
    var addr = $("input:checked").val();
    var symbol = $('#selectsymbol option:checked').val()
    var Tstart = $("#tempstart").val() === "" ? '300' :
$("#tempstart").val();
    var Tstep = $("#tempstep").val() === "" ? '50' : $("#tempstep").val();
    var Tend = $("#tempend").val() === "" ? '5000' : $("#tempend").val();

    Tstart = parseInt(Tstart);
    Tstep = parseInt(Tstep);
    Tend = parseInt(Tend);

    //compute range
    var T = [];
    for(var i = Tstart ; i <= Tend ; i += Tstep) {
        T.push(i);
    }

    var data = {
        inputs: [
            ["therm.sod"],
            [symbol],
            [Tstart, Tstep, Tend]
        ],
        files: ['/home/therm.sod'],
        token: token
    };

    $('#outputs').text('');
    $('#inputs').text(JSON.stringify(data, null, 2));
    var start = new Date().getTime();
    $.ajax({
        url: addr + "/rest/scilab/therm/therm",
        contentType: 'application/json',
        method: "POST",
        data: JSON.stringify(data)
    }).done(function(v) {
        var end = new Date().getTime();
        var time = end - start;
        $('#outputs').text(JSON.stringify(v.outputs, null, 2));
```

```
        $('#btnCompute').text('Compute (' + (time/1000) + ' s)');

        //clear plot div
        $('.plot').children().remove();

        //plot 1st
        var o = v.outputs[0];

        var heatData = o.map(function(value, index) { return value[0]});
        var enthalpyData = o.map(function(value, index) { return
value[1]});
        var entropyData = o.map(function(value, index) { return value[2]});

        var layout1 = {
            showlegend: false,
            autosze: false,
            height: 500,
            width: 350,
            margin: {
                l: 50,
                r: 50,
                b: 50,
                t: 0,
                pad: 5
            },
            xaxis: {
                gridcolor: 'rgb(204, 204, 204)',
                showgrid: true,
                autotick: true,
                title: 'Temperature, K'
            },
            yaxis: {
                gridcolor: 'rgb(204, 204, 204)',
                showgrid: true,
                autotick: true,
                title: 'Specific Heat, Cp/R'
            }
        };

        var trace1 = {
            x: T,
            y: heatData,
            type: 'scatter',
            line: {
                color: 'rgb(0, 0, 255)',
                width: 1
            }
        };

        Plotly.newPlot($('#entropy')[0], [trace3], layout3,
{displayModeBar:false, showLink:false, scrollZoom:false});
    }).fail(function(v) {
        $('#outputs').text(v.responseText);
    }).always(function() {
    });
});
```

Debugging

The last part of every function described so far is to display in a textarea for debugging:

Inputs

```
{
  "inputs": [
    [
      "therm.sod"
    ],
    [
      "AL"
    ],
    [
```

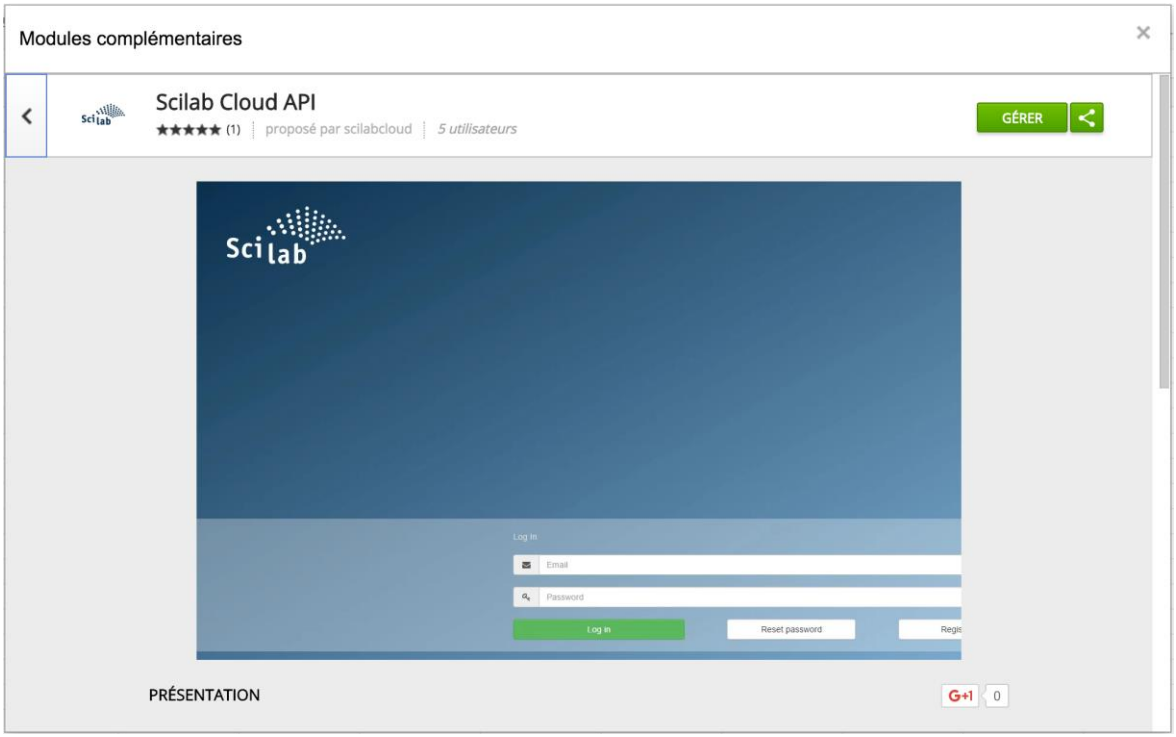
Outputs

```
      19.796057779225958
    ],
    [
      2.55445198063125,
      113.68794594081821,
      20.191168436062423
    ],
    [
      2.5407536496,
```

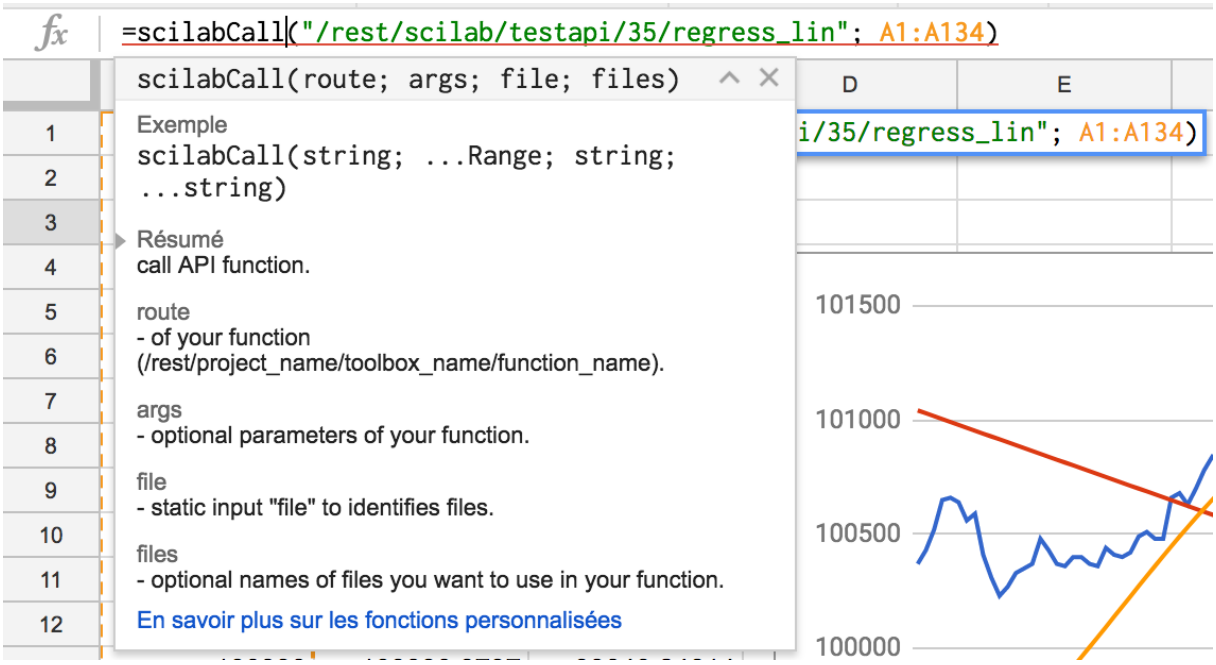
```
<div class="container">
            <div class="col-md-offset-2 col-md-4">Inputs</div>
            <div class="col-md-4">Outputs</div>
            <textarea readonly class="col-md-offset-2 col-md-4"id="inputs"
style="font-family: monospace"></textarea>
            <textarea readonly class="col-md-4" id="outputs" style="font-
family: monospace"></textarea>
</div>
```

Integration in Google Spreadsheet

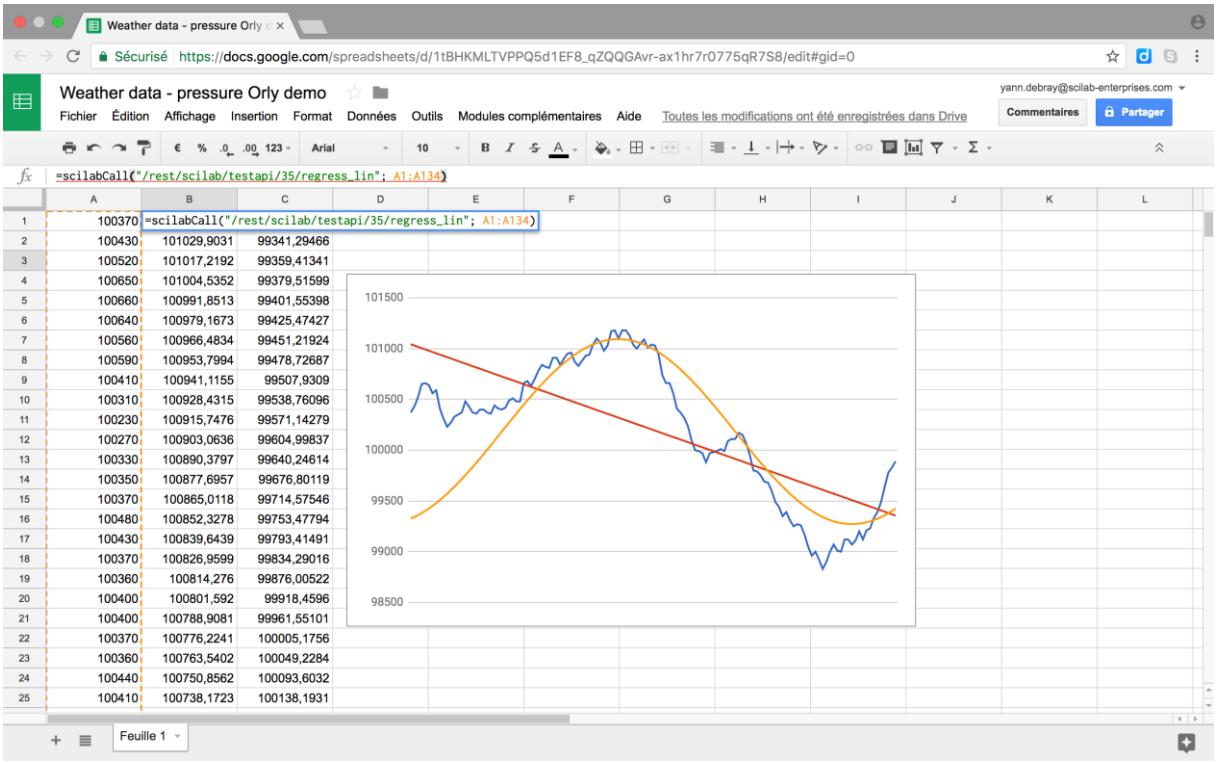
In order to integrate calls to Scilab Cloud in Google Spreadsheet, you have to install this complementary module from the Google Marketplace:



This module provides the following function calling the Scilab Cloud API on the data present in the Spreadsheet (the same way as macros in Excel):



In this first example, we simply call a linear regression on the data from the first column (represented with the red line). It automatically computes the results as a vector and returns it over the whole column.



In a second example, we use the same algorithm as in the previous sections, computing thermodynamic properties of chemical entities (specific heat, enthalpy and entropy).

The function call in this case is mentioning a file argument *therm.sod* located in the home directory of the user */home/therm.sod*:

=scilabCall("/rest/scilab/therm/symbol"; "therm.sod"; "files"; "/home/therm.sod")

	A	B	C	D	E	F
1		AL	Températures	Specific Heat	Enthalpy	Entropy
2		AL2H6	300	2,572	132,209	19,796
3		AL2ME6	350	2,554	113,688	20,191
4		ALAS	400	2,541	99,795	20,531
5		ALH	450	2,531	88,989	20,830
6		ALH2	500	2,524	80,343	21,096
7		ALH3	550	2,520	73,268	21,337
8		ALME	600	2,517	67,372	21,556
9		ALME2	650	2,515	62,383	21,757
10		ALME3	700	2,514	58,107	21,944
11		AR	750	2,512	54,401	22,117
12		AR+	800	2,511	51,158	22,279

After returning the list of entities, with the call of the function *symbol* , we compute the specific heat, enthalpy and entropy of a selected entity over a range of temperature:

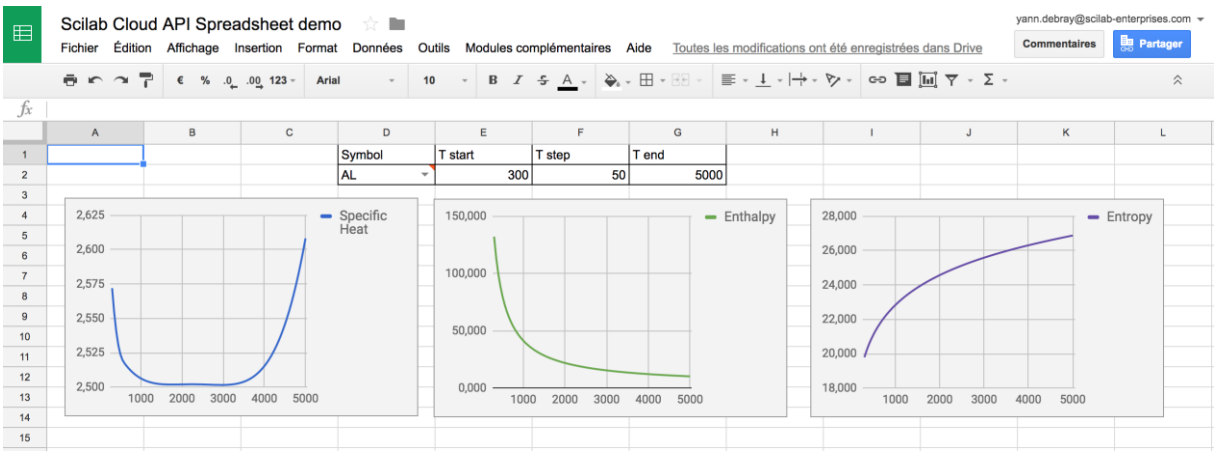
Scilab Cloud API Spreadsheet demo

Fichier Édition Affichage Insertion Format Données Outils Modules complémentaires Aide Toutes les mod

=scilabCall("/rest/scilab/therm/therm"; "therm.sod"; Therm!D2; Therm!E2:G2;"files"; "/home/therm.sod")

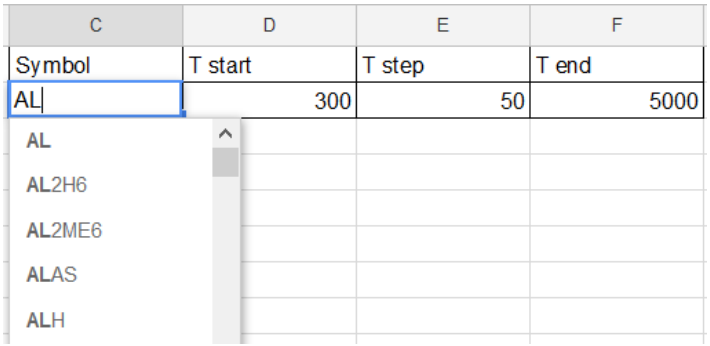
	A	B	C	D	E	F
1		AL	Températures	Specific Heat	Enthalpy	Entropy
2		AL2H6	300	2,572	132,209	19,796
3		AL2ME6	350	2,554	113,688	20,191
4		ALAS	400	2,541	99,795	20,531
5		ALH	450	2,531	88,989	20,830
6		ALH2	500	2,524	80,343	21,096
7		ALH3	550	2,520	73,268	21,337
8		ALME	600	2,517	67,372	21,556
9		ALME2	650	2,515	62,383	21,757
10		ALME3	700	2,514	58,107	21,944
11		AR	750	2,512	54,401	22,117
12		AR+	800	2,511	51,158	22,279
13		AS	850	2,509	48,296	22,431
14		AS2	900	2,508	45,752	22,575
15		AS3	950	2,507	43,476	22,710
16		AS4	1000	2,506	41,428	22,839
17		ASALME	1050	2,506	39,574	22,961
18		ASALME2	1100	2,505	37,889	23,078
19		ASGAET	1150	2,504	36,351	23,189
20		ASGAET2	1200	2,504	34,941	23,295

Once we have done that, we can summarize the results in a separated tab.



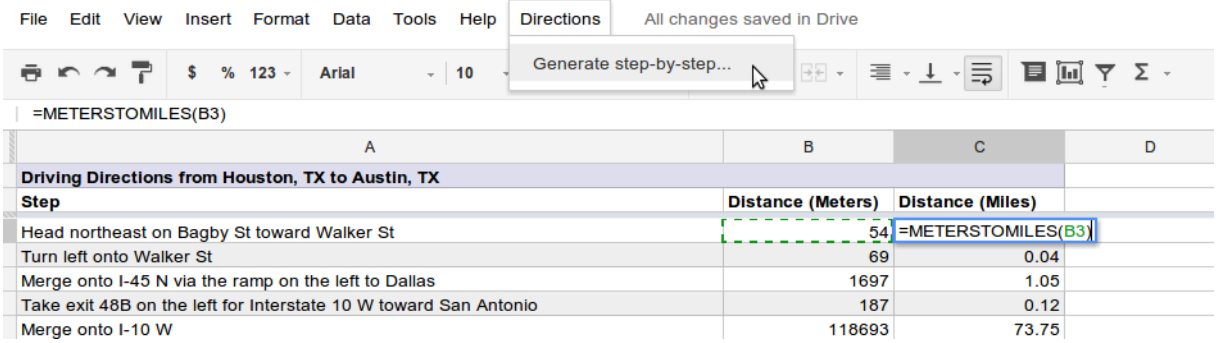
Add a picklist

https://support.google.com/docs/answer/186103



Add Macros, Menu, and Scilab Custom functions

https://developers.google.com/apps-script/quickstart/macros



Integration in a Scilab Application (New in Scilab 6.1)

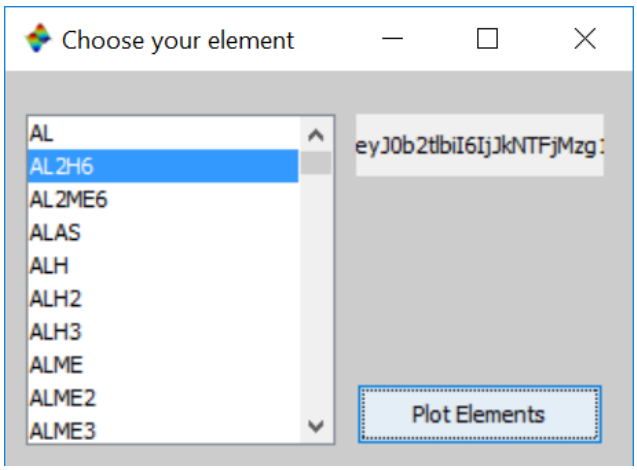
First you need to authenticate yourself to Scilab Cloud:

scilabCloudAuth.sci

```
function [token]=scilabCloudAuth(email, password)
   data.email = email;
   data.password  = password;
   token = http_post("https://scilab.cloud/rest/auth", data).token
endfunction
```

Display List:

```
url_symbol="https://scilab.cloud/rest/scilab/therm/symbol";
   data.inputs=list("therm.sod");
   data.files="/home/therm.sod";
   data.token=token;
   tic();symbol=http_post(url_symbol, data),toc

   elements=symbol.outputs

   //graphical user interface
   //listbox to select the element

   f2 = createWindow();
   f2.figure_size=[320 230];
   f2.figure_name = "Choose your element";
   h=uicontrol(f2,'style','listbox', ..
    'position', [10 10 150 160],..
    'string', elements,..
    'tag','element');

   g=uicontrol(f2,'style','pushbutton', ..
    'position', [170 10 120 30],..
    'string', "Plot Elements",..
    'callback','plotElements()');

   t=uicontrol('style','text',..
    'position', [170 140 120 30],..
    'string',token,..
    'tag','token');
```



Plot Elements:

```
h=get("element");
   elements=h.string;
   symbol=elements(h.Value);

   t=get("token");
```

```
token=t.string;

url_therm="https://scilab.cloud/rest/scilab/therm/therm";
symbol="AL"
Tstart=300;
Tstep=50;
Tend=5000;

data.inputs=list(...
   "therm.sod", ...
   symbol, ...
   [Tstart Tstep Tend]);
data.files="/home/therm.sod";
data.token=token;
tic();results=http_post(url_therm, data),toc


values=results.outputs
T=[Tstart:Tstep:Tend]

//plots

scf();
subplot(1,3,1);plot2d(T',values(:,1),style=2);xgrid() // Cp/R
xtitle('','Temperature, K','Specific Heat, Cp/R')
subplot(1,3,2);plot2d(T',values(:,2),style=14);xgrid() // H/R/T
xtitle('','Temperature, K','Enthalpy,H/R/T')
subplot(1,3,3);plot2d(T',values(:,3),style=24);xgrid() // S/R
xtitle('','Temperature, K','Entropy,S/R')
```