BANANA ACCOUNTING 9

User's guide - english only

Banana Accounting 9, User's guide - english only, 17/04/2019
© Copyright 1998-2019 Banana.ch SA - www.banana.ch

Index

Developers 7
BananaApps 7
Introduction to BananaApps 8
How-tos for BananaApps 10
Build your first BananaApp 10
Experiment with the API 14
Install your BananaApp 15
Create your invoice Layout 16
Working with Report Tables 21
Create a Cash Flow Report 28
Journal reporting 33
Working with import BananaApps 37
Working with BananaApps TestFramework 41
App Design 48
App File 48
Apps Attributes 53
Apps Parameters 57
Import Apps 57
Export Apps 58
Report Apps 60
Invoice App 60
Statement 68
Reminder 71
Dialogs 74
API 80
API Versions 81
Banana (Objects) 81
Banana.Application 83
Banana.Application.ProgressBar 84
Banana.Console 87
Banana.Converter 88
Banana.Document (Accounting) 92
Banana.Document (Base) 109
Banana.Document.Cursor 115
Banana.Document.Row 116
Banana.Document.Table 117
Banana.IO 120
Banana.IO.LocalFile 122
Banana.Report 123
Banana.Report.ReportElement 131
Banana.Report.ReportStyle 138
Banana.Report.ReportStyleSheet 141
Report FAQ 143
Banana.SDecimal 143
Banana.Script 146
Banana.Test 147
Banana.Test.Logger 151

Banana.Ui 155
Banana.Xml 159
Banana.Xml.XmlElement 160
Debugging 163
FAQ 164
Command line 165
Web Server 170
Import data 190
Exporting data 196
JCSV file format 197
Translate Banana Software 202
Open source 205
GitHub BananaAccounting 206

Excel Reports Add-in (Beta) 210
Installation 211
Documentation Excel Add-in 213
Troubleshooting 233
Installation for developers 239

ExcelSync functions 243

How to report a bug 255

Rare Cases 257
Banana9 hangs on startup on a system with two monitors 257
Banana9 hangs on startup with the message "LoadLibrary failed with error XX: Wrong parameter." 258
Banana9 hangs on startup with the message "dll is missing" or "error 0xc000007b" 258
Banana9 hangs some seconds after the main page of the program is showed 259
Banana9 hangs when the Open File Dialog or the Save File Dialog is opened 260
Banana9 hangs when trying to print 261

Developers

BananaApps

Introduction

With BananaApps you can extend the functionalities of Banana Accounting Software:

- **Create own calculations or reports.**
  - Interest calculation.
  - Statistics on transactions data.
  - Report based on special query.
- **Import data from proprietary file format.**
  - Import data from a bank file format and automatically assign accounts.
  - Import data from an invoice software.
  - Consolidate accounting data entered with other software.
- **Export data in a custom format**.
  - Create an Export file for the Tax autorities.
  - Export the balances of some accounts in a format ready to be published on internet or inserted in other documents.
- **Check the accounting data.**
  - Test transactions for a particular condition; You can show the rows that didn't meet the condition in a table, or display a message with detailed information to the user;

At the moment, BananaApps can only retrieve data from Banana Accounting and not modify values in accounting files.

Install and run BananaApps

See documentation on the Menu Apps and command Manage Apps.

Quick "How To" guides

- Build your first BananaApp
- Experiment with the API
- Install your BananaApp
- Create your invoice Layout
- Working with Report Tables
- Journal reporting
- Working with import BananaApps
- Working with BananaApps TestFramework

Examples file

- On github.com/BananaAccounting you will find the GitHub repository of BananaApps.
- On the Embedded BananaApps JavaScript Tutorial you will find two files that contains different basic examples.

Introduction to BananaApps

BananaApps are JavaScript programs that extend the Banana Accounting functionalities. Other therminologies are extension, add-ins, add-on, scripts, apps.

BananaApps file format

BananaApps can be packed in different format:

- **Embedded in the accounting file**: the BananaApp is saved in the Documents table of the accounting file.
- **Included in a JavaScript plain text file**: the BananaApp is saved within a file that is stored on the local disk. Banana reads the file and executes it.
- **Included in a packaged file**: a packaged file can contain different JavaScript files and also other files, like images.

BananaApps file structure

BananaApps files have two parts:

- **Apps Attributes**
  Apps Attributes are special formatted JavaScript comment lines, at the beginning of the file.
  The apps attributes have a left part (name of the attribute) and right part, with the value.
  Attributes give information about the script, like its purpose, description and so on. For more information, see Apps Attributes documentation.
- **JavaScript code**
  The code must be included within functions. Functions are diveded in startup functions, settings functions and normal functions.
  - **Startup functions**
    Are called by Banana software when the script is executed.
    The name of the function called depend on the type of the App.
    - exec() for following types:
      - app.command
      - export.file
      - export.rows
      - export.transactions
      - import.transactions
      - import.rows
      - import.accounts
      - import.categories
      - import.exchangerates
      - import.vatcodes
      - report.general
    - printDocument() for the following types:
      - report.customer.invoice
      - report.customer.statement
      - report.customer.reminder
  - **settingsDialog() function**
    It is called by the Banana Software when the user click on the Setting button, relative to the specific app.
    The setting data is saved within the Accounting file.

- **Other JavaScript functions**
  You can write any functions that is necessary.

BananaApps types, startup functions and how to run them

The BananaApp type is defined within the attribute **@task**.
There are many types, and some of them are started in different ways.

app.command

Is a general application. It can contain any command.

- Types: **app.command**, **export.file**, **export.rows**, **export.transactions**, **report.general**
- Startup function: **exec()**
- How to run it:
  - **File based Apps** are started from the **menu App**.
  - **Embedded Apps** are run with the button within the **Document table**

import

The purpose is to translate the content of a file to a Banana Compatible format.
It is used in Import to Accounting.

- import.transactions
  - Type: **import.transactions**
  - Startup function: **exce(fileContent)** with the content of the file as parameter. The function should return a comma separated file.
  - How to run it:
    - Select from the **menu Account1** the command **Import to accounting...**
    - As **Import** type select **Transactions**
    - Select an import app from the list
    - Click **Browse** to select the file with the data to import in Banana
- import.rows
- import.accounts
- import.categories
- import.exchangerates
- import.vatcodes

report

The purpose of this app is to create a report.
The report is run by the specific function in Banana, for example the print invoice function.
The result is displayed on the preview windows.
The function should return a Banana.Report document.

- report.customer.invoice
  - Type: **report.customer.invoice**
  - Startup function **printDocument(jsonInvoice, repDocObj, repStyleObj)**
  - How to run it:
    - Select from the **menu Account2 > Customers** the command **Print invoices...**
- report.customer.statement
  - Type: **report.customer.statement**

        - Startup function **printDocument(jsonInvoice, repDocObj, repStyleObj)**
        - How to run it:
            - Select from the **menu Account2 > Customers** the command **Print statements...**
- report.customer.reminder
    - Type: **report.customer.reminder**
    - Startup function **printDocument(jsonInvoice, repDocObj, repStyleObj)**
    - How to run it:
        - Select from the **menu Account2 > Customers** the command **Print reminders...**

Installing file based BananaApps

Before using a BananaApps you need to install it through the ManageApps menu.

If you develop a new app, you have to Install the BananaApp from a local file.

How-tos for BananaApps

Build your first BananaApp

Introduction

This walkthrough provides step-by-step guidance for creating a simple BananaApp that uses JavaScript API to interact with Banana Accounting software.

The "Hello World!" program is a classic tradition in computer programming. It is a short and complete first program for beginners, and it is perfect as first BananaApp example.

There are three basic steps in order to experiment with BananaApps:

1. Create the JavaScript file
2. Install the BananaApp
3. Run the BananaApp

Create the JavaScript file

1. Use a text editor. Download a text editor from your choice (Notepad++, Sublime Text, etc.) that will let you code in a simple way.
   It is important to be sure you can save with the UTF-8 encoding.

2. Copy the following JavaScript code and paste it on your text editor.

```
// @id = ch.banana.app.helloworldexample
// @api = 1.0
// @pubdate = 2018-10-24
// @publisher = Banana.ch SA
// @description = BananaApp example: Hello world
// @task = app.command
```

```
// @doctype = *.*
// @docproperties =
// @outputformat = none
// @inputdataform = none
// @timeout = -1
function exec() {
   //Create the report
   var report = Banana.Report.newReport("Report title");
   //Add a paragraph with the "hello world" text
   report.addParagraph("Hello World!");
   //Print the report
   var stylesheet = Banana.Report.newStyleSheet();
   Banana.Report.preview(report, stylesheet);
}
```

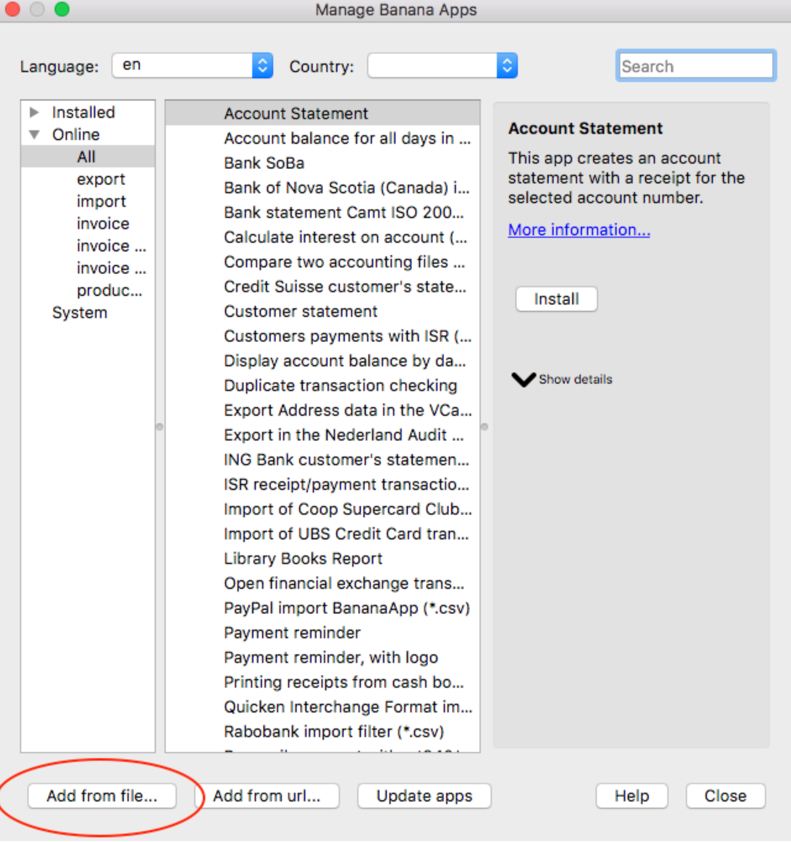
3. Change the attributes of the BananaApp (for more information, see Apps attributes):
   - **@id = <your_script_id>**
     This is the identification of the script.
     In order to avoid duplicates, it is important to assign a unique id at avery script.
   - **@description = <your_script_description>**
     This is the name of the BananaApp. The text will be displayed in the dialogs.
4. Save the file as **helloworld.js**.

You have now created your first BananaApp!

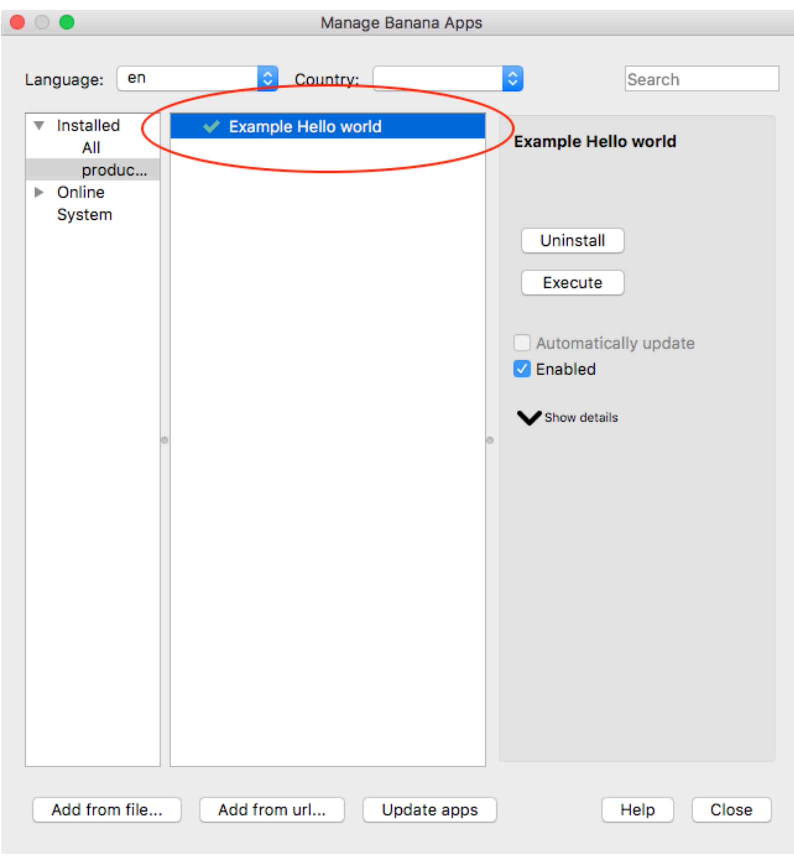
Install the BananaApp

What next? The next step is to install your BananaApp into the Banana Accounting software.
Before to use the BananaApp, and see the "Hello World!" text displayed as report in Banana, the App needs to be installed.
So, let's see how to install the "Hello World!" BananaApp.

- Open an accounting file in Banana Accounting
- In Banana select from the **menu Apps** the command **Manage Apps...**
- Click on **Add from file...**



- Select the **helloworld.js** file.
- Click on **Open** to install the App.
- The BananaApp is displayed in the dialog.
  By Selecting **Installed** from the left, all the installed BananaApps (both local and online apps) will be displayed.



- Click on **Close** to close the Manage Banana Apps dialog

You have now installed the BananaApp!

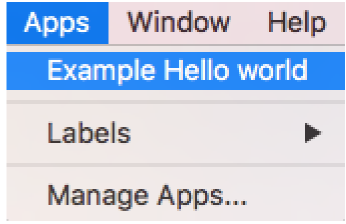
Important:

- Once installed, the JavaScript file needs to always remain in the same directory.
- If the JavaScript file is modified, the program will always use the last version.

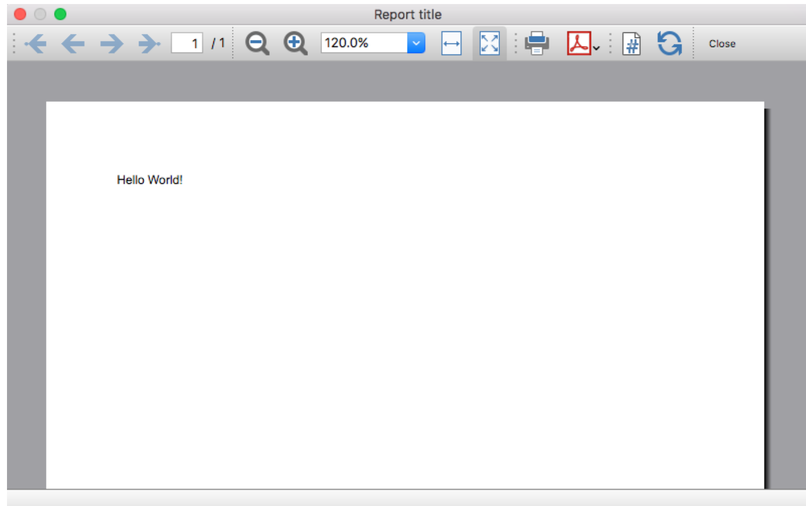
Run the BananaApp

Finally, now it is possible to run the "Hello World!" BananaApp and see the results.
To run the app:

- In Banana select from the **menu Apps** the **Example Hello World** app.



- The app is executed and returns the following reports

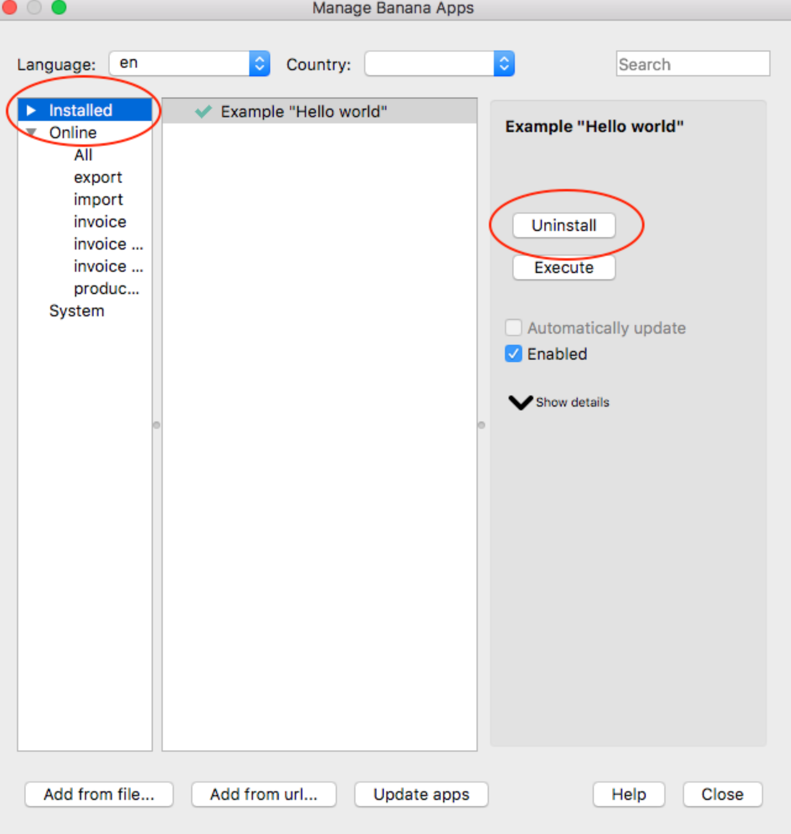


Congratulations, you have now created, installed and executed your own BananaApp!

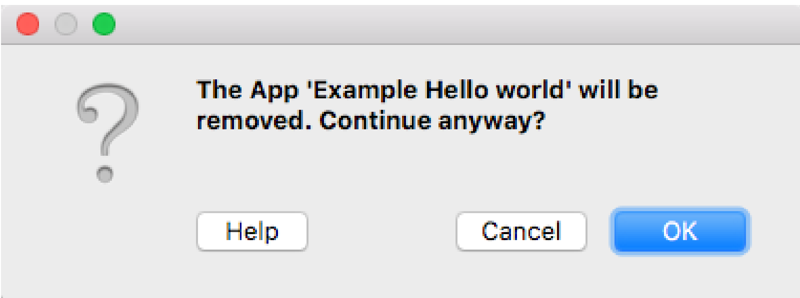
Uninstall the BananaApp

In case you don't need an installed BananaApp anymore, it is also possible to remove it from Banana Accounting software using the unistall command.

- In Banana select from the **menu Apps** the command **Manage Apps...**
- Select the **Installed** section on the left in order to display all the currently installed BananaApps.
- Select the BananaApp you want to remove and click **Uninstall.**



- Confirm with **Ok** to remove the App from Banana Accounting software.



The BananaApp is now removed from Banana Accounting software, but the JavaScript file (i.e. helloworld.js) is not removed from you computer.

More about BananaApps

- JavaScript API
- BananaApps documentation
- Working with Report Tables
- Experiment with embedded BananaApps

Experiment with the API

Banana does allow to have BananaApps that are embedded within a Banana File.

We have prepared tutorial files that include samples code for most API.

- You can see how the API works and experiment with it.
- Just downaload and open a tutorial file in Banana.

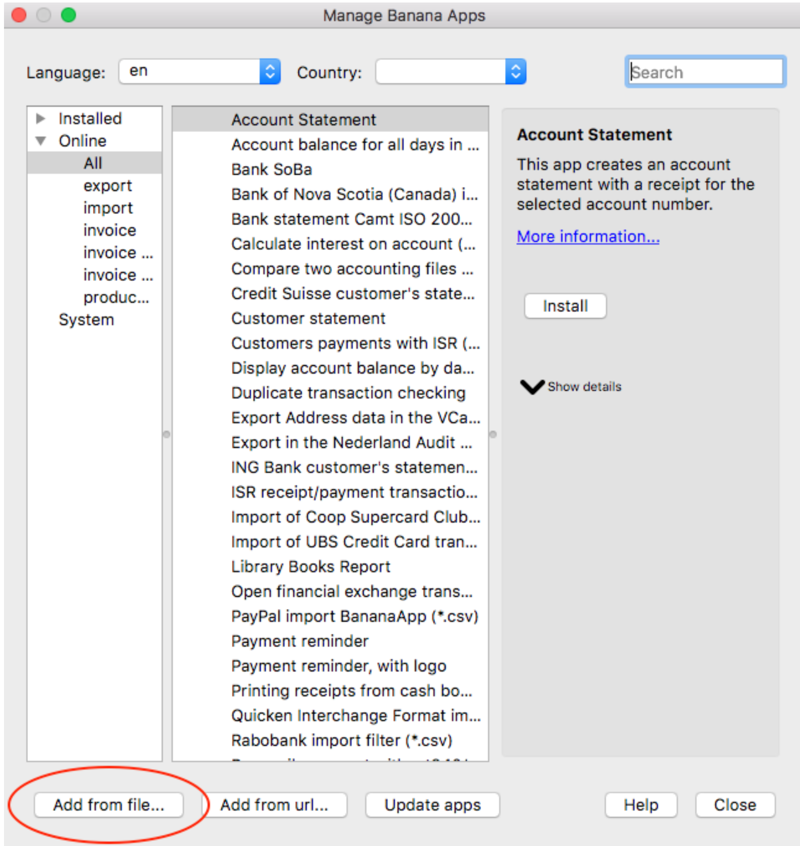
See expalanation [Embedded BananaApps JavaScript Tutorial](#) on Github.

Install your BananaApp

In Banana anyone can create BananaApps that extend the functionality of the program.

Install a BananaApp from a local file

- In Banana select from the **menu Apps** the command **Manage Apps...**
- Click on **Add from file...**



Manage Banana Apps dialog

- Choose your JavaScript (**.js**) file
- Click on **Open** to install the App
- Click on **Close** to close the Manage Banana Apps dialog

At this point, the BananaApp is installed and ready to be used.

Important information:

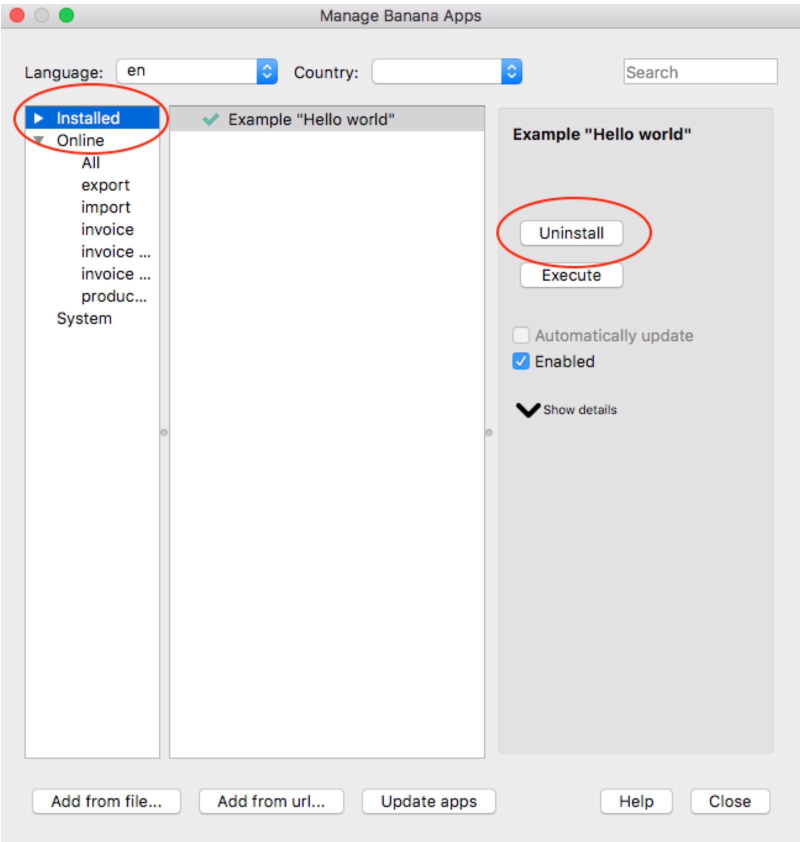
- Once installed, the file .js needs to always remain in the same directory.
- If the App is modified, the program will always use the last version.

Run the BananaApp

In Banana select from the **menu Apps** the App you have installed.

Uninstall the BananaApp

- In Banana select from the **menu Apps** the command **Manage Apps...**
- Select an App from the **Installed** section and click on **Uninstall**



Manage Banana Apps dialog

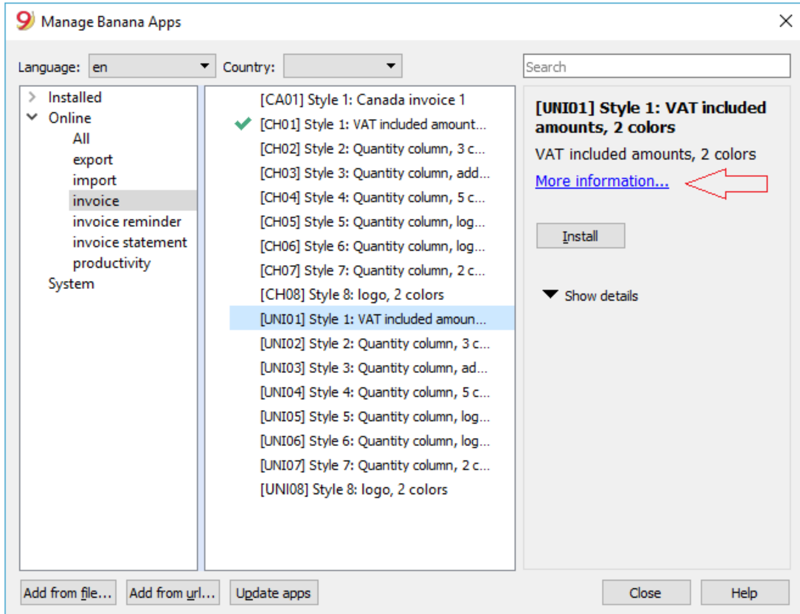
- Click on **Close** to close the Manage Banana Apps dialog

Create your invoice Layout

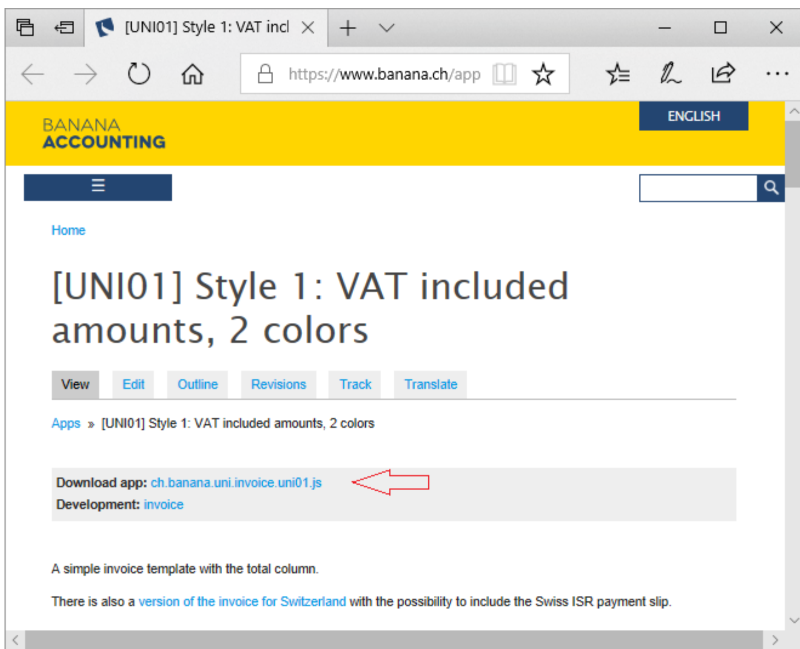
In Banana you can create your own report from scratch or from an existing model. The following steps describe how to create an invoice template starting from an existing one and adapting it to your needs.

1. Choose your print style to start from.

- In Banana select from the **menu Apps** the command **Manage Apps...**
- Select Online, Invoice.
- Choose one of the existing template you want (i.e. [UNI01])
- Click on **More information...**



2. Click on the link **ch.banana.uni.invoice.uni01.js** and save the file to your documents folder (menu **File** -> **Save Page As**...).



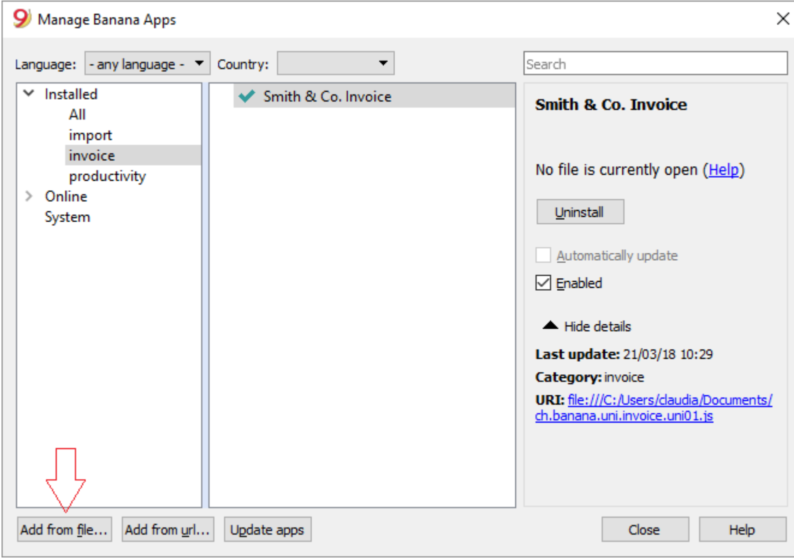
3. Modify the template

- Open the local file with a text editor program
  - Right click on the file
  - Open with .... and select the text editor program
- Changing the **@description** and the **@id**.
- Save the file.

```
ch.banana.uni.invoice.uni01.js
13  // limitations under the License.
14  //
15  // @id = ch.banana.uni.invoice.uni01
16  // @api = 1.0
17  // @pubdate = 2018-01-31
18  // @publisher = Banana.ch SA
19  // @description = Smith & Co. Invoice
20  // @doctype = *
21  // @task = report.customer.invoice
22
23  var rowNumber = 0;
24  var pageNr = 1;
25  var repTableObj = "";
26  var max_items_per_page = "";
27  var max_items_per_page_with_isr = 12;
28  var isFirstPage = true;
29
30  /*Update script's parameters*/
31  function settingsDialog() {
32     var param = initParam();
33     var savedParam = Banana.document.getScriptSettings();
34     if (savedParam.length > 0) {
35        param = JSON.parse(savedParam);
36     }
37     param = verifyParam(param);
38     var lang = Banana.document.locale;
length : 32057   lines : { Ln : 20   Col : 1   Sel : 38 | 2   UNIX   ANSI as UTF-8   INS
```

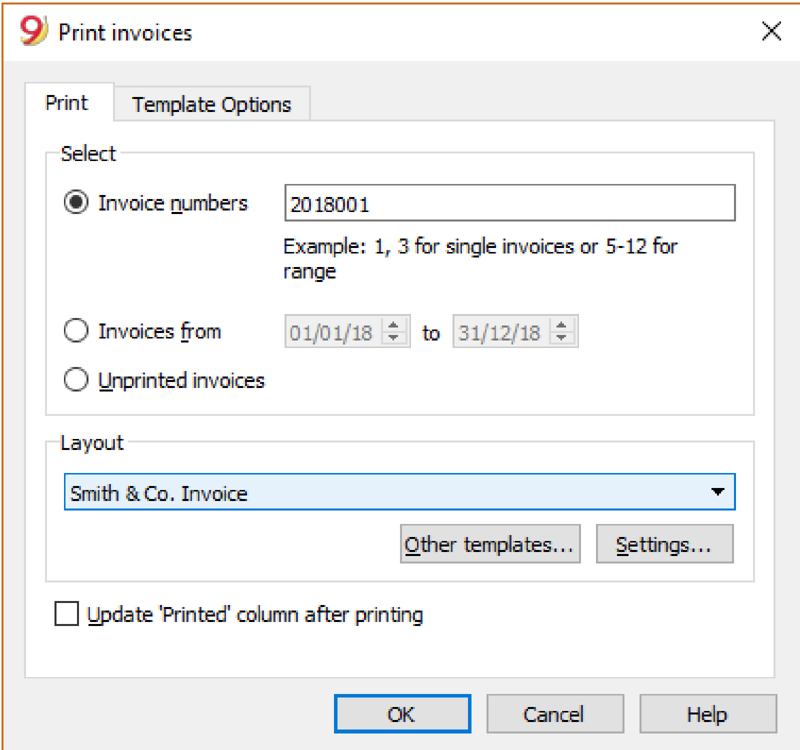
4. Add your template to the Banana Apps.

- In **menu Apps** from Banana select **Manage Apps....**
- Click on the button **Add from file...** and choose the file you just downloaded and modified.

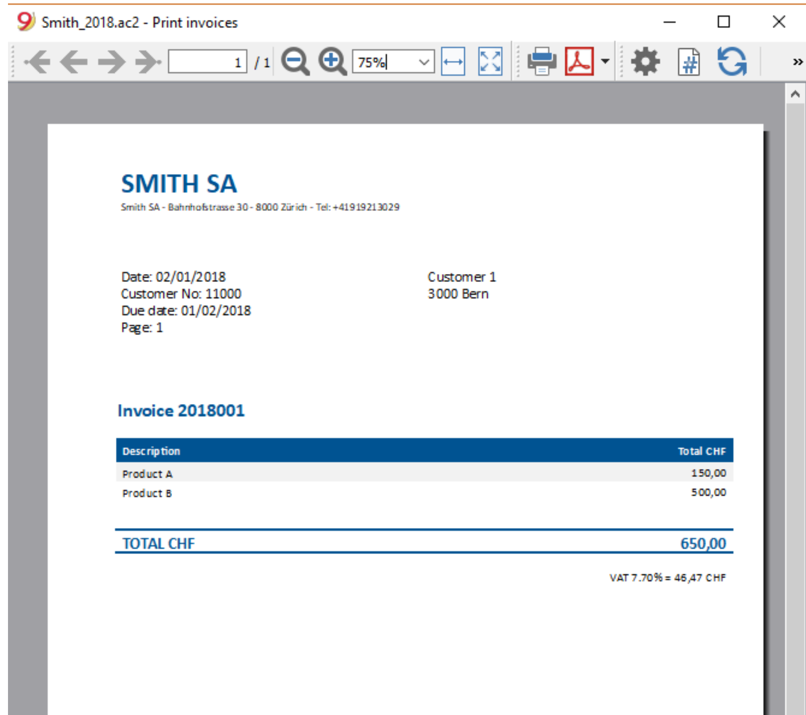


5 View an invoice with the custom template.

- In menu **Account2** from Banana select **Customers**, **Print Invoices...** and select the layout **Smith & Co. Invoice**
- Click Ok



Here the previews



5. Makes more changes

- For example change the header position, we put it on the right.
- Modify the lines as described here.
- Save the changes

C:\Users\claudia\Documents\ch.banana.uni.invoice.uni01.js - Notepad++

Datei Bearbeiten Suchen Ansicht Kodierung Sprachen Einstellungen Makro Ausführen Erweiterungen Fenster ?

ch.banana.uni.invoice.uni01.js

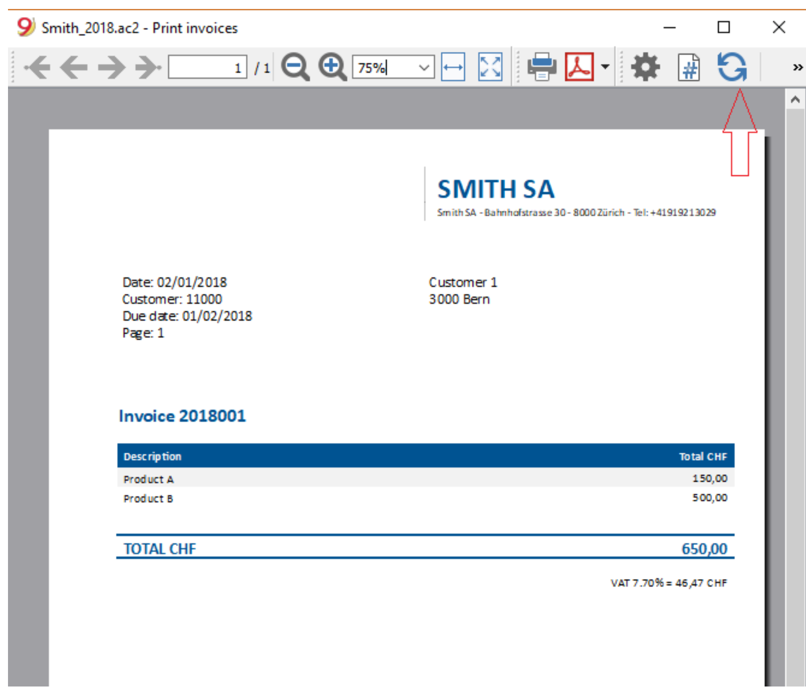
```
663    repStyleObj.addStyle(".repTableCol3", "width:%");
664    //repStyleObj.addStyle(".repTableCol4","width:%");
665
666
667    /*
668        Text begin
669    */
670    var beginStyle = repStyleObj.addStyle(".begin_text");
671    beginStyle.setAttribute("position", "absolute");
672    beginStyle.setAttribute("top", "90mm");
673    beginStyle.setAttribute("left", "20mm");
674    beginStyle.setAttribute("right", "10mm");
675    beginStyle.setAttribute("font-size", "10px");
676
677    //=====================================================================//
678    // TABLES
679    //=====================================================================//
680    var headerStyle = repStyleObj.addStyle(".header_table");
681    headerStyle.setAttribute("position", "absolute");
682    headerStyle.setAttribute("margin-top", "10mm"); //106
683    headerStyle.setAttribute("margin-left", "110mm"); //20
684    headerStyle.setAttribute("margin-right", "4mm");
685    //headerStyle.setAttribute("width", "100%");
686    repStyleObj.addStyle("table.header_table td", "border-right: thin solid
       black;padding-left:10px");
687
688    var infoStyle = repStyleObj.addStyle(".title_table");
689    infoStyle.setAttribute("position", "absolute");
690    infoStyle.setAttribute("margin-top", "90mm");
691    infoStyle.setAttribute("margin-left", "22mm");
692    infoStyle.setAttribute("margin-right", "10mm");
693    //repStyleObj.addStyle("table.info_table td", "border: thin solid black");
694    infoStyle.setAttribute("width", "100%");
```

length : 32087 lines : 8 Ln : 687 Col : 1 Sel : 260 | 5 UNIX ANSI as UTF-8 INS

- Change the text "Customer No" to "Customer"

```
855     texts.qty = 'Hoeveelheid';
856     texts.unit_ref = 'Eenheid';
857     texts.unit_price = 'Eenheidsprijs';
858     texts.vat_number = 'BTW-nummer: ';
859     texts.bill_to = 'Factuuradres';
860     texts.shipping_to = 'Leveringsadres';
861     texts.from = 'VAN';
862     texts.to = 'TOT';
863     texts.param_color_1 = 'Achtergrond kleur';
864     texts.param_color_2 = 'Tekstkleur';
865     texts.param_font_family = 'Lettertype';
866     texts.param_print_header = 'Pagina-koptekst opnemen (1=ja, 0=nee)';
867     texts.payment_due_date_label = 'Vervaldatum';
868     texts.payment_terms_label = 'Betaling';
869     //texts.param_max_items_per_page = 'Aantal artikelen op iedere pagina';
870   } else {
871     texts.customer = 'Customer';
872     texts.date = 'Date';
873     texts.description = 'Description';
874     texts.invoice = 'Invoice';
875     texts.page = 'Page';
876     texts.rounding = 'Rounding';
877     texts.total = 'Total';
878     texts.totalnet = 'Total net';
879     texts.vat = 'VAT';
880     texts.qty = 'Quantity';
881     texts.unit_ref = 'Unit';
882     texts.unit_price = 'Unit price';
883     texts.vat_number = 'VAT Number: ';
884     texts.bill_to = 'Billing address';
885     texts.shipping_to = 'Shipping address';
886     texts.from = 'FROM';
887     texts.to = 'TO';
```

6. See your changes
Simply click on the refresh button



Working with Report Tables

Introduction

All the following code samples are used in the embedded_javascript_tutorial1.ac2 file as embedded BananaApps. This file contains a list of complete and working examples that can be run.

Another complete and working example of BananaApp that use a table can be found here.

Table object

Tables are used in BananaApps to present tabular data as reports.

A Table object is defined using the addTable([classes]) function, and contains a number of **table cells** which are organized into **table rows**.

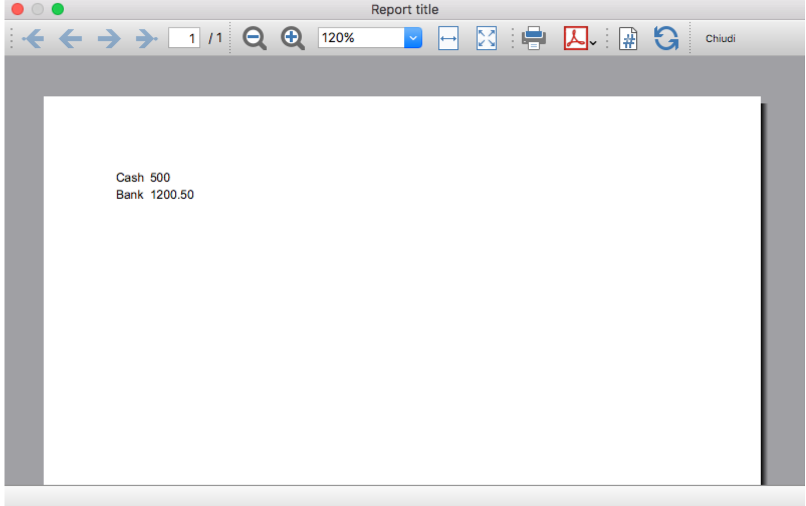
The process for creating a table is the following:

1. Create the report object that willl contain the table
2. Add a table object to the report
3. Add a row object to the table using the addRow([classes]) method.
4. Add cells objects to the row using the addCell([span]) or addCell(text [,classes, span]) methods.
5. Repeat steps 3 and 4

Table Code Sample: Simple Table

```
var report = Banana.Report.newReport("Report title"); // create the report
var myTable = report.addTable("myTable");             // create and add a
table to the report

var tableRow = myTable.addRow();                      // add a row to the
table
tableRow.addCell("Cash");                             // add a first cell to
the row
tableRow.addCell("500.00");                           // add a second cell to
the row

var tableRow = myTable.addRow();                      // add a row to the
table
tableRow.addCell("Bank");                             // add a first cell to
the row
tableRow.addCell("1200.50");                          // add a second cell to
the row
```

Output:



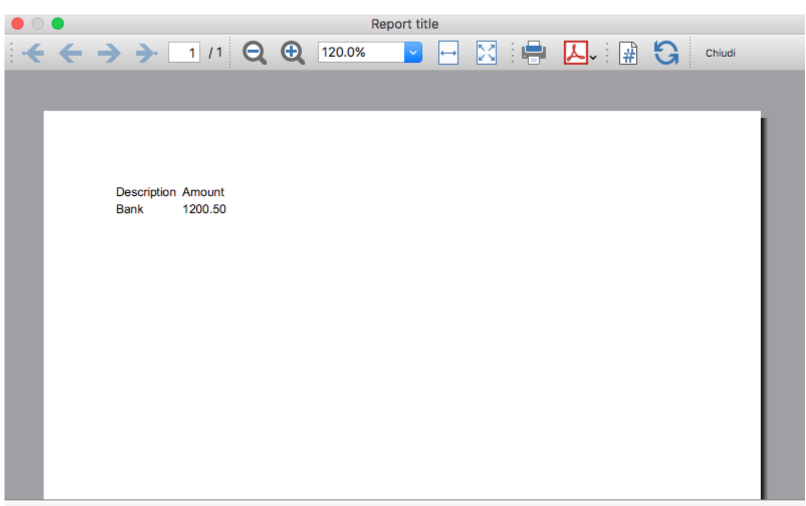
Column headers

Table cells may should act as **column headers,** in such cases it should be used the getHeader() function.

If the table goes over several pages, this would allow you to repeat at the beginning of each page the headers of the columns.

```
var report = Banana.Report.newReport("Report title");
var table = report.addTable("myTable");

// add the table header
var tableHeader = table.getHeader();
var tableRow = tableHeader.addRow();
tableRow.addCell("Description");
tableRow.addCell("Amount");

// add the first row of the table
tableRow = table.addRow();
tableRow.addCell('Cash');
tableRow.addCell('1200');
```

Output:

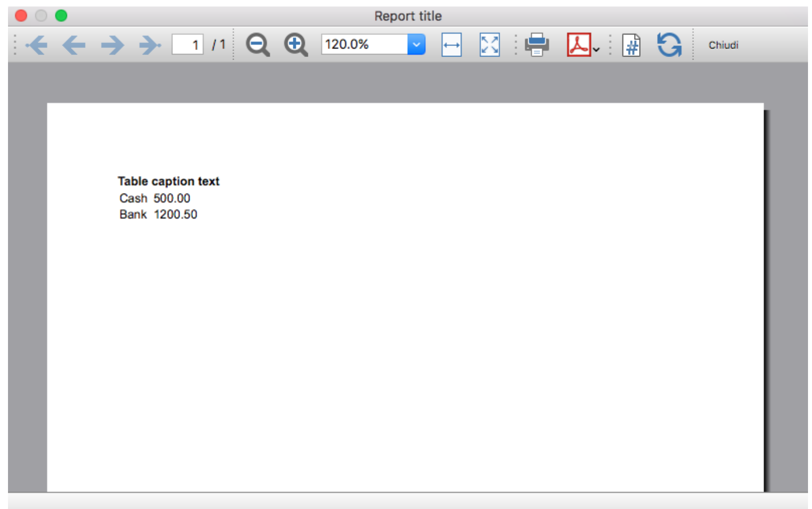


Caption

A caption, which is a descriptive text associated with the element, can be added to a table using the getCaption() function.

```
var table = report.addTable("MyTable");
var caption = table.getCaption();
caption.addText("Table caption text", "captionStyle");
```

Output:



Merge cells

Table **cells can be merged** using the **span** attribute of the addCell([span]) or addCell(text [,classes, span]) functions.

```
tableRow.addCell();                          // span empty cell over 1 column
(default value)
tableRow.addCell("", 3);                     // span empty cell over 3 columns
tableRow.addCell("Cash", 2);                 // span the cell over 2 columns
tableRow.addCell("Cash", "classStyle", 2); // span the cell over 2 columns
```

Columns and Cells attributes

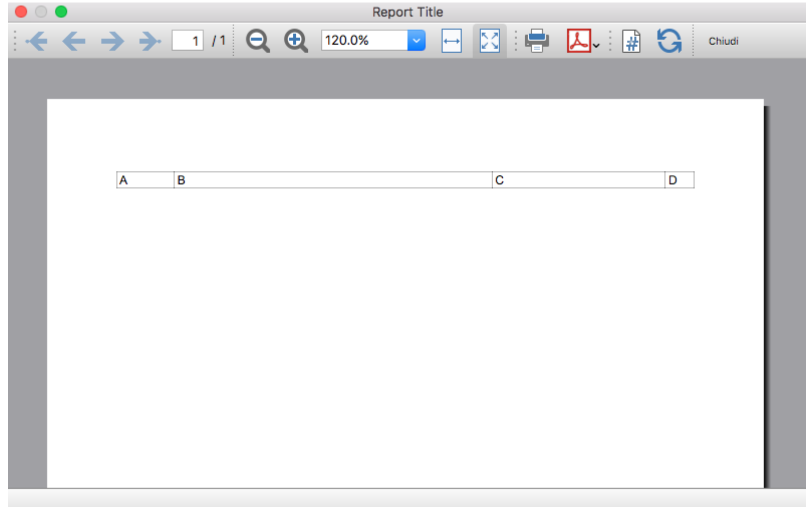
Styles attributes can be defined to set, for example, the **columns width** and **cells borders** using the addColumn([classes]) and setStyleAttributes(attributes) functions.

- The **width attribute** (applied to a table) specifies the width of a table. If the width attribute is not set, a table takes up the space of the report page.
- The **width attribute** (applied to a column) specifies the width of a column. If the width attribute is not set, a column takes up the space it needs to display the data.
- The **border attribute** (applied to a cell) specifies the border of a cell. If the border attribute is not set, a cell will be displayed without borders.

```
var report = Banana.Report.newReport("Report Title");
var table = report.addTable("MyTable");
table.setStyleAttributes("width:100%;"); // specifies the width of the table
```

```
var column1 = table.addColumn("col1");
column1.setStyleAttributes("width:10%"); // specifies the width of the
column1
var column2 = table.addColumn("col2");
column2.setStyleAttributes("width:55%"); // specifies the width of the
column2
var column3 = table.addColumn("col3");
column3.setStyleAttributes("width:30%"); // specifies the width of the
column3
var column4 = table.addColumn("col4");
column4.setStyleAttributes("width:5%");  // specifies the width of the
column4

var tableRow = table.addRow();
tableRow.addCell("A", "", 1).setStyleAttributes("border:thin solid black");
// specifies the cell border
tableRow.addCell("B", "", 1).setStyleAttributes("border:thin solid black");
// specifies the cell border
tableRow.addCell("C", "", 1).setStyleAttributes("border:thin solid black");
// specifies the cell border
tableRow.addCell("D", "", 1).setStyleAttributes("border:thin solid black");
// specifies the cell border
```

Output:



Cell with multiple paragraphs

Table cells can contain **multiple paragraphs** of text or data. Use the addParagraph([text, classes]) function to add many paragraphs to a cell.

```
var report = Banana.Report.newReport("Report Title");
var table = report.addTable("MyTable");
table.setStyleAttributes("width:100%;");
tableRow = table.addRow();

// Add first cell with paragraphs
var cell1 = tableRow.addCell("", "", 1);
```

```
cell1.setStyleAttributes("border:thin solid black");
cell1.addParagraph("First paragraph...", "");
cell1.addParagraph("Second paragraph...", "");
cell1.addParagraph(" "); //empty paragraph
cell1.addParagraph("Fourth paragraph...", "");

// Add second cell without paragraphs
var cell2 = tableRow.addCell("Cell2...", "",
1).setStyleAttributes("border:thin solid black");
```

Output:

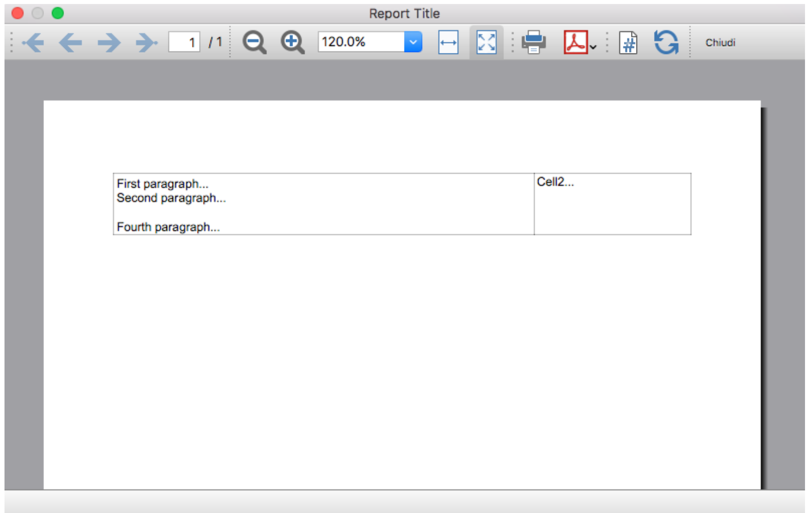


Table Code Sample: Complex Table

```
var report = Banana.Report.newReport("Report Title");
var table = report.addTable("MyTable");
table.setStyleAttributes("width:100%;");

var column1 = table.addColumn("col1");
column1.setStyleAttributes("width:25%");
var column2 = table.addColumn("col2");
column2.setStyleAttributes("width:25%");
var column3 = table.addColumn("col3");
column3.setStyleAttributes("width:25%");
var column4 = table.addColumn("col4");
column4.setStyleAttributes("width:25%");

// 1st row
tableRow = table.addRow();
tableRow.addCell("Row 1, Cell 1: span cell over 4 columns", "",
4).setStyleAttributes("border:thin solid black");

// 2nd row
tableRow = table.addRow();
tableRow.addCell("Row 2, Cell 1: span cell over 2 columns", "",
2).setStyleAttributes("border:thin solid black");
tableRow.addCell("Row 2, Cell 3: span cell over 2 columns", "",
```

```
2).setStyleAttributes("border:thin solid black");

// 3rd row
tableRow = table.addRow();
tableRow.addCell("Row 3, Cell 1", "", 1).setStyleAttributes("border:thin
solid black");
tableRow.addCell("Row 3, Cell 2: span cell over 2 columns", "",
2).setStyleAttributes("border:thin solid black");
tableRow.addCell("Row 3, Cell 4", "", 1).setStyleAttributes("border:thin
solid black");

// 4th row
tableRow = table.addRow();
tableRow.addCell("Row 4, Cell 1", "", 1).setStyleAttributes("border:thin
solid black");
tableRow.addCell("Row 4, Cell 2", "", 1).setStyleAttributes("border:thin
solid black");
tableRow.addCell("Row 4, Cell 3", "", 1).setStyleAttributes("border:thin
solid black");
tableRow.addCell("Row 4, Cell 4", "", 1).setStyleAttributes("border:thin
solid black");

// 5th row
tableRow = table.addRow();
tableRow.addCell("Row 5, Cell 1: span cell over 3 columns", "",
3).setStyleAttributes("border:thin solid black");
tableRow.addCell("Row 5, Cell 4", "", 1).setStyleAttributes("border:thin
solid black");
```

Output:

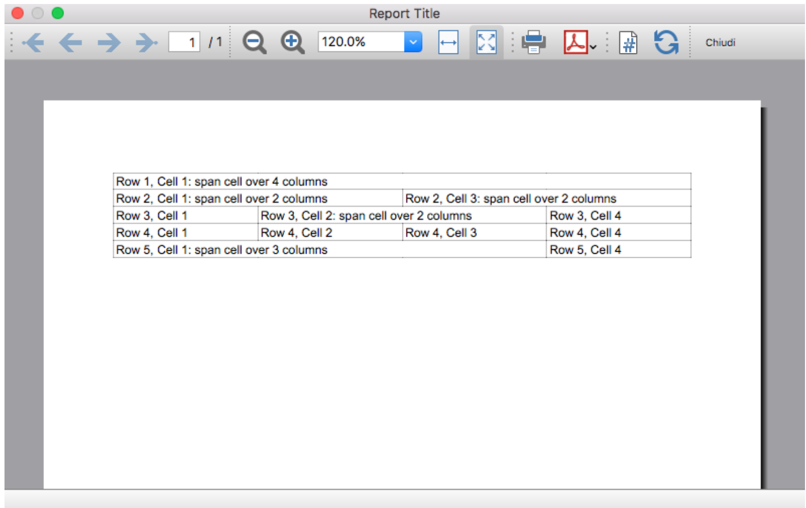


Table within a Table

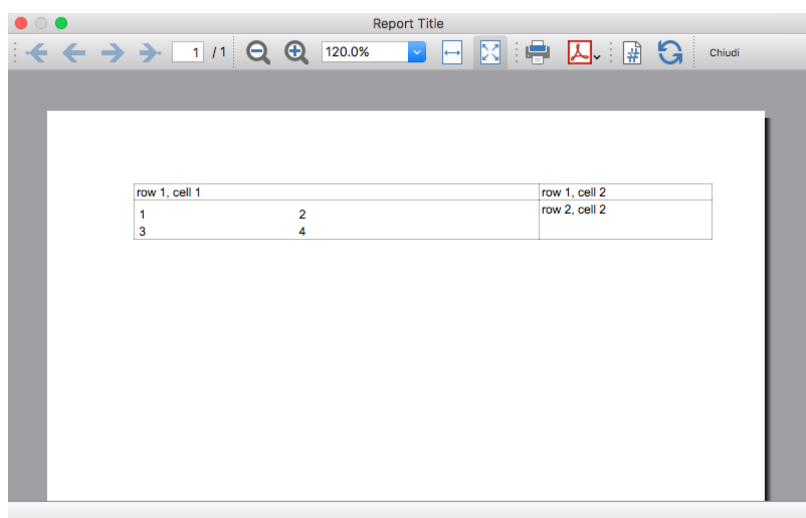
Tables can also be added within other tables, in particular to cells of "externals" tables. This might be useful when you need to create tables with more complex structures.

In these cases it is only necessary to add a tables to the cell object of the external tables instead of

the report. These tables are treated as a separate table, with their own rows, cells and attributes.

```
var report = Banana.Report.newReport("Report Title");

/*  EXTERNAL TABLE */
var table = report.addTable("outTable");
// ... add style attributes, rows, cells, etc. for the first external table

/* INTERNAL TABLE */
row_out = table.addRow(); // add a new row and a cell
cell_out = row_out.addCell("", "", 1);

var insideTable = cell_out.addTable("inTable"); // add a second table within
a cell of the first table

var row_in = insideTable.addRow();
var cell_in = row_in.addCell("1", "", 1);
cell_in = row_in.addCell("2", "", 1);

row_in = insideTable.addRow();
cell_in = row_in.addCell("3", "", 1);
cell_in = row_in.addCell("4", "", 1);

// add a second cell to the first table
cell_out = row_out.addCell("row 2, cell 2", "", 1);
```

Output example:



Create a Cash Flow Report

Introduction

This walkthrough provides step-by-step guidance for creating a cash flow report BananaApp.

As example we use the Cash Flow Report that is part of the Rapports comptables (OHADA - RDC) BananaApp, developed following the specifications for the OHADA-RDC in Africa (for more information, visit the GitHub documentation).

There are three basic steps in order to experiment with this BananaApp:

1. Create a JavaScript programming
2. Install the BananaApp
3. Run the BananaApp

Create the JavaScript programming

The script retrieves all the required data from Banana Accounting files, makes some addition and subtraction operations, and present the data in a table.

By looking a the source code for Cash Flow Report (OHADA - RDC) you will understand how the report is setup.

If you want to experiment with the script, copy and paste it on your text editor and save the file as .js (i.e. **cashflow.js**). Otherwise you can just install and run the app following the GitHub documentation.

Retrieve data from Banana

In order to build a cash flow it is required to get different data from the accounting. These data are values related to specific accounts or groups and to some columns of the table accounts.

But how to do that? How to retrieve a specific account value for a specific column of the accounts table and period?

Function currentBalance()

To retrieve the various amounts of the report, we use the currentBalance(account, startDate, endDate) function.
The function sums the amounts of opening, debit, credit, total and balance calculated based on the opening and all transactions for the given accounts/group and period.

To build the cash flow report we use this function for **accounts** and **groups**:

```
// example for account 1000
var currentBal =
Banana.document.currentBalance('1000','2019-01-01','2019-12-31');

// example for group 10
var currentBal =
Banana.document.currentBalance('Gr=10','2019-01-01','2019-12-31');
```

The parameters of the function are:

- account or group number (the group number is preceded by "**Gr=**")
- start date of the period we are intrested
- end date of the period we are intrested

The returned value of the currentBalance() function is an **object**, which has **name:values** pairs called **properties**. These properties are the values we need.

The object structure is like the following one:

```
{
   "amount":"17570.00",
   "amountCurrency":"",
   "bClass":"1",
   "balance":"17570.00",
   "balanceCurrency":"",
   "credit":"30.00",
   "creditCurrency":"",
   "debit":"16600.00",
   "debitCurrency":"",
   "opening":"1000.00",
   "openingCurrency":"1000.00",
   "rowCount":"6",
   "total":"16570.00",
   "totalCurrency":""
}
```

As you can see this object has many properties, but the cash flow report we want to build uses only four of them:

- **opening** the amount at the beginning of the period (the Opening column of the Accounts table). Can be positive or negative.
- **debit** the amount of debit transactions for the period (the Debit column of the Accounts table). Only positive values.
- **credit** the amount of credit transactions for the period (the Credit column of the Accounts table). Only positive values.
- **total** the difference between debit-credit for the period. Can be positive or negative.
- balance the balance for the period. (opening + total). Can be positive or negative.

For more information, see the documentation here.

Accessing Object Properties

Ok, now we have the object with all the properties. But how to get a single property value?

There are three methods for accessing the property of an object:

```
// method 1: objectName.property
var value = currentBal.debit; // returns 16600.00

// method 2: objectName["property"]
var value = currentBal["credit"]; // returns 30.00

// method 3: objectName[expression]
var x = "total";
var value = currentBal[x]; // returns 16570.00
```

It doesn't matter which method is used, the result does not change.

Calculate totals

The cash flow report requires to do addition and subtraction operations using some specific values retrieved from the accounting file.

To build all the various totals we encounter in the report we use the add(value1, value2) and the subtract(value1, value2) functions.

```
// example sum the amounts of accounts 6541 and 6542
var acc6541 =
Banana.document.currentBalance('6541','2019-01-01','2019-12-31').total;
var acc6542 =
Banana.document.currentBalance('6542','2019-01-01','2019-12-31').total;
var sum = Banana.SDecimal.add(acc6541, acc6542);
```

Previous year Banana document

To generate the report, the BananaApp retrieves data from the **current year accounting file** and from the **previous year accounting file**.
The current year accounting file is the one that is opened in Banana, the one that starts the execution of the BananaApp.
The previous year accounting file is not opened in Banana, it is just selected from the **menu File** -> **File and accounting properties...** -> **Options tab** -> **File from previous year**.

In order to retrieve data from the previous year we use the previousYear([nrYears]) function.
The function returns the previous year as a Banana.Document object. If the previous year is not defined or it is not found it returns null.

```
/* CURRENT year file: the opened document in Banana */
var current = Banana.document;

/* PREVIOUS year file: open a dialog window to select the previous year .ac2
file */
var previous = Banana.document.previousYear();
```

The object **Banana.document** represent the current document opened in the application.

The **previous** variable represent the defined previous year document.

Function getAmount()

We have added to the script a parameterized function that calls the currentBalance() and retrieves the value for the given parameters.
With this function it is possible to define which value to extract and from which Banana document file.

```
function getAmount(banDoc,account,property,startDate,endDate) {
   var currentBal = banDoc.currentBalance(account,startDate,endDate);
```

```
    var value = currentBal[property];
    return value;
}
```

The parameters are:

- **banDoc**: the Banana document from which retrieve the data (see Open Banana document);
- **account**: the account or group;
- **property**: the property of the returned currentBalance() object (i.e. opening, debit, credit, total);
- **startDate**: the opening date of the accounting period;
- **endDate**: the closing date of the accounting period;

```
// retrieve from the current year Banana document the 6541 account's total
value
var current6541 =
getAmount(current,'6541','total','2019-01-01','2019-12-31');

// retrieve from the previous year Banana document the 6541 account's total
value
var previous6541 =
getAmount(previous,'6541','total','2018-01-01','2018-12-31');
```

The use of the function is the same, but the returned values are different: one returns the value of the current year and the other the value of the previous year.

The Dates

Dates that we use in the script are taken from the accounting file using the info(section, id) function.

These dates are retrieved from the **Opening** and **Closing** dates of the accounting file (File properties > Accounting Tab).

```
// Accounting period for the current year file
var currentStartDate = current.info("AccountingDataBase","OpeningDate");
var currentEndDate = current.info("AccountingDataBase","ClosureDate");

// Accounting period for the previous year file
var previousStartDate = previous.info("AccountingDataBase","OpeningDate");
var previousEndDate = previous.info("AccountingDataBase","ClosureDate");
```

Function toLocaleNumberFormat()

The function toLocaleNumberFormat is used to convert all the amount numbers to the local format.

```
Banana.Converter.toLocaleNumberFormat('16570.00'); // returns 16'570.00
```

Install and run the BananaApp

Visit the Install your BananaApp documentation to install and run the app.

Report example:

Journal reporting

In the following we will explain what need to be considered when creating a new BananaApps for a journal reporting.

- Retrieving the transactions data.
- Creating specific reports using the functionalities offered by the Banana API.

Documentation

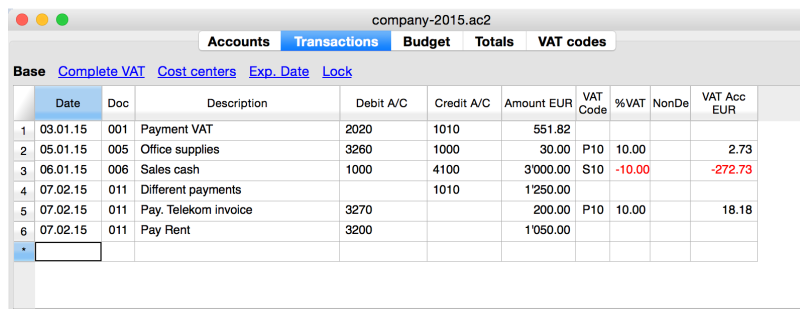
- Journal's API documentation

Example files

- The examples files are available on github/General/CaseStudies/JournalReport.
- Solutions making use of the journal api:
  - China, Voucher report
  - Netherlands, Auditfile

Transactions table

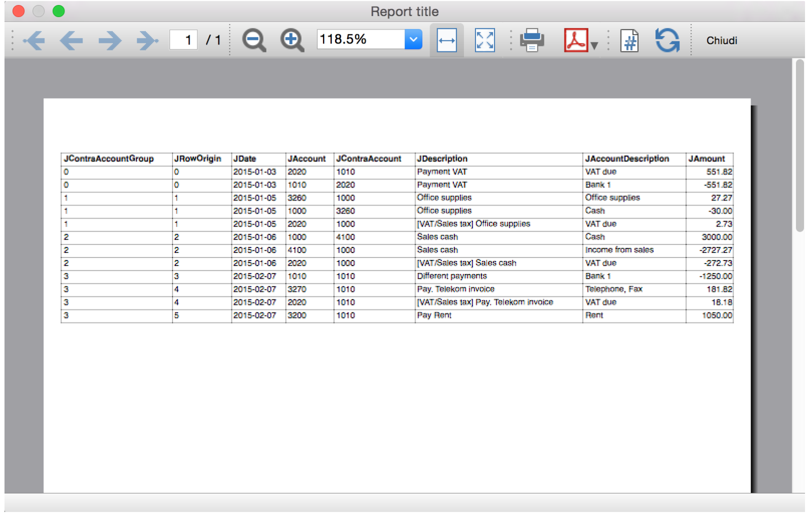
The following table is an example of transactions:


company-2015.ac2
Accounts | Transactions | Budget | Totals | VAT codes
Base | Complete VAT | Cost centers | Exp. Date | Lock

	Date	Doc	Description	Debit A/C	Credit A/C	Amount EUR	VAT Code	%VAT	NonDe	VAT Acc EUR
1	03.01.15	001	Payment VAT	2020	1010	551.82				
2	05.01.15	005	Office supplies	3260	1000	30.00	P10	10.00		2.73
3	06.01.15	006	Sales cash	1000	4100	3'000.00	S10	-10.00		-272.73
4	07.02.15	011	Different payments		1010	1'250.00				
5	07.02.15	011	Pay. Telekom invoice	3270		200.00	P10	10.00		18.18
6	07.02.15	011	Pay Rent	3200		1'050.00				

We see above different types of transactions. The transactions can be on a single line or over multiple lines, with or without VAT.

The idea here is to print a journal's table that contains all the accounts and the transactions. The final result it's the following one:



JContraAccountGroup	JRowOrigin	JDate	JAccount	JContraAccount	JDescription	JAccountDescription	JAmount
0	0	2015-01-03	2020	1010	Payment VAT	VAT due	551.82
0	0	2015-01-03	1010	2020	Payment VAT	Bank 1	-551.82
1	1	2015-01-05	3260	1000	Office supplies	Office supplies	27.27
1	1	2015-01-05	1000	3260	Office supplies	Cash	-30.00
1	1	2015-01-05	2020	1000	[VAT/Sales tax] Office supplies	VAT due	2.73
2	2	2015-01-06	1000	4100	Sales cash	Cash	3000.00
2	2	2015-01-06	4100	1000	Sales cash	Income from sales	-2727.27
2	2	2015-01-06	2020	1000	[VAT/Sales tax] Sales cash	VAT due	-272.73
3	3	2015-02-07	1010	1010	Different payments	Bank 1	-1250.00
3	4	2015-02-07	3270	1010	Pay. Telekom invoice	Telephone, Fax	181.82
3	4	2015-02-07	2020	1010	[VAT/Sales tax] Pay. Telekom invoice	VAT due	18.18
3	5	2015-02-07	3200	1010	Pay Rent	Rent	1050.00

Javascript API equivalent

To retrieve a Table object with all the amount registered on the accounts, we use the Journal's API:

```
var journal = Banana.document.journal(Banana.document.originType,
Banana.document.accountType);
```

where

originType specifies the row to be filtered for. Can be one of:

- ORIGINTYPE_NONE no filter is applyied and all rows are returned (current and budget)
- ORIGINTYPE_CURRENT only the normal transactions are returned
- ORIGINTYPE_BUDGET only the budget transactions are returned

accountType specifies the row to be filtered for. Can be one of:

- ACCOUNTTYPE_NONE no filter is applied and all rows are returned.
- ACCOUNTTYPE_NORMAL only rows for normal accounts are returned
- ACCOUNTTYPE_CC1 only rows for Cost Center 1 are returned
- ACCOUNTTYPE_CC2 only rows for Cost Center 2 are returned
- ACCOUNTTYPE_CC3 only rows for Cost Center 1 are returned
- ACCOUNTTYPE_CC Cost Center rows are returned same as using (ACCOUNTTYPE_CC1 | ACCOUNTTYPE_CC2 | ACCOUNTTYPE_CC3)

The returned table has all the columns of the transaction's table plus many other (please, visit the Journal's API for more information).

Code example

A common use to create and use a journal table to retrieve transactions data would be:

```
// Create the journal table
var journal = Banana.document.journal(Banana.document.ORIGINTYPE_CURRENT,
Banana.document.ACCOUNTTYPE_NORMAL);

// Read the table row by row and save some values
for (var i = 0; i < journal.rowCount; i++) {
```

```
        var tRow = journal.row(i);

        // From the journal table we take only the transactions rows
        if (tRow.value('JOperationType') ==
Banana.document.OPERATIONTYPE_TRANSACTION) {

                // Save some column values
                var jContraAccountGroup = tRow.value('JContraAccountGroup');
                var jRowOrigin = tRow.value('JRowOrigin');
                var jDate = tRow.value('JDate');
                var jAccount = tRow.value('JAccount');
                var jContraAccount = tRow.value('JContraAccount');
                var jDescription = tRow.value('JDescription');
                var jAccountDescription = tRow.value('JAccountDescription');
                var jAmount = tRow.value('JAmount');
                //...
        }
}
```

Results of the Journal's table for each transaction

The journal's table above is useful to better understand exactly how the journal works.

In general:

- For each account used in the transaction table (AccountDebit, AccountCredit, CC1, CC2, CC3) the program generates a journal row with the JAccount column set with the specific account.
- For a double entry account transaction that use AccountDebit, AccountCredit, AccountVat, CC1, CC2, CC3 the Journal will contain six rows. If the transaction has only AccountDebit and AccountCredit, then two rows will be generated.

All transactions in specific:

- Doc 001 – Single line transaction without VAT

	Date	Doc	Description	Debit A/C	Credit A/C	Amount EUR	VAT Code	%VAT	NonDe	VAT Acc EUR
1	03.01.15	001	Payment VAT	2020	1010	551.82				
2	05.01.15	005	Office supplies	3260	1000	30.00	P10	10.00		2.73
3	06.01.15	006	Sales cash	1000	4100	3'000.00	S10	-10.00		-272.73
4	07.02.15	011	Different payments		1010	1'250.00				
5	07.02.15	011	Pay. Telekom invoice	3270		200.00	P10	10.00		18.18
6	07.02.15	011	Pay Rent	3200		1'050.00				
*										

Journal:

JContraAccountGroup	JRowOrigin	JDate	JAccount	JContraAccount	JDescription	JAccountDescription	JAmount
0	0	2015-01-03	2020	1010	Payment VAT	VAT due	551.82
0	0	2015-01-03	1010	2020	Payment VAT	Bank 1	-551.82

One line for the 2020 JAccount
One line for the 1010 JAccount

- Doc 005 – Single line transaction with VAT

	Date	Doc	Description	Debit A/C	Credit A/C	Amount EUR	VAT Code	%VAT	NonDe	VAT Acc EUR
1	03.01.15	001	Payment VAT	2020	1010	551.82				
2	05.01.15	005	Office supplies	3260	1000	30.00	P10	10.00		2.73
3	06.01.15	006	Sales cash	1000	4100	3'000.00	S10	-10.00		-272.73
4	07.02.15	011	Different payments		1010	1'250.00				
5	07.02.15	011	Pay. Telekom invoice	3270		200.00	P10	10.00		18.18
6	07.02.15	011	Pay Rent	3200		1'050.00				
*										

Journal:

JContraAccountGroup	JRowOrigin	JDate	JAccount	JContraAccount	JDescription	JAccountDescription	JAmount
1	1	2015-01-05	3260	1000	Office supplies	Office supplies	27.27
1	1	2015-01-05	1000	3260	Office supplies	Cash	-30.00
1	1	2015-01-05	2020	1000	[VAT/Sales tax] Office supplies	VAT due	2.73

One line for the 3260 JAccount
One line for the 1000 JAccount
One line for the 2020 JAccount

- Doc 006 – Single line transaction with negative VAT

	Date	Doc	Description	Debit A/C	Credit A/C	Amount EUR	VAT Code	%VAT	NonDe	VAT Acc EUR
1	03.01.15	001	Payment VAT	2020	1010	551.82				
2	05.01.15	005	Office supplies	3260	1000	30.00	P10	10.00		2.73
3	06.01.15	006	Sales cash	1000	4100	3'000.00	S10	-10.00		-272.73
4	07.02.15	011	Different payments		1010	1'250.00				
5	07.02.15	011	Pay. Telekom invoice	3270		200.00	P10	10.00		18.18
6	07.02.15	011	Pay Rent	3200		1'050.00				
*										

Journal:

JContraAccountGroup	JRowOrigin	JDate	JAccount	JContraAccount	JDescription	JAccountDescription	JAmount
2	2	2015-01-06	1000	4100	Sales cash	Cash	3000.00
2	2	2015-01-06	4100	1000	Sales cash	Income from sales	-2727.27
2	2	2015-01-06	2020	1000	[VAT/Sales tax] Sales cash	VAT due	-272.73

One line for the 1000 JAccount
One line for the 4100 JAccount
One line for the 2020 JAccount. The VAT amount is in negative for the fact that the VAT amount is registered in credit, and therefore the amount must be pay to the tax authority

- Doc 011 – Multiple lines transaction with VAT

	Date	Doc	Description	Debit A/C	Credit A/C	Amount EUR	VAT Code	%VAT	NonDe	VAT Acc EUR
1	03.01.15	001	Payment VAT	2020	1010	551.82				
2	05.01.15	005	Office supplies	3260	1000	30.00	P10	10.00		2.73
3	06.01.15	006	Sales cash	1000	4100	3'000.00	S10	-10.00		-272.73
4	07.02.15	011	Different payments		1010	1'250.00				
5	07.02.15	011	Pay. Telekom invoice	3270		200.00	P10	10.00		18.18
6	07.02.15	011	Pay Rent	3200		1'050.00				
*										

Journal:

JContraAccountGroup	JRowOrigin	JDate	JAccount	JContraAccount	JDescription	JAccountDescription	JAmount
3	3	2015-02-07	1010	1010	Different payments	Bank 1	-1250.00
3	4	2015-02-07	3270	1010	Pay. Telekom invoice	Telephone, Fax	181.82
3	4	2015-02-07	2020	1010	[VAT/Sales tax] Pay. Telekom invoice	VAT due	18.18
3	5	2015-02-07	3200	1010	Pay Rent	Rent	1050.00

One line for the 1010 JAccount
One line for the 3270 JAccount
One line for the 2020 JAccount
One line for the 3200 JAccount

Working with import BananaApps

Introduction

This walkthrough provides step-by-step guidance for creating a simple BananaApp to import income & expenses transactions in CSV format, that reads the data to import from a **CSV file**, trasform and then return them in a format comptatible with Banana Accounting.

The steps in order to experiment with import BananaApps are the following:

1. Create a CSV file example
2. Create the import BananaApp
3. Install the BananaApp
4. Run the import BananaApp

Create the CSV file

First we need a data source in order to import them in Banana Accounting, and for this example we want to use a CSV file.

Copy the following CSV example, paste it on your text editor and save it as **csv_example.csv**:

```
"Date","Description","Income","Expenses"
"2019-01-01","Income transaction text","100.00",""
"2019-02-02","Expense transaction text","","200.00"
```

- First line is the fields header. Fields names are case sensitive and must correspond to the NameXml (English) of the columns in Banana Accounting.
- Fields names and data values are between double quotes.
- Fields and values are separated with a comma
- Each line is a new record
- The format for the Date fields is yyyy-mm-dd

Create the import BananaApp

Copy the following JavaScript code, paste it on your text editor and save it as **import_transaction_example.js**:

```
// @id = ch.banana.app.importtransactionexample
// @api = 1.0
// @pubdate = 2018-10-30
// @publisher = Banana.ch SA
// @description = Example Import Transactions (*.csv)
// @doctype = *
// @docproperties =
// @task = import.transactions
// @outputformat = transactions.simple
// @inputdatasource = openfiledialog
// @inputencoding = latin1
// @inputfilefilter = Text files (*.txt *.csv);;All files (*.*)

/* CSV file example:
```

```
"Date","Description","Income","Expenses"
"2019-01-01","Income transaction text","100.00",""
"2019-02-02","Expense transaction text","","200.00"
*/

// Parse the data and return the data to be imported as a tab separated file.
function exec(inText) {

   // Convert a csv file to an array of array.
   // Parameters are: text to convert, values separator, delimiter for text
values
   var csvFile = Banana.Converter.csvToArray(inText, ',', '"');
   // Converts a table (array of array) to a tsv file (tabulator separated
values)
   var tsvFile = Banana.Converter.arrayToTsv(csvFile);
   // Return the converted tsv file
   return tsvFile;

}
```

When it is used **transaction.simple** as **@outputformat** attribute in the script, it's important that CSV file includes "Income" and "Expenses" fields.

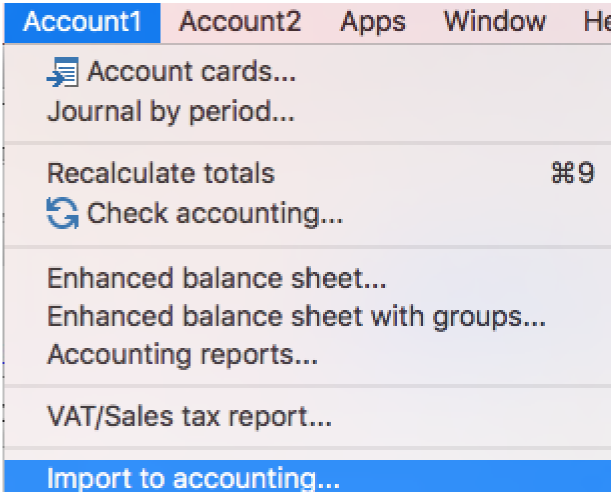
Install the BananaApp

For the installation of the BananaApp, see [Install your BananaApp](#).

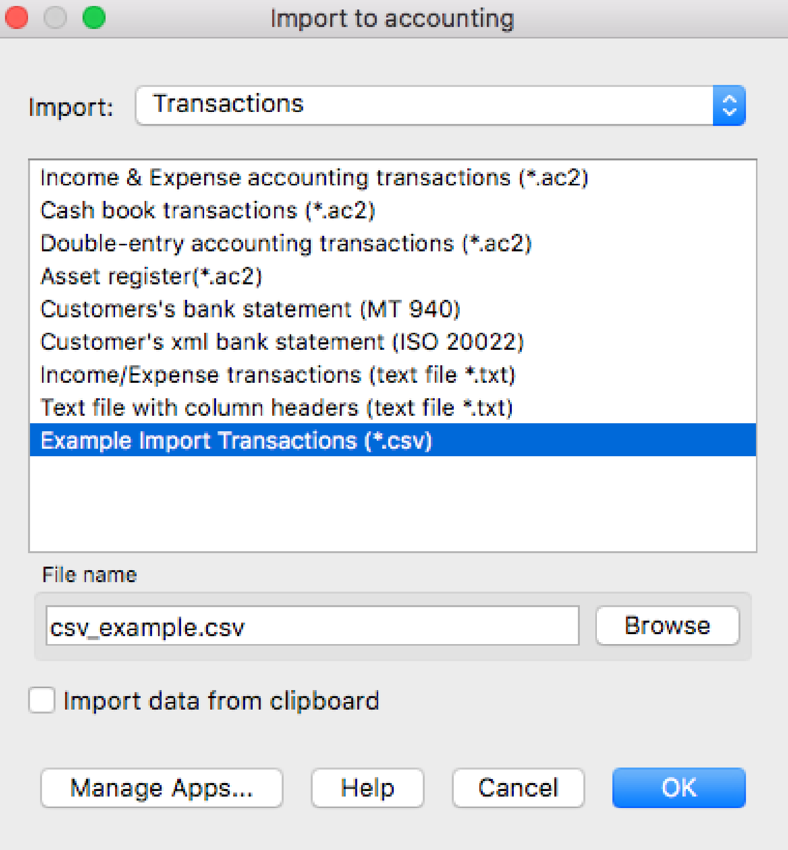
Run the import BananaApp

To run an import BananaApp follow the steps below:

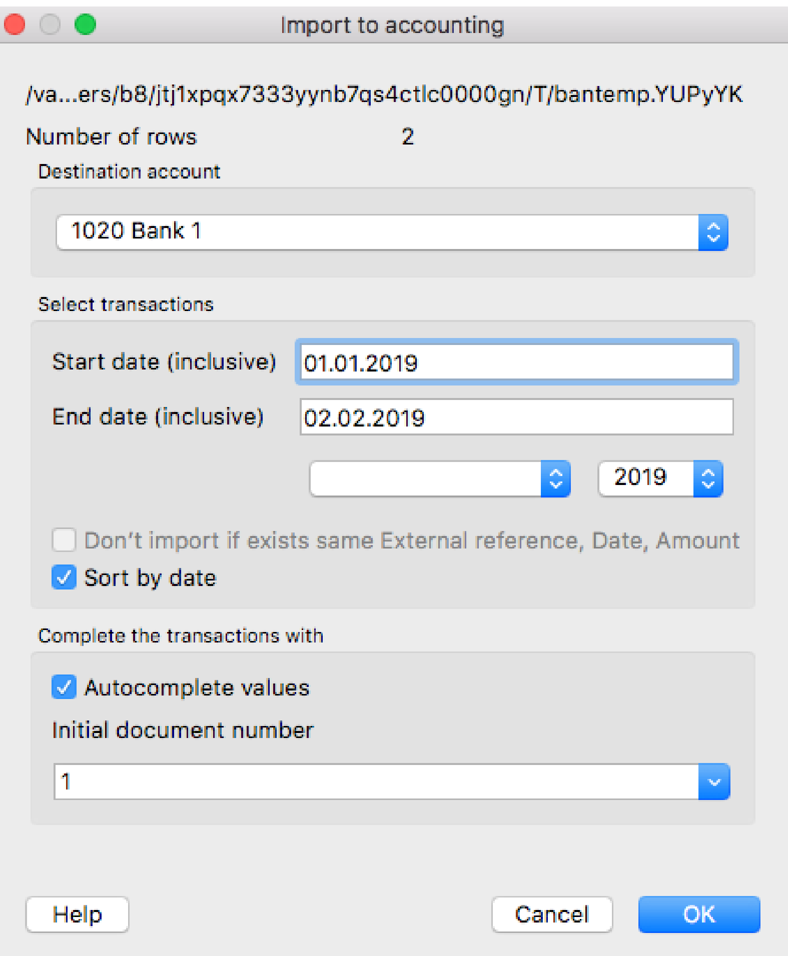
1. Open an accounting file in Banana Accounting.
2. In Banana select from the **menu Account1** the command **Import to accounting...**



3. From the import type selection select **Transactions**.
4. From the list select the **Example Import Transactions (*.csv)** BananaApp.
5. Click on **Browse** and look for the **csv_example.csv** file, then click to **Open**.

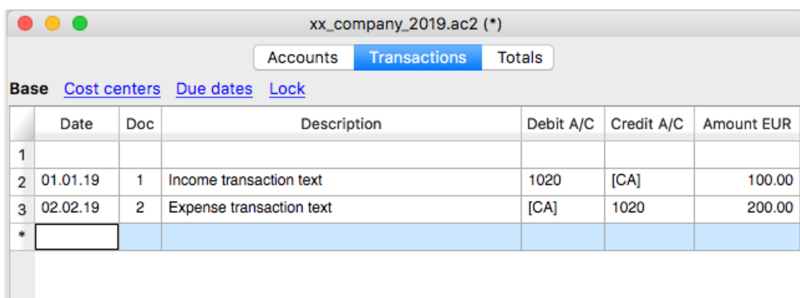


6. Click Ok to begin the import process.
7. On the dialog window select a **Destination account** and click on **Ok** to terminate and import the data.



The data from the CSV file are imported into the Transactions table of your accounting file like the following examples.

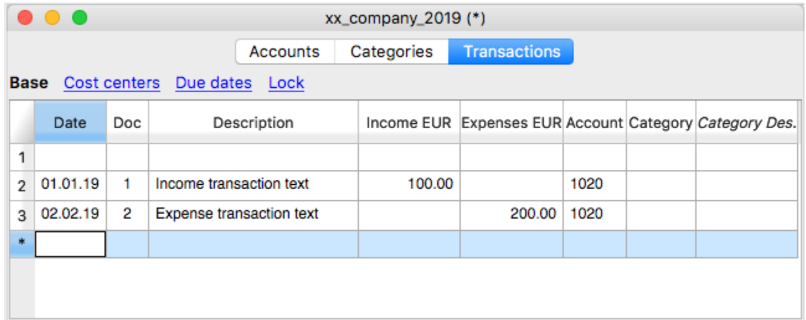
- For a Double-Entry accounting:



	Date	Doc	Description	Debit A/C	Credit A/C	Amount EUR
1						
2	01.01.19	1	Income transaction text	1020	[CA]	100.00
3	02.02.19	2	Expense transaction text	[CA]	1020	200.00
*						

You can now replace all the [CA] values with the appropriate contra-account, so that the Credit transactions will be balanced with the Debit transactions.

- For an Income & Expenses accounting:



	Date	Doc	Description	Income EUR	Expenses EUR	Account	Category	Category Des.
1								
2	01.01.19	1	Income transaction text	100.00		1020		
3	02.02.19	2	Expense transaction text		200.00	1020		
*								

For each transaction you can now enter an income or expense category, as defined in the Categories table.

More about Import BananaApps

- Import Apps
- Import into accounting
- Import transactions

Working with BananaApps TestFramework

Introduction

This walkthrough provides step-by-step guidance for creating a simple test case example that uses BananaApps Test Framework to run unit tests of BananaApps.

The steps in order to experiment with BananaApps Test Framework are the following:

1. Download and use the Banana Experimental version
2. Create a BananaApp to be tested
3. Create a test case
4. Run the test
5. Verify the test results

Create a BananaApp to be tested

In order to create a test case for a BananaApp it is required a working BananaApp to be tested.
For this example we use a modified version of "Hello World!" BananaApp example (see Build your first BananaApp).

1. Copy the code below, paste it on your text editor and save the file as **helloworld2.js**

```
// @id = ch.banana.app.helloworldexample
// @api = 1.0
// @pubdate = 2018-10-23
// @publisher = Banana.ch SA
// @description = Example Hello world 2
// @task = app.command
// @doctype = *.*
// @docproperties =
```

```
// @outputformat = none
// @inputdataform = none
// @timeout = -1
function exec() {
   //Call the function to create the report
   var report = createReport();

   //Print the report
   var stylesheet = Banana.Report.newStyleSheet();
   Banana.Report.preview(report, stylesheet);
}

function createReport() {
    //Create the report
    var report = Banana.Report.newReport("Report title");

    //Add a paragraph with the "hello world" text
    report.addParagraph("Hello World!");

    //Return the report
    return report;
}
```

Create a test case

Follow the instructions below to create a working test case:

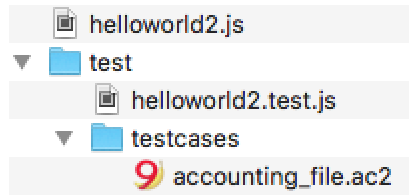
1. Create a folder **test** in the same folder where the BananaApp **helloworld2.js** is located.
2. Copy the following JavaScript code and paste it into a text editor.

```
// @id = ch.banana.app.helloworld2example.test
// @api = 1.0
// @pubdate = 2018-10-30
// @publisher = Banana.ch SA
// @description = [Test] Example Hello world 2
// @task = app.command
// @doctype = *.*
// @docproperties =
// @outputformat = none
// @inputdataform = none
// @timeout = -1
// @includejs = ../helloworld2.js

// Register this test case to be executed
Test.registerTestCase(new TestFrameworkExample());

// Define the test class, the name of the class is not important
function TestFrameworkExample() {
}

// This method will be called at the beginning of the test case
```

```
TestFrameworkExample.prototype.initTestCase = function() {
   this.progressBar = Banana.application.progressBar;
}

// This method will be called at the end of the test case
TestFrameworkExample.prototype.cleanupTestCase = function() {
}

// This method will be called before every test method is executed
TestFrameworkExample.prototype.init = function() {
}

// This method will be called after every test method is executed
TestFrameworkExample.prototype.cleanup = function() {
}

// Every method with the prefix 'test' are executed automatically as
test method
// You can defiend as many test methods as you need

TestFrameworkExample.prototype.testVerifyMethods = function() {
   Test.logger.addText("The object Test defines methods to verify
conditions.");

   // This method verify that the condition is true
   Test.assert(true);
   Test.assert(true, "message"); // You can specify a message to be
logged in case of failure

   // This method verify that the two parameters are equals
   Test.assertIsEqual("Same text", "Same text");
}

TestFrameworkExample.prototype.testBananaApps = function() {
   Test.logger.addText("This test will tests the BananaApp
helloworld.js");
   var document =
Banana.application.openDocument("file:script/../test/testcases/accountin
g_file.ac2");
   Test.assert(document, "File ac2 not found");
   // Add the report content text to the result txt file
   var report = createReport();
   Test.logger.addReport("ReportName", report);
}
```

3. Modify the script: find the row "**var document = Banana.application.openDocument("file:script/../test/testcases/accounting_file.ac2")**" and replace "**accounting_file.ac2**" with the name of your Banana accounting file.
4. Save the file into **test** folder as **helloworld2.test.js** (the file name must always be **<same_name_bananaapp>.test.js**).
5. Create a folder **test/testcases** and copy there your file **.ac2**.

6. Now you should have something like this:



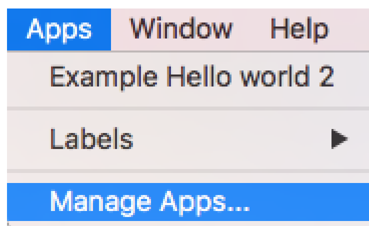
   - **helloworld2.js**: the BananaApp
   - **test**: the test folder
   - **helloworld2.test.js**: the test script for the BananaApp
   - **testcases**: the folder for test ac2 files
   - **xxx.ac2**: your Banana accounting file

Run the test case

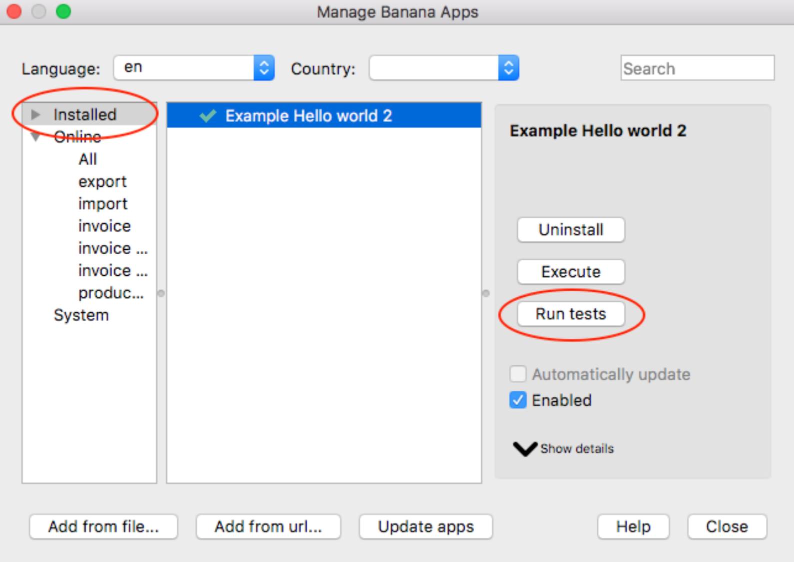
Finally, now it is possible to run the test case and see the results.

To run the test:

1. In Banana select from the **menu Apps** the command **Manage Apps...**



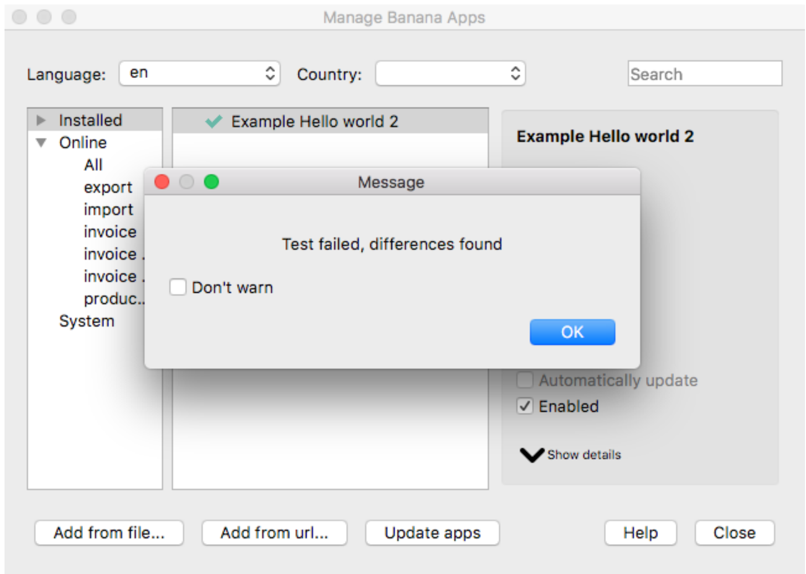
2. Select from the **Installed** element the “Example Hello World 2” BananaApp
3. Click on the button **Run tests**



Test case results

The test compare the current results with the expected results and checks if there are differences. If differences are found a dialog message warn you. All the details can be found inside the **.test.diff.txt file**.

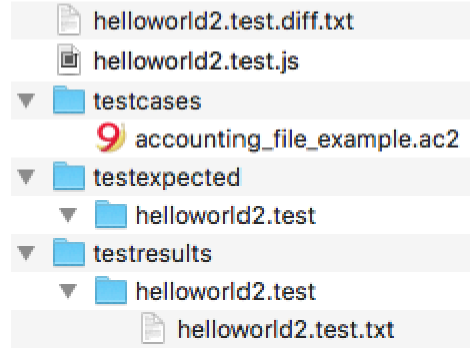
The first time you run a test probably you will see the following "Test failed" dialog message:



In your **test** folder you can see two new auto generated folders:

- **test/testresults**:
  contains the **helloworld2.test** folder with a **helloworld2.test.txt** file with the results of the test.
  When the test is run, the folder and the file are automatically generated.
- **test/testexpected**:
  contains the **helloworld2.test** folder that should contain a **helloworld2.test.txt** file with the expected results.
  When the test is run, the folder is automatically generated, but the file **NOT**. The first time you run the test the folder is **empty**. This is why the test fails.

You now should have something like that:



Inside the **test/testresults/helloworld2.test** folder there is the **helloworld2.test.txt** file with all the results of the test, like the following one:

```
%%info_test_name{helloworld2.test}
\documentclass{report}
\usepackage{longtable}
\usepackage{color}
\usepackage{listings}
\usepackage[margin=1cm]{geometry}
\begin{document}
\newenvironment{tablevalue}[2]{\textbf{Table: #1}\begin{longtable}[l]{#2}}{\end{longtable}}
\newenvironment{reportvalue}[1]{\textbf{Report: #1}\\}{}
\lstnewenvironment{jsonvalue}[1]{\textbf{Json: #1}\lstset{language=Java}}{}
\lstnewenvironment{xmlvalue}[1]{\textbf{Xml: #1}\lstset{language=Xml}}{}
\newcommand{\info}[2]{\textit{#1: #2}\\}
\newcommand{\fatalerror}[1]{\textbf{\textcolor[rgb]{1,0,0}{Fatal error: #1}}\\}
\newcommand{\keyvalue}[2]{\textbf{Keyvalue: #1} #2\\}
\newcommand{\textvalue}[1]{#1\\}
\newcommand{\excltest}[1]{\\Excluded from compare: #1}
\newcommand{\lognewpage}{\newpage}
\newenvironment{testcase}[1]{\section*{Test: #1}}{\newpage}
\newenvironment{test}[1]{\section*{Test case: #1}}{}
%Begin test data
%%SCRIPT{/Users/user_name/Desktop/BananaApps/test_case/./test/helloworld2.test.js}
%
\begin{testcase}{TestFrameworkExample}
\begin{test}{testVerifyMethods}
\textvalue{The object Test defines methods to verify conditions.}\\
\keyvalue{Result}{Passed}
\end{test}
\begin{test}{testBananaApps}
\textvalue{This test will tests the BananaApp helloworld.js}\\
\begin{reportvalue}{ReportName}
Hello World! \\
\end{reportvalue}
\keyvalue{Result}{Passed}
\end{test}
\end{testcase}
\end{document}
```

The file **helloworld2.test.diff.txt** is a resume of the results, with all the differences the test has found.

The image below shows an example of test summary with differences:

- with the sign "+" are indicated the rows added to the .txt file of the test/testresults folder (compared with the testexpected folder)
- with the sign "-" are indicated the rows removed from the .txt file of the test/testresults folder (compared with the testexpected folder)

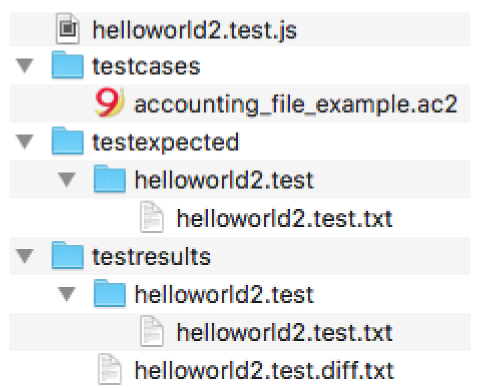
```
Test Banana Apps
----------------

Date:        2018-10-31 08:31:20.778
Duration:    00:00:00.043
Application: BananaExpm90 9.0.3.180906
OS:          macOS Sierra (10.12)
Qt:          5.11.1

Test summary
------------

Test failed

> Fatal errors:  0
> Differences:   1
> Added:         14
> Deleted:       0
> Identical:     0
> Total results: 15
> Total files:   1

Differences Summary
-------------------

> Files with differences
>> testresults/helloworld2.test/helloworld2.test.txt

> Files with results only in current
>> testresults/helloworld2.test/helloworld2.test.txt

Differences Details
-------------------

>> testresults/helloworld2.test/helloworld2.test.txt
                        1  + \begin{document}
                        2  + \begin{testcase}{TestFrameworkExample}
                        3  + \begin{test}{testVerifyMethods}
                        4  + \textvalue{The object Test defines methods to verify conditions.}\\
                        5  + \keyvalue{Result}{Passed}
                        6  + \end{test}
                        7  + \begin{test}{testBananaApps}
                        8  + \textvalue{This test will tests the BananaApp helloworld.js}\\
                        9  + \begin{reportvalue}{ReportName}
                        10  + Hello World! \\
                        11  + \end{reportvalue}
                        12  + \keyvalue{Result}{Passed}
                        13  + \end{test}
                        14  + \end{testcase}
                        15  + \end{document}
                         1 -
```

As mentioned above, the folder test/testexpected is empty. This is why we can see a lot of added rows to the .txt file of the test/testresults folder.

If you have differences and and you know these differences are correct (like in this case):

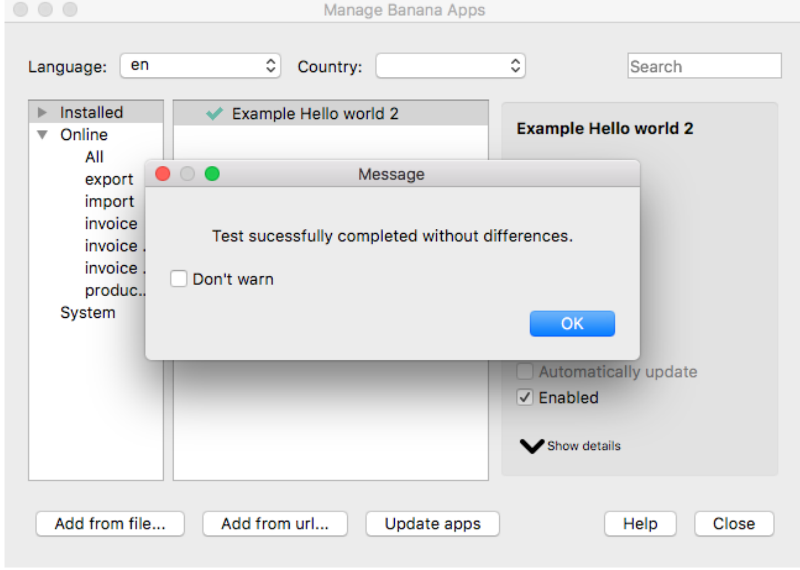
- copy the test results **.txt** file from the folder **test/testresults** to the folder **test/testexpected**.

You should have a structure like this:



Note that the **helloworld2.test.txt** file is now located in both folders, in the **test/testresults/helloworld2.test** folder and in the **test/testexpected/helloworld2.test** folder.

If you run again the test, this time the result is different. You now should see a positive message from the dialog:



This means that the results .txt file generated from the test, is perfectly identical of the expected .txt file. So, the test is successfully passed.

You can also check the **helloworld2.test.diff.txt** file of the **test/testresults/helloworld2.test** folder to see the differences results, and there should not be differences found.

```
Test Banana Apps
----------------

Date:        2018-10-31 08:56:12.512
Duration:    00:00:00.037
Application: BananaExpm90 9.0.3.180906
OS:          macOS Sierra (10.12)
Qt:          5.11.1

Test summary
------------

Test succesfully completed

> Fatal errors:  0
> Differences:   0
> Added:         0
> Deleted:       0
> Identical:     15
> Total results: 15
> Total files:   1
```

From that point, every time you do changes to the BananaApp you can test it and see if the changes you made works as you expected.

Remember: if you have differences and you know these differences are correct, you have to replace the expected .txt file.

More about BananaApps Test

- BananaApps Test Framework
- TestFramework GitHub documentation
- JavaScript code example (ch.banana.script.testframework.test.js)

App Design

App File

Javascript compliant script

BananaApps are essentially javascript compliant script files (ECMA-262). People knowing javascript can easily write BananaApps.

A BananaApp file contains the following two sections:

- **Apps's attributes**
  The apps's attributes give information about the script, like it purpose, description and so on. They are inserted at the beginning of the file through tags in comment's lines.

- The **exec([inData, options]) function**
  The function exec() is the function that is called every time a BananaApps is executed.
  It has some optional arguments:
  - ***inData***: the requested input data as a string or a Banana.Document object;
    This parameters is only used for import scripts, for all other tasks this parameter is just null;
  - ***options***: options as an object that can contains those parameters;
  - ***useLastSettings***: if true the script executes with the last used setttings and doesn't show

a setting's dialog;
The script return a string formatted according the tag @outputformat.
Errors are notified through exceptions (clause throw), or just by returning a string beginning with "@Error:"

- [Optional] The **settingsDialog() function**
  If the script has a dialog for settings some parameters it is advised to put the code for the dialog in this function. In this way the application can call this function just for showing or editing the parameters without executing the whole script.
  This method should return null if the user click on cancel button, or a value different of null if the user click on ok button.

For a list of supported javascript functions, objects and properties see: Qt ECMAScript Reference.

BananaApps interact with Banana Accounting through some global objects made available by Banana Accounting, like for example 'Banana.document'. Those objects are described under the Banana Script API.

BananaApp "Hello World" example

Here an example that open a print preview windows, and show a document with the text "Hello world!!!". Other examples are found in the BananaApps tutorial.

```
// @id = ch.banana.report.helloworld
// @version = 1.0
// @doctype = nodocument
// @publisher = Banana.ch SA
// @description = Hello world
// @task = app.command
// @timeout = -1
function exec() {
   //Create the report
   var report = Banana.Report.newReport('Report title');
   //Add a paragraph with some text
   report.addParagraph('Hello World!!!');
   //Preview the report
   var stylesheet = Banana.Report.newStyleSheet();
   Banana.Report.preview(report, stylesheet);
}
```

BananaApp with a setting's dialog example

Here an example that use a dialog to input a text. Other examples are found in the BananaApps tutorial.

```
// @id = ch.banana.report.settingsdialog
// @version = 1.0
// @doctype = *
// @publisher = Banana.ch SA
// @description = Example for settings dialog
// @task = app.command
// @timeout = -1
```

```
function exec(inData, options) {
   // Show dialog if options.useLastSettings is not set or is false
   if (!options || !options.useLastSettings) {
      if (!settingsDialog())
         return; // return if user pressed cancel
   }

   // Get the settings
   var text = Banana.document.getScriptSettings();

   //Create the report
   var report = Banana.Report.newReport('Report title');
   report.addParagraph('You entered: "' + text + '"');
   report.addParagraph(new Date().toString());

   //Stylesheet
   var stylesheet = Banana.Report.newStyleSheet();

   //Preview the report
   Banana.Report.preview(report, stylesheet);
}

function settingsDialog() {
   // Ask the user to enter a text that will be printed in the report
   var text = Banana.document.getScriptSettings();
   text = Banana.Ui.getText("Enter a text", "The text will be printed in the
report", text);
   if (typeof(text) === 'string') {
      Banana.document.setScriptSettings(text);
      return true;
   }
   return false; // cancel pressed
}
```

BananaApps have a strong Security model

BananaApps are secure for the fact that are confined within Banana.
BananaApps are NOT ALLOWED to directly write or read file, web resource, change computer setting or execute programs.

BananaApps, contrary to Excel Macro, can be run with the confidence, they will not change any data and modify any file or computer settings.

To accesso or write to file you need to use the Banana Api that display a dialog box to the user.

- To write file you need to use the export functionality, that display a dialog where the user indicate the file name where to save.
- To import file you need to use the import functionality that display a dialog where the user specify the file name.

Best way to distribute the BananaApp

- Single App file (javascript file)
  - Easier to edit (external text editors), move and update.
  - Can be included in the menu Apps.
  - Can be used by different accounting file.
- Embedded apps
  - Not available in the menu Apps.
  - Only relative to the file where it is included.
  - More difficult to edit and update.
- Packaged App file
  - Cannot be easily changed.
  - Can be included in the menu Apps.
  - Can be used by different accounting file.
  - Protected from user modification.

BananaApps as a single javascript file

A single javascript (.js) file that includes all the code of the app.

This is how it works:

- BananaApp are saved in UTF-8 file without BOOM.
- The BananaApp needs to be installed through the Manage Apps command.
- Once the BananaApp is installed, it appears in the menu Apps.
- The BananaApp can be run from the menu Apps.

Embedded BananaApps in documents

Banana allows to have BananaApps that are embedded within a Banana File. Embedded apps run only for the specific file, but don't need to be installed.

To create embedded BananaApps you can add script files in the table Documents.

On the Embedded BananaApps JavaScript Tutorial you will find the documentation and different basic examples embedded in a document that you can run and edit.

BananaApps as packaged file

It is possible to package one or more apps composed by one or more files (.js, .qml and other files) in one single **.sbaa BananaApp file** (see documentation below).

It's very practical for distributing Apps composed by two or more files, or packages with two or more BananaApps.

This is how it works:

- The .sbaa BananaApp needs to be installed through the Manage Apps command.
- Once the BananaApp is installed, it appears in the menu Apps.
- The BananaApp can be run from the menu Apps.

BananaApps file extention '.sbaa'

A **.sbaa** file can be either a text file containing javascript code (.js files) or a packaged qt resource file (.sbaa). The application determine automatically the type of the file.

When Banana loads a packaged .sbaa file, it looks for all .js files contained in the package that have an attribute section. Those files are readen and a corresponding entry is inserted in the menu Apps.

Javascript files in packages can include other javascript files in the same package using the directive **@includejs** or the method Banana.include(fileName). It is not possibile to include files outside the package.

```
// Include a script via @includejs attribute
// @includejs = somescript.js"

// Include a script via Banana.include() method
Banana.include(somescript.js);
```

Here is how to create a packaged .sbaa file:

- Create a manifest.json file with the information regarding the package
- Create a .qrc file with the list of the files to be included.
- It is also possibile to create package files with the 'rcc' tool from the QT (see below)
- Open Banana Accounting
- Drag the .qrc file in Banana Accounting.
- It will ask you if you want to compile the file and will generate a .sbaa file.

Qrc resource file (.qrc)

For more information see the Qt Resource system.

Example: ch.banana.script.report.jaml.qrc

```
<!DOCTYPE RCC><RCC version="1.0">
<qresource>
     <file>ch.banana.script.report.jaml.js</file>
     <file>lib/jaml-all.js</file>
     <file>manifest.json</file>
</qresource>
</RCC>
```

Qrc file file can also be compiled with the QT

```
rcc -binary ch.banana.script.report.jaml.qrc -o
ch.banana.script.report.jaml.rcc
```

Manifest file

If you create a .sbaa file, also include a manifest file. The **manifest.json file** is a JSON-formatted file, which you can include in the .sbaa file through the .qrc file.
Using manifest.json, you specify basic metadata about your package such as the title, description and version.
The file name must end with the extension 'manifest.json'

Example: ch.banana.script.report.vat-ch.qrc

```
<!DOCTYPE RCC><RCC version="1.0">
```

```
<qresource>
     <file alias="manifest.json">ch.banana.script.report.vat-
ch.manifest.json</file>
     <file>ch.banana.script.report.vat-ch.js</file>
</qresource>
</RCC>
```

Example: ch.banana.script.report.vat-ch.manifest.json

```
{
    "category": "productivity",
    "country":"switzerland",
    "countryCode":"ch",
    "description": "Postfinance Schweiz (Bewegungen importieren): ISO20022
und CSV Format",
    "description.en": "Swiss Postfinance (Import transactions): ISO20022 and
CSV File Format",
    "language":"de",
    "publisher": "Banana.ch",
    "title": "Postfinance Schweiz (Bewegungen importieren)",
    "title.en": "Swiss Postfinance (Import transactions)",
    "version": "1.0"
}
```

- **Available categories**: export, import, invoice, invoice reminder, invoice statement, productivity.
  If you don't specify the category ("category": ""), the program will take the category from the first app included in the package. If you don't specify country or language, the app will be shown for any country or language.
- All tags are optional

Apps Attributes

At the beginning at the script there should be a part that define the Apps Attribute.

```
// @api = 1.0
// @id = ch.banana.apps.example.docfilepath
// @description = Hello world
// @task = app.command
// @doctype = nodocument
// @publisher = Banana.ch SA
// @pubdate = 2015-05-12
// @inputdatasource = none
// @timeout = -1
```

The attribute is a commented text line

- Sart with //
- Followed by the attribute that start with @
- Fossowed by the " = " and the value

Tags defines the purpose (import, export, extract, ...), the name displayed in the dialogs, the kind of data it expect to receive, the kind of data it returns, and other information of the script. Tags are inserted at the beginning of the script in comment's lines though the following syntax: "// @tag-name = tag-value".

Attribute list

Attribute name	**Required**	**Value** [1)]	**Description**
@api	Required	The required API version. Available API versions: 1.0	Define the required API to be executed in the form of MAIN_VERSION.SUB_VERSION The implemented API in the application have to be equal or newer.
@contributors		List of contributors separated by ';'	This attribute contains the list of people that contribute to the developping of the BananaApp.
@description[.lang]	Required	The name or description of the script	This text will be displayed in the dialogs. This tag is localisable.
@docproperties		any text	Define a property the script is writen for. With this attribute you can manually select for what document the script is visbile in the menu Add-ons and can be run. The property can be added to the document though the dialog Add-Ons. The property can be any text (ex.: "datev", "realestate", ...). Mulitple properties can be defined wiht a ';' as separator (ex.: "datev;skr03").
@doctype	Required	nodocument * XXX.* XXX.YYY !XXX.YYY ...	Define the type of document the script is writen for. With this attribute you can define for what type of document the script is visbile in the menu Add-ons and can be run. nodocument = doesn't require an open document, the add-on is always visible * = for any type of document, always visible if a document is open *.* = for any type of document, always visible if a document is open 100.* = for Double entry accountings 110.* = for Income & Expenses accountings 130.* = for Cash Books 400.* = for Addresses and Labels 100.100 = for accountings without VAT and without foreign currrencies 100.110 = for accountings wiht VAT 100.120 = for accountings with foreing currencies 100.130 = for accountings with VAT and foreign currencies The sign ! is used to invert the definition. The above codes can be combined togheter like the following examples: 100.130 = for double entries with VAT and with foreign currencies 100.120;100.130 = for double entry with foreign currencies 100.*;110.*;130.* = for all accounting files !130.* = for any filesexcept cash books
@exportfilename		A string defining the name of the file where to export the data.	If the string contains the text <Date>, it will be replaced by the current date in the format of yyyyMMdd-hhmm.

@exportfiletype		A string defining the type of data exported txt ...	This parameter is used for export scripts, it defines the type of exported data and it is used for the extension in the save file dialog.
@id	Required	An identification of the script	It is used when setting and reading the preferences. In order to avoid duplicate banana.ch use the following scheme. country.developper.app.domain.name for example: ch.banana.app.patriziato.consuntivopersubtotali
@includejs		Relative path to a javascript .js file to load before the execution of the script.	Include the javascript file. Every function and object defined in the file are then available to the current script.
@inputdatasource		One of the following values: none **openfiledialog** fixedfilepath [2)]	With this attribute you can specify if you don't need to input data, if you want the user to select a file to import (openfiledialog), or if you want to get a file which path is defined in @inputfilepath. If you set fixedfilepath the program will ask the user the permission to open this file, the user's answer will be saved for the next execution.
@inputencoding		The encoding of the input data. One of the following values: latin1 **utf-8** iso 8859-1 to 10 ...	The encoding used to read the input file (for import apps). If the attribute is empty or not defined, the application try to decode the input data with utf-8, if it fails, the application decode the input data with latin1. For a complete list see *QTextCodec*
@inputfilefilter[.lang]		The file filter for the open file dialog Ex.: Text files (*.txt *.csv);;All files (*.*)	This value describes the file filters you want to show in the input file dialog. If you need multiple filters, separate them with ';;' for instance. This tag is localizable.
@inputfilepath		The file to read for the input data	If the script has the value fixedfilepath as @inputdatasource, you can define here the path of the file to load.
@inputformat		One of the following values: **text** ac2	If "text" the filter receive the selected file in inData as a text. If "ac2" the filter receive the selected file in inData as a Banana.Document object.
@outputencoding		The encoding of the input data. One of the following values: latin1 **utf-8** iso 8859-1 to 10	The encoding used to write the output file (for export apps). For a complete list see *QTextCodec*
@outputformat		One of the follwing values: **tablewithheaders** transactions.simple	If the script has an import tasks this value define the format of the returned value. The format transaction.simple contains the transaction as income / expenses. For details of the formats see Import data from a txt file.
@pubdate	Required	The publication date in the format YYYY-MM-DD	This publication date is also used for scripts published by Banana.ch to check if newer version exist.
@publisher		The publisher of the script	

@task	Required	One of following values: app.command export.file export.rows export.transactions import.rows **import.transactions** import.accounts import.categories import.exchangerates import.vatcodes report.general report.customer.invoice report.customer.statement report.customer.reminder	This value define the purpouse of the script, and determine in which dialog or menu the script is visible. • app.command for general script • export.* for Export Apps • import.* for Import Apps • report.customer.invoice for invoice templates • report.customer.statement for statement templates • report.customer.reminder for reminder templates
@testapp		Path and name of the test app. Default is './test/<bananaapp_name>.test.js	This can be used if there is a test app and the path to the test app is different to the default path. Since Banana 9.0.4
@testappversionmin **@testappversionmax**		Only for test cases. Minimum and mximum application's version to whitch the test is applicable.	
@timeout		The timeout for the script in milliseconds, default is **2000** (2 seconds). If you set -1 the timeout is disabled and the application allow you to abort it though a progress bar.	If the script takes longer than this value to finish, it will be aborted and a message showed. If you have a script with a very long run time, you can increase the timeout or set it to -1.

1) Default values are listed in bold.

2) Function not yet available

Example:

```
// @api = 1.0
// @id = ch.banana.apps.example.docfilepath
// @description = Hello world
// @task = app.command
// @doctype = nodocument
// @publisher = Banana.ch SA
// @pubdate = 2015-05-12
// @inputdatasource = none
// @timeout = -1

/**
 * Hello world example for Banana Accounting.
 */
function exec(inData) {
    Banana.Ui.showInformation("", "Hello World");

    if (Banana.document) {
       var fileName = Banana.document.info("Base","FileName");
       Banana.Ui.showInformation("Current document", fileName);
    }
}
```

Apps Parameters

Apps parameters allow to initialize and set parameters that are relative to a BananaApps, for example:

- Parameters for the printing.
- Header of a report that are set once only.

The script should provide a function settingDialog() that is called when the user click on the Set Parameters on the Manage Apps dialog.

The function settinDialog() should:

1. Read the existing setting with the Banana.document.getScriptSettings();
2. Request user to enter the information
3. Set the modified values with the function Banana.document.setScriptSettings(paramToString);
   The JSon text will be saved within the accounting file.

```
function settingsDialog() {
   var param = initParam();
   var savedParam = Banana.document.getScriptSettings();
   if (savedParam.length > 0) {
      param = JSON.parse(savedParam);
   }
   param = verifyParam(param);
   param.isr_bank_name = Banana.Ui.getText('Settings',
texts.param_isr_bank_name, param.isr_bank_name);
   if (param.isr_bank_name === undefined)
      return;
   var paramToString = JSON.stringify(param);
   Banana.document.setScriptSettings(paramToString);
}
```

the function Exec() should then read the setting.
It is a good practice to check and verify if the setting are valid.

```
function printDocument(jsonInvoice, repDocObj, repStyleObj) {
  var param = initParam();
  var savedParam = Banana.document.getScriptSettings();
  if (savedParam.length > 0) {
    param = JSON.parse(savedParam);
    param = verifyParam(param);
  }
  printInvoice(jsonInvoice, repDocObj, repStyleObj, param);
}
```

Import Apps

Import filters are import BananaApps that read a custom format and convert in an import format

suitable for using with the with the command "Import to accounting".

- For details of the formats see Import data from a txt file.
- For examples see the Github.com template page.

Import Apps

Imports BananaApps are JavaScript program that read the data to import and trasform and return them as text, in a format comptatible with Banana.

Import BananaApps have:

- the attribute **@task** defined as one of the import for example **//@task = import.transactions** (for more information, see Apps attributes documentation)
- The parameter in the function exec contains the import data (the content of the file specified in the input box)
- You can specify that the data is read from the file specified on the input box or that the user can select the file with "// @inputdatasource = openfiledialog"
- The import text is returned as a String in the function exec with the return statement

```
// @api = 1.0
// @id = ch.banana.scripts.import.creditsuisse
// @description = Credit Suisse bank (*.csv)
// @task = import.transactions
// @doctype = nodocument
// @publisher = Banana.ch SA
// @pubdate = 2015-06-21
// @outputformat = transactions.simple
// @inputdatasource = openfiledialog
// @inputfilefilter = Text files (*.txt *.csv);;All files (*.*)
// @inputfilefilter.de = Text (*.txt *.csv);;Alle Dateien (*.*)
// @inputfilefilter.fr = Texte (*.txt *.csv);;Tous (*.*)
// @inputfilefilter.it = Testo (*.txt *.csv);;Tutti i files (*.*)

/**
* Parse the data and return the data to be imported as a tab separated file.
*/
function exec(inText) {
  // parse the inText and set to outText
  // in the return text the data is tab separated
  var outText = "";
  outText += "Date\tDescription\tIncome\tExpenses\n";
  outText += "2015-01-01\tIncome text\t100.25\t\n";
  outText += "2015-01-02\tExpense text\t\t73.50\n";
  return outText;
}
```

Export Apps

Export apps are used to export data in a custom format.

- Define attribute @task as export.file
  // @task = export.file

- Define the extension of the file to be exported.
  // @exportfiletype = xml

- The text to be written to the export file is the return value of the exec function and must be a return.
  return "exported text".

- When the script terminate and if the return text is not null and does not start with "@Cancel ", the user will be promped with a dialog to choose a file name where to export.

Example

Export all the accounting with description and balance in a xml file.

```
// @id = ch.banana.apps.export
// @api = 1.0
// @pubdate = 2016-04-08
// @doctype = *.*
// @description = Export into a text file (.txt)
// @task = export.file
// @exportfiletype = txt
// @timeout = -1

function exec() {
var exportResult = '<accounts>';
    var tableAccounts = Banana.document.table('Accounts');
    if ( !tableAccounts) {
      return;
   }
   for (i=0;i<tableAccounts.rowCount;i++) {
        if (tableAccounts.row(i).value('Account')) {
            exportResult += '<account>';
            exportResult += '<accountnr>' +
tableAccounts.row(i).value('Account') + '</accountnr>';
            exportResult += '<description>' +
tableAccounts.row(i).value('Description') + '</description>';
            exportResult += '<balance>' +
tableAccounts.row(i).value('Balance') + '</balance>';
            exportResult += '</account>';
        }
    }
    exportResult += '</accounts>';
    //return the string
    return exportResult;
}
```

Report Apps

Report apps are java-script programs that are used to customize printouts like invoice documents. The main function printDocument() receives the json object from Banana, writes the document and lunches the result in a print preview window.

Copies of some report apps that you can use as starting point are available at the following address: github.com/BananaAccounting

- Invoice Report Apps
- Reminder Report Apps
- Statement Report Apps

Important notes

- Banana Accounting uses Qt script engine to execute report apps.
- Mandatory functions: printDocument(jsonInvoice, repDocObj, repStyleObj) which is the main function and settingsDialog() which is called from user to set up parameters like colour or additional text.
- Available json objects: invoice json object, statement json object, reminder json object
- The extension of custom report apps should be .js and the script must contains the main attributes, see Apps Attributes.
- The attribute @id of the script should correspond to the file name.
- System report apps are downloaded to the folder /User/.../AppData/Local/Banana.ch/.../Apps (Mac Users: /Users/.../Library/Application Support/Banana.ch/.../Apps)
- Do not overwrite system report apps because updates will overwrite your changes.
- You can save your report app anywhere, Banana Accounting saves the path to your app in the configuration file /AppData/Local/Banana.ch/.../Apps/apps.cfg

Invoice App

Create personalized invoice report apps

We have published our templates on:

- How to create your own invoice
- github.com/BananaAccounting/Universal: in this section you will find different basic examples that works for every country.
- github.com/BananaAccounting/Switzerland: in this section you will find different basic examples made for Switzerland (with the Swiss ISR payment slip).

You can save a copy of one template in your computer and make the changes you wish. In order to use your custom template in Banana you have to:

- select the command **Account2 - Customers - Print invoices...**
- In the **Print invoices** dialog select **Manage apps...**
- In the **Manage apps** dialog select **Add from file...** and choose your invoice report file you just created

Apps attributes

```
// @id = scriptfilename.js
// @api = 1.0
// @pubdate = yyyy-mm-dd
// @publisher = yourName
// @description = script description
// @task = report.customer.invoice
```

Report code

The main function is **printDocument(jsonInvoice, repDocObj, repStyleObj)**. The parameter jsonInvoice object contains the data, repDocObj is the document object and repStyleObj is the stylesheet object where you can add styles.

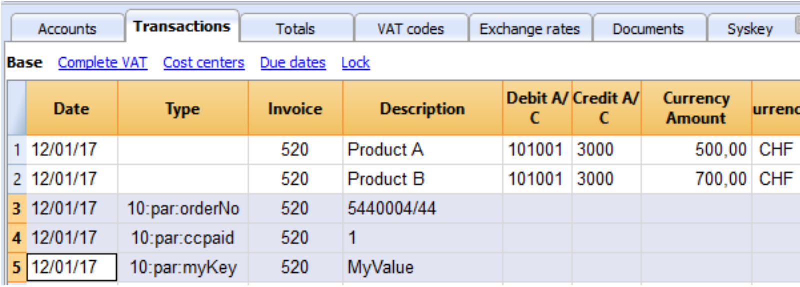
```
function printDocument(jsonInvoice, repDocObj, repStyleObj) {
  var param = initParam();
  var savedParam = Banana.document.getScriptSettings();
  if (savedParam.length > 0) {
    param = JSON.parse(savedParam);
    param = verifyParam(param);
  }
  printInvoice(jsonInvoice, repDocObj, repStyleObj, param);
}
```

The function settingsDialog() is called from Banana when you select the button **Params...** from dialog **Manage apps**. You can write any code you need for your script.

```
/*Update script's parameters*/
function settingsDialog() {
   var param = initParam();
   var savedParam = Banana.document.getScriptSettings();
   if (savedParam.length > 0) {
      param = JSON.parse(savedParam);
   }
   param = verifyParam(param);
   ...
   var paramToString = JSON.stringify(param);
   var value = Banana.document.setScriptSettings(paramToString);
}
```

Printing custom data

You can add your own parameters in order to print specific data. For instance printing a reference order number or removing payments information if the invoice has already been paid.



Accounts | Transactions | Totals | VAT codes | Exchange rates | Documents | Syskey

Base | Complete VAT | Cost centers | Due dates | Lock

	Date	Type	Invoice	Description	Debit A/C	Credit A/C	Currency Amount	urrenc
1	12/01/17		520	Product A	101001	3000	500,00	CHF
2	12/01/17		520	Product B	101001	3000	700,00	CHF
3	12/01/17	10:par:orderNo	520	5440004/44				
4	12/01/17	10:par:ccpaid	520	1				
5	12/01/17	10:par:myKey	520	MyValue				

```
if (invoiceObj.parameters.orderNo) {
    cell1.addParagraph("Reference order: " + invoiceObj.parameters.orderNo);
  }
```

Printing images

With the command **addImage** it is possible to add images into the document. For instance

```
var reportObj = Banana.Report;
var repDocObj = reportObj.newReport();
repDocObj.addImage("documents:logo", "logoStyle");

var logoStyle = repStyleObj.addStyle(".logoStyle");
logoStyle.setAttribute("position", "absolute");
logoStyle.setAttribute("margin-top", "5mm");
logoStyle.setAttribute("margin-left", "20mm");
logoStyle.setAttribute("width", "120px");
```

If you set the width, the image will be resized to the given width. If the width is not specified the image will be printed with a 72dpi resolution.

Invoice Json Fields

Invoice Json Object Source Data

This list explains where the actual information on your invoice json object is coming from

Invoice Object Property	Source
customer_info.address1	Table: Accounts, View: Address, Column: Street
customer_info.address2	Table: Accounts, View: Address, Column: AddressExtra
customer_info.address3	Table: Accounts, View: Address, Column: POBox
customer_info.balance	Table: Accounts, View: Address, Column: Balance
customer_info.balance_base_currency	Table: Accounts, View: Address, Column: BalanceCurrency
customer_info.bank_account	Table: Accounts, View: Address, Column: BankAccount
customer_info.bank_clearing	Table: Accounts, View: Address, Column: BankClearing
customer_info.bank_name	Table: Accounts, View: Address, Column: BankName
customer_info.business_name	Table: Accounts, View: Address, Column: OrganisationName

Invoice Object Property	Source
customer_info.city	Table: Accounts, View: Address, Column: Locality
customer_info.country	Table: Accounts, View: Address, Column: Country
customer_info.country_code	Table: Accounts, View: Address, Column: CountryCode
customer_info.courtesy	Table: Accounts, View: Address, Column: NamePrefix
customer_info.credit_limit	Table: Accounts, View: Address, Column: CreditLimit
customer_info.currency	Table: Accounts, View: Address, Column: Currency
customer_info.date_birth	Table: Accounts, View: Address, Column: DateOfBirth
customer_info.email	Table: Accounts, View: Address, Column: EmailWork
customer_info.fax	Table: Accounts, View: Address, Column: Fax
customer_info.first_name	Table: Accounts, View: Address, Column: FirstName
customer_info.fiscal_number	Table: Accounts, View: Address, Column: FiscalNumber
customer_info.iban_number	Table: Accounts, View: Address, Column: BankIban
customer_info.lang	Table: Accounts, View: Address, Column: Language
customer_info.last_name	Table: Accounts, View: Address, Column: FamilyName
customer_info.member_fee	Table: Accounts, View: Address, Column: MemberFee
customer_info.mobile	Table: Accounts, View: Address, Column: PhoneMobile
customer_info.number	Table: Accounts, View: Address, Column: Account
customer_info.payment_term_days	Table: Accounts, View: Address, Column: PaymentTermInDays
customer_info.phone	Table: Accounts, View: Address, Column: PhoneMain
customer_info.postal_code	Table: Accounts, View: Address, Column: PostalCode
customer_info.state	Table: Accounts, View: Address, Column: Region
customer_info.vat_number	Table: Accounts, View: Address, Column: VatNumber
customer_info.web	Table: Accounts, View: Address, Column: Website
document_info.currency	Invoice currency which usually corresponds to the customer account currency
document_info.date	Table: Transactions, Column: DateDocument or Date
document_info.decimals_amounts	Decimals are the same as the decimals used in the accounting file
document_info.description	Not used
document_info.doc_type	Table: Transactions, Column: DocType
document_info.greetings	Table: Transactions, Column: DocType Transactions with DocType=**10:gre** If there are many rows with 10:gre the texts are joined with ','. More info...

Invoice Object Property	Source
document_info.locale	Menu: File-File and accounting properties, Other, current Language
document_info.number	Table: Transactions, Column: DocInvoice
document_info.origin_row	Row index of source transaction
document_info.origin_table	Table name of source transaction
document_info.rounding_total	Default value for CHF: 0.05 You can overwrite this value with the menu command: Account2 - Customers - Settings - Advanced - Invoice rounding For multicurrency accounting: you can setup the rounding value for each currency in the table ExchangeRates, column DecimalPoints
document_info.type	invoice
items	Table: Transactions All rows with the same invoice number and transaction date are invoice's items (lines)
note	Table: Transactions, Column: DocType Transactions with DocType=**10:not**. More info...
parameters	Table: Transactions, Column: DocType Transactions with DocType=**10:par:key** Key: any key text you wish Value: is taken from column Description More info...
payment_info	Calculated from journal
shipping_info	Delivery address if different from the invoice address (customer_info) Table: Transactions, Column: DocType Transactions with DocType=**10:sadr** More info...
supplier_info.address1	Menu: File-File and accounting properties, Address, Address 1
supplier_info.address2	Menu: File-File and accounting properties, Address, Address 2
supplier_info.business_name	Menu: File-File and accounting properties, Address, Company
supplier_info.city	Menu: File-File and accounting properties, Address, City
supplier_info.country	Menu: File-File and accounting properties, Address, Country
supplier_info.courtesy	Menu: File-File and accounting properties, Address, Courtesy
supplier_info.email	Menu: File-File and accounting properties, Address, Email
supplier_info.fax	Menu: File-File and accounting properties, Address, Fax
supplier_info.first_name	Menu: File-File and accounting properties, Address, Name

Invoice Object Property	Source
supplier_info.fiscal_number	Menu: File-File and accounting properties, Address, Fiscal Number
supplier_info.last_name	Menu: FilevFile and accounting properties, Address, Family Name
supplier_info.mobile	Menu: File-File and accounting properties, Address, Mobile
supplier_info.phone	Menu: File-File and accounting properties, Address, Phone
supplier_info.postal_code	Menu: File-File and accounting properties, Address, Zip
supplier_info.state	Menu: File-File and accounting properties, Address, Region
supplier_info.vat_number	Menu: File-File and accounting properties, Address, Vat Number
supplier_info.web	Menu: File-File and accounting properties, Address, Web
transactions	Table: Transactions All rows with the same invoice number and different transaction date, which are not considered invoice items, like payments transactions

Json Object

Invoice Json Object

Data structure you can access through the report:

```
{
    "billing_info": {
        "payment_term": "",
        "total_amount_vat_exclusive": "500.00",
        "total_amount_vat_exclusive_before_discount": "500.00",
        "total_amount_vat_inclusive": "540.00",
        "total_amount_vat_inclusive_before_discount": "540.00",
        "total_categories": [
        ],
        "total_discount_percent": "",
        "total_discount_vat_exclusive": "",
        "total_discount_vat_inclusive": "",
        "total_rounding_difference": "",
        "total_to_pay": "540.00",
        "total_vat_amount": "40.00",
        "total_vat_amount_before_discount": "40.00",
        "total_vat_codes": [
        ],
        "total_vat_rates": [
            {
                "total_amount_vat_exclusive": "500.00",
                "total_amount_vat_inclusive": "540.00",
                "total_vat_amount": "40.00",
                "vat_rate": "8.00"
```

```
        }
    ]
},
"customer_info": {
    "address1": "Viale Stazione 11",
    "address2": "",
    "address3": "",
    "balance": "102.60",
    "balance_base_currency": "102.60",
    "business_name": "Rossi SA",
    "city": "Bellinzona",
    "country": "Switzerland",
    "country_code": "CH",
    "courtesy": "Signor",
    "currency": "CHF",
    "date_birth": "1999-10-06",
    "email": "info@test.com",
    "first_name": "Andrea",
    "lang": "it",
    "last_name": "Rossi",
    "mobile": "0033608405",
    "number": "1100",
    "origin_row": "26",
    "origin_table": "Accounts",
    "postal_code": "6500",
    "vat_number": "1234"
},
"document_info": {
    "currency": "CHF",
    "date": "20160101",
    "decimals_amounts": 2,
    "description": "",
    "doc_type": "10",
    "locale": "it",
    "number": "201710",
    "origin_row": "1",
    "origin_table": "Transactions",
    "printed": "1",
    "rounding_total": "0.05",
    "type": "invoice"
},
"items": [
    {
        "account_assignment": "3000",
        "description": "Prodotto A",
        "details": "",
        "index": "0",
        "item_type": "item",
        "mesure_unit": "",
        "number": "",
        "origin_row": "1",
```

```
        "origin_table": "Transactions",
        "quantity": "1",
        "total_amount_vat_exclusive": "500.00",
        "total_amount_vat_inclusive": "540.00",
        "total_vat_amount": "40.00",
        "unit_price": {
            "amount_vat_inclusive": "540.00",
            "calculated_amount_vat_exclusive": "500.00",
            "calculated_amount_vat_inclusive": "540.00",
            "calculated_vat_amount": "40.00",
            "currency": "CHF",
            "vat_code": "V80",
            "vat_rate": "8.00"
        }
    }
],
"note": [
        {
            "date": "2017-04-24",
            "description": "commande=AWO-003530",
            "origin_row": "968",
            "origin_table": "Transactions"
        }
],
"parameters": {
    "ccpaid":"1",
    "orderNo":"5440004/44",
    "myKey":"MyValue"
},
"payment_info": {
    "date_expected": "2017-05-24",
    "due_date": "20160131",
    "due_days": "240",
    "due_period": "0_>90",
    "last_reminder": "",
    "last_reminder_date": "",
    "payment_date": ""
},
"shipping_info": {
     "address1": "26, lotissement Bellevue",
     "address2": "street2",
     "address3": "street3",
     "business_name": "Company",
     "city": "Clairac",
     "country": "FR",
     "different_shipping_address": true,
     "first_name": "Carla",
     "last_name": "Francine",
     "postal_code": "47320"
},
"supplier_info": {
```

```
        "address1": "Indirizzo 1",
        "address2": "Indirizzo 2",
        "business_name": "Società",
        "city": "Loc",
        "courtesy": "Signor",
        "email": "info@myweb",
        "fax": "+419100000",
        "first_name": "Nome",
        "fiscal_number": "222",
        "last_name": "Cognome",
        "phone": "+419100000",
        "postal_code": "CAP",
        "state": "Suisse",
        "vat_number": "1111",
        "web": "http://www.myweb"
    }
    "transactions": [
        {
            "balance": "-219.30",
            "balance_base_currency": "",
            "currency": "CHF",
            "date": "20170430",
            "description": "Paiement CERAT DES ALPES* 50 ml - Réf. AL07010",
            "origin_row": "1006",
            "origin_table": "Transactions"
        }
    ],
    "type": "invoice",
    "version": "1.0"
}
```


Statement

Create personalized statement report apps

We have published our templates on github.com/BananaAccounting. In this section you will find different basic examples.

You can save a copy of one template in your computer and make the changes you wish. In order to use your custom template in Banana you have to:

- select the command **Account2 - Customers - Print statements...**
- In the **Print statements** dialog select **Manage apps...**
- In the **Manage apps** dialog select **Add from file...** and choose your statement report file you just created

Apps attributes

```
// @id = scriptfilename.js
// @api = 1.0
// @pubdate = yyyy-mm-dd
// @publisher = yourName
// @description = script description
// @task = report.customer.statement
```

Report code

The main function is **printDocument(jsonStatement, repDocObj, repStyleObj)**. The parameter jsonStatement object contains the data, repDocObj is the document object and repStyleObj is the stylesheet object where you can add styles.

```
function printDocument(jsonStatement, repDocObj, repStyleObj) {
  var param = initParam();
  var savedParam = Banana.document.getScriptSettings();
  if (savedParam.length > 0) {
    param = JSON.parse(savedParam);
    param = verifyParam(param);
  }
  printInvoice(jsonInvoice, repDocObj, repStyleObj, param);
}
```

The function settingsDialog() is called from Banana when you select the button **Params...** from dialog **Manage apps**. You can write any code you need for your script.

```
/*Update script's parameters*/
function settingsDialog() {
   var param = initParam();
   var savedParam = Banana.document.getScriptSettings();
   if (savedParam.length > 0) {
      param = JSON.parse(savedParam);
   }
   param = verifyParam(param);
   ...
   var paramToString = JSON.stringify(param);
   var value = Banana.document.scriptSaveSettings(paramToString);
}
```

Json Object

Statement Json Object

Data structure you can access through the report:

```
{
    "customer_info": {
```

```
        "address1": "Viale Stazione 11",
        "business_name": "Rossi SA",
        "city": "Bellinzona",
        "first_name": "Andrea",
        "last_name": "Rossi",
        "number": "1100",
        "origin_row": "26",
        "origin_table": "Accounts",
        "postal_code": "6500"
    },
    "document_info": {
        "date": "20160927",
        "decimals_amounts": 2,
        "description": "",
        "locale": "it",
        "number": "",
        "type": "statement"
    },
    "items": [
        {
            "balance": "540.00",
            "credit": "",
            "currency": "CHF",
            "date": "20160101",
            "debit": "540.00",
            "due_date": "20160131",
            "due_days": "240",
            "item_type": "invoice",
            "last_reminder": "",
            "last_reminder_date": "",
            "number": "10",
            "payment_date": "",
            "status": "",
            "total_amount_vat_exclusive": "",
            "total_amount_vat_inclusive": "",
            "total_vat_amount": "",
            "unit_price": {
            }
        },
        {
            "balance": "540.00",
            "credit": "",
            "currency": "",
            "date": "",
            "debit": "540.00",
            "item_type": "total",
            "number": "",
            "status": "",
            "total_amount_vat_exclusive": "",
            "total_amount_vat_inclusive": "",
            "total_vat_amount": "",
```

```
            "unit_price": {
            }
        }
    ],
    "supplier_info": {
        "address1": "Indirizzo 1",
        "address2": "Indirizzo 2",
        "business_name": "Società",
        "city": "Loc",
        "email": "info@myweb",
        "first_name": "Nome",
        "fiscal_number": "numerofiscale",
        "last_name": "Cognome",
        "postal_code": "CAP",
        "web": "http://www.myweb"
    }
}
```


Reminder

Create personalized reminder report apps

We have published our templates on github.com/BananaAccounting. In this section you will find different basic examples.

You can save a copy of one template in your computer and make the changes you wish. In order to use your custom template in Banana you have to:

- select the command **Account2 - Customers - Print reminders...**
- In the **Print payment reminders** dialog select **Manage apps...**
- In the **Manage apps** dialog select **Add from file...** and choose your reminder report file you just created

Apps attributes

```
// @id = scriptfilename.js
// @api = 1.0
// @pubdate = yyyy-mm-dd
// @publisher = yourName
// @description = script description
// @task = report.customer.reminder
```

Report code

The main function is **printDocument(jsonReminder, repDocObj, repStyleObj)**. The parameter jsonReminder object contains the data, repDocObj is the document object and repStyleObj is the stylesheet object where you can add styles.

```
function printDocument(jsonReminder, repDocObj, repStyleObj) {
```

```
  var param = initParam();
  var savedParam = Banana.document.getScriptSettings();
  if (savedParam.length > 0) {
    param = JSON.parse(savedParam);
    param = verifyParam(param);
  }
  printReminder(jsonReminder, repDocObj, repStyleObj, param);
}
```

The function settingsDialog() is called from Banana when you select the button **Params...** from dialog **Manage apps**. You can write any code you need for your script.

```
/*Update script's parameters*/
function settingsDialog() {
   var param = initParam();
   var savedParam = Banana.document.getScriptSettings();
   if (savedParam.length > 0) {
      param = JSON.parse(savedParam);
   }
   param = verifyParam(param);
   ...
   var paramString = JSON.stringify(param);
   var value = Banana.document.setScriptSettings(paramString);
}
```

Json Object

Reminder Json Object

Data structure you can access through the report:

```
{
    "customer_info": {
        "address1": "Viale Stazione 11",
        "business_name": "Rossi SA",
        "city": "Bellinzona",
        "first_name": "Andrea",
        "last_name": "Rossi",
        "number": "1100",
        "origin_row": "26",
        "origin_table": "Accounts",
        "postal_code": "6500"
    },
    "document_info": {
        "date": "20160927",
        "decimals_amounts": 2,
        "description": "",
```

```
        "locale": "it",
        "number": "",
        "type": "reminder"
    },
    "items": [
        {
            "balance": "540.00",
            "balance_base_currency": "540.00",
            "base_currency": "CHF",
            "credit": "",
            "credit_base_currency": "",
            "currency": "CHF",
            "date": "20160101",
            "debit": "540.00",
            "debit_base_currency": "540.00",
            "item_type": "invoice",
            "number": "10",
            "status": "1. reminder",
            "total_amount_vat_exclusive": "",
            "total_amount_vat_inclusive": "",
            "total_vat_amount": "",
            "unit_price": {
            }
        },
        {
            "balance": "540.00",
            "balance_base_currency": "540.00",
            "base_currency": "",
            "credit": "",
            "credit_base_currency": "",
            "currency": "",
            "date": "",
            "debit": "540.00",
            "debit_base_currency": "540.00",
            "item_type": "total",
            "number": "",
            "status": "",
            "total_amount_vat_exclusive": "",
            "total_amount_vat_inclusive": "",
            "total_vat_amount": "",
            "unit_price": {
            }
        }
    ],
    "supplier_info": {
        "address1": "Indirizzo 1",
        "address2": "Indirizzo 2",
        "business_name": "Società",
        "city": "Loc",
        "email": "info@myweb",
        "first_name": "Nome",
```

```
        "fiscal_number": "numerofiscale",
        "last_name": "Cognome",
        "postal_code": "CAP",
        "web": "http://www.myweb"
    }
}
```

Dialogs

For simple interactions with the user you can use the predefined dialogs of the class Banana.Ui. With those dialogs you can ask the user to insert a value, answer a question, or show to the user an information.

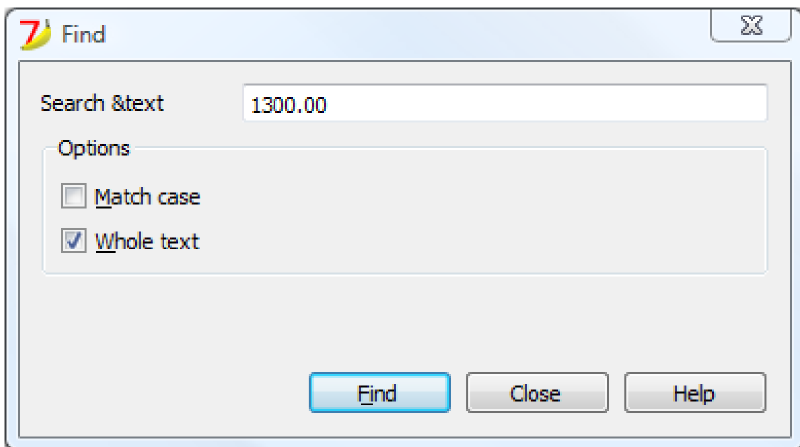
For a more complex dialog:

- Install Qt Creator
- Draw the dialog with Qt Creator
- Save the dialog in a .ui file,
- Load the .ui file in the script through the function Banana.Ui.createUi()

All the properties and public slots of the widgets in the dialogs will be accessible from the script.

Example: a script to search in the whole accounting a text.

The dialog:



The script file ch.banana.scripts.find.js:

```
/**
* This example search a text in all the tables of the document,
* and show the matches in the messages pane.
*/
// @id = ch.banana.scripts.find
// @version = 1.2
// @date = 2014-08-29
// @publisher = Banana.ch SA
// @description = Find in whole accounting
```

```
// @description.it = Cerca in tutta la contabilità
// @description.de = Suchen in der gesamten Buchhaltung
// @description.fr = Chercher dans toute la comptabilité
// @task = app.command
// @inputdatasource = none
// @timeout = -1

/**
* param values are loaded from Banana.document, edited through dialog and
saved to Banana.document
* This array object is like a map (associative array) i.e. "key":"value", see
initParam()
* Examples of keys: searchText, wholeText, ...
*/
var param = {};

/** Dialog's functions declaration */
var dialog = Banana.Ui.createUi("ch.banana.scripts.find.ui");

dialog.checkdata = function () {
    var valid = true;

    if (dialog.searchTextLineEdit.text.length <= 0) {
        Banana.Ui.showInformation("Error", "Search text can't be empty");
        valid = false;
    }

    if (valid) {
        dialog.accept();
    }
}

dialog.showHelp = function () {
    Banana.Ui.showHelp("ch.banana.script.find");
}

/** Dialog's events declaration */
dialog.findNextButton.clicked.connect(dialog, "checkdata");
dialog.buttonBox.accepted.connect(dialog, "checkdata");
dialog.buttonBox.rejected.connect(dialog, "close");
dialog.buttonBox.helpRequested.connect(dialog, "showHelp");

/** Main function */
function exec(inData) {

    //calls dialog
    var rtnDialog = true;
    rtnDialog = dialogExec();

    //search text in the whole accounting
    if (rtnDialog && Banana.document) {
```

```
        Banana.document.clearMessages();
        searchInTables();
    }
}

/** Show the dialog and set the parameters */
function dialogExec() {

    // Read saved script settings
    initParam();
    if (Banana.document) {
        var data = Banana.document.getScriptSettings();
        if (data.length > 0) {
            param = JSON.parse(data);
        }
    }

    // Text at cursor position
    var cursor = Banana.document.cursor;
    param["searchText"] =
Banana.document.value(cursor.table,cursor.row,cursor.column);

    // Set dialog parameters
    dialog.searchTextLineEdit.text = param["searchText"];
    if (param["matchCase"] == "true")
        dialog.groupBox.matchCaseCheckBox.checked = true;
    else
        dialog.groupBox.matchCaseCheckBox.checked = false;
    if (param["wholeText"] == "true")
        dialog.groupBox.wholeTextCheckBox.checked = true;
    else
        dialog.groupBox.wholeTextCheckBox.checked = false;

    Banana.application.progressBar.pause();
    var dlgResult = dialog.exec();
    Banana.application.progressBar.resume();

    if (dlgResult !== 1)
        return false;

    // Read dialog parameters
    param["searchText"] = dialog.searchTextLineEdit.text;
    if (dialog.groupBox.matchCaseCheckBox.checked)
        param["matchCase"] = "true";
    else
        param["matchCase"] = "false";
    if (dialog.groupBox.wholeTextCheckBox.checked)
        param["wholeText"] = "true";
    else
        param["wholeText"] = "false";
    // Save script settings
```

```
    var paramString = JSON.stringify(param);
    var value = Banana.document.setScriptSettings(paramString);

    return true;
}

/** Initialize dialog values with default values */
function initParam() {
    param = {
        "searchText": "",
        "matchCase": "false",
        "wholeText": "false"
    };
}

/** Search a text in the accounting's tables */
function searchInTables() {
    var searchText = param["searchText"];
    if (param["matchCase"] === "false")
        searchText = searchText.toLowerCase();
    var tables = Banana.document.tableNames;
    // Tables
    for (var t=0; t < tables.length; t++) {
        var table = Banana.document.table(tables[t]);
        var columns = table.columnNames;
        // Rows
        for (var r=0; r < table.rowCount; r++) {
            // Columns
            for (var c=0; c < columns.length; c++) {
                var textFound = false;
                var text = table.value(r, columns[c]);
                if (param["matchCase"] === "false")
                    text = text.toLowerCase();
                // Find text
                if (param["wholeText"] === "true") {
                    if (text === searchText)
                        textFound = true;
                } else {
                    if (text.indexOf(searchText) >= 0)
                        textFound = true;
                }
                // Show message
                if (textFound) {
                    table.addMessage("Text \"" + param["searchText"] +
                        "\" found in \"" + table.value(r, columns[c]) + "\"",
r, columns[c]);
                }
            }
        }
    }
}
```

The .ui file: ch.banana.scripts.find.ui

```
<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
 <class>DlgFind</class>
 <widget class="QDialog" name="DlgFind">
  <property name="geometry">
   <rect>
    <x>0</x>
    <y>0</y>
    <width>395</width>
    <height>192</height>
   </rect>
  </property>
  <property name="windowTitle">
   <string>Find</string>
  </property>
  <layout class="QVBoxLayout" name="verticalLayout_2">
   <item>
    <layout class="QGridLayout" name="gridLayout">
     <property name="horizontalSpacing">
      <number>40</number>
     </property>
     <item row="0" column="0">
      <widget class="QLabel" name="searchTextLabel">
       <property name="text">
        <string>Search &amp;text</string>
       </property>
      </widget>
     </item>
     <item row="0" column="1">
      <widget class="QLineEdit" name="searchTextLineEdit"/>
     </item>
    </layout>
   </item>
   <item>
    <widget class="QGroupBox" name="groupBox">
     <property name="title">
      <string>Options</string>
     </property>
     <property name="flat">
      <bool>false</bool>
     </property>
     <property name="checkable">
      <bool>false</bool>
     </property>
     <layout class="QVBoxLayout" name="verticalLayout">
      <item>
       <widget class="QCheckBox" name="matchCaseCheckBox">
        <property name="text">
         <string>&amp;Match case</string>
```

```
     </property>
    </widget>
   </item>
   <item>
    <widget class="QCheckBox" name="wholeTextCheckBox">
     <property name="text">
      <string>&amp;Whole text</string>
     </property>
    </widget>
   </item>
  </layout>
 </widget>
</item>
<item>
 <spacer name="verticalSpacer">
  <property name="orientation">
   <enum>Qt::Vertical</enum>
  </property>
  <property name="sizeHint" stdset="0">
   <size>
    <width>20</width>
    <height>15</height>
   </size>
  </property>
 </spacer>
</item>
<item>
 <layout class="QHBoxLayout" name="horizontalLayout">
  <item>
   <spacer name="horizontalSpacer">
    <property name="orientation">
     <enum>Qt::Horizontal</enum>
    </property>
    <property name="sizeHint" stdset="0">
     <size>
      <width>80</width>
      <height>20</height>
     </size>
    </property>
   </spacer>
  </item>
  <item>
   <widget class="QPushButton" name="findNextButton">
    <property name="text">
     <string>&amp;Find</string>
    </property>
   </widget>
  </item>
  <item>
   <widget class="QDialogButtonBox" name="buttonBox">
    <property name="sizePolicy">
```

```
        <sizepolicy hsizetype="Minimum" vsizetype="Fixed">
         <horstretch>0</horstretch>
         <verstretch>0</verstretch>
        </sizepolicy>
       </property>
       <property name="standardButtons">
        <set>QDialogButtonBox::Close|QDialogButtonBox::Help</set>
       </property>
      </widget>
     </item>
    </layout>
   </item>
  </layout>
 </widget>
 <tabstops>
  <tabstop>matchCaseCheckBox</tabstop>
  <tabstop>findNextButton</tabstop>
  <tabstop>buttonBox</tabstop>
 </tabstops>
 <resources/>
 <connections/>
</ui>
```

API

Banana namespace

The whole API (Application Program Interface) made available for Banana is under the namespace "Banana".
The are different objects and methods that belong to the name space Banana, that can be accessed by the javascript at run time:

Data formats

Date

Date values are in ISO 8601 format "YYYY-MM-DD".

Decimal

Decimal values have a '.' (dot) as decimal separator and doesn't have a group separator. For example: "12345.67".
Decimal values are rounded according to the accounting settings.

Text

Text values can contain any character supported by UTF-8.

Time

Time values are in ISO 8601 format "HH:MM:SS". The formats "HH:MM" and "HH:MM:SS.ZZZ" are also

accepted.

API Versions

List of API Version made available by Banana Accounting.

Banana Accounting Version	API Version
7.0.6	1.0
8.0.7 or more recent	1.0
9.0.0 or more recent	1.0

Banana (Objects)

Banana is the namespace (object) through which all Banana script's methods, class and objects are accessible.

Banana.application

The object Banana.application represent the running application.

Banana.console

The object Banana.console is used to sent message to the debug the script.

Banana.Converter

The class Banana.Converter contains methods useful to convert data from and to various formats.

Banana.document

The object Banana.document represent the current document opened in the application. It contains base properties and methods, see Banana.Document (Base), and if the document represent an accounting document it contains additional accounting's properties and methods, see Banana.Document (Accounting). If any document is opened this object is of type Undefined.

Banana.IO

The class Banana.IO is used to read and write files.

Banana.Report

The class Banana.Report enable you to create reports, preview and print them in Banana Accounting.

Banana.script

The object Banana.script is used to get informations about the running script.

Banana.SDecimal

The class Banana.SDecimal contains methods useful to do decimal math calculation.

Banana.Test

The class Banana.Test contains methods to run test units.

Banana.Ui

The class Banana.Ui contains predefined dialogs to interact with the user, and methods to load dialogs from .ui or .qml files.

Banana.Xml

The class Banana.Xml contains methods to parse and access Xml data.

Banana Methods

Banana.compareVersion(v1, v2)

Compare two version strings. Versions string are in the form of "x.y.w.z". Returns 0 if v1 and v2 are equal, -1 if v2 is later and 1 if v1 is later.

```
var requiredVersion = "8.0.5";
if (Banana.compareVersion &&
Banana.compareVersion(Banana.application.version, requiredVersion) >= 0)
   Banana.Ui.showInformation("Message", "More recent or equal than version "
+ requiredVersion);
else
   Banana.Ui.showInformation("Message", "Older than version " +
requiredVersion);
```

Banana.include(path)

The method Banana.include(path) include a javascript file evaluating it.
If an error occour, i.e. the file is not found or is not valid, the method throws an exception.

The path is relative to the current script being executed, if no protocol is specified. Otherwise depending on the protocol it can be relative to the main script's folder, the document's folder or the name of a document attacched to the current file.
- <relative_path_to_current_script>/<file_name>
- file:script/<relative_path_to_main_script>/<file_name>
- file:document/<relative_path_to_file>/<file_name>
- documents:<attachment_name>

Scripst included though the method Banana.include(path) can include other scripts trhough the method Banana.include(path), but not via the script's attibute @includejs. The method Banana.include(path) garantees that each distinct script is evaluated once, even if it is included more than one time from differents scripts. Path can contain ".." (parent folder), in the case the destination path is outside the main script's folder, the method will throw a security exception.

```
Banana.include("cashflowlib.js");
Banana.include("folder/cashflowlib.js");
```

Banana.Application

Banana.Application represent the interface to the program and can be accessed through Banana.application.

Properties

isBeta

Return true if the application is a beta version.

```
var isBeta = Banana.application.isBeta;
```

isExperimental

Return true if the application is a beta version.

```
var isExperimental = Banana.application.isExperimental;
```

serial

Return the serial of the application in the form of "80006-170428".

```
var serial = Banana.application.serial;
```

version

Return the version of the application in the form of "8.0.4".

```
var version = Banana.application.version;
```

locale

Return the locale of the application in the form of "language_country", where language is a lowercase, two-letter ISO 639 language code, and country is an uppercase, two- or three-letter ISO 3166 country code.

```
var locale = Banana.application.locale;
```

progressBar

Return an object of type ProgressBar used to give the user an indication of the progress of an operation and the ability to cancel it.

```
var progerssBar = Banana.application.progressBar;
```

Methods

addMessage(msg [, idMsg])

Add the message msg to the application. The message is showed in the pane "Messages", and in a

dialog if the application option "Show Messages" is turned on.

If idMsg is not empty, the help button calls an url with script's id and message's id (idMsg) as parameters.

```
Banana.application.addMessage("Hello World");
```

See also: Table.AddMessage, Row.AddMessage, Document.AddMessage.

clearMessages()

Clear all the messages showed in the pane "Messages".

```
Banana.application.clearMessages();
```

showMessages([show])

Enable or disable the notification of new messages through the message dialog.

```
Banana.application.showMessages(); // Next messages are showed to the user
through the message dialog.
Banana.application.showMessages(false); // Next messages will not pop up the
message dialog.
```

openDocument(ac2FilePath [, password] [, title])

Open the ac2 file located in filePath and return an Object of type Banana.Document or undefined if the file is not found. The path can be relative, in this case the base directory is the path of the current document.

If the path is empty or contains a "*" or a "?" an open file dialog is showed to the user, and the title is used in the caption of the file open dialog.

With this function you can also open ISO 20022 and MT940 files, in this case a cash book with the transactions of the file is returned.

```
var file1 = Banana.application.openDocument("*.*");
if (!file1)
   return;

var file2 = Banana.application.openDocument("c:/temp/accounting_2015.ac2");
if (!file2)
   return;
```

Banana.Application.ProgressBar

Banana.Application.ProgressBar is the interface to the program progress bar and can be accessed through Banana.application.progressBar. The progressBar object is used to give the user an indication of the progress of an operation and the possibility to interrupt the running process. The progress bar is showed in bottom left corner of the application windows.

Properties

showDetails

If true details are showed in the progess bar. If false only the text set by the first progressBar.start() call is showed.

```
progressBar.showDetails = false;
// Example without details: "VAT Report"
progressBar.showDetails = true;
// Example with details: "Vat Report; Sales; Row: 120"
```

Since Banana Accounting 9.0.3.

Methods

finish()

Notify that the operation has been completed and close the progress bar.

Returns **false** if the user canceled the operation, otherwise **true**.

```
progressBar.finish();
```

pause()

Notify that the operation has been paused, the cursor icon is set to the arrow cursor or poiting hand cursor. This is usually called before showing a dialog.

```
Banana.application.progressBar.pause();
var result = dialog.exec();
Banana.application.progressBar.resume();
```

resume()

Notify that the operation has been resumed, the cursor icon is set back to an hourglass or watch cursor . This is usually called after a dialog has been closed.

```
Banana.application.progressBar.pause();
var result = dialog.exec();
Banana.application.progressBar.resume();
```

start(maxSteps)

Start the progress indicator and define the number of steps this operation needs before being complete.

You can call several times this method to split the progress in main and sub steps. Every call of the method start() should be paired with a call of the method finish().

Returns **false** if the user canceled the operation, otherwise **true**.

```
// Example use of a progress bar
var progressBar = Banana.application.progressBar;
progressBar.start(10);
```

```
for (var i = 0; i < 10; i++) {
    ...
    if (!progressBar.step(1)) {
       return; // Operation canceled by the user
    }
}
progressBar.finish();
```

start(text, maxSteps)

Start the progress indicator and define the text to be showed in the progress bar and the number of steps this operation needs before being complete.

You can call several times this method to split the progress in main and sub steps. Every call of the method start() should be paired with a call of the method finish().

Returns **false** if the user canceled the operation, otherwise **true**.

```
// Example use of a progress bar
var progressBar = Banana.application.progressBar;
progressBar.start("Checking rows", 10);
for (var i = 0; i < 10; i++) {
    ...
    if (!progressBar.step("Row: " + i.toString())) {
       return; // Operation canceled by the user
    }
}
progressBar.finish();
```

Since Banana Accounting 9.0.3.

setText(text)

Set the text to show in the progress bar.

```
progressBar.setText("Checking vat rates");
```

Since Banana Accounting 9.0.3.

step([stepCount])

Advance the progress indicator of stepCount steps. If stepCount is not defined it advance of one step.

Returns **false** if the user canceled the operation, otherwise **true**.

```
progressBar.step(1);
```

step(text, [stepCount])

Advance the progress indicator of stepCount steps and set the text of the progress bar. If stepCount is not defined it advance of one step.

Returns **false** if the user canceled the operation, otherwise **true**.

```
progressBar.step("Row: " i.toString());
```

Since Banana Accounting 9.0.3.

Example multiple steps inside a block

```
// Two blocks of progress bar inside a progressBar

var progressBar = Banana.application.progressBar;

progressBar.start("Vat Report", 2);

// Block 1
progressBar.start("Sales", 10)
for (i=0;i < 10; i++) {
     progressBar.step("Row: " + i.toString());
}
progressBar.finish();

// Block 2
progressBar.start("Purchases", 10)
for (i=0;i < 10; i++) {
     progressBar.step("Row: " + i.toString());
}
progressBar.finish();

progressBar.finish();
```

Banana.Console

The Banana.console object is used to output messages to the Debug output panel or to the terminal (command prompt). The methods in this object are mainly used for Debugging purposes.

- To open the debug panel you have to enable the option "**Display Debug output panel**" under -> Program Options -> Tab Developer options
- The debug panel is located on the bottom of the main widow near the Info and Messages panels
- In the debug panel you can choose the level of messages to shows
- Debug messages can also be displayed in the terminal (command prompt) if the applicaiton is started from a terminal (command prompt)

Methods

console.critical(msg)

Display the msg in the debug panel as critical message.

```
Banana.console.critical("critical message");
```

console.debug(msg)

Display the msg in the debug panel ad a debug message.
Debug messages are show if in the panel the 'Debug level' or 'Info level' is selected.

```
Banana.console.debug("Debug message");
```

console.info(msg)

Display the msg in the debug panel as an info message.
Info messages are show in the panel only if the 'Info level' is selected.

```
Banana.console.info("Debug message");
```

console.log(msg)

Display the msg in the debug panel as an info message.
Log messages are show if in the panel only if 'Info level' is selected.

```
Banana.console.log("Debug message");
```

Since Banana 9.0.4

console.warn(msg)

Display the msg in the message inthe debug panel as a warning.

```
Banana.console.warn("Warning message");
```

Deprecated since Banana 9.0.4 use console.warning method instead.

console.warning(msg)

Display the msg in the debug panel as a warning.

```
Banana.console.warning("Warning message");
```

Since Banana 9.0.4

Banana.Converter

The class Banana.Converter is a collection of methods useful to convert various formats to and from data tables (array of array).

Methods

arrayToObject(headers, arrData, skipVoid)

Converts an array of array string to an array of objects

- **headers** is an array of strings that will become the properties of the objects.
- **arrData** is an array containing array of strings
- **skipVoid** if true skip void lines, if not present is set to false

```
    // read a CSV file
    var ppData = Banana.Converter.csvToArray(string, ',');
    // first line is header
    var headers = ppData[0];
    // remove first line
    ppData.splice(0, 1);
    // convert in array of objects
    var arraOfObjects = Banana.Converter.arrayToObject(fileData.headers,
ppData, true);
```

csvToArray(string [, separator, textdelim])

Convert a csv file (coma separated values) to an array of array.

The parameter string contains the text to convert. The parameter separator specify the character that separates the values, default is a comma ','. The parameter textDelim specify the delimiter character for text values, default is a double quote '"'.

Example:

```
var text = "1, 2, 3\n4, 5, 6\n7, 8, 9";
var table = Banana.Converter.csvToArray(text);
var value = table[0][1];
value == '2'; // true
```

flvToArray(string, fieldLengths)

Convert a flv file (fixed length values) to an array of array.

The parameter string contains the text to convert. The parameter fieldLengths is an array with the lengths of the fields.

Example:

```
//               6                  20       8
var text = "120608Phone               00002345";
var table = Banana.Converter.flvToArray(text, [6,20,8]);
var value = table[0][2];
value == '00002345'; // true
```

mt940ToTable(string)

Converts mt940 file to a table (array of array).

naturalCompare(a, b [, caseSensitive])

Compare two string so that the string "2" is considered less then "100" as it would be with normal string compare.
This function can be passed to array.sort function.

- **a** first value to compare
- **b** second value to compare
- return value is -1 if a < b, 1 if a > b and 0 if a == b

```
Banana.Converter.naturalCompare(a,b);
```

objectArrayToCsv(headers, objArray, [separator])

Converts an array of objects (with identical schemas) into a CSV table.

- **headers** An array of strings with the list of properties to export
- **objArray** An array of objects. Each object in the array must have the same property list.
- **separator** The CSV column delimiter. Defaults to a comma (,) if omitted.
- **return value** a string containing the CSV text.

```
var csvText = Banana.Converter.objectArrayToCsv(headers, objArray, ";");
```

stringToCamelCase(string)

Converts a text to camel case, where only the first letter every word is upper case.

```
Banana.Converter.stringToCamelCase("this is an example");
// returns "This Is An Example"
```

stringToLines(string)

Convert a text in an array of lines. The end line character can be '\n', \r' or a combination of both.

```
Banana.Converter.stringToLines("this is\nan\nexample");
//returns ["this is", "an", "example"]
```

stringToTitleCase(string)

Converts a text to title case, where only the first letter of the text is upper case.

```
Banana.Converter.stringToTitleCase("this is an example");
// returns "This is an example"
```

arrayToTsv(table [, defaultChar])

Converts a table (array of array) to a tsv file (tabulator separated values). If a string contains a tab it will be replaced with defaultChar or a space if defaultChar is undefined.

```
Banana.Converter.arrayToTsv(table);
```

arrayToCsv(table)

Converts a table (array of array) to a csv file (coma separated values). Doubles quotes in text are replaced by apos. Texts containing comas are inserted in doubles quotes.

```
Banana.Converter.arrayToCsv(table);
```

toDate(date[, time])

Convert a date and/or time to a javascript date object.

The parameter date is a string in the formats YYYYMMDD or YYYY-MM-DD.

The time parameter is a string in the fromats HHMM[SSZZZ] or HH:MM[:SS.ZZZ].

```
Banana.Converter.toDate("2015-12-31");
Banana.Converter.toDate("20151231");
```

toInternalDateFormat(date [, inputFormat])

Converts a date to the internal format: YYYY-MM-DD.

The parameter date can be a string or a date object.

The parameter inputFormat specifies the date input format, if it is not specified the local date format is used.

Example:

```
Banana.Converter.toInternalDateFormat("31-12-13", "dd-mm-yy");
// returns "2013-12-31"
Banana.Converter.toInternalDateFormat(new Date());
// return current date in format "yyyy-mm-dd"
```

toInternalNumberFormat(value [, decimalSeparator])

Converts a number to the internal format: 123456.78.The internal number format use the character '.' as decimal separator, and doesn't contain a group separator.

The parameter value can be a string or a number object.

The parameter decimalSeparator specifies the character used to separate the decimals, if it is not specified the local decimal separator is used.

Example:

```
Banana.Converter.toInternalNumberFormat("1200,25", ",");
// returns "1200.25"
```

toInternalTimeFormat(string)

Converts a time to the internal format: HH:MM:SS.ZZZ.

```
Banana.Converter.toInternalTimeFormat("11:24");
// returns "11:24:00"
```

toLocaleDateFormat(date [, format])

Converts a date to the local format.

The parameter date can be a string or a date object.

The parameter format specifies the date ouptut format, if it is not specified the local date format is used.

```
Banana.Converter.toLocaleDateFormat("2014-02-24")
// returns "24.02.2014"
```

toLocaleNumberFormat(value [, decimals = 2, convZero = true])

Converts a number to the local format.

The parameter value can be a string or a number object.

The parameter decimals set the number of decimals.

The parameter convZero set the format returned for zero values. If false the method returns an empty string, if true it returns the zero value as string.

Example:

```
Banana.Converter.toLocaleNumberFormat("1200.25")
// returns "1'200,25"
```

toLocalePeriodFormat(startDate , endDate [, format])

Converts a period defined by startDate and endDate to a readable string.

The parameter startDate specifies the start date of the preriod, can be a date object or a string.
The parameter endDate specifies the end date of the preriod, can be a date object or a string.
The parameter format can be empty or one of the following strings: 'short', 'long'. Default is 'long'.

Return the period as a readable string in the current language of the application.

```
Banana.Converter.toLocalePeriodFormat("2017-01-01", "2017-01-31"); // returns
"January '17"
Banana.Converter.toLocalePeriodFormat("2017-01-01", "2017-01-31", "short");
// returns "Jan '17"
Banana.Converter.toLocalePeriodFormat("2017-01-01", "2017-01-31", "long"); //
returns "January '17"
```

toLocaleTimeFormat(string [, format])

Converts a time to the local format.

The parameter format specifies the time ouptut format, if it is not specified the local time format is used.

```
Banana.Converter.toLocaleTimeFormat("11:24:42");
// returns "11:24"
```

Banana.Document (Accounting)

Methods for accounting's files

Banana is build with object oriented techologies. Each file is a Banana.document class.

Accunting files are element of class that derive from the Banana.document.
For each file there are:

- Methods, that apply to all documents, see Banana.Document (Base),
- Method specific to the Accounting class.

The following explanations relate to the accounting functions.

Date functions

endPeriod([period])

Return the end date in the form of 'YYYY-MM-DD'.

The endDate and startDate functions are used to retrieve the date of the accounting, so that you can create scripts that works on file of different years.

```
var dateEnd = Banana.document.endPeriod();
var dateStartFebruary = Banana.document.endPeriod('2M');
```

- Period:
  - If period is not present the return value is the end date of the accounting.
  - The period is added the starting account date, and than is returned the last date of the period..
  - Period (for example 2M = 2 months) is a number followed by one of the following charachters
    - D for days
    - M for months
    - Q for quarters
    - S for semesters
    - Y for years
  - Assuming that the Start accounting date is 2015-01-01
    - 1M return 2015-01-02
    - 2M return 2015-02-28
    - 2Q return 2015-06-30
    - 2S return 2015-12-31
    - 2Y return 2016-12-31

startPeriod ([period])

Return the end date in the form of 'YYYY-MM-DD'.

The endPeriod and startPeriod functions are used to retrieve the date of the accounting, so that you can create scripts that works on file of different years.

```
var dateStart = Banana.document.endPeriod();
var dateStart3Q = Banana.document.endPeriod('3Q');
```

- Period:
  If period is not present return the start date.
  - Period (for example 2M = 2 months) is a number followed by one of the following charachters
    - D is for Days
    - M for Months

        - Q for Quorters
        - S for Semester
        - Y for year
    - With 1 the starting date of the accounting is returned.
    - Assuming that the Start accounting date is 2015-01-01
        - 1M return 2015-01-01
        - 2M return 2015-02-01
        - 2Q return 2015-04-01
        - 2S return 2015-07-01
        - 2Y return 2016-01-01

previousYear([nrYears])

Return the previous year as a Banana.Document object. If the previous year is not defined or it is not foud it return null.

- **nrYears** is the number of years to go back, default is one.

```
var previousYearDoc = Banana.document.previousYear();
var previousTwoYearDoc = Banana.document.previousYear(2);
```

Current accounting functions

The functions that start with "current" retrieve values calculated based on the actual accounting data, comprising:

- Opening amounts (Table accounts)
- Transactions entered in the Transactions table

currentBalance(account [, startDate, endDate, function(rowObj, rowNr, table)])

Sum the amounts of opening, debit, credit, total and balance calculated based on the opening and all transactions for this accounts / group.

The calculations are perfermed by traversing by creating a journal (see journal() function) with all the transactions , and selecting the transctions with the parameters specified.
The computation is usually very fast. But if you have a file with many transactions especially the first query could take some time.

```
var currentBal = Banana.document.currentBalance('1000','','');
var openingBalance = currentBal.opening;
var endBalance = currentBal.balance;
```

- **Return value**
  Is an object that has
    - **opening** the amount at the begining of the period (all transactions before)
    - **debit** the amount of debit transactions for the period
    - **credit** the amount of credit transactions for the period
    - **total** the difference between debit-credit for the period
    - **balance** opening + debit-credit for the period
    - **amount** it the "normalized" amount based on the bclass of the account or group.
      If there are multiple accounts or groups, takes the first BClass of the first.
        - for BClass 1 or 2 it return the balance (value at a specific instant).

        - for BClass 3 or 4 it return the total (value for the duration).
        - For BClass 2 and 4 the amount is inverted.
    - **openingCurrency** the amount at the begining of the period in the account currency
    - **debitCurrency** the amount of debit transactions for the period in the account currency
    - **creditCurrency** the amount of credit transactions for the period in the account currency
    - **totalCurrency** the difference between debit-credit for the period in the account currency
    - **balanceCurrency** opening + debit-credit for the period in the account currency
    - **rowCount** the number of lines that have bben found and used for this computation
    - **bclass** (double entry accounting only) is the bclass of the account or group used to express the amount.
      The bclass is the value entered in the columns bclass.
      It is taken in consideration the first account or group specified. If for examble you query two account that first that has bclass 2 and the second that has bclass 1. The bclass would be 2.
      The bclass is assigned by following this steps. :
        - The bclass of the specified account or group.
        - The blcass of the partent group, for the same section.
        - The blcass of the section.
- **Account**
    - can be an account id, a cost center, a segment or a group.
    - can be a cobination of account and segments, separeted by the semicolon ":"
      In this case it returns all the transactions that have the indicated account and segments
        - 1000:A1:B1
    - can be differents accounts and multiple segments separated by the "|"
      In this case it include all transactions that have the one of the specified accounts and one of the specified segments
        - 1000|1001
        - 1000|1001:A1:B1
        - 1000|1001:A1|A2:B1
        - can be a wildCardMatching
          Wildcards can be used for accounts, segments, Groups or BClass and in combination
            - ? Matches any single character.
            - * Matches zero or more of any characters
            - [...] Set of charachtes
            - "100?" match "1001, 1002, 1003, 100A, ...)
            - "100*" Matches all accounts starting with 100
            - "100*|200*:A?" Matches all accounts starting with 100 or 200 and with segments with A and of two charachters.
            - "[1234]000" Matches "1000 2000 3000 4000"
    - Can be a group or a BClass.
      It include all the transactions where the account used belong to a specified Group o r BClass.
      It is also possible to use wildcards.
      The program first create a list of accounts and then use the account list.
      Do non mix mix groups relative to normal accounts, with groups relative to cost center or segments. Calculation could provide unexpected results.
        - BClass (for the double entry accounting only)
            - BClass=1
            - BClass=1|2
        - Gr for groups that are in Accounts table.

            - Gr=100
            - Gr=10*
            - Gr=100|101|102
        - GrC for group that are in the Category table of the income and expenses accounting type.
            - GrC=300
            - GrC=300|301
    - Contra Accounts or other fields selection
      Transactions are included only if they have also a value corresponding
      After the "&&" you can insert a field name of the table journal.
        - 1000&&JContraAccount=2000 return all transctions of the account 1000 that have a contra account 2000.
          As per accounts you can specify multiple contra accounts, BClass=, Gr= with also wildcards.
        - 1000&&JCC1=P1|P2 will use only transactions on account 1000 and that also have the CC1=.P1 or .P2
- **StartDate**
    - is a string in form 'YYYY-MM-DD' or a date object.
    - If startDate is empty the accounting start date is taken.
- **End date**:
    - is a string in form 'YYYY-MM-DD' or a date object.
    - If endDate is empty the accounting end date is taken.
- **function(rowObj, rowNr, table)**
  This fuction will be called for each row of the selected account.
  The function should return true if you want this row to be included in the calculation.

```
function exec( string) {
    // We retrive the total sales (account 4000) only for the cost center P1
    var balanceData = Banana.document.currentBalance('4000','', '',
onlyCostCenter);
    // sales is a revenue so is negative and we invert the value
    var salesCC1 = -balanceData.total;
    // display the information
    Banana.Ui.showText("Sales of Project P1: " + salesCC1);
}

// this function return true only if the row has the cost center code  "P1"
function onlyCostCenter( row, rowNr, table){
   if(row.value('JCC1') === 'P1') {
      return true;
   }
   return false;
}
```

Examples

```
Banana.document.currentBalance("1000")              // Account 1000
Banana.document.currentBalance("1000|1010")         // Account 1000 or  1010
Banana.document.currentBalance("10*|20*")           // All account that start
with 10 or with 20
Banana.document.currentBalance("Gr=10")             // Group 10
Banana.document.currentBalance("Gr=10| Gr=20")      // Group 10 or  29
```

```
Banana.document.currentBalance(".P1")              // Cost center .P1
Banana.document.currentBalance(";C01|;C02")        // Cost center ;C01 and 
C2
Banana.document.currentBalance(":S1|S2")           // Segment :S1  and :S2
Banana.document.currentBalance("1000:S1:T1")       // Account 1000 with 
segment :S1 or ::T1
Banana.document.currentBalance("1000:{}")          // Account 1000 with 
segment not assigned
Banana.document.currentBalance("1000:S1|S2:T1|T2")  // Account 1000 with 
segment :S1 or ::S2 and ::T1
                                                   //   or ::T
Banana.document.currentBalance("1000&&JCC1=P1")     // Account 1000 and cost 
center .P1
```

currentBalances(account, frequency [, startDate, endDate, function(rowObj, rowNr, table)])

It return the time series balance for the specified periods. It used for chart rendering, with one command you can have the monthly data for an account.

Sum the amounts of opening, debit, credit, total and balance for all transactions for this accounts and returns the values according to the indicated frequency indicated.

The calculations are perfermed by traversing by creating a journal (see journal() function) with all the transactions , and selecting the transctions with the parameters specified.
The computation is usually very fast. But if you have a file with many transactions especially the first query could take some time.

```
var currentBalances = Banana.document.currentBalances('1000', 'M');
var openingBalance = currentBalances[0].opening;
var endBalance = currentBalances[0].balance;
```

- **Return value**
  Return an array of objects that have
  - **opening** the amount at the begining of the period (all transactions before)
  - **debit** the amount of debit transactions for the period
  - **credit** the amount of credit transactions for the period
  - **total** the difference between debit-credit for the period
  - **balance** opening + debit-credit for the period
  - **amount** it the "normalized" amount based on the bclass of the account or group.
    If there are multiple accounts or groups, takes the first BClass of the first.
    - for BClass 1 or 2 it return the balance (value at a specific instant).
    - for BClass 3 or 4 it return the total (value for the duration).
    - For BClass 2 and 4 the amount is inverted.
  - **openingCurrency** the amount at the begining of the period in the account currency
  - **debitCurrency** the amount of debit transactions for the period in the account currency
  - **creditCurrency** the amount of credit transactions for the period in the account currency
  - **totalCurrency** the difference between debit-credit for the period in the account currency
  - **balanceCurrency** opening + debit-credit for the period in the account currency
  - **rowCount** the number of lines that have bben found and used for this computation
  - **bclass** (double entry accounting only) is the bclass of the account or group used to express the amount.

The bclass is the value entered in the columns bclass.
It is taken in consideration the first account or group specified. If for examble you query two account that first that has bclass 2 and the second that has bclass 1. The bclass would be 2.
The bclass is assigned by following this steps. :
        - The bclass of the specified account or group.
        - The blcass of the partent group, for the same section.
        - The blcass of the section.
    - **startDate** period's start date
    - **endDate** period's end date
- **Account**
    - can be an account id, a cost center or a segment.
    - can be a cobination of account and segments, separeted by the semicolon ":"
      In this case it returns all the transactions that have the indicated account and segments
        - 1000:A1:B1
    - can be differents accounts and multiple segments separated by the "|"
      In this case it include all transactions that have the one of the specified accounts and one of the specified segments
        - 1000|1001
        - 1000|1001:A1:B1
        - 1000|1001:A1|A2:B1
        - can be a wildCardMatching
          Wildcards can be used for accounts, segments, Groups or BClass and in combination
            - ? Matches any single character.
            - * Matches zero or more of any characters
            - [...] Set of charachtes
            - "100?" match "1001, 1002, 1003, 100A, ...)
            - "100*" Matches all accounts starting with 100
            - "100*|200*:A?" Matches all accounts starting with 100 or 200 and with segments with A and of two charachters.
            - "[1234]000" Matches "1000 2000 3000 4000"
    - Can be a group or a BClass.
      It include all the transactions where the account used belong to a specified Group o r BClass.
      It is also possible to use wildcards.
      The program first create a list of accounts and then use the account list.
      Do non mix mix groups relative to normal accounts, with groups relative to cost center or segments. Calculation could provide unexpected results.
        - BClass (for the double entry accounting only)
            - BClass=1
            - BClass=1|2
        - Gr for groups that are in Accounts table.
            - Gr=100
            - Gr=10*
            - Gr=100|101|102
        - GrC for group that are in the Category table of the income and expenses accounting type.
            - GrC=300
            - GrC=300|301
    - Contra Accounts or other fields selection

Transactions are included only if they have also a value corresponding
After the "&&" you can insert a field name of the table journal.
    - 1000&&JContraAccount=2000 return all transctions of the account 1000 that have a contra account 2000.
      As per accounts you can specify multiple contra accounts, BClass=, Gr= with also wildcards.
    - 1000&&JCC1=P1|P2 will use only transactions on account 1000 and that also have the CC1=.P1 or .P2

- **Frequency**
  - Specifiy the frequency to be returned, is one of the following charachters
    - D for daily
    - W for weekly
    - M for montly
    - Q for quarterly
    - S for semeterly
    - Y for yearly
- **StartDate**
  - is a string in form 'YYYY-MM-DD' or a date object.
  - If startDate is empty the accounting start date is taken.
- **End date**:
  - is a string in form 'YYYY-MM-DD' or a date object.
  - If endDate is empty the accounting end date is taken.
- **function(rowObj, rowNr, table)**
  This fuction will be called for each row of the selected account.
  The function should return true if you want this row to be included in the calculation.

```
function exec( string) {
    // We retrive the montly total sales (account 4000) only for the cost
center P1
    var balanceData = Banana.document.currentBalances('4000', 'M', '', '',
onlyCostCenter);
    // sales is a revenue so is negative and we invert the value
    var salesCC1 = -balanceData[0].total;
    // display the information
    Banana.Ui.showText("Sales of Project P1: " + salesCC1);
}

// this function return true only if the row has the cost center code  "P1"
function onlyCostCenter( row, rowNr, table){
   if(row.value('JCC1') === 'P1') {
      return true;
   }
   return false;
}
```

Examples

```
Banana.document.currentBalances("1000", 'M')         // Montly values for
account 1000 and for the accounting start and end period
// See also the examples for the function currentBalance
```

currentCard(account [, startDate, endDate, function(rowObj, rowNr, table)])

Return for the given account and period a Table object with the transactions for this account.

Row are sorted by JDate

parameters:

- account can be any accounts, cost center or segment as specifiend in currentBalance.
- startDate any date or symbol as specifiend in currentBalance.
- endDate any date or symbol as specifiend in currentBalance.

Return columns the same as for the Journal() function.

```
var transactions =
Banana.document.currentCard('1000','2015-01-01','2015-12-31');
```

currentInterest(account, interestRate, [startDate, endDate, , function(rowObj, rowNr, table)])

Return the calculated interest on the specified account.
Interest is calculate on the effective number o days for 365 days in the years.

- **account** is the account or group (same as in the function currentBalance)
- interestRate. In percentage "5", "3.25". Decimal separator must be a "."
  - If positive it calculate the interst on the debit amount.
  - If negative it calculate the interest on the credit amounts.
- startDate, endDate, function see the currentBalance description.
  If no parameters are specified it calculate the interest for the whole year.

```
// calculate the interest debit for the whole period
var interestDebit = Banana.document.currentInterest('1000','5.75');

// calculate the interest credit for the whole period
var interestDebit = Banana.document.currentInterest('1000','-4.75');
```

Budget Functions

This functions are similar to the "current" functions, but they work on the budget data. They consider:

- The opening amount (table Accounts and Categories)
- Transactions entered in the Budget table.

For detailed information check the documentation on the equivalent "current" functions.

When using the API in the column Formula of the Table Budget a special API is available.

budgetBalance(account [, startDate, endDate, function(rowObj, rowNr, table)])

Sum the amounts of opening, debit, credit, total and balance for all budget transactions for this accounts .

```
var budget = Banana.document.budgetBalance('1000');
```

It works the same as the function currentBalance, but for the budgeting data.

budgetBalances(account, frequency [, startDate, endDate, function(rowObj, rowNr, table)])

Time seies function. Sum the amounts of opening, debit, credit, total and balance for all budget transactions for this accounts and returns the values according to the indicated frequency indicated.

```
var budgets = Banana.document.budgetBalances('1000', 'M');
```

It works the same as the function currentBalances, but for the budgeting data.

budgetCard(account [, startDate, endDate, function(rowObj, rowNr, table)])

Return for the given account and period a Table object with the budget account card.

```
var card = Banana.document.budgetCard('1000');
```

It works the same as the function currentCard, but for the budgeting data.

budgetExchangeDifference(account, [date, exchangeRate])

Return the unrealized exchange rate Profit or Loss fo the account at the specified date.

- account must be a valid account number not in base currency
- date
  - a date that is used to calculate the balance
  - if empty calculate up to the end of the period
- exchangeRate
  - if empty use the historic exchange rate for the specified date or the current if not a valid exchange rate for the date are found.
  - if "current" use the current exchange
  - if number for example "1.95" use the specified exchange rate.
- Return value
  - Positive number (debit) are exchange rate Loss.
  - Negative number (credit) are exchange rate Profit.
  - empty if no difference or if the account has not been found or not a multicurrency accounting file.

```
// unrealized exchange rate profit or loss for the account 1000
// last balance and current exchange rate
var exchangeRateDifference =
Banana.document.budgetExchangeRateDifference('1000');

// at the end of Semptember and hystoric exchange rate
var exchangeRateDifference =
Banana.document.budgetExchangeRateDifference('1000', "2017-09-31");

// at the end of Semptember and current exchange rate
var exchangeRateDifference =
Banana.document.budgetExchangeRateDifference('1000', "2017-09-31",
"current");

// at the end of Semptember and specified exchange rate
```

```
var exchangeRateDifference =
Banana.document.budgetExchangeRateDifference('1000', "2017-09-31", "1.65");
```

budgetInterest(account, interestRate, [startDate, endDate, function(rowObj, rowNr, table)])

Return the calculated interest for the budget transactions.

It works the same as the function currentInterest, but for the budgeting data.

```
// calculate the interest debit for the whole period
var interestDebit = Banana.document.budgetInterest('1000','5.75');

// calculate the interest credit for the whole period
var interestDebit = Banana.document.budgetInterest('1000','-4.75');
```

Projections functions

This function are a mix of actual and budget data. It has a parameter projectionStartDate that define up to when to use actual data and budget data.

The value of a projection is calculated as following:

- It use the actual data up to the day before the projectionStartDate.
- From the projectionStartDate it use the budgeting data.

Assume you have prepared the budget for the year and you have entered accounting data up to the end of March (2015-03-31).
With the projectionBalance function you can have the projected balance up to the end of year, comprised from the actual data up end of March and the budgeting data starting form 1. April.
In this case the projectionStartDate should be "2015-04-01".

projectionBalance(account, projectionStartDate [, startDate, endDate, function(rowObj, rowNr, table)])

Same as currentBalance but use the budget data starting from the projectionStartDate.

This functions calculate a projection of the end of year result (or specified period) combining the current data and the budget data for the period non yet booked.

if projectionStartDate is empty the result will be the same as currentBalance.

If you have already booked the 1. semester and would like to have a projection up to the end of the year

```
// We have booked the 1. semester and would like to have
// a projection up to the end of the yer
var cashProjection = Banana.document.projectionBalance('1000','2015-07-01');
var cashEnd = projection.balance;
var salesProjection =
Banana.document.projectionBalance('3000','2015-07-01').total;
var salesForYear = -salesProjection.total;
```

projectionBalances(account, projectionStartDate, frequency [, startDate, endDate, function(rowObj, rowNr, table)])

Same as currentBalances but use the budget data starting from the projectionStartDate.

projectionCard(account, projectionStartDate [, startDate, endDate, function(rowObj, rowNr, table)])

Same as currentCard but use the budget data starting from the projectionStartDate.

If projectionStart date is empty result will be the same s currentCard.

```
var transactions = Banana.document.projectionCard('1000','2015-01-01','','');
```

Descriptions and Reporting functions

accountDescription(account [,column])

Return the Description of the specified account.

- Account can be an account or a Group (Gr=)
- Column can be an alternative column name to retrieve.

```
var descriptionAccount = Banana.document.accountDescription('1000');
var descriptionGroup = Banana.document.accountDescription('Gr=10');
var gr = Banana.document.accountDescription('1000','Gr');
```

accountsReport([startDate, endDate])

Return the account report for the specified period. Start and end date can be a string in form 'YYYY-MM-DD' or a date object.

```
var report = Banana.document.accountsReport();
var report = Banana.document.accountsReport('2017-01-01', '2017-03-31');
```

Journal

This function retrieve an array of transactions, with one line for each account.
This is the accounting information that is used for all accounting related calculation.

1. The software first prepare the Journal with one line for each account amount mouvement.
2. It use the Journal data to do calculation and reporting.

One line for each account mouvement

For each account mouvement there will be a corresponding line in the journal.

- Opening amounts.
  One line is generated for each account with an opening amount.
- Double entry transactions or Income and expense accounts.
  - Transactions with Debit and Credit accounts.
    In the Journal there will be two lines, one for each Credit and Debit account, with the relative amount.

    - Transactions with Debit and Credit accounts plus VAT.
      In the Journal there will be have three lines, one for each Credit, Debit and VAT account, with the relative amount.
    - Transactions with Cost center.
      For each cost center used CC1, CC2, CC3 a journal line is created with the Cost center account and relative amount.

Current and Budget data

Each Journal line indicate where is the orgin of the data. It can be the Current or Budget.
In case you have both transactions and budget data, for each account you will have Journla line for Current and Budget.

journal([originType = ORIGINTYPE_NONE, int accountType = ACCOUNTTYPE_NONE])

Return for the given parameters a Table object with all the amount registered on the accounts.
The journal contain a row for each account used.

- **originType** specify the row to be filtered for
  Can be on of
    - ORIGINTYPE_NONE no filter is applied and all rows are returned (current and budget)
    - ORIGINTYPE_CURRENT only the normal transactions are returned
    - ORIGINTYPE_BUDGET only the budget transactions are returned
- **accountType** specify the row to be filtered for
    - ACCOUNTTYPE_NONE no filter is appplyied and all rows are returned
    - ACCOUNTTYPE_NORMAL only rows for normal accounts are returned
    - ACCOUNTTYPE_CC1 only rows for Cost Center 1 are returned
    - ACCOUNTTYPE_CC2 only rows for Cost Center 2 are returned
    - ACCOUNTTYPE_CC3 only rows for Cost Center 1 are returned
    - ACCOUNTTYPE_CC Cost Center rows are returned same as using (ACCOUNTTYPE_CC1 | ACCOUNTTYPE_CC2 | ACCOUNTTYPE_CC3)

```
// get all transactions for normal accounts
var journal = Banana.document.journal(Banana.document.ORIGINTYPE_CURRENT,
Banana.document.ACCOUNTTYPE_NORMAL );
```

For each account used in the transaction table (AccountDebit, AccountCredit, CC1, CC2, CC3) the program generate a Journal row with the JAccount column set with the specific account.
For a double entry account transaction that use AccountDebit, AccountCredit, AccountVat, CC1, CC2, CC3 the Journal will contain 6 rows. If the transaction has only AccountDebit and AccountCredit only 2 rows will be generated.

The column JAmount contains the exact amount registered on the specific account.

The returned table has all the columns of the transaction's table plus the following columns.

The return columns are:

- Origin Information
    - **JOriginType** as defined above
        - ORIGINTYPE_CURRENT
        - ORIGINTYPE_BUDGET

- **JOriginFile** the file name where the transaction orginate.
  - **JTableOrigin** the source table.
  - **JRowOrigin** the row number in the transaction's table (rows begin from 0).
  - **JRepeatNumber** the progressive number of the repetition of budget transactions.
- **JOperationType** on of
  - OPERATIONTYPE_NONE = 0
  - OPERATIONTYPE_OPENING = 1
    The row is generated from the opening balance
  - OPERATIONTYPE_CARRYFORWARD = 2
    The row is used from the account card and is the balance of the account at this moment.
  - OPERATIONTYPE_TRANSACTION = 3
    The row is generated from the Transactions table if it is ORIGINTYPE_CURRENT
    or from the budget table if the row is ORIGINTYPE_BUDGET

  - OPERATIONTYPE_INVOICESETTLEMENT = 21

- **JDate** the date of the transction.
- **JDescription** the transaction's description.
- **JAccount** the account for this line.
  There is one row for each account (AccountDebit, AccountCredit, AccountVat, CC1, CC2, CC3).
- **JAccountDescription** the Description for this account.
- **JAccountClass** the BClass number for this account.
- **JAccountGr** the Gr for this account.
- **JAccountGrDescription** the Gr for this account.
- **JAccountGrPath** the whole Gr path.
- **JAccountCurrency** the currency of thi account.
- **JAccountType** as defined above (ACCOUNTTYPE_NORMAL, ACCOUNTTYPE_CC1, ...)
- **JAmount** the amount in basic currency registered on the account (positive is debit, negative is credit).
- **JAmountAccountCurrency** the amount in the account currency (positive i debit, regative is credit).
- **JTransactionCurrency** the transaction's currency.
- **JAmountTransactionCurrency** the amount in transaction's currency.
  For account with currency not in tranasctions currency the exchange rate of the transaction is used.
- **JTransactionCurrencyConversionRate** is the conversion rate to obtain amounts in transaction's currency.
  Multiply the transcation's amount in basic currency with the JTransactionCurrencyConversionRate and you will have the amount converted in transaction's currency.
  The conversion rate has 12 significant decimal so only by very large conversion should be have conversion differences.
- **JVatIsVatOperation** true if this row has a Vat code.
- **JVatCodeWithoutSign** the Vat code without the evetually preceeding '-'. For example "-V10" becomomes "V10".
- **JVatCodeDescription** the Description for this Vat code.
- **JVatCodeWithMinus** true if the Vat code is preceeded by "-".
- **JVatCodeNegative** true if the Vat amount is negative (deductible).
- **JVatTaxable** the amount VatTaxable with the sign that follow the JVatCodeNegative
- **VatTwinAccount** the account where the net amount (without VAT) is being registered .
  In case of a transaction where the Gross amount is CHF 1100, then the VAT is CHF 100 and the

net amount is CHF 1000. The VatTwin account will be the account where the CHF 1000 is being registered.
We use the name Twin for the fact that the VatTwinAccount follows the sign of the VatAccount.
If the Vat amount is registered in debit, the VatTwinAccount will be the AccountDebit.
If the Vat amount is registered in credit, the VatTwinAccount will be the AccountCredit.

- **JContraAccount** the contra account.
  The contra account is deducted based on the other accounts and the sequence in the transactions table.
- **JContraAccountType** one of the following value:
  - CONTRAACCOUNTTYPE_NONE for no contra account
  - CONTRAACCOUNTTYPE_DIRECT when there is on the same line credit and debit accounts.
  - CONTRAACCOUNTTYPE_MULTIPLEFIRST the first line of a transaction on more accounts. The first transactions after a line with debit and credit accounts or with a different date.
  - CONTRAACCOUNTTYPE_MULTIPLEFOLLOW the second or following line of a MULTIPLEFIRST with the same date.
  - CONTRAACCOUNTTYPE_VAT the line for the Vat Account
- **JContraAccountGroup** the line number corresponding to the row number of the CONTRAACCOUNTTYPE_MULTIPLEFIRST
- **JCC1** the CC1 without the preceeding sign
- **JCC2** the CC2 without the preceeding sign
- **JCC3** the CC3 without the preceeding sign
- **JSegment1 .. JSegment10** the segment relative to the account
- **JDebitAmount** the amount debit in basic currency
- **JCreditAmount** the amount credit in basic currency
- **JDebitAmountAccountCurrency** the amount debit in account currency
- **JCreditAmountAccountCurrency** the amount credit in account currency
- **JBalance** the balance amount (for account card) in basic currency
- **JBalanceAccountCurrency** the balance amount (for account card) in account currency

journalCustomersSuppliers([originType = ORIGINTYPE_NONE, int accountType = ACCOUNTTYPE_NONE])

Same as journal with additional settlements rows for closing invoices and additional columns:

- **JInvoiceDocType**: specifies the type of document (see column DocType)
- **JInvoiceAccountId**: customer account id from table accounts
- **JInvoiceCurrency**: the currency of the invoice, same as customer account currency from table accounts
- **JInvoiceStatus**: paidInvoice (the invoice was offset with the payment that refers to the same document), paidOpening (the invoice was offset with the opening balance of the customer account)
- **JInvoiceDueDate**: invoice expiration date
- JInvoiceDaysPastDue
- JInvoiceLastReminder
- JInvoiceLastReminderDate
- JInvoiceIssueDate
- JInvoiceExpectedDate
- JInvoicePaymentDate
- JInvoiceDuePeriod
- JInvoiceRowCustomer (1=Customer, 2=Supplier)

Invoices

invoicesCustomers()

Return a table with the customers invoices from the transaction table. A customer group must be defined and invoices must be numbered using the column DocInvoice.

See Invoice Json Object.

invoicesSuppliers()

Return a table with the suppliers invoices from the transaction table. A supplier group must be defined and invoices must be numbered using the column DocInvoice.

See Invoice Json Object.

Exchange rate functions

exchangeRate(currency, [date])

Return the exchange rate that convert the amount in currency in basic currency as object with the properties 'date' and 'exchangeRate'. Return null if no exchange rate is found.

The exchange rate is retrieved from the Currency table, already considering the multiplier.

- If no date is specified the exchange rate without date is used.
- If a date is specified it retrieve the exchange rate with the date minor or equal the specified date.

Vat functions

vatBudgetBalance(vatCode[, startDate, endDate, function(rowObj, rowNr, table)])

Sum the vat amounts for the specified vat code and period, using the Budget data.

```
var vatTotal = Banana.document.vatBudgetBalance('V15');
```

vatBudgetBalances(vatCode, frequency, [, startDate, endDate, function(rowObj, rowNr, table)])

Sum the vat amounts for the specified vat code and period, using the Budget data and returns the values according to the indicated frequency indicated.

```
var vatTotal = Banana.document.vatBudgetBalances('V15', 'Q');
```

vatCurrentCard(vatCode[, startDate, endDate, function(rowObj, rowNr, table)])

Retrieve the transactions relative to the specified VatCode.

```
var vatTransactions = Banana.document.vatCurrentCard('V15');
```

vatCurrentBalance(vatCode[, startDate, endDate, function(rowObj, rowNr, table)])

Sum the vat amounts for the specified vat code and period.
For more info see :

- explanations of the function currentBalance.
- Examples files are available on github/General/CaseStudies.
- Solutions making use of the VAT api.
  - Vat report Germany 2016
  - Vat Saudi Arabia

Example:

```
var currentVat = Banana.document.vatCurrentBalance('V15','','');
var vatTaxable = currentVat.vatTaxable;
var vatPosted = currentVat.vatPosted;
```

- **Return value**:
  Is an object that has
  - **vatTaxable** the amount of the taxable column
    (the sign is the same as the vatAmount)
  - **vatAmount** the amount of vat
  - **vatNotDeductible** the amount not deductible
  - **vatPosted** VatAmount - VatNotDeductible
  - **rowCount** the number of lines that have bben found and used for this computation
- **VatCode**
  One or more VatCode defined in the tabel Vat Codes.
  Multiple vat code can be separated by "|" for example "V10|V20", or you can use vildcard "V*".

vatCurrentBalances(vatCode, frequency [, startDate, endDate, function(rowObj, rowNr, table)])

Sum the vat amounts for the specified vat code and period and returns the values according to the indicated frequency indicated.
For more info see :

- explanations of the function currentBalances.
- Examples files are available on github/General/CaseStudies.
- Solutions making use of the VAT api.
  - Vat report Germany 2016
  - Vat Saudi Arabia

Example:

```
var currentVat = Banana.document.vatCurrentBalances('V15', 'Q');
var vatTaxable = currentVat[0].vatTaxable;
var vatPosted = currentVat[0].vatPosted;
```

- **Return value**:
  Is an object that has
  - **vatTaxable** the amount of the taxable column
    (the sign is the same as the vatAmount)
  - **vatAmount** the amount of vat
  - **vatNotDeductible** the amount not deductible
  - **vatPosted** VatAmount - VatNotDeductible

    - **rowCount** the number of lines that have bben found and used for this computation
- **VatCode**
  One or more VatCode defined in the tabel Vat Codes.
  Multiple vat code can be separated by "|" for example "V10|V20", or you can use vildcard "V*".
- **Frequency**
    - Specifiy the frequency to be returned, is one of the following charachters
        - D for daily
        - W for weekly
        - M for montly
        - Q for quarterly
        - S for semeterly
        - Y for yearly

vatProjectionBalance(vatCode, projectionStartDate, [, startDate, endDate, function(rowObj, rowNr, table)])

Same as vatCurrenBalance but use the budget data starting from the projectionStartDate.

```
var projectionVat = Banana.document.vatProjectionBalance('V15','','');
var vatTaxable = projectionVat.vatTaxable;
var vatPosted = projectionVat.vatPosted;
```

vatProjectionBalances(vatCode, projectionStartDate, frequency, [, startDate, endDate, function(rowObj, rowNr, table)])

Same as vatCurrenBalances but use the budget data starting from the projectionStartDate.

```
var projectionVat = Banana.document.vatProjectionBalances('V15',
'2017-03-01', 'Q');
var vatTaxable = projectionVat[0].vatTaxable;
var vatPosted = projectionVat[0].vatPosted;
```

vatProjectiontCard(vatCode, projectionStartDate, [, startDate, endDate, function(rowObj, rowNr, table)])

Same as vatCurrentCard but use the budget data starting from the projectionStartDate.

```
var vatTransactions =
Banana.document.vatProjectiontCard('V15','2015-01-01','','');
```

vatReport([startDate, endDate])

Return the vat report for the specified period.

Start and end date are strings in form 'YYYY-MM-DD' or a date object. If startDate is empty the accounting start date is taken. If endDate is empty the accounting end date is taken.

```
var vatReport = Banana.document.vatReport('','');
```

Banana.Document (Base)

Banana.Document is the interface to a document in Banana Accounting. The current opened document can be accessed through the property Banana.document. A document can be also opened with the method Banana.application.openDocument.

Properties

cursor

Return a Cursor object with the current position of the cursor and the range of the selected rows.

```
var currentCursor = Banana.document.cursor;
```

locale

Return the locale of the document in the form of "language_country", where language is a lowercase, two-letter ISO 639 language code, and country is an uppercase, two- or three-letter ISO 3166 country code.

```
var locale = Banana.document.locale;
```

rounding

Return the rounding context of the current file that can be used with the SDecimal math functions.

```
var rounding = Banana.document.rounding;
```

tableNames

Return an array with the xml names of the tables in the document.

```
var tableNames = Banana.document.tableNames;
```

Methods

addAttachment(name, content)

Add an attachment to the document.

The param *name* define the name of the attachment inclusive extention (.png, .pdf, .xml) that will appear in print preview attachment's list on in the printed pdf.

The param *content* contains the path to the file to attach or the data to attach. The path can be relative to the script's folder, the document's folder, the name of a document attacched to the file or a data uri scheme (for images imbedded in the document).
- file:script/<relative_path_to_script_folder>/<image_name>
- file:document/<relative_path_to_file_folder>/<image_name>
- documents:<document_name>
- data:[<media type>][;charset=<character set>][;base64],<data>

Example:

```
//Create the report
var report = Banana.Report.newReport('Report attachments');

//Add a paragraph with some text
report.addParagraph('Report with attachments');

//Attach text files created on the fly
//We use the prefix 'data:...' to tell that the string is not an url but is
itself the content of the file
report.addAttachment('text file 1.txt', 'data:text/plain;utf8,This is the
content of the text file 1.');
report.addAttachment('text file 2.txt', 'data:text/plain;utf8,This is the
content of the text file 2.');
report.addAttachment('text file 3.txt', 'data:text/plain;utf8,This is the
content of the text file 3.');

//Attach an image stored in the document table
//We use the prefix 'document:...'
report.addAttachment('logo.jpg', 'documents:logo');

//Add an xml element
//We just add the new created Banana.Xml.newDocument
var xmlDocument = Banana.Xml.newDocument("eCH-0217:VATDeclaration");
var rootNode = xmlDocument.addElement("eCH-0217:VATDeclaration");
rootNode.addElement("title").addTextNode("Vat Declaration 2018");
report.addAttachment('vat_declaration.xml', xmlDocument);

//Print the report
var stylesheet = Banana.Report.newStyleSheet();
Banana.Report.preview(report, stylesheet);
```

Since Banana Accounting 9.0.4

addMessage(msg[, idMsg])

Add the message msg to the document. The message is showed in the pane "Messages", and in a dialog if the application option "Show Messages" is turned on.
If idMsg is not empty, the help button calls an url with script's id and message's id (idMsg) as parameters.

See also: Application.AddMessage, Table.AddMessage, Row.AddMessage.

```
Banana.document.addMessage("Message text");
```

clearMessages()

Clear all the document's messages showed in the pane "Messages".

```
Banana.document.clearMessages();
```

getScriptSettings()

Get the settings of the script saved in the document. You use this method to get settings that are private to the running script. It is possible to save the settings of the script through the method "setScriptSettings".

With this method Settings are saved and restored under the script id, If you change the script's id you will lose the saved settings.

Example:

```
// Initialise parameter
param = {
   "searchText": "",
   "matchCase": "false",
   "wholeText": "false"
};

// Readscript settings
var strData = Banana.document.getScriptSettings();
if (strData.length > 0) {
   var objData = JSON.parse(strData);
   if (objData)
      param = objData;
}
```

getScriptSettings(id)

Return the settings saved in the document under the id 'id'.

You use this method to get settings that are shared between scripts. As id we recommend to use a substring of the script's id. For example if you have the scripts 'ch.banana.vat.montlyreport' and 'ch.banana.vat.endofyearreport', then you can use as id 'ch.banana.vat'.

Example:

```
// Initialise parameter
param = {
   "searchText": "",
   "matchCase": "false",
   "wholeText": "false"
};

// Readscript settings
var strData = Banana.document.getScriptSettings('ch.banana.vat');
if (strData.length > 0) {
   var objData = JSON.parse(strData);
   if (objData)
      param = objData;
}
```

info(section, id)

Return the info value of the document referenced by section and id. Section and Id correspond to the

xml name listed in the Info table, see command File info in menu "Tools" and set the view to complete to see the XML columns. If the value referenced by section and id doesn't exist, an object of type undefined is returned.

Example:

```
// Get some value of the accounting file
var FileName = Banana.document.info("Base","FileName");
var DecimalsAmounts = Banana.document.info("Base","DecimalsAmounts");
var HeaderLeft = Banana.document.info("Base","HeaderLeft");
var HeaderRight = Banana.document.info("Base","HeaderRight");
var BasicCurrency =
Banana.document.info("AccountingDataBase","BasicCurrency");

// For openingDate and closureDate use instead startDate and endDate
var openingDate = Banana.document.info("AccountingDataBase","OpeningDate");
var closureDate = Banana.document.info("AccountingDataBase","ClosureDate");

// For file accounting type
var FileType = Banana.document.info("Base","FileType");
var FileGroup = Banana.document.info("Base","FileTypeGroup");
var FileNumber = Banana.document.info("Base","FileTypeNumber");
```

FileTypeGroup / FileTypeNumber combinations:

- 100 Double entry accounting
  - 100 No VAT
  - 110 With VAT
  - 120 Multi Currency
  - 130 Multi Currency with VAT
- 110 Income and Expense accounting
  - 100 No VAT
  - 110 With VAT
- 130 Cash Book
  - 100 No VAT
  - 110 With VAT
- 400 Address / Labels
  - 110 Labels
  - 120 Address

scriptSaveSettings(string)

Save the settings of the script in the document. The next time the script is run, it si possible to read the saved settings with "scriptReadSettings".

With this method Settings are saved and restored under the script id, If you change the script's id you will lose the saved settings.

Example:

```
// Save script settings
var paramString = JSON.stringify(param);
```

```
var value = Banana.document.scriptSaveSettings(paramString);
```

Deprecated. Use setScriptSettings instead.

scriptReadSettings()

Return the saved settings of the script.

With this method Settings are saved and restored under the script id, If you change the script's id you will lose the saved settings.

Example:

```
// Initialise parameter
param = {
   "searchText": "",
   "matchCase": "false",
   "wholeText": "false"
};

// Readscript settings
var strData = Banana.document.scriptReadSettings();
if (strData.length > 0) {
   var objData = JSON.parse(strData);
   if (objData)
      param = objData;
}
```

Deprecated. Use getScriptSettings instead.

setScriptSettings(value)

Save the settings of the script in the document. It is possible to read the saved settings of the script with the method "getScriptSettings".

With this method Settings are saved and restored under the script id, If you change the script's id you will lose the saved settings.

Example:

```
// Save script settings
var paramString = JSON.stringify(param);
var value = Banana.document.setScriptSettings(paramString);
```

setScriptSettings(id, value)

Save the settings in the document under the id 'id'. It is possible to read the saved settings with "getScriptSettings(id)".

You use this method to set settings that are shared between scripts. As id we recommend to use a substring of the script's id. For example if you have the scripts 'ch.banana.vat.montlyreport' and 'ch.banana.vat.endofyearreport', then you can use as id 'ch.banana.vat'.

Example:

```
// Save script settings
var paramString = JSON.stringify(param);
var value = Banana.document.setScriptSettings('ch.banana.vat', paramString);
```

table(xmlTableName)

Return the table referenced by the name xmlTableName as a [Table object](), or undefined if it doesn't exist.

```
Banana.document.table("Accounts");
```

table(xmlTableName, xmlListName)

Return the table referenced by the name xmlTableName with the rows of the list xmlListName as a [Table object](), or undefined if the table or the list don't exist. The default list is the 'Data' list.

```
Banana.document.table("Transactions", "Examples");
Banana.document.table("Transactions").list("Examples");  // alternative way
```

See also: [Table.list](), [Table.listNames]().

value(tableName, rowNr, columnName)

Return the value in table tableName, row rowNr and column columnName as string. If the table, row or column are not founds it return an object of type undefined.

```
Banana.document.value("Accounts", 5, "Description")
```

Banana.Document.Cursor

Banana.Document.Cursor is the interface to the cursor and can be accessed through Banana.document.cursor.

Properties

tableName

Return the xml name of the current table.

```
var currentTable = Banana.document.cursor.tableName;
```

rowNr

Return the number of the current row.

```
var currentRow = Banana.document.cursor.rowNr;
```

columnName

Return the xml name of the current column.

```
var currentColumn = Banana.document.cursor.columnName;
```

selectionTop

Return the index of the top row of the current selection.

```
var currentSelectionTop = Banana.document.cursor.selectionTop;
```

selectionBottom

Return the index of the bottom row of the current selection.

```
var currentSelectionBottom = Banana.document.cursor.selectionBottom;
```

Banana.Document.Row

Banana.Document.Row is the interface of a row.

Properties

isEmpty

Return true if the row is completly empty.

```
var isEmpty = tRow.isEmpty;
```

rowNr

Return the index of the row.

```
var rowNr = tRow.rowNr;
```

uniqueId

Return the unique id (an interger value) of the row.
Banana assign to every new row a unique id, this value is fix a will never change.

```
var uniqueId = tRow.uniqueId;
```

Methods

addMessage(msg [, columnName] [, idMsg])

Add the message msg to the document. The message is showed in the pane "Messages", and in a dialog if the application option "Show Messages" is turned on.

If idMsg is not empty, the help button calls an url with message's id (idMsg) as parameter.

If columnName is not empty, the message is connected to the column columnName. With a double click over message in the message pane, the cursor jump to the corresponding table, rowNr and

columnName.

See also: Application.AddMessage, Table.AddMessage, Document.AddMessage.

```
var accountsTable = Banana.document.table("Accounts");
var tRow = accountsTable.row(4);
tRow.addMessage("Message text");
```

toJSON([columnNames])

Return the row as JSON string. If the parameter columnNames is defined, only the columns in the array are included in the file.

```
// Return all the columns of the row
var json = tRow.toJSON();

// Return only the defined columns of the row
var json = tRow.toJSON(["Account", "Description", "Balance"]);
```

value(columnName)

Return the value in column columnName. If the column is not found or the object is invalid it return the value undefined.

```
var accountsTable = Banana.document.table("Accounts");
var tRow = accountsTable.row(4);
tRow.value("Description");
```

Banana.Document.Table

Banana.Document.Table is the interface of a table.

Properties

name

Return the xml name of the table.

```
var table = Banana.document.table("Accounts");
var tName = table.name;
```

columnNames

Return the xml names of the table's columns as an array.

```
var table = Banana.document.table("Accounts");
var tColumnNames = table.columnNames;
```

listName

Return the xml name of the list that this table object reference to. The default list is the 'Data' list.

```
var table = Banana.document.table("Accounts");
var tListName = table.listName;
```

listNames

Return the xml names of the available lists as an array. The default list is the 'Data' list.

```
var table = Banana.document.table("Accounts");
var tListNames = table.listNames;
```

rowCount

Return the number of rows in the table.

```
var table = Banana.document.table("Accounts");
var tRowCount = table.rowCount;
```

rows

Return the rows of the table as an array of Row objects.

```
var table = Banana.document.table("Accounts");
var tRows = table.rows;
```

Methods

addMessage(msg, rowNr [, columnName] [, idMsg])

Add the message msg to the queue of the document. The message is showed in the pane "Messages", and in a dialog if the application option "Show Messages" is turned on.

If idMsg is not empty, the help button calls an url with message's id (idMsg) as parameter.

If rowNr is different than "-1" the message is connected to the row rowNr. if columnName is not empty, the message is connected to the column columnName. With a double click over message in the message pane, the cursor jump to the corresponding table, rowNr and columnName.

See also: Application.AddMessage, Row.AddMessage, Document.AddMessage.

```
var table = Banana.document.table("Accounts");
table.addMessage("Message string", 3, "description");
```

extractRows(function(rowObj, rowNr, table), tableTitle)

Return an array of rows filled with all row elements that pass a test (provided as a function) and show them in the table "Selections".
The title of the table is set to tableTitle.

```
function accountStartsWith201(rowObj,rowNr,table) {
   // Return true if account start with '201'
   return rowObj.value('Account').startsWith('201');
```

```
}
var tableAccount = Banana.document.table('Accounts');
// Show a table with all accounts that start with '201'
tableAccount.extractRows(accountStartsWith201, 'Accounts that start with
201');
```

findRows(function(rowObj, rowNr, table))

Return an array of Row objects that pass a test (provided as a function).

```
function accountStartsWith201(rowObj,rowNr,table) {
   // Return true if account start with '201'
   return rowObj.value('Account').startsWith('201');
}
var tableAccount = Banana.document.table('Accounts');
// Find rows of all accounts that start with '201'
var rows = tableAccount.findRows(accountStartsWith201);
```

findRowByValue(columnName, value)

Return the first row as Row object that contains the value in the the column columnName. Or undefined if any row is found.

```
var cashAccountRow =
Banana.document.table('Accounts').findRowByValue('Account','1000');
if (!cashAccountRow)
   //Row not found
```

list(xmlListName)

Return a new table object with the rows of the list xmlListName, or undefined if the list xmlListName doesn't exist.

```
var recurringTransactions =
Banana.document.table('Transactions').list('Examples');
var archivedProducts = Banana.document.table('Products').list('Archive');
```

row(rowNr)

Return the Row at index rownr as Row Object, or undefined if rowNr si outside the valid range.

```
var table = Banana.document.table("Accounts");
var row = table.row(3);
```

toJSON([columnNames])

Return the table as JSON string. If the parameter columnNames is defined, only the columns in the array are included in the file.

```
var table = Banana.document.table("Accounts");
var json = table.toJSON();
```

toHtml([columnNames, formatValues])

Return the table as Html file. If the parameter columnNames is defined, only the columns in the array are included in the file. If formatValues is set to true, the values are converted to the locale format.

Example:

```
//Show the whole row content of the table Accounts
Banana.Ui.showText(Banana.document.table('Accounts').toHtml());

//Show some columns and format dates, amounts, ... as displayed in the
program
Banana.Ui.showText(
Banana.document.table('Accounts').toHtml(['Account','Group','Description','Ba
lance'],true)
);
```

toTsv([columnNames])

Return the table as Tsv file (Tab separated values). If the parameter columnNames is defined, only the columns in the array are included in the file.

```
var table = Banana.document.table("Accounts");
var tsv = table.toTsv();
```

value(rowNr, columnName)

Return the value in row rowNr and column columnName as string. Or undefined if the row or column are not found.

```
var table = Banana.document.table("Accounts");
var account = table.value(3,'Account');
var description = table.value(3,'Description');
```

Banana.IO

The Banana.IO class is used to read and write to files.

Introduction

The API Banana.IO and Banana.IO.LocalFile allow a script to read or write to files in a secure way. The script can only read or writes to files that are first selected by the user though the corresponding dialogs. The script has no direct access to files on the file system. After the script finished, the permissions to write or read files are canceled.

For example to write the result of a script to a file:

```
var fileName = Banana.IO.getSaveFileName("Select save file", "", "Text file
(*.txt);;All files (*)");
if (fileName.length) {
   var file = Banana.IO.getLocalFile(fileName)
```

```
   file.codecName = "latin1";  // Default is UTF-8
   file.write("Text to save ...");
   if (!file.errorString) {
      Banana.IO.openPath(fileContent);
   } else {
      Banana.Ui.showInformation("Write error", file.errorString);
   }
} else {
   Banana.Ui.showInformation("Info", "no file selected");
}
```

To read the content of a file:

```
var fileName = Banana.IO.getOpenFileName("Select open file", "", "Text file
(*.txt);;All files (*)")
if (fileName.length) {
   var file = Banana.IO.getLocalFile(fileName)
   file.codecName = "latin1";  // Default is UTF-8
   var fileContent = file.read();
   if (!file.errorString) {
       Banana.IO.openPath(fileContent);
   } else {
      Banana.Ui.showInformation("Read error", file.errorString);
   }
} else {
   Banana.Ui.showInformation("Info", "no file selected");
}
```

Methods

getOpenFileName(caption, path, filter)

The method getOpenFileName returns an existing file selected by the user. If the user presses Cancel, it returns an empty string. The file selected by the user is then allowed to be readen, but not written.

The parameter *caption* is the caption of the dialog.

The parameter *path* is path inclusive the file name to be selected. If the path is relative, the current open document path or the user's document path is used.

The parameter *filter* set the files types to be showed. If you want multiple filters, separate them with ';;', for example: "Text file (*.txt);;All files (*)".

```
var fileName = Banana.IO.getOpenFileName("Select file to read", "", "Text
file (*.txt);;All files (*)")
```

Since: Banana Accounting 9.0.7, only in Banana Experimental

getSaveFileName(caption, path, filter)

The method getSaveFileName returns an existing file selected by the user. If the user presses Cancel, it returns an empty string. The file selected by the user is then allowed to be readen and written.

The parameter *caption* is the caption of the dialog.

The parameter *path* is path inclusive the file name to be selected. If the path is relative, the current open document path or the user's document path is used.

The parameter *filter* set the files types to be showed. If you want multiple filters, separate them with ';;', for example: "Text file (*.txt);;All files (*)".

```
var fileName = Banana.IO.getSaveFileName("Select file to write", "", "Text
file(*.txt);;All files (*)")
```

getLocalFile(path)

The method getLocalFile(path) return an object of type [Banana.IO.LocalFile]() that represent the requested file. This method always return a valid Banana.IO.LocalFile object.

The parameter *path* to the file.

openUrl(path)

The method openUrl(path) open the file referred by *path* in the system default application.

The parameter *path* to the file.

openPath(path)

The method openPath(path) show the folder containing the file referred by *path* in the system file manager.

The parameter *path* to the file.

Banana.IO.LocalFile

The LocalFile class represent a file on the local file system. See [Banana.IO]() for an example.

Properties

codecName

The name of the codec to be used for reading or writing the file. Default is 'UTF-8'.

errorString

Read only. The string of the last occured error. If no error occured it is empty.

Methods

read()

Returns the content of the file. This function has no way of reporting errors. Returning an empty string can mean either that the file is empty, or that an error occurred. Check the content of the property errorString to see if an error occured.

write(text [, append])

Write text to the file. If append is set to true, text is appended to the file. Return true if the operation was succesfully, false otherwise.

Banana.Report

The class Banana.Report enable you to create reports, preview and print in Banana Accounting.

Introduction

The report logic is similar to the HTML / CSS logic:

1. Create a Report object .
    - A report contain a list of ReportElements (paragraphs, texts, tables and other)
    - The element can contains other sub-elements
    - For each element you can add a class that is used for rendering the element
2. Create a StyleSheet
    - A stylesheet is composed of StyleElements.
3. You preview and print a report by passing the Report and the Stylesheet object.

Each report sturcture has:

- a ReportElement list
- a Header Element list
- a Footer Element list

```
// Report
var report = Banana.Report.newReport("Report title");
report.addParagraph("Hello World !!!", "styleHelloWorld");

// Styles
var stylesheet = Banana.Report.newStyleSheet();
var style = stylesheet.addStyle(".styleHelloWorld");
style.setAttribute("font-size", "96pt");
style.setAttribute("text-align", "center");
style.setAttribute("margin-top", "50mm");

var style2 = stylesheet.addStyle("@page");
style2.setAttribute("size", "landscape");

// Print preview
Banana.Report.preview(report, stylesheet);
```

Methods

newReport(title)

Creates a report with title 'title'. The returned object is of type Banana.Report.ReportElement.

To the report you can then add the desired elements, like paragraphs, texts, tables, and so on that construct the structure of the report.

```
var report = Banana.Report.newReport("Report title");
```

newStyleSheet()

Creates an empty stylesheet. The returned object is of type Banana.Report.ReportStyleSheet.

To the stylesheet you can add the styles that format the report.

```
var stylesheet = Banana.Report.newStyleSheet();
```

newStyleSheet(fileName)

Creates a stylesheet from a file. The file has the same syntax as CSS stylesheets. The file path is relative to the script's path. The path can't contain a '..'' (parent directory).

The returned object is of type Banana.Report.ReportStyleSheet.

You can add further styles to the returned stylesheet.

```
var reportStyles = Banana.Report.newStyleSheet("styles.css");

*** Content of file styles.css ***
.helloWorldStyle
{
font-size: 96pt;
text-align: center;
margin-top: 50mm;
}

@page
{
size: landscape;
}
*** End of file styles.css ***
```

preview(report, stylesheet)

Opens a print preview Dialog and shows the report with the given stylesheet.

The param report is an object of type Banana.Report.ReportElement. The param stylesheet is an object of type Banana.Report.ReportStyle.

The page orientation is given by the stylesheet. The default size and orientation is taken from the default printer, or can be set through the stylesheet.

```
// Set landscape orientation
stylesheet.addStyle("@page {size: landscape}");
```

```
// Set page size and orientation
stylesheet.addStyle("@page {size: A5 lanscape}");

// Displays the report
Banana.Report.preview(report, stylesheet);
```

preview(title, reports, stylesheets)

Opens a print preview Dialog with title 'title' and shows the reports with the given stylesheets. This method allow you to print two or more distinct reports together.

The param report is an array of objects of type Banana.Report.ReportElement. The param stylesheet is an array of objects Banana.Report.ReportStylesheet.

Each report's title will appear in the index of the printed pdf. The numbering of pages will restart from 1 at the beginning of each printed report.

The page orientation is given by the stylesheet. The default size and orientation is taken from the default printer, or can be set through the stylesheet.

```
var docs = [];
var styles = [];

// Report
for (var i = 0; i < 10; i++) {
   var report = Banana.Report.newReport("Report title");
   report.addParagraph("Hello World #" + i + " !!!", "styleHelloWorld");
   report.setTitle("Document " + i); // The report's title will appear in the
pdf's index
   report.getFooter().addFieldPageNr();
   docs.push(report);

   // Styles
   var stylesheet = Banana.Report.newStyleSheet();
   var style = stylesheet.addStyle(".styleHelloWorld");
   style.setAttribute("font-size", "24pt");
   style.setAttribute("text-align", "center");
   style.setAttribute("margin-top", "10mm");
   var style2 = stylesheet.addStyle("@page");
   style2.setAttribute("size", "landscape");
   styles.push(stylesheet);
}

// Print preview of 10 documents together
Banana.Report.preview("Multi documents printing example", docs, styles);
```

Since Banana Accounting 9.0.4

qrCodeImage(text, qrCodeParam)

Creates a QRCode image according to the passed text. The returned object is a svg image.

- qrCodParam.**errorCorrectionLevel**
  string value H (high), L (low), M (medium), Q (quartile) (default value M)
- qrCodeParam.**binaryCodingVersion**
  int value from 0 to 40 (default value 40)
- qrCodeParam.**border**
  int value from 0 to 100 (default value 0)
- qrCodeParam.**errorMsg**
  in case of error the method returns the error message in this property

since: Banana Accounting 9.0.4

```
var text = 'hello world';
var qrCodeParam = {};
qrCodeParam.errorCorrectionLevel = 'M';
qrCodeParam.binaryCodingVersion = 25;
qrCodeParam.border = 0;

var qrCodeSvgImage = Banana.Report.qrCodeImage(text, qrCodeParam);
if (qrCodeParam.errorMsg && qrCodeParam.errorMsg.length>0) {
   Banana.document.addMessage(qrCodeParam.errorMsg);
}
if (qrCodeSvgImage) {
   var repDocObj = Banana.Report.newReport('Printing QRCode img');
   repDocObj.addImage(qrCodeSvgImage, 'qrcode');
}
```

Example: Hello world

```
// Simple test script using Banana.Report
//
// @id = ch.banana.script.report.helloworld
// @api = 1.0
// @pubdate = 2017-01-02
// @publisher = Banana.ch SA
// @description = Report Hello world
// @task = app.command
// @doctype = *
// @inputdatasource = none
// @timeout = -1
//

function exec(string) {

   // Create the report
   var report = Banana.Report.newReport("Report title");
   // Add a paragraph to the report
   report.addParagraph("Hello World !!!", "helloWorldStyle");

   // Define some styles
```

```
    var stylesheet = Banana.Report.newStyleSheet();
    var style = stylesheet.addStyle(".helloWorldStyle");
    style.setAttribute("font-size", "96pt");
    style.setAttribute("text-align", "center");
    style.setAttribute("margin-top", "50mm");

    var style2 = stylesheet.addStyle("@page");
    style2.setAttribute("size", "landscape");

    // Open Preview
    Banana.Report.preview(report, stylesheet);
}
```

An example with tables, page breaks and differents styles

Result

Page header

Document title

1. Text

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip **ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit** esse cillum dolore eu fugiat nulla pariatur.
Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

2. Table

Table caption

Description		Income	Expense	Balance
	Initial balance			157.00
11.02.2014	Transfer from post office account	500.00		657.00
20.02.2014	Various payments		7'250.00	-6'593.00
	Total transactions	**500.00**	**7'250.00**	**-6'593.00**

3. Bookmarks and links

3.1 Internal links

-> link to bookmark on page 2

3.2 External links

-> link to Banana.ch web page

1

Sample built with Script Report API

Script

```
// Test script using Banana.Report
```

```
//
// @id = ch.banana.script.report.report
// @api = 1.0
// @pubdate = 2017-01-02
// @publisher = Banana.ch SA
// @description = Test report api
// @task = app.command
// @doctype = *
// @outputformat = none
// @inputdatasource = none
// @timeout = -1
//

function exec(string) {

   // Report
   var report = Banana.Report.newReport("Report title");

   var pageHeader = report.getHeader()
   pageHeader.addClass("header");
   pageHeader.addText("Page header");
   report.getFooter().addFieldPageNr();

   var watermark = report.getWatermark();
   watermark.addParagraph("Sample built with Script Report API");

   report.addParagraph("Report title", "titleStyle");
   report.addParagraph("1. Text", "chapterStyle").setOutline(1);

   report.addParagraph("Lorem ipsum dolor sit amet, consectetur adipisicing
elit, sed do " +
    "eiusmod tempor incididunt ut labore et dolore magna aliqua. " +
    "Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi
ut aliquip " +
    "ex ea commodo consequat. Duis aute irure dolor in reprehenderit in
voluptate velit " +
    "esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat
cupidatat non " +
    "proident, sunt in culpa qui officia deserunt mollit anim id est
laborum.");

   var paragraph2 = report.addParagraph();
   paragraph2.addText("Lorem ipsum dolor sit amet, consectetur adipisicing
elit, sed do ");
   paragraph2.addText("eiusmod tempor incididunt ut labore et dolore magna
aliqua. ", "blueStyle");
   paragraph2.addText("Ut enim ad minim veniam, quis nostrud exercitation
ullamco laboris nisi ut aliquip ", "boldStlyle");
   paragraph2.addText("ex ea commodo consequat. Duis aute irure dolor in
reprehenderit in voluptate velit ", "underlineStyle boldStyle");
   paragraph2.addText("esse cillum dolore eu fugiat nulla pariatur.");
```

```
   paragraph2.addLineBreak();
   paragraph2.addText("Excepteur sint occaecat cupidatat non proident, sunt
in culpa qui officia deserunt mollit anim id est laborum.", "italicStyle");

   report.addParagraph("2. Table", "chapterStyle").setOutline(1);

   var table = report.addTable();
   table.getCaption().addText("Table caption");

   var tableHeader = table.getHeader();
   var tableHeaderRow = tableHeader.addRow();
   tableHeaderRow.addCell("Description", "", 2);
   tableHeaderRow.addCell("Income");
   tableHeaderRow.addCell("Expense");
   tableHeaderRow.addCell("Balance");

   var tableRow = table.addRow();
   tableRow.addCell();
   tableRow.addCell("Initial balance");
   tableRow.addCell();
   tableRow.addCell();

tableRow.addCell(Banana.Converter.toLocaleNumberFormat("157.00")).addClass("b
alanceStyle");

   var tableRow = table.addRow();
   tableRow.addCell(Banana.Converter.toLocaleDateFormat("2014-02-11"));
   tableRow.addCell("Transfer from post office account");
   tableRow.addCell(Banana.Converter.toLocaleNumberFormat("500.00"));
   tableRow.addCell();

tableRow.addCell(Banana.Converter.toLocaleNumberFormat("657.00")).addClass("b
alanceStyle");

   var tableRow = table.addRow();
   tableRow.addCell(Banana.Converter.toLocaleDateFormat("2014-02-20"));
   tableRow.addCell("Various payments");
   tableRow.addCell();
   tableRow.addCell(Banana.Converter.toLocaleNumberFormat("7250.00"));

tableRow.addCell(Banana.Converter.toLocaleNumberFormat("-6593.00")).addClass(
"balanceStyle negativeStyle");

   var tableRow = table.addRow("totalrowStyle");
   tableRow.addCell();
   tableRow.addCell("Total transactions");
   tableRow.addCell(Banana.Converter.toLocaleNumberFormat("500.00"));
   tableRow.addCell(Banana.Converter.toLocaleNumberFormat("7250.00"));

tableRow.addCell(Banana.Converter.toLocaleNumberFormat("-6593.00")).addClass(
"balanceStyle negativeStyle");
```

```
   report.addParagraph("3. Bookmarks and links",
"chapterStyle").setOutline(1);

   report.addParagraph("3.1 Internal links", "chapter2Style").setOutline(2);
   report.addParagraph("-> link to bookmark on page
2").setLink("bookmarkpage2");

   report.addParagraph("3.2 External links", "chapter2Style").setOutline(2);
   report.addParagraph("-> link to Banana.ch web
page").setUrlLink("http://www.banana.ch");

   report.addPageBreak();

   var chapter4 = report.addParagraph("4. Pages", "chapterStyle");
   chapter4.setOutline(1);

   report.addParagraph("Bookmark on page 2").setBookmark("bookmarkpage2");


   // Styles
   var docStyles = Banana.Report.newStyleSheet();

   var pageStyle = docStyles.addStyle("@page");
   pageStyle.setAttribute("margin", "20mm 20mm 20mm 20mm");

   var headerStyle = docStyles.addStyle("phead");
   headerStyle.setAttribute("padding-bottom", "1em");
   headerStyle.setAttribute("margin-bottom", "1em");
   headerStyle.setAttribute("border-bottom", "solid black 1px");

   var footerStyle = docStyles.addStyle("pfoot");
   footerStyle.setAttribute("text-align", "right");

   var paragraphStyle = docStyles.addStyle("p");
   paragraphStyle.setAttribute("margin-top", "0.5em");

   var captionStyle = docStyles.addStyle("caption");
   captionStyle.setAttribute("margin-top", "1em");

   var titleStyle = docStyles.addStyle(".titleStyle");
   titleStyle.setAttribute("font-size", "24");
   titleStyle.setAttribute("text-align", "center");
   titleStyle.setAttribute("margin-bottom", "1.2em");

   docStyles.addStyle(".chapterStyle", "font-size:16; margin-top:2em; margin-
bottom:0.2em");
   docStyles.addStyle(".chapter2Style", "font-size:12; margin-top:1.4em;
margin-bottom:0.2em");

   var tableStyle = docStyles.addStyle("table");
```

```
    tableStyle.setAttribute("border", "2px solid red");

    docStyles.addStyle("td", "border: 1px dashed black; padding: 2px;");

    var tableColStyle = docStyles.addStyle(".balanceStyle");
    tableColStyle.setAttribute("background-color", "#E0EFF6");
    tableColStyle.setAttribute("text-align", "right");

    var totalRowStyle = docStyles.addStyle(".totalrowStyle");
    totalRowStyle.setAttribute("font-weight", "bold");

    var totalBalanceStyle = docStyles.addStyle(".totalrowStyle
td.balanceStyle");
    totalBalanceStyle.setAttribute("text-decoration", "double-underline");

    docStyles.addStyle(".blueStyle", "color:blue");
    docStyles.addStyle(".underlineStyle", "text-decoration:underline;");
    docStyles.addStyle(".italicStyle", "font-style:italic;");
    docStyles.addStyle(".boldStyle", "font-weight:bold");

    // Open Preview
    Banana.Report.preview(report, docStyles);
}
```

Banana.Report.ReportElement

The class Banana.Report.ReportElement represents the report itself and every element in the report, like sections, paragraphs, tables, texts and the report itself.

Once you create a new report through the method Banana.Report.newReport() you can start adding sections, paragraphs, texts, tables and so on.

When you add an element with one of the add methods, you get as return value an object of type

Elements as a container of other elements.

Banana.Report.ReportElement that represents the added element.
To this object you can add further elements and by this way construct the structure of the report.

```
Report
  +  Paragraph
  +  Table
    +  Row
      + Cell
      + Cell
    +  Row
      + Cell
      + Cell
    ...
```

Even if this interface enable you to add tables to text elements or columns to paragraphs, the result

will be undefined.

Formatting like text size, text color, margins, and so on are set separately through a Banana.Report.ReportStyleSheet object.

Methods

addClass(classes)

Add classes to the node. A class binds the element to the corresponding class style definend in Banana.Report.ReportStyleSheet as used in CSS Stylesheets.

```
var report = Banana.Report.newReport("Report title");
report.addParagraph("1250.00").addClass("balanceStyle");
```

addSection([classes])

Add a section and return the created section as a Banana.Report.ReportElement object.

You can add sections only to sections, cells, captions, headers or footers.

```
var report = Banana.Report.newReport("Report title");

//Add a section with a style
var section = report.addSection("sectionStyle");
section.addParagraph("First paragraph");
section.addParagraph("Second paragraph");
```

addParagraph([text, classes])

Add a paragraph and return the created paragraph as a Banana.Report.ReportElement object.

You can add paragraphs only to sections, cells, captions, headers or footers.

```
var report = Banana.Report.newReport("Report title");

//Add an empty paragraph
report.addParagraph(" ");

//Add a paragraph with a text
report.addParagraph("Hello World !!!");

//Add a paragraph with a text and a style
report.addParagraph("Hello World !!!", "styleHelloWorld");
```

addText(text [, classes])

Add a text node and return the create text as a Banana.Report.ReportElement object.

You can add texts only to sections, paragraphs, cells, captions, headers or footers.

```
var report = Banana.Report.newReport("Report title");
```

```
//Add a text
report.addText("Hello world !!!");

//Add a text with a style
report.addText("Hello world !!!", "styleHelloWorld");
```

addTable([classes])

Add a table and return the created table as a Banana.Report.ReportElement object.

You can add tables only to the report or sections.

```
var report = Banana.Report.newReport("Report title");
var myTable = report.addTable("myTable");
```

addColumn([classes])

Add a column and return the created column as a Banana.Report.ReportElement object.

You can add columns only to tables.

```
var column1 = myTable.addColumn("column1");
var column2 = myTable.addColumn("column2");
var column3 = myTable.addColumn("column3");
```

addRow([classes])

Add a row and return the created row as a Banana.Report.ReportElement object.

You can add rows only to tables, table headers or table footers.

```
var tableRow = myTable.addRow();
...
```

addCell([span])

Add an empty cell and return the created cell as a Banana.Report.ReportElement object.

You can add cells only to rows. You can span cells over columns but not over rows.

```
tableRow.addCell();      //span empty cell over 1 column (default value)
tableRow.addCell("", 3); //span empty cell over 3 columns
...
```

addCell(text [,classes, span])

Add a cell to the node and return the created cell as a Banana.Report.ReportElement object.

You can add cells only to rows.You can span cells over columns but not over rows.

```
tableRow.addCell("Bank", "firstCellStyle", 3);      //span cell over 3
columns
tableRow.addCell("1200.65", "secondCellStyle, 1);   //span cell over 1 column
...
```

addLineBreak()

Add a line break and return the created line break as a Banana.Report.ReportElement object.

You can add line breaks only to paragraphs or cells.

```
// Add a line break to a paragraph
var p = report.addParagraph(" ");
p.addLineBreak();

// Add a line break to a cell
var c = tableRow.addCell();
c.addLineBreak();
```

addPageBreak()

Add a page break node and return the created page beak as a Banana.Report.ReportElement object.

You can add page breaks only to the report or sections.

```
var report = Banana.Report.newReport("Report title");
...
report.addPageBreak();
...
```

addImage(path [,classes])

Add an image and return the created image as a Banana.Report.ReportElement object. Supported formats are png and jpg.

The path can be relative to the script's folder, the document's folder, the name of a document attacched to the file or a data uri scheme (for images imbedded in the document).
- file:script/<relative_path_to_script_folder>/<image_name>
- file:document/<relative_path_to_file_folder>/<image_name>
- documents:<document_name>
- data:[<media type>][;charset=<character set>][;base64],<data>

You can add images only to sections, paragraphs, cells, captions, headers or footers.

The parameter path can be absolute or relative to the script path.

```
var report = Banana.Report.newReport("Report title");

// Add an image located in the script folder
report.addImage("file:script/logo_abc.jpg");

// Add an image located in the dcoument folder
report.addImage("file:document/logo_mnp.jpg");

// Add an image saved in the table documents
report.addImage("documents:logo_xyz.jpg");

// Add an image (a red dot) included in the document
```

```
report.addImage("data:image/png;base64,iVBORw0KGgoAAAANSUhEUgAAAAUA
AAAFCAYAAACNbyblAAAAHElEQVQI12P4//8/w38GIAXDIBKE0DHxgljNBAAO
9TXL0Y4OHwAAAABJRU5ErkJggg==");
```

addImage(path, widht, height [,classes])

Overloaded method to add an image and return the created image as a Banana.Report.ReportElement object.

The parameters widht and height have the same syntax as css lenhgt values. They can be absolute (ex.: "30px", "3cm", ...) or relative (ex.: "50%", "3em", ...).

```
var report = Banana.Report.newReport("Report title");
report.addImage("documents:image_logo", "3cm", "5cm");
```

addFieldPageNr([classes])

Add a field with containg the page number and return the created field as a Banana.Report.ReportElement object.

You can add this field only to sections, paragraphs, cells, captions, headers or footers.

```
var report = Banana.Report.newReport("Report title");
...
// Add the page number to the paragraph
report.addParagraph("Page ").addFieldPageNr();

// Add a page number to the footer
var footer = report.getFooter();
footer.addText("Page ");
footer.addFieldPageNr();
```

excludeFromTest()

Mark the field to be not tested during a test case. The value is any case outputed to the test results, it is just ignored during the comparaison of the test results.

This useful for example for text fields containing the current date.

```
var currentDate = Banana.Converter.toLocaleDateFormat(new Date());
var textfield = report.getFooter().addText(currentDate);
textfield.excludeFromTest();
```

getWatermark()

Return the watermak element.

Only the report has a watermak element.

```
var watermark = report.getWatermark();
watermark.addParagraph("Watermark text");
```

getHeader()

Return the header of the element.

Only tables and the report have an header element.

```
var report = Banana.Report.newReport("Report title");

//Report
var reportHeader = report.getHeader();
reportHeader.addClass("header");
reportHeader.addText("Header text");

//Table
var table = report.addTable("myTable");
var tableHeader = table.getHeader();
tableRow = tableHeader.addRow();
tableRow.addCell("Description");
tableRow.addCell("Amount");
```

getFooter()

Return the footer of the element.

Only tables and the report have a footer element.

```
//Report
var footer = report.getFooter();
footer.addText("Footer text");
```

getCaption()

Return the caption of the element.

Only tables have a caption element.

```
var table = report.addTable("MyTable");
var caption = table.getCaption();
caption.addText("Table caption text", "captionStyle");
```

getTag()

Return the tag of the element, like 'body', 'p', 'table', 'td' and so on.

```
var report = Banana.Report.newReport("Report title");
...
report.getTag(); // returns 'body'
footer.getTag(); // returns 'pfoot'
...
```

getTitle()

Return the title of the element.

Only a document element have a title.

```
var report = Banana.Report.newReport("My Report Title");
var title = report.getTitle(); // return 'My Report Title'
```

setOutline(level)

Set the outline level, this is used to create the index when exporting to pdf.

```
report.addParagraph("1. Text").setOutline(1);
```

setBookmark(bookmark)

Set a bookmark (or anchor), this is used in conjunction with setLink().

```
report.addParagraph("Bookmark on page 2").setBookmark("bookmarkpage2");
```

setLink(bookmark)

Set a link to a bookmark. See setBookmark().

```
report.addParagraph("-> link to bookmark on page
2").setLink("bookmarkpage2");
```

setPageBreakBefore()

Set to insert a page break before the element.

```
// Insert a page break before a paragraph
report.addParagraph("Hello world!!!").setPageBreakBefore();

// Insert a page break before a table
   /* first create a table then... */
myTable.setPageBreakBefore();
```

setSize(width, height)

Set the size of the element.

The parameters widht and height have the same syntax as css lenhgt values. They can be absolute (ex.: "30px", "3cm", ...) or relative (ex.: "50%", "3em", ...).

You can only set the size of an image element.

```
var image = report.addImage("C:/Documents/Images/img.jpg");
image.setSize("3cm", "6cm");
```

setStyleAttribute(attr_name, attr_value)

Set a style attribute to the element. Attributes ids and values follow the CSS specification. This attibute correspont to the inline styles in Css.

```
paragraph.setAttribute("font-size", "24pt");
```

setStyleAttributes(attributes)

Set style attributes to the element. Attributes ids and values follow the CSS specification.
Those attributes correspond to the inline styles in Css.

```
paragraph.setAttribute("font-size:24pt;font-weight:bold;");
```

setTitle(title)

Set the title of the element.

Title can be only set to a document element.

```
document.setTitle("Annual report");
```

setUrlLink(link)

Set a link to an external file (file://...) or a web page (http://....).

To the element the class "link" is automatically added.

```
report.addParagraph("Link to Banana.ch web
page").setUrlLink("http://www.banana.ch");
```

Banana.Report.ReportStyle

The class Banana.Report.ReportStyle represent a single style in a stylesheet. It is used to set the style attributes.

Methods

setAttribute(attr_name, attr_value)

Set the attribute value. Attributes ids and values follow the CSS specification.

```
style.setAttribute("font-size", "24pt");
```

setAttributes(attributes)

Set attributes values. Attributes ids and values follow the CSS specification.

```
style.setAttributes("font-size:24pt;font-weight:bold;");
```

Supported attributes

font
font-family
font-size
font-style
font-weight

margin [top, right, bottom, left]
margin-top
margin-bottom
margin-left
margin-right

padding
padding-top
padding-bottom
padding-left
padding-right

hanging-ident
text-align
text-decoration
text-ellipsis
vertical-align
color
background-color

border
border-top
border-top-style
border-top-color
border-top-width
border-bottom
border-bottom-style
border-bottom-color
border-bottom-width
border-left
border-left-style
border-left-color
border-left-width
border-right
border-right-style
border-right-color
border-right-width

display
overflow
float
text-wrap
width
max-width
min-width
height

page-break-after
column-break-after
line-break-after
page-break-before
column-break-before

line-break-before
page-break-inside
line-break-inside

size
position
left
top
right
bottom

transform (matrix, translateX, translateY, translate, rotate, scaleX, scaleY, scale, skewX, skewY and skew)
transformOrigin

orphans

fill-empty-area

Non standard attributes and values

width-sym
This attribute contain a string. Columns with the same width-sym will be layouted with the same width.

layout-sym
This attribute is a string. Tables with the same layout-sym attribute will have the same layout for the width of the columns.

overflow
This attribute has the non standard value "shrink". The content of the node will be down scaled to fit given space.

```
style.setAttribute("overflow", "shrink");
```

overflow-shrink-max
This attibute the maximal down scaling factor (like 0.8).

```
style.setAttribute("overflow-shrink-max", "0.6");
```

text-decoration
This attribute can also contains the values "double-underline" or "double-strong-underline".

```
style.setAttribute("text-decoration", "underline");
```

border-style
This attribute can also contain the values "double" and "double-strong".

```
style.setAttribute("border-style", "double");
```

flexible-width
This attribute can contain the value "always" and is only used with columns. If in a table one or more columns have the attribute "flexible-widht", only those columns are enlarged to get the desired table

widht, untouching the other. Otherwise all columns are enlarged.

fill-empty-area
With this attribute you can fill the remaing space of your page with lines. Lines can be defined through the attribute, which is a string and contains the color, the style and the width of the line.

Style can be: solid, dash and dot.
Examples:

```
var style1 = stylesheet.addStyle("@page", "black solid 1");
var style2 = stylesheet.addStyle("@page", "green dash 0.5");
```

Banana.Report.ReportStyleSheet

The class Banana.Report.ReportStyleSheet is used to set the styles to format a report.

Page size and orientation

At this moment the report is rendered based on the page size defined in the default printer device. You can't define a page size, bu you can set the orientation with the Style @page.

Page orientation can't be set only once per report, you can't switch from potrait to landscape.

```
var stylesheet = Banana.Report.newStyleSheet();
stylesheet.addStyle("@page").setAttribute("size", "landscape");
```

Methods

addStyle(selector)

Create a new style with the given selector. The return object is of type Banana.Report.ReportStyle.

The syntax of selector follow the CSS specification.

- Style name without a preceding dot are reserved predefined tags like "td", "p", "table"
- Style name for new class need a preceding point ".myStyle" in the addStyle method.
  The dot name is not used when adding the class name to the element

```
report.addParagraph("Text to print 24pt", "myStyle");
var style = stylesheet.addStyle(".myStyle");
myStyle.setAttribute("font-size", "24pt");
myStyle.setAttribute("text-align", "center");

report.addCell("Text to print");
var styleTd = stylesheet.addStyle("td");
styleTd.setAttribute("font-weight", "bold");
```

addStyle(selector, attributes)

Create a new style with the given selector and attributes. The return object is of type Banana.Report.ReportStyle.

The syntax of selector and attributes follow the CSS specification.

```
var style2 = stylesheet.addStyle(".style2", "font.size: 24pt; text-align:
center");
```

parse(text)

Load the styles from the given text. The text follow the CSS specification.

```
stylesheet.parse(
  "p.style1 {font-size:24pt; text-align:center;}" +
  "@page {size:A4 landscape;}"
  );
```

The selector

The selector follow the css syntax, following you will find some examples:

Selector	Selected elements
.xyz	Select all elements with class xyz NB.: When you set the class to a ReportElement you enter the name without '.'
table	Select all tables
table.xyz	Select all tables with class xyz
table.xyz td	Select all cells in tables with class xyz

Tag selectors

@page	page
body	content of the report
phead	page header
pfoot	page footer
div	section
p	paragraph
table	table
caption	table caption
thead	table header
tbody	table body
tfoot	table footer
tr	table row
td	table cell

You can get the tag of an element through the method getTag();

How can I set the orientation of the page and the margins

```
// Set landscape orientation
styleSheet.addStyle("@page", "size: landscape");

// Page margins top, right, bottom, left
styleSheet.addStyle("@page", "margin: 20mm 20mm 20mm 25mm");
```

How can I set the size of the page

```
// Set page size
styleSheet.addStyle("@page", "size: A5");
```

How can I set the margins of page header and footer

```
styleSheet.addStyle("phead", "margin-bottom:2em");
styleSheet.addStyle("pfoot", "margin-top:2em");
```

How can I print the page number on the right of the page footer

```
document.getFooter().addFieldPageNr("alignright");
stylesheet.addStyle("pfoot", "text-align:right");
```

Can I print the total number of pages

No

I like a style implemented in a report of Banana Accounting, where can I get the used stylesheet?

In print preview export the report as html and look at the code. You will find the used styles.

Banana.SDecimal

The Banana.SDecimal (String Decimal) provide functions to do decimal math calculation that

- use decimal string in the form of '10.00' or '-10' as argument and return value
  - '.' is interpreted as the decimal separator
  - thousand separator are not allowed
- has up to 34 digits of numeric precision
- do accurate decimal rounding

You can use this functions instead of the javascript Number that use floating point arithmetic and are not very suitable for accounting calculation do the rounding differences.

```
var r = Banana.SDecimal.add('6.50', '3.50');   // return '10.00'
var r = Banana.SDecimal.divide('10', '2');     // return '5.00'
var r = Banana.SDecimal.divide('10', '2', ''); // return '5'
var r = Banana.SDecimal.divide('10', '2');     // return '5.00000'
```

Rounding context

Functions can be passed a rounding context that specify the rounding properties:

- **decimals** is the number of decimal digits (default value is 2)
  - **null** returns the value unrounded.
  - **'0'** returns with no decimals.
  - **'1'** to **'33'** returns the value with the indicated number o decimals.
- **mode** is the rounding mode (default value is HALF_UP)
  - Banana.SDecimal.**HALF_UP** the amount are rounded to the nearest. The 0.5 are rounded up.
  - Banana.SDecimal.**HALF_EVEN** the amount are rounded to the nearest. The 0.5 are rounded up or down based on the preceding digit.

If the rounding context is omitted no rounding is done.

Rounding context of the accounting file

All Banana document file have a rounding context that can be retrieved with the property **Banana.document.rounding** (see Banana.document).

Examples:

```
// no context
var r = Banana.SDecimal.divide('10', '3'); // return
'3.3333333333333333333333333'

// with context
var context = {'decimals' : 4, 'mode' : Banana.SDecimal.HALF_UP};
var r = Banana.SDecimal.divide('10', '3', context);        // return '3.3333'
var r = Banana.SDecimal.divide('10', '3', {'decimals':0}); // return '3'

// use the rounding property (accunting file 2 decimals)
var r = Banana.SDecimal.divide('10', '3', Banana.document.rounding); //
return '3.33'
```

Functions

abs(value1, [, rounding])

Returns the value1 without the sign and rounded as indicated

```
var r = Banana.SDecimal.abs('-10') // return '10.00'
```

add(value1, value2 [, rounding])

Returns the sum of value1 and value2.

```
var r = Banana.SDecimal.add('6.50', '3.50'); // return '10.00'
```

compare(value1, value2)

Returns an integer value

- **1** if value1 > value2
- **0** if value1 = value2
- **-1** if value1 < value2

```
Banana.SDecimal.compare('3.50', '2'); // return '1'
Banana.SDecimal.compare('3.00', '3'); // return '0'
```

divide(value1, value2 [, rounding])

Returns value1 divided by value2.

```
var r = Banana.SDecimal.divide('6', '3'); // return '2.00'
```

isZero(value)

Returns a boolean

- **true** if value is zero
- **false** if value is not zero

```
var r = Banana.SDecimal.isZero('3.00'); // return 'false'
```

max(value1, value2 [, rounding])

Returns the max of value1 and value2.

```
var r = Banana.SDecimal.max('6', '3'); // return '6.00'
```

min(value1, value2 [, rounding])

Returns the min of value1 and value2.

```
var r = Banana.SDecimal.min('6', '3'); // return '3.00'
```

multiply(value1, value2 [, rounding])

Returns value1 multiplied by value2.

```
var r = Banana.SDecimal.multiply('6', '3'); // return '18.00'
```

remainder(value1, value2 [, rounding])

Divide value1 by value2 and returns the reminder.

```
var r = Banana.SDecimal.reminder('10', '3'); // return '1.00'
```

round(value1, [, rounding])

Returns value1 round to the spcified rounding context.

```
var r = Banana.SDecimal.round('6.123456'); // no context no rounding
r = Banana.SDecimal.round('6.123456', {'decimals':2}); // return '6.12'
```

roundNearest(value1, nearest, [, rounding])

Returns value1 round to the specified minimal amount.

```
var r = Banana.SDecimal.roundNearest('6.17', '0.1'); // return '6.1'
r = Banana.SDecimal.roundNearest('6.17', '0.05', {'decimals':2}); // return
'6.15'
```

invert(value, [, rounding])

If positive returns a negative value, if negative returns a positive value.

```
var a = Banana.SDecimal.invert('5'); //return '-5'
var b = Banana.SDecimal.invert('-2.50'); //return '2.50'
```

sign(value)

Returns an integer value

- **1** if value > 0
- **0** if value = 0
- **-1** if value < 0

```
var r = Banana.SDecimal.sign('-5'); // return '-1'
```

subtract(value1, value2 [, rounding])

Subtract value2 from value1 and returns the result.

```
var r = Banana.SDecimal.subtract('10', '3'); // return '7.00'
```

Locale conversion

To convert to and from the locale format use the Banana.Converter functions

- Banana.Converter.toInternalNumberFormat(value [, decimals, convZero])

- Banana.Converter.toLocaleNumberFormat(value [, decimalSeparator])

```
var sum = Banana.SDecimal.add('10000', '2000'); // return '12000.00'
var printValue = Banana.Converter.toLocaleNumberFormat(sum); // return
"12'000.00"
```

Banana.Script

Banana.Script represent the interface to the script file and can be accessed through Banana.script. It is used to get the parameters values defined in the script. For example if you want to print out in a report the publishing date of the script.

Properties

Methods

getParamValue(paramName)

Return the value definend in the script file of the parameter *paramName*. Return an empty string or the internal default value if the parameter is not defined. Return the first found value, if the parameter is defined multiple times.

```
Banana.script.getParamValue('pubdate'); // returns for example ' 2016-05-11'
```

getParamValues(paramName)

Return all the values defined in the script file of the param *paramName* . Return an empty array if the parameter *paramName* is not defined.

```
// Script.js example:
// ...
// @authors = Pinco
// @authors = Pallino
// ...

Banana.script.getParamValues('authors'); // returns ['Pinco', 'Pallino']
```

getParamLocaleValue(paramName)

Return the localized value definend in the script file of the param *paramName*. Return an empty string the parameter is not defined.

```
// Script.js example:
// ...
// @description = English desciption
// @description.it = Descrizione italiana
// @description.de = German Beschreibung
 ...

Banana.script.getParamLocaleValue('description'); // returns 'Descrizione italiana' for a system running with the locale 'italian'.
```

Banana.Test

The Banana.Test class is used to run unit tests of BananaApps.

BananaApps TestFramework

The BananaApps Test Framework is like an usual Unit Test Framework.

Two methologies are available:

- Verify results
  Through assert methods the user can verify some conditions, in case of a condition didn't meet the test fail and it is interrupted.
  For example you verify that a method return a determined value.

Ex.: Test.assertIsEqual(totalVatAmount(), "5000.00");

- Log results and compare them with previous results (expected results)
  Through the Banana.Test.Logger methods you can log test results. Test results are stored under the *test/testresults* folder. They will be compared at the end of the test with the expected results stored under the folder *test/testexpected* (results form previous tests), if any difference is found the test is marked as failed and the differences showed to the user.
  In this case you don't care about the exact value returned by a method, but you verify that the method returns the same value across differents versions of the BananaApp or Banana Accounting.

  Ex.: Test.logger.addKeyValue("Total VAT amount", totalVatAmount());

Create a test case

To create a test case look at the example SampleTests/TestFramework.

Run a test case

You can run a test case in two ways (both availables starting Banana Accounting 9.0.4):

- Through the Manage Apps dialog
  - Open the Manage Apps dialog
  - Select the BananaApp to test
  - Click over 'Show details'
  - Click on the button 'Run test case'

- Through the Command line
  - *banana90.exe -cmd=runtestsapps -p1=path_to_testcase.js|folder*
  - As parameter p1 you can specify the path to a test case file, or the path of a folder
  - In case of a folder all files in the folder and subfolders ending with test.js are run

Test case folder structure

This is the default test structure of a test case. All the files used for the test case are stored in a folder named `test`.

In the dialog Manage apps the button 'Run `test case' button` is showed only if the application find a file named `test/<same_name_bananaapps>.test.js`.

```
ch.banana.script.bananaapp.js                   # BananaApps
ch.banana.script.bananaapp2.js
...
test/
  ch.banana.script.bananaapp.test.js            # BananaApps Test Cases
  ch.banana.script.bananaapp2.test.js
  ...
  testcases/
    *.ac2                                       # ac2 files for the test cases
```

```
    ...
  testexpected/                              # Expected test results used for
verifying the current results
    ch.banana.script.bananaapp.test/
      *.txt
    ch.banana.script.bananaapp2.test/
      *.txt
    ...
  testresults/                               # Current test results
    ch.banana.script.bananaapp.test/
      *.txt
    ch.banana.script.bananaapp2.test/
      *.txt
    ...
```

Logger output format

The result are saved in .txt with the Latex format. Yes, it means that you can convert the output files in pdf, and look at the results without the log structure commands.

Short example

For a complete example look a SampleTests/SampleTest.

```
// @id = ch.banana.script.bananaapp.test
// @api = 1.0
// @pubdate = 2018-03-30
// @publisher = Banana.ch SA
// @description = Simple test case
// @task = app.command
// @doctype = *.*
// @docproperties =
// @outputformat = none
// @inputdataform = none
// @timeout = -1

// Register test case to be executed
Test.registerTestCase(new TestLoggerSimpleExample());

// Here we define the class, the name of the class is not important
function TestLoggerSimpleExample() {
}

// Test method, every method starting with 'test' will be automatically
executed
TestLoggerSimpleExample.prototype.testOk = function() {
   Test.logger.addText("This test will pass :-)");
   Test.assert(true);
}
```

The Test object

When a script is run as a test case, a global object named Test is exposed to the script. This object defines properties and methods for executing the test case.

```
Test.logger.addKeyValue("count", 4);
Test.assert(true);
```

Properties

logger

The property logger returns an object of type Banana.Test.Logger that permit to log test's results. If the script is not run though the Banana Apps functionality this object is null.

```
var testLogger = Test.logger;
testLogger.addKeyValue("count", 4);
```

Methods

assert(condition, message)

This method verify if condition is true. If condition is true the test continue, else an exception is thrown and the message message is inserted in the test results.

```
Test.assert(true, "This test will pass");
```

assertEndsWidth(string, endString)

This method verify if text string ends with the text endString. If the condition is true the test continue, else an exception is thrown and a fatal error is inserted in the test results.

```
Test.assertEndsWith("This string ends with the text", "the text");
```

assertIsEqual(actual, expected)

This method verify if actual iequal to expected. If the condition is true the test continue, else an exception is thrown and a fatal error is inserted in the test results.

```
Test.assertIsEqual("Those strings are equal", "Those strings are equal");
```

assertGreaterThan(actual, expected)

This method verify if actual is greather than expected. If the condition is true the test continue, else an exception is thrown and a fatal error is inserted in the test results.

```
Test.assertGreaterThan(10, 8);
```

assertGreaterThanOrEqual(actual, expected)

This method verify if actual is greather than or equal to expected. If the condition is true the test continue, else an exception is thrown and a fatal error is inserted in the test results.

```
Test.assertGreaterThanOrEqual(8, 8);
```

assertLessThan(actual, expected)

This method verify if actual less than expected. If the condition is true the test continue, else an exception is thrown and a fatal error is inserted in the test results.

```
Test.assertLessThan(8, 10);
```

assertLessThanOrEqual(actual, expected)

This method verify if actual less than or equal to expected. If the condition is true the test continue, else an exception is thrown and a fatal error is inserted in the test results.

```
Test.assertLessThanOrEqual(10, 10);
```

assertMatchRegExp(string, pattern)

This method verify if string math the regula expression defined by pattern. If the condition is true the test continue, else an exception is thrown and a fatal error is inserted in the test results.

```
Test.assertMatchRegExp("This string match the regual expression", /match/);
```

assertStartsWith(string, startString)

This method verify if text string starts with the text startString. If the condition is true the test continue, else an exception is thrown and a fatal error is inserted in the test results.

```
Test.assertStartsWith("This string start with the text", "This string");
```

registerTestCase(testCase)

This method register a testCase (an object) to be run as test.

```
// Register test case to be executed
Test.registerTestCase(new TestLoggerSimpleExample());
```

Banana.Test.Logger

The class Banana.Test.Logger contains methods to log test results.

Methods

addSection(key)

This method insert a new section namen key in the test results. A section is like a chapter 1. A section ends at the end of the test or when the method addSection is called again.

addSubSection(key)

This method insert a new subsection namen key in the test results. A section is like a chapter 1.1. A subsection ends at the end of the test or when the methods addSection or addSubSection are called again.

addSubSubSection(key)

This method insert a new subsubsection namen key in the test results. A section is like a chapter 1.1.1. A subsubsection ends at the end of the test or when the methods addSection, addSubSection or addSubSubSection are called again.

addComment(key)

This method insert a comment in the test results. Test comments are discarded when comparing with the expected test results.

addCsv(key, table [, columns, separator, comment])

This method insert the content of a csv text (comma separated values) in the test results.

The optional argument columns let you specify the subset of columns to output and in which order. The optional argument separator if defined is used as the value's separator. If it is not defined the programm automatically determine the separator from one of '\t' (tabulator), ',' (comma) and ';' (semicolon).

```
// This output all columns
Test.logger.addTable("This is a table", Banana.document, columns);
// This output only the columns Date, Description and Amount
Test.logger.addTable("This is a table", Banana.document, ["Date",
"Description", "Amount"]);
```

addInfo(key, value)

This method insert an information in the test results. Test information are discarded when comparing with the expected test results. Compared to *addComment*' the key/value information is printed in case you publish the latex result as a pdf.

```
Test.logger.addInfo("Current date", new Date().toLocaleDateString());
```

addFatalError(key)

This method insert a fatal error in the test. If a fatal error is inserted, event in case the results are identical to the expected results, the test fails, and the error message reported to the test differences.

```
Test.logger.addFatalError("This is a fatal error message");
```

addJson(key, jsonString [, comment])

This method insert a json string in the test results. The json string will formatted with identation and the formatted string outputted line by line. If the jsonString contains 'carriage return' characters, then it will be outputted as it is. If the string is not a valid json, the string is outputted as it is.

```
var obj = {
   'count': 100,
   'color': "yellow"
};
Test.logger.addJsonValue("Param", JSON.stringify(obj));
```

addKeyValue(key, value [, comment])

This method insert a test value in form of key and value. The parameter value is of type string.

```
Test.logger.addKeyValue("Row count",
Banana.document.table("Transactions").rowCount);
```

addPageBreak()

This method insert a page break. Page Breaks are discarded when comparing with the expected test results. They are just useful when the output si converted to a pdf file for inspecting visually the results.

Since Banana 9.0.4

addRawText(text [, insertEndl])

This method insert a raw string in the test results without any modification or cleaning.

The optional parameter insertEndl defaults to true. If true an endl in inserted after the text.

addReport(key, report [, comment])

This method insert the content of a Banana.Report object in the test results. Only the text elements are inserted, not the element's styles.

```
var report = Banana.Report.newReport("Report title");
report.addParagraph("Hello World !!!", "styleHelloWorld");
Test.logger.addReport("This is a report", report);
```

addTable(key, table [, columns, comment])

This method insert the content of a Banana.Table object in the test results.

The optional argument columns let you specify the subset of columns to output and in whitch order.

```
// This output all columns
Test.logger.addTable("This is a table", Banana.document, columns);
// This output only the columns Date, Description and Amount
Test.logger.addTable("This is a table", Banana.document, ["Date",
"Description", "Amount"]);
```

addText(key)

This method insert a simple string in the test results.

addXml(key, xmlString [, comment])

This method insert an xml string in the test results. The xml string will be formatted with identation and the formatted string outputted line by line. If the xmlString contains 'carriage return' characters, then it will be outputted as it is. If the string is not a valid xml, the string is outputted as it is.

```
var xml =
   "<note>" +
      "<to>Pinco</to>" +
      "<from>Pallino</from>" +
      "<heading>Reminder</heading>" +
      "<body>Don't forget me this weekend!</body>" +
   "</note>";
Test.logger.addXmlValue("This is a xml value", xml);
```

getElapsedTime()

This method return the elapsed test execution time in milliseconds.

newLogger(logname)

This method return a new logger, results are written in a separated log file named *logname*. With this methods you can split test results over several files.

If you have a lot of test resutls it is adviced to split the results over more folder and files. This make it easy to verify the differences between tests.

As a generale rule, if you feed the test with two or more *.ac2 files, split the results in separate files.

```
Test.logger.addText("This test split the results over more files and
folder");

// Write results in a new file called testresults
var testLogger = Test.logger.newLogger("testresults");
testLogger.addText("This text will be written in file testresults.txt");
testLogger.close();

// Write results in a new folder called testgroup
var groupLogger = Test.logger.newGroupLogger("testgroup");

// Write results in a new file called testgroup/testresults1
var test1Logger = groupLogger.newLogger("testresults1");
test1Logger.addText("This text will be written in file
testgroup/testresults1.txt");
test1Logger.close();

// Write results in a new file called testgroup/testresults2
var test2Logger = groupLogger.newLogger("testresults2");
test1Logger.addText("This text will be written in file
testgroup/testresults2.txt")
test2Logger.close();

groupLogger.close();
```

newGoupLogger(groupname)

This method return a new logger, results are written in a separated folder named *groupname*. With

this methods you can split test results over several folders.

If you have a lot of test resutls it is adviced to split the results over more folder and files. This make it easy to verify the differences between tests.

close()

Close the logger for writting and free the reserved system resources (handle, ,memory, ...). This method shold be called for every new logger created with the methods *newLogger* and *newGroupLogger*.

Reserved methods

Those methods are used by the BananaApps Test Framework, and should not be directly used.

addTestInfo(key, value)

This method is called automatically by the framerk to insert in a test info value like the test name, the running date and time, ...

addTestBegin(key [, comment])

This method is called automatically by the framerk to insert in the log file an indication when a test method is started.

addTestEnd()

This method is called automatically by the framerk to insert in the log file an indication when a test method is finished. The framework will also automatically insert an information about the elapsed time.

addTestCaseBegin(key [, comment])

This method is called automatically by the framerk to insert in the log file an indication when a test case is started.

addTestCaseEnd()

This method is called automatically by the framerk to insert in the log file an indication when a test case is finished. The framework will also automatically insert an information about the elapsed time.

Banana.Ui

This class Banana.Ui contains methods to interact with user interface.

Methods

createUi(uiFilePath)

Read the file uiFilepath and return an object representing the dialog. The uiFileName has to be in the same directory as the running script. For details and examples see Script dialogs.

If an error occurred undefiend is returned.

Example:

```
@includejs = ch.banana.calculator.dialog.js;  // Define the class Calculator
                                              // that control the .ui file
...
var calculatorUi = Banana.Ui.createUi("ch.banana.calculator.dialog.ui");
var calcJs = new Calculator(calculatorUi);
calclatorUi.exec();  //Show the dialog
```

getDouble(title, label [, value , min, max, decimals])

Show the user a dialog asking to insert a double. Return the inserted double or undefined if the user clicked cancel.

```
var a = Banana.Ui.getDouble("Title text", "Label text");
var b = Banana.Ui.getDouble("Title text", "Label text", "10.0");
```

getInt(title, label [, value, min, max, steps])

Show the user a dialog asking to insert a integer. Return the inserted integer or undefined if the user clicked cancel.

```
var a = Banana.Ui.getInt("Title text", "Label text");
var b = Banana.Ui.getInt("Title text", "Label text", "5", "1", "10","1");
```

getItem(title, label, items [, current, editable])

Show the user a dialog asking to select an intem from a list. Return the selected item or undefined if the user clicked cancel.

```
var value = Banana.Ui.getItem("Input", "Choose a value",
["a","b","c","d","e"], 2, false);
```

getPeriod(title, startDate, endDate [, selectionStartDate, selectionEndDate, selectionChecked])

Show the user a dialog asking to select a period like the tab Period. Return an object with the atributes 'startDate', 'endDate' and 'hasSelection' or undefined if the user clicked cancel. Date values are in the format "YYYY-MM-DD".

```
var period = Banana.Ui.getPeriod("Title text", "2016-01-01", "2016-12-31");
if (period) {
    var selectedStartDate = period.startDate; // return the start date of the
selected period
    var selectedEndDate = period.endDate; // return the end date of the
selected period
}
```

getText(title, label [, text])

Show the user a dialog asking to insert a text. Return the inserted text or undefined if the user clicked

cancel.

```
var text = Banana.Ui.getText("Title text","Label text");
```

openPropertyEditor(title, params, [dialogId])

Show the user a dialog asking to set given params. Return the modified params or undefined if the user clicked cancel.
Invoice Apps contains examples with this method (https://www.banana.ch/doc9/en/node/8381)

Param properties:

- **name**: param's key (unique id)
- **title**: param's title, which appears in the left column "property"
- **type (optional)**: string, multilinestring, bool (default: string). if boolean a checkbox will appear, if multilinestring a textarea will appear
- **value**: the value, which appears in the right column "value". For type bool: true/false, for type string and multilinestring a text
- **defaultvalue (optional)**: value used by Restore Defaults button. If the param has this property the button Restore Defaults will be available.
- **enabled (optional)**: true/false. If false the value will appears grey and the user cannot change the value (default: true)
- **editable (optional)**: true/false. If false the user cannot change the value, available only for string and multilinestring types (default: true)
- **tooltip (optional)**: text which appears over the item without clicking on the item

```
var paramList = {};
paramList.version = '1.0';
paramList.data = [];
var param = {};
param.name = 'print_header';
param.title = 'print header';
//type: bool, string, number
param.type = 'bool';
param.value = true;
param.readValue = function() {
  param.print_header = this.value;
}
paramList.data.push(param);
var dialogTitle = 'Settings';
var pageAnchor = 'dlgSettings';
Banana.Ui.openPropertyEditor(dialogTitle, paramList, pageAnchor))
for (var i = 0; i < paramList.data.length; i++) {
   // Read values to param (through the readValue function)
   paramList.data[i].readValue();
}
```

showHelp(uiFileName)

Show the help of a dialog. The help is loaded from the Banana.ch web site.

showInformation(title, msg)

Show the user an information dialog.

```
Banana.Ui.showInformation("Information", 'Insert here the text of the information.');
```

showQuestion(title, question)

Show the user a question dialog with Yes and No buttons. Return **true** if the user clicked Yes, otherwise **false**.

```
var answer = Banana.Ui.showQuestion("Question title", "Insert here the text of the question");
```

showText(text)

Show the given text in a dialog. The text can be plain text of html and span over multiple lines. If the text is in html the title is taken form the html. The dialog enable the user to save the content in the formats html, pdf, odf and txt.
The use of pixels to set the font sizes is not supported, the text is not rendered properly.

```
// Normal text
Banana.Ui.showText("Insert here the text.");

// Html text
Banana.Ui.showText('<html><header><title>This is title</title></header><body>Hello world</body></html>');
```

showText(title, text)

This is an overloaded function.

Show the given text in a dialog with the given title. The text can be plain text of html and span over multiple lines. The dialog enable the user to save the content in the formats html, pdf, odf and txt.

showText(title, text, options)

This is an overloaded function.

Show the given text in a dialog with the given title. The text can be plain text of html and span over multiple lines. The dialog enable the user to save the content in the formats html, pdf, odf and txt.

Through the object options it is possible to set the following additional parameters:

- codecName: the name of the codec to be used in case the content will be saved as txt file. Default is 'UTF-8'
- outputFileName: the file name without path to be used in case the content will be saved. The path is current open document path or the user's document path.

```
var options = {
   'codecName' : "latin1", // Default is UTF-8
   'outputFileName' : "prova.txt"
}
Banana.Ui.showText("Title", "some text...", options);
```

Banana.Xml

The Banana.Xml class is used to parse and access xml data.

Since: Banana Accounting 9.0.5

Introduction

The API Banana.Xml and Banana.Xml.XmlElement implement a subset of the DOM Document Object Model interface. The most used properties and methos are implemented.

For example the list of books in the following xml file:

```
<Library updated="2016-10-31">
   <Book>
      <Title>Paths of colours</Title>
      <Author>Rosa Indaco</Author>
   </Book>
   <Book>
      <Title>Accounting exercises</Title>
      <Author>Su Zhang</Author>
   </Book>
</Library>
```

Can be retrieved with the following code:

```
var xmlFile = Banana.Xml.parse(xml);
var xmlRoot = xmlFile.firstChildElement('Library');
var updateDate = xmlRoot.attribute('updated');
var bookNode = xmlRoot.firstChildElement('Book'); // First book
while (bookNode) {
   // For each book in the library
   var title = bookNode.firstChildElement('Title').text;
   var authorNode = bookNode.firstChildElement('Author');
   var author = authorNode ? authorNode.text : 'unknow';
   bookNode = bookNode.nextSiblingElement('Book'); // Next book
}
```

Properties

errorString

Read only. The string of the last occured error. If no error occured it is empty.

Since Banana 9.0.4

Methods

newDocument(name)

The method newDocument(name) create a new Xml document and return it as a Banana.Xml.XmlElment object.

Since Banana 9.0.4

parse(xml)

The method parse(xml) parses a xml data and returns an object of type Banana.Xml.XmlElment that represents the parsed xml. If the xml data is not valid, this method returns null, the occured error can be retrieved through the propertry errorString.

```
var xmlFile = Banana.Xml.parse(xml);
var xmlRoot = xmlFile.firstChildElement('Bookshelf'); // The root element is
named 'Bookshelf' in this example
```

save(xmlElement)

The method newDocument(name) return a Banana.Xml.XmlElment as a string.

Since Banana 9.0.4

validate(xmlElement, schemaFilePath)

The method validate(xmlElement, schemaFilePath) validate a Banana.Xml.XmlElment against a shema. The schema is passed as a path relative to the script path. The method return true if the validation passed, otherwise return false. The occured validation error can be retrieved though the property errorString.

```
// Create document
var xmlDocument = Banana.Xml.newDocument("eCH-0217:VATDeclaration");
var rootNode = xmlDocument.addElement("eCH-0217:VATDeclaration");
...

// Validate against schema (schema is passed as a file path relative to the
script)
var schemaFileName = "eCH-0217-1-0.xsd";
if (Banana.Xml.validate(xmlDocument, schemaFileName)) {
   Banana.Ui.showInformation("Validation result", "Xml document is valid
against " + schemaFileName);
} else {
   Banana.Ui.showInformation("Validation result", "Xml document is not valid
againts " + schemaFileName + ": " + Banana.Xml.errorString);
}
```

Since Banana 9.0.4

Banana.Xml.XmlElement

The XmlElement class represent an Xml element. See Banana.Xml for an example.

Since: Banana Accounting 9.0.3

Properties

nodeName

The read only property *nodeName* returns the node name of the xml element.

parent

The read only property *parent* returns the parent of this Xml element as a Banana.Xml.XmlElment object. If this is the root element, it return null.

text

The read only property *text* returns the text of this Xml element and their childs.

value

This is a synomin of the property *text*.

Methods

addProcessingInstruction(target, data)

Add a new processing instruction to the document.

```
xmlDoc.addProcessingInstruction('xml-stylesheet', 'href="mycss.css"
type="text/css"');
```

Since Banana 9.0.4

addElement(name)

Adds a new Banana.Xml.XmlElement with the specified name to the document and return it.

Since Banana 9.0.4

addElementNs(ns, name)

Adds a new Banana.Xml.XmlElement with the specified name and namespace to the document and return it.

Since Banana 9.0.4

addComment(text)

Adds a comment note to the document, and return it as a Banana.Xml.XmlElement object.

Since Banana 9.0.4

addTextNode(text)

Add a new Xml TextNode to the document and return it as a Banana.Xml.XmlElement object.

Since Banana 9.0.4

attibute(name [, defaultValue])

Returns the value of the attribute with the specified name as a string. If no attibute with the specified name is found, the defaultValue or an empty string is returned.

attibuteNS(ns, name [, defaultValue])

Returns the value of the attribute with the specified name and namespace as a string. If no attibute with the specified name is found, the defaultValue or an empty string is returned.

elementsByTagName(name)

Returns an array containing all descendants of this element with the specified name.

firstChildElement([name])

Returns the first child element with the specified name if *name* is non-empty, otherwise it returns the first child element. Returns null if no such child exists.

```
var bookNode = xmlRoot.firstChildElement('Book');  // First book
while (bookNode) {
   // For each book in the library
   var title = xmlFile.firstChildElement('Title').text();
   bookNode = bookNode.nextSiblingElement('Book');  // Next book
}
```

hasChildElements([name])

Returns true if this element contains one or mode elemets with the specified name.

hasAttribute(name)

Returns true is the attribute with the specified name exists.

hasAttributeNS(ns, name)

Returns true is the attribute with the specified name and namespace exists.

lastChildElement([name])

Returns the last child element with the specified name if *name* is non-empty, otherwise it returns the last child element. Returns null if no such child exists.

namespaceURI()

Returns the namespace URI of this node or an empty string if the node has no namespace URI.

Since Banana 9.0.4

nextSiblingElement([name])

Returns the next sibling element with the specified name if *name* is non-empty, otherwise returns any next sibling element. Returns null if no such sibling exists.

prefix()

Returns the namespace prefix of the node or an empty string if the node has no namespace prefix.

Since Banana 9.0.4

previousSiblingElement([name])

Returns the previous sibling element with the specified name if *name* is non-empty, otherwise returns any previous sibling element. Returns null if no such sibling exists.

setAttribute(name, value)

Adds an attribute with the qualified name name with the value value.

Since Banana 9.0.4

setAttributeNs(ns, name, value)

Adds an attribute with the qualified name name and the namespace URI ns with the value value.

Since Banana 9.0.4

setPrefix(value)

If the node has a namespace prefix, this function changes the namespace prefix of the node to pre. Otherwise this function does nothing.

Since Banana 9.0.4

Debugging

Output messages to the debug panel

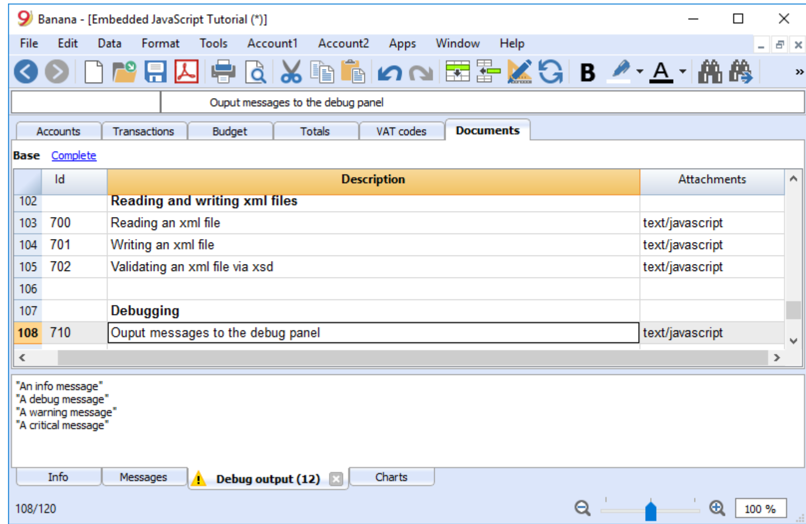
For debugging you can use the methods in Banana.Console object to output useful informations to the debug panel during the execution of the script. If you have to notify the user use instead the methods Banana.application.addMessage, Banana.document.addMessage, Banana.Document.Table.addMessage or Banana.Document.Row.addMessage

Example

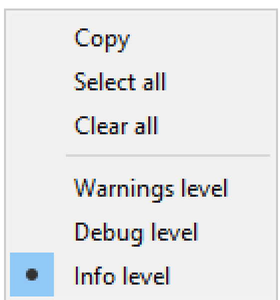
```
Banana.console.log("An info message");
Banana.console.debug("A debug message");
Banana.console.warning("A warning message");
Banana.console.crtitcal("A critical message");
```

Debug panel

The debug panel when enabled is located on the bottom of the main window near the Info and Messages panels. To open the debug panel you have to enable the option "**Display Debug output panel**" under -> Program Options -> Developer options.



In the debug panel you can choose the type of message to shows (only warnings, debug or info messages), click on the panel with the right mouse key and select the desired level.



FAQ

Can I call an external program within a BananaApp?

For the moment, for security reason we do not allow BananaApps to works directly on file and call external programs.

Can I create QML (QtQuick) apps?

With QML application have extensive access to the computer.
Fot the moment, for security reason we do not allow BananaApps to use QML.

How can I get the start and end date of the accounting?

```
var openingDate = Banana.document.info("AccountingDataBase","OpeningDate");
var closureDate = Banana.document.info("AccountingDataBase","ClosureDate");
```

Note: the keywords "AccountingsDataBase", "OpeningDate" and "ClosureDate" correspond to the values in the columns "Section Xml" and ID Xml" of the table "Info file". See command "Info table" under the menu "Tools".

Can I save and recall in a script the values entered by the user?

Yes, use the functions Banana.Document.scriptSaveSettings and Banana.Document.scriptReadSettings.

Settings are saved and restored in the current accounting file under the script id, if you change the id your settings will not be retrieved.

```
// Initialise parameter
param = {
   "searchText": "",
   "matchCase": "false",
   "wholeText": "false"
};

// Read script settings
var data = Banana.document.getScriptSettings();
if (data.length > 0) {
   param = JSON.parse(data);
}

...

// Save script settings
var paramString = JSON.stringify(param);
var value = Banana.document.setScriptSettings(paramString);
```

Accented letters are displayed wrong

Save the script file in UTF-8.

Can I protect the app?

If you don't want to someone easily change the js file, you can package it in a rcc file.

Command line

Banana can be started by giving a series of command (for a list of command and examples file see below).

> *Example: open a file*
> banana90.exe c:\temp\example.ac2

Rule for command line command

- The arguments need to be preceded by a minus "-" sign. If an argument is missing of the "-" sign, it is interpreted as the name of the file to open.
- Include the whole argument within the delimiter "…" if the text include whitespace.
- Running import as command in the command line save the accounting file on exit

If a command fail, than a return code different than 0 is returned, and the error is inserted in the log file (only if the option –log_file was used).

Examples

Example: open a file:

```
banana90.exe c:\temp\example.ac2
```

Export to xml file

```
banana90.exe -cmd=export "-cmd_file=c:\temp\my example.ac2" "-
cmd_p1=c:\temp\myexample.xml" -cmd_p2=xml -period_begin=2006-01-01
–period_end=2005-03-30
```

Example: import transactions (use the file name with the proper directory name)
Use also a log so that you know the error

```
banana90.exe -cmd=import -cmd_file="company.ac2" -cmd_table=Transactions -
cmd_p1=import_mov.txt -cmd_exit=1 -log_file=log.txt
```

For detail information regarding the import of transaction see the page "Importing in txt format".

Available Command

The argument "–cmd=…" specifies the command to be executed. The other arguments specify the option for this command.

The command can be used as a command line or a DDE request.

Argument	Description

cmd=	The command to execute • file_open (cmd_p1=noshow) • file_close (cmd_file) • file_save (cmd_file) • file_saveas (cmd_file, cmd_p1) • file_show (cmd_file) • get_tableinfo (cmd_file , cmd_table) • get_getcell (cmd_file , cmd_table, cmd_column, cmd_row) • get_getline (cmd_file , cmd_table, cmd_column, cmd_row) • get_lasterror • set_language(cmd_p1) • calc_all (cmd_file) • calc_simple (cmd_file) • deleterows (…) ***)** • export (…) • fileinfo (…) • import (…) ***)** • acc_accountcard (…) • acc_externalreport (…) • acc_vatreport (…) • version. Program return the version number and terminate. 1) Running import in the command line save the file on exit; *) If you use the commands “deleterows” and “import” directly from a command line the file is automatically saved on exit

List of arguments

Command	Argument	Description
From command line	cmd_exit=1	The program should exit and terminate Note if you use the command import= then the file that has been opened is automatically saved when the program terminate.
	nonetwork	Turn off all connections to the network (i.e. to check for updates, integrated web-server, ...)
For all commands	cmd_file=	the file to use or open
	cmd_pw=	password to open the file
	cmd_names=	A - Field name in XML (default on) a - Field name in original language
	log_file=	set the log file name for writing messages (if no file name no log)
deletelines	cmd_p1=	start of line to delete (number)
	cmd_p2=	how many lines to delete (if not present = 1)
	cmd_table=	The name of table
set_language	cmd_p1=	The two letter ISO639 language code (de, fr, en, it)
file_open	cmd_p1=	noshow – do not show the file
file_saveas	cmd_p1=	file name of saved file
get_tableinfo	cmd_table=	The name of the table to get info

get_getcell	cmd_table=	The name of the table
	cmd_row=	The number of the row, or an expression like "Account=1000:3" (In this ex. the third row where the field Account is equal to 1000 is used)
	cmd_column=	The name of the column
	cmd_op=	A – Format value (default on)
get_getline	cmd_table=	The name of the table
	cmd_row=	The number of the row, or an expression like "Account=1000:3" (In this ex. the third row where the field Account is equal to 1000 is used)
	cmd_op=	A – Format value (default on)

export	export_use_param	Instead of the default parameters use the last saved parameters (set with the dialog) and then applies the specified options with the other arguments
	cmd_p1=	file name of the export file
	cmd_p2=	Type: html, excel, xml
	cmd_table=	The name of table to export (only the table is exported)
	export_include=	Options: Upper case(A) = on; Lower Case(a) = off A - Recheck accounting (default on) B - Include statistics table (default on) C - Include transaction table (default on) D - Include account table (default on) E - Include category table (default on) F - Include total table (default on) G - Include Exchange rate table (default on) H - Inclue Vat code table and vat report (default on) I - Include Period Accounts (default on) L - Include Period VAT (default on) M - Create periods for the whole year (default off) N – Create accounts card
	export_include_ma=	number of months for accounts period, for option I, (default 1) -1 for daily
	export_include_mv=	number of months VAT period, for option L (default 3)
	export_include_mm=	max numbers of periods (default 36)
	export_op_html=	Options for html A - Use style sheet B - Use predefined style sheet (default on) C - Include style shett within html file (default on) D - Export visible fields only (default on) E - Table with borders (default on) F - Columns with headers (default on) G - Preserve page breaks within the table (default on)
	export_op_excel=	Options for Excel export A - Define cell name (default on) B - Define table name (default on) C - Use Xml names (default on) D - Protect tables (default on)
	export_op_xml=	Upper case(A) = on; Lower Case(a) = off A – Visible field only (default off) B – Include view list (default off)
	period_all	period All
	period_begin=	Begin date (yyyy-mm-dd)
	period_end=	period End date (yyyy-mm-dd)
	vat_use_param=	Instead of the default parameters use the last saved parameters (set with the dialog) and then applies the options specified with vat_op
	vat_op=	A - Include transactions B - Include total account C - Include total codes D - Include total percentage E - Use own group schema F - Only code specified G - Only group specified
	vat_sort=	sort field
	vat_text=	single code or groups (to use with –vat_op F and G)

fileinfo	cmd_op=	A - Recalculate all (default off)
import	cmd_p1=	File name of the file to import. Data have to be separated by tabulator, and the first row has to contain the name of the fields.
	cmd_p2=	Insert al line number (0=Append to end)
	cmd_op=	A - Complete imported raws
	cmd_table=	The name of table where to insert the data (Accounts, Transactions, ...)
acc_accountcard	cmd_p1=	account number
	cmd_p2=	field name for sorting
	period_all	period All (default)
	period_begin=	Begin date (yyyy-mm-dd)
	period_end=	period End date (yyyy-mm-dd)
acc_externalreport	cmd_p1=	file name of the report
	cmd_p2=	column name for grouping
	cmd_op=	Upper case(A) = on; Lower Case(a) = off A - Include account with balance = 0 B - Include account with no transactions C - Include totals only D - Notify if an account is without a group
acc_vatreport	period_all	period All (default)
	period_begin=	Begin date (yyyy-mm-dd)
	period_end=	period End date (yyyy-mm-dd)
	vat_use_param=	use parameters from existing dialog + specified options like vat_op
	vat_op=	Vat options Upper case(A) = on; Lower Case(a) = off A - Include transactions B - Include total account C - Include total codes D - Include total percentage E - Use own group schema F - Only code specified G - Only group specified H - Not used groups
	vat_sort=	sort field
	vat_text=	single code or groups (to use with –vat_op F and G)

Web Server

Excel, Word, Access and other software have the ability to integrate documents and data that is made available throught the internet protocol.
To have the possibility to retrieve the Banana Accounting data from other software, Banana include a web server, and a RESTful API, that can be accessed through http protocol.

Starting the web server

The web server enable you to access the accounting data through http.
The web server is started from the dialog Program Options unter the menu Tools.
Once the web server is started you can access to the server by typing the address "http://localhost:8081/" in your browser or in your application.

Settings

The settings of the file server, like the listening port number and others, are stored in the following

file:

Windows: "C:/Users/{user_name}/AppData/Local/Banana.ch/Banana/8.0/httpconfig.ini"
Mac: "/Users/{user_name}/Library/Application Support/Banana.ch/Banana/8.0/httpconfig.ini"
Linux: "/home/{user_name}/.local/share/data/Banana.ch/Banana/8.0/httpconfig.ini"

Resources API v1.0

/v1

Show the home page of the web server and enable you to navigate the content of the accounting files.

/v1/application[/{value_name}]

Return a JSON object with some information about the running application like 'version', 'serial', ... (since Banana 9.0.7).

Examples:

```
/v1/applicaiton
Returns:
   {
      "isbeta": false,
      "isexperimental": true,
      "name": "BananaExpm90",
      "osdetails": "Macintosh; Intel Mac OS X 10_12_4; it_CH",
      "osname": "macOS Sierra (10.12)",
      "qtversion": "5.8.0",
      "serial": "80006-170510",
      "version": "8.0.6.170510"
   }

/v1/application/serial
Returns: "80006-170510"
```

/v1/docs

Return the list of opened documents as json array.

Examples:

```
/v1/docs
Returns: ["accounting.ac2","accounting previous year.ac2", ...]
```

/v1/doc/{doc_name}

Return the list of available http requests for the file doc_name as html page.

To access the previous years files just postfix doc_name with '_p1', '_p2', ... (since Banana 9.0.6).
Deprecated: To access the previous year file just postfix doc_name with '_previous'.

Examples:

```
/v1/doc/accounting.ac2
/v1/doc/accounting.ac2_p1
/v1/doc/accounting.ac2_p2
```

/v1/doc/{doc_name}/tablenames

Return the list of tables in document doc_name as json array.

Examples:

```
/v1/doc/accounting.ac2/tablenames
Returns: ["Accounting","Transactions", ...]
```

/v1/doc/{doc_name}/table/{table_name}

Return the content of table table_name in document doc_name as html.

Parameters:

view Contains the the xml name of the view to be returned.

columns Contains the xml names of the columns to be returned.

navigation If set to true the html page navigation is showed, else only the data are showed.

format Contains the format to be returned. Supported formats are 'html' or 'json'. Default is 'html'. Since Banana 9.0.5.

Examples:

```
/v1/doc/accounting.ac2/table/Accounts
/v1/doc/accounting.ac2/table/Accounts?view=Base
/v1/doc/accounting.ac2/table/Accounts?columns=Account,Group,Description,Balan
ce
/v1/doc/accounting.ac2/table/Accounts?format=json
```

/v1/doc/{doc_name}/table/{table_name}/rowcount

Return the number of rows in the table table_name as text.

/v1/doc/{doc_name}/table/{table_name}/columnnames

Return the list of columns as json array.

/v1/doc/{doc_name}/table/{table_name}/row/{row_nr}/column/{col_name}

Return the value of cell at row row_nr and column col_name as text.

The part row_nr can be a row number starting from 1 or an expression like 'Account=1000' (In this ex. the first row where the field Account is equal to 1000 is used)

The part col_name is the xml name of the requested column.

Examples:

```
/v1/doc/accounting.ac2/table/Accounts/row/2/column/Description
/v1/doc/accounting.ac2/table/Accounts/row/Account=1000/column/Balance
```

/v1/doc/{doc_name}/table/{table_name}/rowlistnames

Return the names of row lists present in the table table_name as json array.

Examples:

```
/v1/doc/accounting.ac2/table/Transactions/rowlistnames
Returns: ["Data","Examples", "Archives", ...]
```

/v1/doc/{doc_name}/table/{table_name}/rowlist/{rowlist_name}

Return the content of the row list rowlist_name in table_name of document doc_name as html.

Parameters:

view Contains the the xml name of the view to be returned.

columns Contains the xml names of the columns to be returned.

navigation If set to true the html page navigation is showed, else only the data are showed.

format Contains the format to be returned. Supported formats are 'html' or 'json'. Default is 'html'. Since Banana 9.0.5.

Examples:

```
/v1/doc/accounting.ac2/table/Transactions/rowlist/Examples
/v1/doc/accounting.ac2/table/Transactions/rowlist/Examples?view=Base
/v1/doc/accounting.ac2/table/Accounts?columns=Account,Group,Description,Balan
ce
/v1/doc/accounting.ac2/table/Accounts?format=json
```

/v1/doc/{doc_name}/table/{table_name}/rowlist/{rowlist_name}/rowcount

Return the number of rows in the row list rowlist_name of table table_name as text.

/v1/doc/{doc_name}/table/{table_name}/rowlist/{rowlist_name}/row/{row_nr}/column/{col_name}

Return the value of cell in row list rowlist_name at row row_nr and column col_name as text.

The part row_nr can be a row number starting from 1 or an expression like 'Account=1000' (In this ex. the first row where the field Account is equal to 1000 is used)

The part col_name is the xml name of the requested column.

Examples:

```
/v1/doc/accounting.ac2/table/Accounts/row/2/column/Description
/v1/doc/accounting.ac2/table/Accounts/row/Account=1000/column/Balance
```

/v1/doc/{doc_name}/accounts

Return the list of accounts as json array.

Examples:

```
/v1/doc/accounting.ac2/accounts
Returns: [{"id":"1000","descr":"1000 Cash"}, {"id":"1010","descr":"1000
Post"}, ...]
```

/v1/doc/{doc_name}/accountdescription/{account_id|Gr=group_id}[/{col_name}]

Return the description of the requested account or group as text.
The part col_name is optional, it is the xml name of the requested column. Default is the column 'Description'.

Examples:

```
/v1/doc/accounting.ac2/accountdescription/1000
/v1/doc/accounting.ac2/accountdescription/Gr=1
/v1/doc/accounting.ac2/accountdescription/1000/Curreny
```

/v1/doc/{doc_name}/groups

Return the list of groups as json array.

Examples:

```
/v1/doc/accounting.ac2/groups
Returns: [{"id":"100","descr":"100 Current Assets"}, ...]
```

/v1/doc/{doc_name}/segments

Return the list of segments as json array.

Examples:

```
/v1/doc/accounting.ac2/segments
Returns: [{"id":":lugano","descr":"Lugano"}, ...]
```

/v1/doc/{doc_name}/vatcodes

Return the list of vatcodes as json array.

Examples:

```
/v1/doc/accounting.ac2/vatcodes
Returns: [{"id":"V80","descr":"V80 Sales and services 8.0%"}, ...]
```

/v1/doc/{doc_name}/vatdescription/{vat_code}[/{col_name}]

Return the description of the requested vat code as text.
The part col_name is optional, it is the xml name of the requested column. Default is the column 'Description'.

Examples:

```
/v1/doc/accounting.ac2/vatdescription/V80
/v1/doc/accounting.ac2/vatdescription/V80/Gr1
```

/v1/doc/{doc_name}/balance/{account_id|Gr=group_id|BClass=class_id}/{opening|credit|debit|total|balance|openingcurrency|...}

Return the current balance of the requested account, group or bclass as text.

To access the balances of the previous year file just postfix doc_name with '_p1', '_p2', ...

The last part or the url can be one of the followings strings:

- opening
- credit
- debit
- total
- balance
- openingcurrency
- ceditcurrency
- debitcurrency
- totalcurrency
- balancecurrency
- rowcount

Parameters:

period Define the start and end date for the request.
It can contain a period abbreviation like 'Q1' or a start and end date like '2014-01-01/2014-03-31'.
A period abbreviation is defined by a number followed by one of the following charachters:

- M for months
- Q for quarters
- S for semesters
- Y for years

frequency Define the frequency for the request.
The amounts are calculated at the given frequency and returned as an array.
Frequency abbreviation contains one of the following charachters:

- D for daily
- W for weekly
- M for monthly
- Q for quarterly
- S for semesterly
- Y for yearly

filter Contains a javascript expression used to filter the transactions. The object available to the expression are "row", "rowNr", and "table".
For example: filter=row.value("Date")==="2014-01-15"

Examples:

```
/v1/doc/accounting.ac2/balance/1000/opening
/v1/doc/accounting.ac2_p1/balance/1000/opening
/v1/doc/accounting.ac2/balance/1000|1010|1020|1030/opening
/v1/doc/accounting.ac2/balance/Gr=6/totalcurrency?period=Q1
/v1/doc/accounting.ac2/balance/Gr=6/totalcurrency?frequency=M
/v1/doc/accounting.ac2/balance/Gr=6/totalcurrency?period=M1&frequency=D
/v1/doc/accounting.ac2/balance/BClass=1/balance
```

/v1/doc/{doc_name}/budget/{account_id|Gr=group_id|BClass=class_id}/{opening|credit|debit|total|balance|openingcurrency|...}

Return the budget of the requested account, group or bclass as text.

To access the budget balances of the previous year file just postfix doc_name with '_p1', '_p2', ...

The last part or the url can be one of the followings strings:

- opening
- credit
- debit
- total
- balance
- openingcurrency
- ceditcurrency
- debitcurrency
- totalcurrency
- balancecurrency
- rowcount

Parameters:

period Define the start and end date for the request.
It can contain a period abbreviation like 'Q1' or a start and end date like '2014-01-01/2014-03-31'.
A period abbreviation is defined by a number followed by one of the following charachters:

- M for months
- Q for quarters
- S for semesters
- Y for years

frequency Define the frequency for the request.
The amounts are calculated at the given frequency and returned as an array.
Frequency abbreviation contains one of the following charachters:

- D for daily
- W for weekly
- M for monthly

- Q for quarterly
- S for semesterly
- Y for yearly

filter Contains a javascript expression used to filter the transactions. The object available to the expression are "row", "rowNr", and "table".
For example: filter=row.value("Date")==="2014-01-15"

Examples:

```
/v1/doc/accounting.ac2/budget/1000/opening
/v1/doc/accounting.ac2_p1/budget/1000/opening
/v1/doc/accounting.ac2/budget/1000|1010|1020|1030/opening
/v1/doc/accounting.ac2/budget/Gr=6/totalcurrency?period=Q1
/v1/doc/accounting.ac2/budget/Gr=6/totalcurrency?frequency=M
/v1/doc/accounting.ac2/budget/BClass=1/balance
```

/v1/doc/{doc_name}/interest/{account_id|Gr=group_id|BClass=class_id}

Return the calculated interest on the specified account.

Parameters:

rate The interest rate in percentage (ie.: '5', '3.25'). The decimal separator must be a dot '.'. If positive it calculate the interest fo the debit amounts. If negative it calcaulate the interest on the credits amounts.

period Define the start and end date for the request.
It can contain a period abbreviation like 'Q1' or a start and end date like '2014-01-01/2014-03-31'.
A period abbreviation is defined by a number followed by one of the following charachters:

- M for months
- Q for quarters
- S for semesters
- Y for years

frequency Define the frequency for the request.
The amounts are calculated at the given frequency and returned as an array.
Frequency abbreviation contains one of the following charachters:

- D for daily
- W for weekly
- M for monthly
- Q for quarterly
- S for semesterly
- Y for yearly

filter Contains a javascript expression used to filter the transactions. The object available to the expression are "row", "rowNr", and "table".
For example: filter=row.value("Date")==="2014-01-15"

Examples:

```
/v1/doc/accounting.ac2/interest/1000?rate=2.5
/v1/doc/accounting.ac2/interest/1000?rate=-8.0
/v1/doc/accounting.ac2/interest/1000?rate=-8.0&period=Q1
/v1/doc/accounting.ac2/interest/1000?rate=-8.0&frequency=Q
```

/v1/doc/{doc_name}/budgetinterest/{account_id|Gr=group_id|BClass=class_id}

Return the calculated interest on the specified account for the budget transactions.

Parameters:

rate The interest rate in percentage (ie.: '5', '3.25'). The decimal separator must be a dot '.'. If positive it calculate the interest fo the debit amounts. If negative it calcaulate the interest on the credits amounts.

period Define the start and end date for the request.
It can contain a period abbreviation like 'Q1' or a start and end date like '2014-01-01/2014-03-31'.
A period abbreviation is defined by a number followed by one of the following charachters:

- M for months
- Q for quarters
- S for semesters
- Y for years

frequency Define the frequency for the request.
The amounts are calculated at the given frequency and returned as an array.
Frequency abbreviation contains one of the following charachters:

- D for daily
- W for weekly
- M for monthly
- Q for quarterly
- S for semesterly
- Y for yearly

filter Contains a javascript expression used to filter the transactions. The object available to the expression are "row", "rowNr", and "table".
For example: filter=row.value("Date")==="2014-01-15"

Examples:

```
/v1/doc/accounting.ac2/budgetinterest/1000?rate=2.5
/v1/doc/accounting.ac2/budgetinterest/1000?rate=-8.0
/v1/doc/accounting.ac2/budgetinterest/1000?rate=-8.0&period=Q1
/v1/doc/accounting.ac2/budgetinterest/1000?rate=-8.0&frequency=Q
```

/v1/doc/{doc_name}/projection/{account_id|Gr=group_id|BClass=class_id}/{opening|credit|debit|total|balance|openingcurrency|...}

Return the projection of the requested account, group or bclass as text.

To access the budget balances of the previous year file just postfix doc_name with '_p1', '_p2', ...

The last part or the url can be one of the followings strings:

- opening
- credit
- debit
- total
- balance
- openingcurrency
- ceditcurrency
- debitcurrency
- totalcurrency
- balancecurrency
- rowcount

Parameters:

projectionstart This parameter is mandatory and define the start date of the projection.
It can contain a period abbreviation like 'Q1' (start at beginnig of) or a date like '2014-07-01'.

period Define the start and end date for the request.
It can contain a period abbreviation like 'Q1', a start and end date like '2014-01-01/2014-03-31' or a list of periods separated by a coma like 'S1,S2,ALL'.
A period abbreviation is defined by a number followed by one of the following charachters:

- M for months
- Q for quarters
- S for semesters
- Y for years

frequency Define the frequency for the request.
The amounts are calculated at the given frequency and returned as an array.
Frequency abbreviation contains one of the following charachters:

- D for daily
- W for weekly
- M for monthly
- Q for quarterly
- S for semesterly
- Y for yearly

filter Contains a javascript expression used to filter the transactions. The object available to the expression are "row", "rowNr", and "table".
For example: filter=row.value("Date")==="2014-01-15"

Examples:

```
/v1/doc/accounting.ac2/projection/1000/opening?projectionstart=S1
/v1/doc/accounting.ac2/projection/1000/opening?projectionstart=2014-07-01
/v1/doc/accounting.ac2_p1/projection/1000/opening?projectionstart=S1
/v1/doc/accounting.ac2_p1/projection/1000/opening?projectionstart=S1&frequenc
y=M
```

/v1/doc/{doc_name}/accountcard/{account_id}

Return the account card of account account_id as html.

Parameters:

view Contains the the xml name of the view to be returned.

columns Contains the xml names of the columns to be returned.

navigation If set to true the html page navigation is showed, else only the data are showed.

period Define the start and end date for the request.
It can contain a period abbreviation like 'Q1' or a start and end date like '2014-01-01/2014-03-31'.
A period abbreviation is defined by a number followed by one of the following charachters:

- M for months
- Q for quarters
- S for semesters
- Y for years

filter Contains a javascript expression used to filter the transactions. The object available to the expression are "row", "rowNr", and "table".
For example: filter=row.value("Date")==="2014-01-15"

format Contains the format to be returned. Supported formats are 'html' or 'json'. Default is 'html'.
Since Banana 9.0.5.

Examples:

```
/v1/doc/accounting.ac2/accountcard/1000
/v1/doc/accounting.ac2/accountcard/1000?period=Q1
/v1/doc/accounting.ac2/accountcard/1000?period=2014-01-01/2014-03-31
/v1/doc/accounting.ac2/accountcard/1000?filter=row.value("Description").conta
in("xyz")
/v1/doc/accounting.ac2/accountcard/1000?format=json
```

/v1/doc/{doc_name}/budgetcard/{account_id}

Return the budget card of account account_id as html.

Parameters:

view Contains the the xml name of the view to be returned.

columns Contains the xml names of the columns to be returned.

navigation If set to true the html page navigation is showed, else only the data are showed.

period Define the start and end date for the request.
It can contain a period abbreviation like '1Q' or a start and end date like '2014-01-01/2014-03-31'.
A period abbreviation is defined by a number followed by one of the following charachters:

- M for months
- Q for quarters
- S for semesters

- Y for years

filter Contains a javascript expression used to filter the transactions. The object available to the expression are "row", "rowNr", and "table".
For example: filter=row.value("Date")==="2014-01-15"

format Contains the format to be returned. Supported formats are 'html' or 'json'. Default is 'html'. Since Banana 9.0.5.

Introduced in 7.0.7.0

Examples:

```
/v1/doc/accounting.ac2/budgetcard/1000
/v1/doc/accounting.ac2/budgetcard/1000?period=Q1
/v1/doc/accounting.ac2/budgetcard/1000?period=2014-01-01/2014-03-31
/v1/doc/accounting.ac2/budgetcard/1000?format=json
```

/v1/doc/{doc_name}/projectioncard/{account_id}

Return the projection card of account account_id as html.

Parameters:

view Contains the the xml name of the view to be returned.

columns Contains the xml names of the columns to be returned.

navigation If set to true the html page navigation is showed, else only the data are showed.

projectionstart This parameter is mandatory and define the start date of the projection.
It can contain a period abbreviation like 'Q1' (start at beginnig of) or a date like '2014-07-01'.

period Define the start and end date for the request.
It can contain a period abbreviation like '1Q' or a start and end date like '2014-01-01/2014-03-31'.
A period abbreviation is defined by a number followed by one of the following charachters:

- M for months
- Q for quarters
- S for semesters
- Y for years

filter Contains a javascript expression used to filter the transactions. The object available to the expression are "row", "rowNr", and "table".
For example: filter=row.value("Date")==="2014-01-15"

format Contains the format to be returned. Supported formats are 'html' or 'json'. Default is 'html'. Since Banana 9.0.5.

Introduced in 7.0.7.0

Examples:

```
/v1/doc/accounting.ac2/projectioncard/1000
```

```
/v1/doc/accounting.ac2/projectioncard/1000?period=Q1
/v1/doc/accounting.ac2/projectioncard/1000?period=2014-01-01/2014-03-31
/v1/doc/accounting.ac2/projectioncard/1000?format=json
```

/v1/doc/{doc_name}/vatbalance/{vat_code|Gr=vat_group}/{taxable|amount|notdeductible|posted}

Return the current balance of the requested vat code as text.

The last part of the url can be one of the followings strings:

- taxable
- amount
- notdeductible
- posted
- rowcount

Parameters:

period Define the start and end date for the request.
It can contain a period abbreviation like 'Q1' or a start and end date like '2014-01-01/2014-03-31'.
A period abbreviation is defined by a number followed by one of the following charachters:

- M for months
- Q for quarters
- S for semesters
- Y for years

frequency Define the frequency for the request.
The amounts are calculated at the given frequency and returned as an array.
Frequency abbreviation contains one of the following charachters:

- D for daily
- W for weekly
- M for monthly
- Q for quarterly
- S for semesterly
- Y for yearly

filter Contains a javascript expression used to filter the transactions. The object available to the expression are "row", "rowNr", and "table".
For example: filter=row.value("Date")==="2014-01-15"

Examples:

```
/v1/doc/accounting.ac2/vatbalance/V80/balance
```

/v1/doc/{doc_name}/vatbudget/{vat_code|Gr=vat_group}/{taxable|amount|notdeductible|posted}

Return the budget of the requested vat code as text.

The last part or the url can be one of the followings strings:

- taxable
- amount
- notdeductible
- posted
- rowcount

Parameters:

period Define the start and end date for the request.
It can contain a period abbreviation like 'Q1' or a start and end date like '2014-01-01/2014-03-31'.
A period abbreviation is defined by a number followed by one of the following charachters:

- M for months
- Q for quarters
- S for semesters
- Y for years

frequency Define the frequency for the request.
The amounts are calculated at the given frequency and returned as an array.
Frequency abbreviation contains one of the following charachters:

- D for daily
- W for weekly
- M for monthly
- Q for quarterly
- S for semesterly
- Y for yearly

filter Contains a javascript expression used to filter the transactions. The object available to the expression are "row", "rowNr", and "table".
For example: filter=row.value("Date")==="2014-01-15"

Examples:

```
/v1/doc/accounting.ac2/vatbudget/V80/balance
```

/v1/doc/{doc_name}/vatcard/{vat_code}

Return the account card of the vat code vat_code as html.

Parameters:

view Contains the the xml name of the view to be returned.

columns Contains the xml names of the columns to be returned.

navigation If set to true the html page navigation is showed, else only the data are showed.

period Define the start and end date for the request.
It can contain a period abbreviation like 'Q1' or a start and end date like '2014-01-01/2014-03-31'.
A period abbreviation is defined by a number followed by one of the following charachters:

- M for months
- Q for quarters
- S for semesters
- Y for years

filter Contains a javascript expression used to filter the transactions. The object available to the expression are "row", "rowNr", and "table".
For example: filter=row.value("Date")==="2014-01-15"

format Contains the format to be returned. Supported formats are 'html' or 'json'. Default is 'html'.
Since Banana 9.0.5.

Examples:

```
/v1/doc/accounting.ac2/vatcard/V80
/v1/doc/accounting.ac2/vatcard/V0|V25|V80
/v1/doc/accounting.ac2/vatcard/V80?period=Q1
/v1/doc/accounting.ac2/vatcard/V80?period=2014-01-01/2014-03-31
/v1/doc/accounting.ac2/vatcard/V80?filter=row.value("Description").contain("x
yz")
/v1/doc/accounting.ac2/vatcard/V80?format=json
```

/v1/doc/{doc_name}/vatprojection/{vat_code|Gr=vat_group}/{taxable|amount|not deductible|posted}

Return the projection of the requested vat code as text.

The last part or the url can be one of the followings strings:

- taxable
- amount
- notdeductible
- posted
- rowcount

Parameters:

projectionstart This parameter is mandatory and define the start date of the projection.
It can contain a period abbreviation like 'Q1' (start at beginnig of) or a date like '2014-07-01'.

period Define the start and end date for the request.
It can contain a period abbreviation like 'Q1' or a start and end date like '2014-01-01/2014-03-31'.
A period abbreviation is defined by a number followed by one of the following charachters:

- M for months
- Q for quarters
- S for semesters
- Y for years

frequency Define the frequency for the request.
The amounts are calculated at the given frequency and returned as an array.
Frequency abbreviation contains one of the following charachters:

- D for daily
- W for weekly
- M for monthly
- Q for quarterly
- S for semesterly
- Y for yearly

filter Contains a javascript expression used to filter the transactions. The object available to the expression are "row", "rowNr", and "table".
For example: filter=row.value("Date")==="2014-01-15"

Examples:

```
/v1/doc/accounting.ac2/vatprojection/V80?startdate=2016-18-31
/v1/doc/accounting.ac2/vatprojection/V80/balance?startdate=Q3
```

/v1/doc/{doc_name}/accreport

Return the accounting report for the document doc_name as html.

Parameters:

view Contains the the xml name of the view to be returned.

columns Contains the xml names of the columns to be returned.

navigation If set to true the html page navigation is showed, else only the data are showed.

period Define the start and end date for the request.
It can contain a period abbreviation like '1Q' or a start and end date like '2014-01-01/2014-03-31'.
A period abbreviation is defined by a number followed by one of the following charachters:

- M for months
- Q for quarters
- S for semesters
- Y for years

subdivision Define the period subdivision for the request .A period subdivision is defined by one of the following charachters:

- M for monthly subdivision
- Q for quarter sudbivision
- S for semester subdivision
- Y for year subdivision

format Contains the format to be returned. Supported formats are 'html' or 'json'. Default is 'html'.
Since Banana 9.0.5.

Examples:

```
/v1/doc/accounting.ac2/accreport
```

```
/v1/doc/accounting.ac2/accreport?period=Q1
/v1/doc/accounting.ac2/accreport?subdivision=Q
/v1/doc/accounting.ac2/accreport?format=json
```

/v1/doc/{doc_name}/vatreport

Return the vat report for the document doc_name as html.

Parameters:

view Contains the the xml name of the view to be returned.

columns Contains the xml names of the columns to be returned.

navigation If set to true the html page navigation is showed, else only the data are showed.

period Define the start and end date for the request.
It can contain a period abbreviation like '1Q' or a start and end date like '2014-01-01/2014-03-31'.
A period abbreviation is defined by a number followed by one of the following charachters:

- M for months
- Q for quarters
- S for semesters
- Y for years

format Contains the format to be returned. Supported formats are 'html' or 'json'. Default is 'html'.
Since Banana 9.0.5.

Examples:

```
/v1/doc/accounting.ac2/vatreport
/v1/doc/accounting.ac2/vatreport?period=Q3
/v1/doc/accounting.ac2/vatreport?format=json
```

/v1/doc/{doc_name}/journal

Return the journal for the document doc_name as html.

Parameters:

columns Contains the xml names of the columns to be returned.

navigation If set to true the html page navigation is showed, else only the data are showed.

format Contains the format to be returned. Supported formats are 'html' or 'json'. Default is 'html'.
Since Banana 9.0.5.

Examples:

```
/v1/doc/accounting.ac2/journal
/v1/doc/accounting.ac2/journal?format=json
```

doc/{doc_name}/startperiod

Return the start date in the form of 'YYYY-MM-DD'.

Parameters:

period Define the period for the request.
It can contain a period abbreviation like '1Q' or be empry. If period is not present or empty the accounting start date is returned.
A period abbreviation is defined by a number followed by one of the following charachters:

- M for months
- Q for quarters
- S for semesters
- Y for years

See also endPeriod.

Introduced in 7.0.7.0

Examples:

```
/v1/doc/accounting.ac2/startperiod
/v1/doc/accounting.ac2/startPeriod?period=Q3
```

doc/{doc_name}/endPeriod

Return the end date in the form of 'YYYY-MM-DD'.

Parameters:

period Define the period for the request.
It can contain a period abbreviation like '1Q' or be empry. If period is not present or empty the accounting end date is returned.
A period abbreviation is defined by a number followed by one of the following charachters:

- M for months
- Q for quarters
- S for semesters
- Y for years

See also startPeriod.

Introduced in 7.0.7.0

Examples:

```
/v1/doc/accounting.ac2/endPeriod
/v1/doc/accounting.ac2/endperiod?period=Q3
```

/v1/doc/{doc_name}/info

Return the info table as html.

Parameters:

columns Contains the xml names of the columns to be returned.

navigation If set to true the html page navigation is showed, else only the data are showed.

format Contains the format to be returned. Supported formats are 'html' or 'json'. Default is 'html'. Since Banana 9.0.5.

Examples:

```
/v1/doc/accounting.ac2/info
/v1/doc/accounting.ac2/info?format=json
```

/v1/doc/{doc_name}/info/{info_section_name}/{info_id_name}

Return the info value section:id as text.

Examples:

```
/v1/doc/accounting.ac2/info/AccountingDataBase/BasicCurrency
```

/v1/doc/{doc_name}/infos

Return the infos of document doc_name as json object.

Examples:

```
/v1/doc/accounting.ac2/infos
Returns: [{"section":"Base", "id":"FileInfo", "value":""},{"section":"Base",
"id":"Date", "value":"2014-05-13"}, ...]
```

/v1/doc/{doc_name}/messages

Return the list of message as a json array

Parameters:

recheck If set to 'yes' a 'Recheck accounting' is executed.

/v1/doc/{doc_name}/messages/count

Return the number of error messages in the accounting file.

Parameters:

recheck If set to 'yes' a 'Recheck accounting' is executed.

/v1/doc/{doc_name}/apps/{app_name}

Return the www app app_name as a html page. Www apps are a showcase of the powerfull capability of the http api.

A www app is just a html page stored under '{program_folder}/WWW' that is returned by this request. Currently banana accounting has the app "Charts" that permit to display charts of accounts, and the app "Dashboard" that show an overview of the accounting.

```
/v1/doc/accounting.ac2/apps/charts
/v1/doc/accounting.ac2/apps/dashboard
```

/v1/doc/{doc_name}/[*/]bananaapiv1.js

Return a javascript file tha define an object oriented interface to access the webserver. With this interface the http requests are hidden behind the object's methods and properties. Methods and properties follow the schema of the BananaApps API. All the request are synchronous.

```
/v1/doc/accounting.ac2/bananaapiv1.js
/v1/doc/accounting.ac2/charts/bananaapiv1.js // note: only the name is
checked and not the path
```

/v1/appdata/{data_id}

The request appdata permit to save and restore parameters.

```
GET /v1/appdata/chart_xyz      // return the data chart_xyz
PUT /v1/appdata/chart_xyz      // save some data, the data are sent as body
of the request
DELETE /v1/appdata/chart_xyz   // delete the data chart_xyz
```

/v1/appdataform

The request return a form that permit to create, modify or delete app data.

/v1/files/{file_name}

The request return the file file_name stored in the folder user data.

Windows: "C:/Users/{user_name}/AppData/Local/Banana.ch/Banana/8.0/httpconfig.ini"
Mac: "/Users/{user_name}/Library/Application Support/Banana.ch/Banana/8.0/httpconfig.ini"
Linux: "/home/{user_name}/.local/share/data/Banana.ch/Banana/8.0/httpconfig.ini"

/v1/help

Show the help page (this page) of the web server.

/v1/script?scriptfile=path_to_script

Execute the script scriptfile.

Parameters:

scriptfile Define the path of the script to be executed.The path can be an absolute path or a path relative to the user's document directory.

ac2file Define the name of an opened document to be passed to the script. It is optional.

indata Contains text data to be passed to the script. It is optional.

Examples:

```
/v1/script?scriptfile=getresults.js
/v1/script?scriptfile=getresults.js&ac2file=accounting.ac2
```

/v1/settings

Show the settings of the web server.

Data formats API

Date

Date values are in ISO 8601 format 'YYYY-MM-DD'.

Decimal

Decimal values have a '.' (dot) as decimal separator and doesn't have a group separator. For example: '12345.67'.
Decimal values are rounded according to the accounting settings.

Text

Text values can contain any character supported by UTF-8.

Time

Time values are in ISO 8601 format 'HH:MM:SS'. The formats 'HH:MM' and 'HH:MM:SS.ZZZ are also accepted.

Import data

Banana Accounting is a very suitable companion for business solutions that do not provide an integrated accounting solution. Banana Accounting software offers different advantages, therefore many software houses, independent or in house solutions developers offer an integration mechanism and market their solution with Banana:

- Solutions developers can provide a complete business solution, without the need to develop a new accounting module.
- Banana Accounting is very versatile, easy to use, provides many functionalities and it is also affordable.
- In Banana Accounting Importing is a primary function, that is very much advanced.
    - It supports different formats.
    - Once the data imported has been imported it can still be verified and changed by the users.
      Implementing the export is much easier, special cases or errors due to incorrect configurations, can be modified by the users.
      Users can fix the problem directly and the developer can later provide a new patch release. This approach makes supporting the export to Banana much easier and less stressful.
    - Import can be undone and repeated, later.
      During the import one sometimes happens to see error in the original data. The user

undoes the import, fixes the data and repeats the export and import.
- Banana Accounting includes a Web server and an API that can be used to retrieve information. Prior to exporting the data it is possible to verify the status of the accounting.

Currently, the only way to integrate data from other solutions is to use the import function. Due to the flexibility and possibility for users to check and modify the imported data, this has been proven to be a better solution.
Therefore, we did not make available an API for integrating data. Kindly let us know user cases where this would be a better solution.

For the development needs you can download the Banana Accounting software version available on our web site. You can use all functionalities and save up to 70 transactions. This version is normally sufficient for the development and testing process.

User cases

Generally the integration is related to:

- CRM with invoicing or Online Web shop.
  Customer sales are exported in TXT format and imported in Banana Accounting.
  - Customer information is exported as an accounts data and imported into the Accounts table
  - Invoice information or payments are exported as transactions and imported into the Transactions table.
- Payroll and Salary
  End of month's summary data is exported as transactions and imported into Banana Accounting.
- Accounting data from another accounting software is integrated in Banana Accounting.

Other documentation

- End user general import documentation.
- End user documentation for importing accounting data.

How to proceed

If you are a developer and do want to test if the import file is correct:

- Download and install the Banana Software. In Starter Edition mode you can Create a new accounting file, use the import into accounting function and operate all tests you need.
- In case you are an independent software developer and need full functionalities we can provide a time-limited full license.
  Request must come with a link to the developer web site.

How to for

- How to read the balance for an account / customer / supplier
  Use the Web API.
  Replace the "1000" with the account number.
  "/v1/doc/accounting.ac2/table/Accounts/row/Account=1000/column/Balance"
- How to get about Accounts, VatCodes
  Use the Web API.

Replace the "Accounts" with the table you need to query.
"/v1/doc/accounting.ac2/table/Accounts?format=json"
- How to add a customer or supplier
    - Create and export file in format TXT for the Accounts table, with the customers and suppliers.

ImportApp

An importApp is a javascript program that is executed internally by Banana (when the user wants to import data) and usually converts data from a proprietary format to a Banana "Text file with columns header".

You can create an ImportApp that does more sofisticated things like:

- Applying an account number based on the content .
  For example set the ContraAccount to "3000" if the amount is positive and the "Description" starts with "Revenue from "
- Use data contained in another Banana table to complete the transactions.
    - Create a table with (Tools->Add new functionalities->Add simple table) where you list text to look for in the transactions texts and the corresponding account to use.
    - Assume you invoice data using a customer number that is different from the account number in the Account table.
      Add a new column in the Account table and then use the content to retrieve the appropriate account number.
    - Adding a prefix to the supplier invoice number, so that you can easily distinguish incoming and outgoing invoices.

For more information see creating an ImportApp

Import "Text file with columns header"

The Banana import format is of type "**Text file with column headers**".

- Fields and column name separator is to be tab separated "\t"
- Each line (after a "\n") is a new record
- Character code preferably UTF8 or else the local one.
- The first line contains the columns header name
    - You can use any column name existing on the table
    - Names are case sensitive and must correspond to the Banana NameXml (English) of the column.
- Starting from line 2 it contains the data to be imported
    - The format for the Date fields is yyyy-mm-dd
    - The decimal separator is the decimal point "."
    - Amount should not have any thousand separator

Two commands for import

1. Through the Command in Menu Account1->Import in Accounting.
   This is a specialized import for accounting data, with postprocessing of that suitable for the accounting data.
    - Import Accounts, VatCodes, Exchange rates, Invoice numbers
      It uses the column name as in the transactions table

        - Transactions
          You have two options:
            - Double-entry accounting format (same as in the double entry transactions table)
            - Income/Expense accounting (for bank statements)
2. Through the command Data import. Import from txt.
   You can import data in any table. For accounting data, prefer option 1.

Import Double-entry transactions in CSV format

For what is concerning the specifics of the import of Double-entry see the explanations for **Import Double-entry accounting transactions**. The only difference is that the **Complete transactions with** section is active and allows the user to enter the initial document number and the destination account number. Once the import is done, the counterparty account will have to be entered manually.

Menu Account1->**Import into accounting**

The type of file to be used is a "**Text file with column headers**".

Main columns for import

For importing invoices the tab separated import file or translated file usually contains these columns:

- **Date** of the transaction (2014-12-31).
- **DateDocument** with the date of the invoice.
- **DocInvoice** the invoice number.
- **Description** a brief text.
- **AccountDebit** the account number of the customer or the general account for customers.
- **AccountCredit** the account number of the revenue account.
- **Amount** the amount of the accounting currency.
- **VatCode** the vat code that should be used.
- **AmountCurrency** if multi-currency the amount of the invoice in original currency and currency of the AccountDebit.

Example file Double-entry format

Fields are separated by the tab char (ASCII decimal 11, C language "\t").
In the example the tab character is not visible.

```
Date Doc Description AccountDebit AccountCredit Amount VatCode
VatPercentNonDeductible

2018-01-03 Bank to Cash 1000 1020 350.00
2018-01-05 Office Supplies 6500 1000 32.50 E76
```

Options

- Import using clipboard data will use the content of the clipboard instead of the file
- Autocomplete values: Some fields of the transactions are automatically completed (see "Importing transactions for multicurrency Double-entry accounting").
- Unicode (utf-8) The content of the file is in Unicode utf-8 (it supports any character set).

Importing other transaction's columns

You can import any other field that is defined in the Transactions table.
There are other values that we suggest to import if available:

- **DateDocument** the date of the original document (for example the date of the invoice).
- **DocOriginal** the document number for example the invoice number.
- **DocPaid** the document number that has been paid.
- **DocLink** the address of the file that links to a scanned document (pdf, jpg, ..).
- **DateExpiration** due date of the invoice.
- **ExternalReference** an information that help to identify the transactions as unique.
  It will be used in future implementation of Banana (in conjunction with the date) to warn a user that the transaction has already been imported.
  This should be an externaReference generated by the software that creates the transactions to be imported.
  We suggest to use a name of the program and a number that is unique for the accounting period.
  For example "invoice-2013-00001.001" with year, invoice number and a progressive number that is relative to the invoice in case it will be necessary to have more transaction lines for the same invoice.

Importing transactions for multicurrency Double-entry accounting

By importing multicurrency data there can be rounding or calculation differences due to different development tools used. To avoid such differences you should provide only certain fields and while importing the program will calculate the field values that are missing (with the option "Autocomplete values") .

- If you provide only "AmountCurrency" the program will use the default exchange rate and will calculate the "Amount".
- In order to avoid error provide always the "ExchangeCurrency"
- If you provide the "AmountCurrency" and the "ExchangeRate" and the "Amount" are 0 or not present the program will calculate the exchange rate based on the column "Amount" and "AmountCurrency".

Importing Invoice data

The data of your invoice software can be imported in Banana.
There are two ways to do so:

1. Let your invoice software generate a file for Banana as indicated in the "Import Double-entry transactions in txt format".
2. Use the data of the export format of your existing invoicing software .
   In order to import this data from a proprietary format into Banana you need to create a Javascript BananaApp that translates the data into a format acceptable for Banana.
   The script program takes as input the content of a file and creates an output that is a tab separated text file with columns headers.
   See also repository on github.

Invoices on more lines

Most invoices have different items that need to be registered in different revenue accounts or that have different VAT percentages.
In this case, for each invoice you need to have many import lines.

Date, DateDocument, DocInvoice have always the same values.

- The first line you have the
  - AccountDebit the customer account number
  - AccountCredit is void.
  - Amount the total amount of the invoice. The amount due from the Customer.
  - VatCode is void
- For each item with a different revenue account or Vat percentage you should have an additional line
  - AccountDebit is void
  - AccountCredit the revenue account for this item
  - Amount the total amount to be registered on this account.
    If you have a VatCode it could be convenient to use the amount without VAT.
  - VatCode the VatCode that applies to this item.
    If the Amount is Net of VAT you should define a VAT Code that is calculated on the net Value.

Group transactions by invoice number

If the imported data contains the "DocInvoice" columns, when Banana displays you a second DialogBox, you can choose to have Banana group the transactions by DocInvoice.
In this case Banana automatically creates, if necessary, a transaction for rounding differences.

Use Cost center instead for customer account

If you want to keep track of the invoices registered but do not want them to be recorded on individual accounts you can use the Cost center (CC3). See also Client/Suppliers register.

Import Income & Expenses transactions in CSV format

This format is suitable to import Bank statements in electronic format.

- For what is concerning the specifics of the import of income/expenses transactions see the explanations Import Income & Expenses accounting transactions.
- The information about the Double-entry import is also partially applicable.

Menu Account1->Import into accounting

The type of file to be used is a "Receipt/Payment transactions".

- You can use any column name existing on the table
- Columns with special meaning are
  - **Date** of the transaction (2014-12-31)
  - **Description** a brief text
  - **Income**: The amount in credit (can also be a negative amount)
  - **Expenses**: The amount in debit
  - **ContraAccount**: the account number (debit/credit) or category
  - **Account**: If the file contains the movements of multiple accounts, the account of the transaction
  - **VatCode** the vat code that should be used
  - **IsDetail** for composed transactions a "S" identifies a counterpart transaction and a "D" a detail transactions

- Fields header in the first line of the file. Fields names are case sensitive and must correspond to the NameXml (English)
- Fields and column name separator is to be tab separated "\t"
- Each line (after a "\n") is a new record
- The format for the Date fields is yyyy-mm-dd

Example file Income/Expenses format

Fields are separated by the tab char (ASCII decimal 11, C language "\t").
In the example the tab character is not visible.

```
Date Description Income Expenses ContraAccount Account
2007-01-02 Paper  30.00
2007-01-06 material for photographic competition  259.2  3000
```

Options

- Import using clipboard data will use the content of the clipboard instead of the file
- Autocomplete values: Some fields of the transactions are automatically completed (see "Importing transactions for multicurrency Double-entry accounting").
  Once the import is done, the contra account will have to be entered manually.
- Unicode (utf-8) The content of the file is in Unicode utf-8 (it supports any character set).

Import Accounts

For creating new accounts, customers, suppliers, cost centers. See Chart of accounts documentation for the list of columns available.

Menu Account1->Import into accounting->Accounts

The type of file to be used is a "TXT with headers.

- You can use any column name existing on the table
  - Account.
    The account number.
  - Description
    A brief text, organisation or customer name
  - BClass
    Required (1,2,3,4).
  - Gr1
    Obligatory. It is also used to order the data when it is imported.
  - Address fields.

When importing the user can choose to import only the new lines.

Exporting data

Banana Accounting can export in different formats:

- See documentation regarding Data Export.

It is also possible to create Banana Apps that export the data in specific formats:

- Creating Banana Apps that export data in specific formats.

Banana software is also trying to improve the data interchange with a new JCSV file format that bring toghether the simplicity of CSV and the advanced data interchange capabilities of Json.

JCSV file format

JSCV (Json Comma Separated Value) for tabular data

JSCV (Json Comma Separated Value) is an innovative file format for tabular data, that allows to mix data and metadata in a very simple way. You can specify tables and columns names, description, format, attributes without the need to have a separated schema file..JCSV files are CSV files for the internet aera, where metadata are central.

JCSV has been developed by Domenico Zucchetti, founder of Banana.ch, the first to introduce blockchain in the business world in 2002.

Writing a JCSV file is simple as creating a CSV file, but when reading the file you also have no parsing problems and all the necessary information to understand the data. From a technical point of view JCSV brings together the advantages of JSon and the CSV format (Comma Separated Value):

- It maintains the characteristics of CSV, but with a reliable JSon based data format.
- It allows mixing data and metadata (header, column info, descritpions, formats etc.).
- Multiple tables can fit in the same file.

This is the JCSV file in the simplest form. An header row, followed by the data rows.
Each line is writtene in a JSon data format.

```
{"column-names":["Section","Group","Account","Description","Boolean","BClass","Gr","Opening","Balance"]}
["","","1000","Cash on hand",true,"1","10",100,1290.3]
["","","1020","Bank account",false,"1","10",0,10]
["","","2000","Suppliers or Creditors",false,"2","20",-50,-50]
["","28","","Equity",false,"","2",-250,-1450.3]
```

There are 3 type of rows:

- Metadata rows, within {} brackets, in the form of a valid JSon Object in compact form, followed by end of line.
  In the above example the "column-names".
- Data rows, within [] brackets (JSon Array), followed by end of line.
  This is the CSV data, but formatted as a valid JSon Array.
  The data contains the value in the same sequence as the columns headers.
- Other rows, that are ignored.

Try the JCSV format

You can try a new format by installing the Banana Accounting software.

- See JCSV example files on Github\BananaAccounting
- Open or drag a JCSV file in Banana
  Banana will create a new file tables and columns. You can the copy and paste the data in Excel.
- Export in JCSV
  - Export of a single table
    Menu Data -> Export Rows->JCsv
  - Export of all the tables
    Menu File -> Export ->JCsv

JCSV with data and metadata

JCSV it allows to embed in the same file the data and metadata, like table and column name, description, format and attributes, in a very simple form, without requiring a schema file.

- JCSV is a better format for archiving tabular data.
- JCSV simplify the exchange of data .

The adavantages of JCSV comes from the possibility to embed any metadata in JSon.

- Encoding specification.
- File information.
- Table information. It is therefore possible to include more tables in the same file.
- Columns sequence.
- Field information.
- Unlimited expansion. Any metadata information, specific to the application or the data row, can be embedded.
- It is also possible to insert any other text. All information not within {} or [] is ignored.

JCSV references

The JCSV is based on the accepted standards.
See the reference documentation:

- JSon data format (wikipedia Json,, www.json.org).
- The W3C CSV on the Web Working Group specifications for the columns metadata.
- The W3C CSVW Namespace Vocabulary Terms for the terms used.
- The Dublin Core Metadata Element (for elements like dc:description, dc:title, dc:creator).
- The ISO date format 8601 for the date and time format.

JCSV format specification

See example below.

- Use the **UTF8 format**
- JCSV file is composed of text lines terminated by the "\r\n" or "\n". Both end of line terminators should be supported.
- **Metadata lines.**
  - They need to be valid Json Objects (JSon document), in compact format (No space and no end of line).
    Meta data line start "{" and end with the "}" brackets.

- Reserved metadata keyword:
    - "jcsv" with version and encoding.
    - "kind" is a property that uniquely identify the data, so that it make easier to understand what kind of data is contained in the jcsv file and apply appropriate transformation.
    - "table" contain the name of a table.
      All rows following the "table" are considered to belong to this table.
    - "schema" a url to a document that contains the infeormation necessary to verify the document or the table.
    - "column-names" is a required element that contains a Json Array with the columns names.
      The data rows following the columns are considered to be in the sequence of the columns name.
      A "column-names" that does not follow a "table" is considered to start a new table.
    - "columns" contains information relative to the columns.
      The columns information is not necessarily in the sequence of the data, so the "columnsNames" should always be present.
    - "row-attributes" contain supplementary information regarding the following data row.

- **Data lines.**
  Thex need to be Json arrays in compact form (no space or line feed).
  - Data lines start with the "[" and terminate with the "]" an contains the row data.
  - The array size need to be the same size as the preceding column-names.
  - Data is stored in Json format.
    - String are between ".
      - Usual string "Bank account".
    - Date, Time and Timestamp are Json string in the ISO date format 8601
      - Date "2018-01-03".
      - Time "10:18:21.000".
      - Timestamp "2016-11-19T09:52:39".
    - Number in valid Json format, decimal separator "."
    - true or false.
    - null.
    - Other lines not being a valid Json Object or array (like empty lines, text or else) are not considered.

Examples

n this example it used the column-datatypes to specify the datatype of each column.

```
/* text that is not within {} or [] is ignored */
{"jcsv" : {"version" : "1.0", "encoding":"UTF-8"}}
{"kind" : "banana.ch/testfile/test"}
{"fileinfo":{"Application":"Banana","Application version":"8.0.4.160915"}}
{"table":"Accounts"}
{"column-
names":["Section","Group","Account","Description","Boolean","BClass","Gr","Op
ening","Balance"]}{"column-datatypes"
:["string","string","string","string","boolean","number","string","number","n
umber"]}["","","1000","Cash on hand",true,"1","10",100,1290.3]
["","","1020","Bank account",false,"1","10",0,10]
```

```
["","","2000","Suppliers or Creditors",false,"2","20",-50,-50]
{"row-attributes":{"styleNumber":1024}}
["","28","","Equity",false,"","2",-250,-1450.3]

{"table":"Transactions"}
{"column-names":["Date","Time","Doc","Description","AccountDebit","AccountCredit","Amount"]}
{"column-datatypes":["date","time","string","string","string","string","number"]}
["2018-01-03","10:18:21.000","1","Cash to Bank","1020","1000",10]
["2018-02-02","23:55:00.000","2","Sales","1000","3400",1200.3]
```

In the following example it is used the "columns" specification with metadata information regarding each column.

```
{"table":"Accounts"}
{"column-names":["Account","Balance"]}
{"columns":[{"name":"Account",
"datatype":"string","titles":"Account"},{"name":"Balance","titles":"Balance",
"datatype":{"base":"number","scale":1}}]}
["1020",10]
["2000",-50]
```

JCSV files

- Mime type "text/jcsv".
- File extension ".jcsv".
- Encoding "UTF-8".

Advantages and limitation of JCSV format

Advantages of JCSV file format over CSV

- JCSV follows the JSon format.
  There is just one way to write and read the data. No more doubts about encoding, fields separators, number format, decimal separator.
- For generating and reading a JCSV it is possible to use the Json libraries.
- JCSV can contains data of different tables.
- Supplementary information relative to the structure or the attributes of the rows does not interfere with the data.

Advantages over Json

- In JCSV each line is a unique json document, independent from the other lines.
  - It is possible to add lines to an existing document (Append mode).
  - It is possible to parse the lines individually.

Limitation of JCSV

- Like CSV the JCSV format can be used only to exchange tabular data and not nested data structure (like Json).

Using Json library to generate or read the JCSV format

Generating JCSV files

1. Create a Json Object that contains and Json Array with the columns name and convert to Json text, plus the line feed.
2. For each data row create a JSon array that contains the data and convert to Json text, plus the line feed.

Parse JCSV files

1. Read each line.
2. If lines start with "{" and end with "}" parse as JSon and extract the values.
3. If lines start with "[" and end with "]" parse as JSon and get the data.

Javascript example for generating and parsing JCSV

```
// Create and parse JCSV data in Javascript
// header
var text = JSON.stringify({"column-names" : ["Date", "Name", "Amount"]}) +
"\n";
// Data row
text += JSON.stringify(["2018-01-24", "John Smith", 1200.10]) + "\n";
text += JSON.stringify(["2018-12-31", "Maria Callas", -200]) + "\n";

// parse JCSV data
var lines = text.split("\n");
for (i = 0; i < lines.length; i++)
{
  // header lines
  if (lines[i].startsWith("{") && lines[i].endsWith("}"))
  {
    JSON.parse(lines[i]);
  }
  // data lines
  if (lines[i].startsWith("[") && lines[i].endsWith("]"))
  {
    JSON.parse(lines[i]);
  }
}
```

Creating and parsing the JCSV file without the JSon library

Generating the file

- Write the header line as a simple text with the header.

```
"{"column-names" : ["Date", "Name", "Amount"]}\n"
```

- Create a CSV with the data that follow the rules of JSon data.
  Add at the begin the "[" and at the end the "]".

Parse the JCSV file

- Read the file line by line.
- Process the lines that start with "{" and end with "}".
- Process as data the lines that start with "[" and end with "]" as a normal CSV data structure.

Author

The JCSV specification has been conceived and developed by Domenico Zucchetti, founder and CEO of Banana.ch and creator of Banana Accounting.

Domenico Zucchetti has more then 30 year of experience in international accounting and as legal expert and software developers. He has been a blockchain pioneer. In 2002, it was the first in the world to implement a blockchain certification functionality in an accounting software.

D. Zucchetti welcomes feedback.

Translate Banana Software

Prepare for translation

Introduction

This documentation relates to Banana version 9.0.4 and later
Download Banana 9 at https://www.banana.ch/en/download_en

Download and install the tranlsation package

1. The translation package contains
    - QtLinquist
    - The Banana Accounting texts to be translated
    - The QtLibrary text (only few to translate)

2. In order to access the texts to be translated you have to **download** the following file:
   ban80_translations.zip -
   (www.banana.ch/accounting/files/banana9/translations/ban90_translations.zip)
   last update: 14 january 2016

3. Once you downloaded the file, double click on it in order to **extract its content** - once the process is finished you will have a new directory on your Desktop:
   ban80_translations directory, with the following content:
    - "win_start_linguist.cmd" file - to launch under Windows the software QtLinguist used to translate banana accounting
    - "mac_start_linguist.cmd" file - to launch under Mac the software QtLinguist used to translate banana accounting
    - "win_compile_translations.cmd" file - to compile under Windows the translations in a binary form readable by banana accounting
    - "mac_compile_translations.cmd" file - to compile under Mac the translations in a binary form readable by banana accounting
    - "output" directory - once you run "compile_translations.cmd", it contains the compiled

translations
  - "src/translations" directory - contains the Banana Accounting files with the texts to be translated
  - "win_bin", "mac_bin" and "qtrsrc" directories - contain various files

QtLinguist

Is a software tool that allow to insert the translated text.

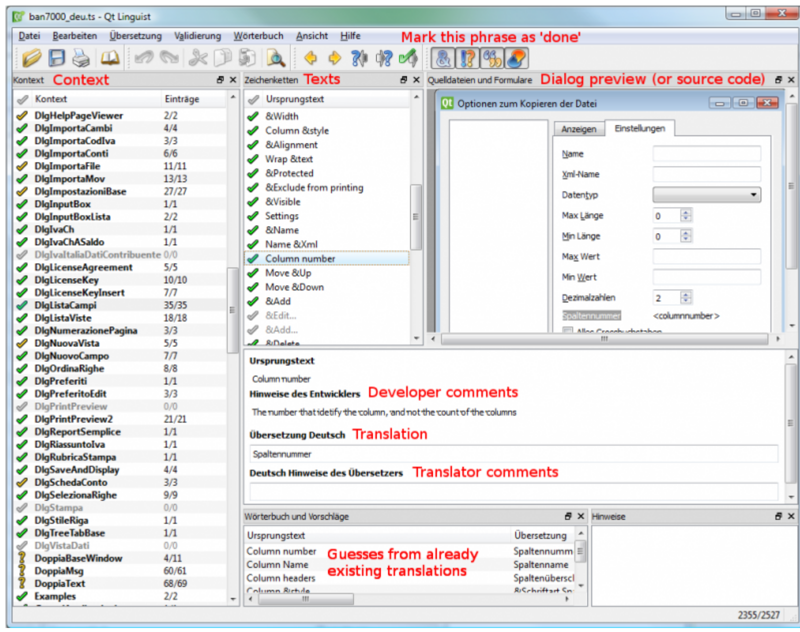
Before opening the QtLinguist sofware we advise you to read the [QtLinquist guide for translators]

Start QtLinuist and open the translation file :

- Launch the QtLinguist software with a double on the file "start_linguist.cmd"
- Open the translation file src/translations/ban8000_xxx.ts (where xxx stand for the abbreviation of your language)

QtLinquist Interface

You will face a screenshot similar to this one:



Translate the texts

- You need to translate the missing text and mark every finished text with a green check mark (you will find it

 in the toolbar) until all texts will be marked with either one of these symbols:

or
.
- We advise you to start with the more general texts, this means leaving for later texts starting with a Dlg (dialog) or Tab symbol - for example "CDate"
- You won't be able to change the English texts - if you find any mistake in English you will have to ask us to correct it.

- If you want to have a comparison with other languages, you just need to open the corresponding *.ts files.
- If it is not Se non è chiaro come tradurre un termine leggere le indicazioni dello sviluppatore
- f you need further information from us in order to translate a specific texts, there is a field called "Translator notes" where you can enter your comments.
  Start with the words: "TODO:" this is just a code so later on we will be able to trace and easily extract all your comments.

Special texts and charachters

- You will find %1, %2, %3 within the text.
  The symbol %1 will be replaced by the program by the appropriate value (account number, date, file name, etc.).
  If there is a %1, %2, %* this should also be present in the translation in the appropriate place.
- Banana Accounting columns name an table names.
  For example "ReportWithMovements;With mov.;with movements"
  - The text is separated by the semicolumn ";"
  - First part is the field name, in the translated text
    - No space or special charachters
    - Camel case (first charachter of the word is Capital)
  - Second part is the column header. Should possible fit on the columns with.
  - Thirt part is the full description, it is displayed as a tooltip when you go with the mouse on the column header.

QtLibrary translation files

- This are the text of the library file. They contains many text but we do need only to translate a fews.
- This texts are in ban80_translations/src/translations/qt_xx.ts file,
- You need to translate it in the same way as the other, and we actually advise you to start from this one. Attention: in the qt_xx.ts file not all the texts are to be translated, but only those that have a green check mark

  in the Italian file qt_it.ts (which you will have to open for comparison).

Do not translate

- It is not necessary to translate the following texts: ResocontoIvaCh*, ReportIvaItalia*I

After the translation

Send the translation back to us:

Once you finish your translation work:

- Send us as an attachement the *.ts files of your languages, this means the file ban8000_xxx.ts and the file qt_xx.ts if you have it in your language.
- Once you have send us the files back do not translate any more text.
  Wait until we will make the next version of the translations files available.

How to get your translation text in Banana Accounting software

You can verify how your translation look in the Banana accounting software - here is how to proceed:

- Check that you have the latest version of Banana Accounting.
  Get in touch with with Banana Tecnictian to know if the latest version is available.
- Double click on the file compile_translations.cmd
- Copy all the files output/*.qm in the Lang folder of the Banana software (usually the location is C:\Program files\Banana90\Lang or , C:\Program files\BananaExpm90\Lang, but it could vary depending on where you have installed Banana 9)
- Restart Banana 9 and select your language from the Option command (Tools menu)
- If some text does not display at the correct place it is due to different software version (can happen with experimental version)

Tools and links for translators

Microsoft Language portal - it contains all Microsoft translation

KDE Qt Translations - suggestion on how to translate the qt_xx.ts texts

Bing Translator - this is an online translator that you can rely on to translate complete sentences if you have not found a suitable solution with the previous links. Please note that technical and specific terms might not always be translated correctly.

Notes

Please let us know if we can improve this documentation.

Open source

Banana Accounting use this open source library:

- Qt Framework Libraries with LGPL 2.1 and LGPL 3.
- Libharu libraries (pdf writing) with zlib/libpng license.
- QtWebApp HTTP Server with the LGPL license.

The exact version of the library is visible within the software under

- Info regarding Banana Accounting
- Patent and legal informations

Building the libraries

The above indicated libraries are dinamically linked.
If you want to use modified libraries:

- Qt Framework. Banana use the dll libraries build made available by the The Qt Company for each platform.
  For building your own libraries simply follow the Qt instrunctions.
- Libharu libraries are also build with the default builds scripts.
- QtWebApp are also build with the default builds scripts.

Replacing the libraries

Once you have re-build the library and created the dll:

- Replace the libraries/dll
  - In Windows the libraries are in the program directory
  - In Mac are under the directory frameworks
  - In Android are in the lib directory
  - In Linux are in the lib directory

Info

If you have question let us know.

GitHub BananaAccounting

Contribute to the BananaAccounting GitHub repository

In order to contribute to the BananaAccounting GitHub repository and submit your changes it is necessary to follow some basic steps:

1. Install GitHub Desktop
2. Fork the repository
3. Clone the repository
4. Modify or add files to the repository
5. Keep in sync with the BananaAccounting repository
   (Repeat this step frequently!! It downloads the changes from the BananaAccounting main repository into your local repository, preventing conflicts).
6. Submit the changes

Install Github Desktop

Install GitHub Desktop on your computer, and (if you don't have it already) create your own account.

Fork the repository

The first step is to fork your own copy of the repository you want to work on to your account:

1. Go to the Banana Accounting's main page in github
2. Click on the repository that you wish to fork
3. Click the button **Fork** on top right of the page
4. Select your Account

Clone the repository

The second step is to clone the forked repository:

1. Click the green button **Clone or download** -> **Open in Desktop** -> **Open link**. GitHub Desktop should automatically open.
2. Click the button **Clone** and wait a moment while the forked repository is cloned on your local machine.

At this point, you have created on your computer your own copy of the repository.

Modify or add files to the repository

To work with the repository:

1. Start GitHub Desktop
2. Choose the repository you want to work on
3. Menu **Repository** -> **Show in Explorer**
4. Work on the repository, create and/or edit the files that you wish to change.
5. Open GitHub desktop again, your changes should be visible on the left side of the window.
6. Commit the changes to your own repository by clicking on **Commit to master** and then on the **Pull** button.

Sync forked repository with the BananaAccounting repository

Prior to work on your local files, make sure you have the latest copy of the files. If you do changes before updating the files you will end up having conflicts.

There are two steps to follow in order to sync a fork of a repository to keep it up-to-date with the upstream repository:

1. Configure a remote for a fork

2. Sync a fork

Configure a remote for a fork

Configure a remote that points to the upstream repository in Git to sync changes you make in a fork with the original repository.
See https://help.github.com/articles/configuring-a-remote-for-a-fork/.

This procedure which creates the upstream have to be executed only once. After that the system remembers this upstream. In this case go directly to Sync a fork.

1. GitHub Desktop -> Menu Repository -> Open in Command prompt
2. Write the following commands:

```
1) git remote -v
2) git remote add upstream
https://github.com/ORIGINAL_OWNER/ORIGINAL_REPOSITORY.git
3) git remote -v
```

Replace ORIGINAL_OWNER with **BananaAccounting**, and ORIGINAL_REPOSITORY with the **name of the repository** (i.e. .../BananaAccounting/Netherlands).

Sync a fork

Sync a fork of a repository to keep it up-to-date with the upstream repository.
See https://help.github.com/articles/syncing-a-fork/.

1. GitHub Desktop -> Menu Repository -> Open in Command prompt
2. Write the following commands:

```
1) git fetch upstream
2) git checkout master
3) git merge upstream/master
```

Your forked repository should be in sync.

Submit the changes (Pull request)

To submit changes with GitHub desktop:

1. Start GitHub Desktop
2. Choose the repository
3. Click on Menu **Branch** -> **Create Pull Request**
4. On the opened page of the browser set:
   1. **base fork**: select the **BananaAccounting repository** (example: BananaAccounting/Netherlands)
   2. **base**: select the **master** branch of the BananaAccounting repository
   3. **head fork**: select **your own repository** (example: YourAccountName/Netherlands)
   4. **compare**: select the **master** branch of the repository
5. Click the button **Create Pull Request**
6. Add a comment title and a comment text for the pull request.
   - Explain exactly what changes you have made, so that the moderator can more easily accept the changes.
     It will also help you later to understand what has been done.
     It it is worth to dedicate some time to doing good comments.
   - You can comment also groups of files you have changed. (see github documentation).
7. Click the button **Create Pull Request**

The repository moderator will receive the pull request, evaluate and approve or refuse the contribution.

You will be notified by email when the pull request has been accepted.

Important:

- **Do not click on the "Close Pull Request" button, or the pull request will be blocked!**
- **If the system tell you that there are conflicts, you should not submit the pull request.**
  Conflicts are probably due to the fact that you have not synched you repository with the Banana Accounting repository.
  Eventually:
  - do a copy of your repository.
  - resync again or in case clone again the main repository.
  - copy the changed files from the copy repository to the correct one.
  - Redo the pull request process.

Excel Reports Add-in (Beta)

With this add-in there will no longer need to make "copy and paste" of the values each time you update your accounting file.

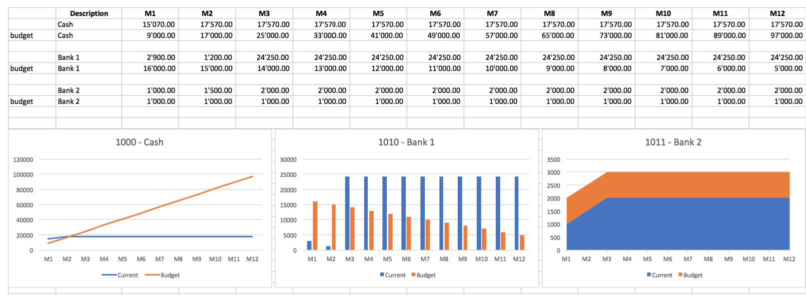
You create worksheets with formulas, charts, formatting and more in Excel, and the add-in will retrieve for you the data from the accounting file.

Just click on the **update button** and your Excel worksheet will be automatically filled with the new values from Banana Accounting, and the results of formulas and charts will be updated accordingly.

See Documentation [Banana Accounting Excel Add-in](#).

Sibex Ltd				
Example of Double-Entry Accounting with VAT				
2019				
BALANCE SHEET				
	01.01.19	**31.12.19**	**% of total**	**+/-**
Cash	1'000.00	17'570.00	9 %	16'570.00
Bank 1	17'000.00	24'250.00	12.4 %	7'250.00
Bank 2	1'000.00	2'000.00	1.1 %	1'000.00
Cash	**19'000.00**	**43'820.00**	**22.4 %**	**24'820.00**
Clients	10'000.00	10'000.00	5.2 %	0.00
Prepaid taxes	0.00	0.00	0 %	0.00
Accounts Receivable	**10'000.00**	**10'000.00**	**5.2 %**	**0.00**
Inventory	7'000.00	7'000.00	3.6 %	0.00
Inventory	**7'000.00**	**7'000.00**	**3.6 %**	**0.00**
Transitory assets	0.00	0.00	0 %	0.00
Prepaid Expenses	**0.00**	**0.00**	**0 %**	**0.00**
Current Assets	**36'000.00**	**60'820.00**	**31.1 %**	**24'820.00**
Machinery and applicances	4'000.00	4'000.00	2.1 %	0.00
Office furniture	6'000.00	6'000.00	3.1 %	0.00
Computer	10'000.00	10'000.00	5.2 %	0.00
Software	4'000.00	4'000.00	2.1 %	0.00
Car	11'000.00	11'000.00	5.7 %	0.00
Equipment	**35'000.00**	**35'000.00**	**17.9 %**	**0.00**
Real Estate	100'000.00	100'000.00	51.1 %	0.00
Real Estate	**100'000.00**	**100'000.00**	**51.1 %**	**0.00**
Fixed Assets	**135'000.00**	**135'000.00**	**69 %**	**0.00**
Total ASSETS	**171'000.00**	**195'820.00**	**100 %**	**24'820.00**

Example of a Balance sheet report created with the Excel Reports add-in



	Description	**M1**	**M2**	**M3**	**M4**	**M5**	**M6**	**M7**	**M8**	**M9**	**M10**	**M11**	**M12**
	Cash	15'070.00	17'570.00	17'570.00	17'570.00	17'570.00	17'570.00	17'570.00	17'570.00	17'570.00	17'570.00	17'570.00	17'570.00
budget	Cash	9'000.00	17'000.00	25'000.00	33'000.00	41'000.00	49'000.00	57'000.00	65'000.00	73'000.00	81'000.00	89'000.00	97'000.00
	Bank 1	2'900.00	1'200.00	24'250.00	24'250.00	24'250.00	24'250.00	24'250.00	24'250.00	24'250.00	24'250.00	24'250.00	24'250.00
budget	Bank 1	16'000.00	15'000.00	14'000.00	13'000.00	12'000.00	11'000.00	10'000.00	9'000.00	8'000.00	7'000.00	6'000.00	5'000.00
	Bank 2	1'000.00	1'500.00	2'000.00	2'000.00	2'000.00	2'000.00	2'000.00	2'000.00	2'000.00	2'000.00	2'000.00	2'000.00
budget	Bank 2	1'000.00	1'000.00	1'000.00	1'000.00	1'000.00	1'000.00	1'000.00	1'000.00	1'000.00	1'000.00	1'000.00	1'000.00

Example of a report with charts created with the Excel Reports add-in

Characteristics

- This add-in is hosted on our server.
  Once you have installed the manifest on your computer, your will automatically use the last version.
- The add-in are secure.
  Unlike Excel-macros the Add-in are secure and cannot compromise your computer.

- The add-in is currently in Beta Test.
    - Please check everything and report any problem.
    - You can use for free, but It is also possible that it will be made available with a cost.

Installation

The steps below walk you through all the setup to run the Banana Office Add-ins for Microsoft Office 2016.

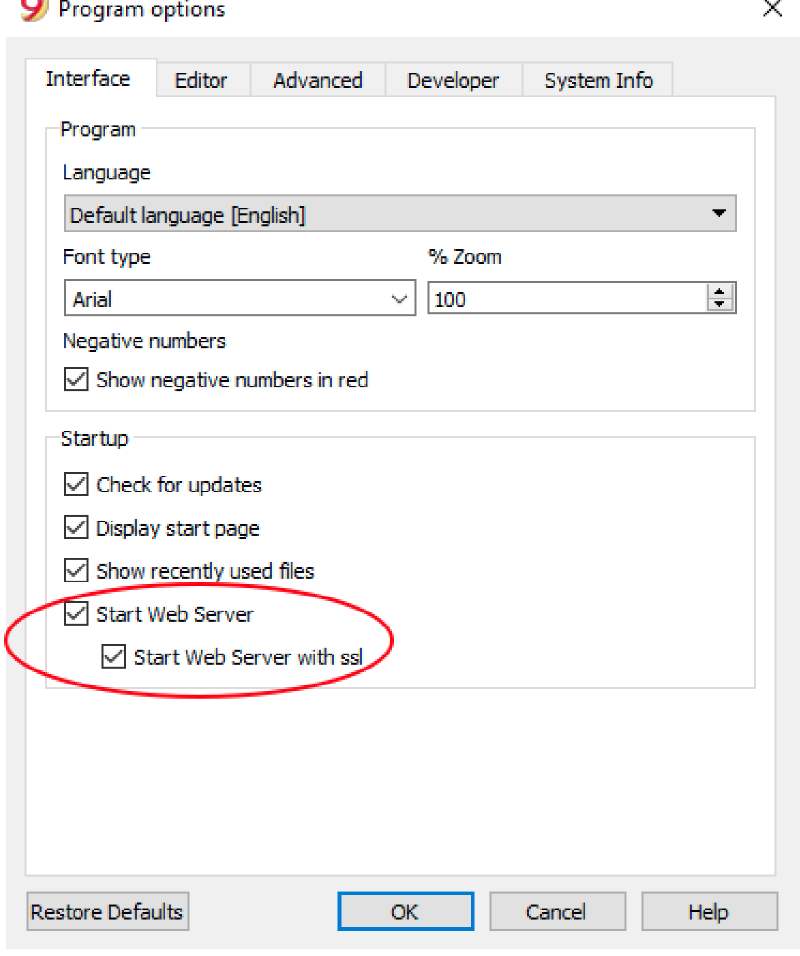
Minimum requirements: Microsoft Office 2016 (Word, Excel, PowerPoint, Outlook).

Get Banana Accounting 9

1. Download Banana Accounting 9 for Windows or Mac.
2. Install it on your pc.

Activate Banana Accounting web server

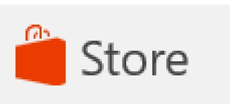
1. Start Banana Accounting 9
2. On Menu bar click **Tools** -> **Program options...** -> select the **Interface** tab
3. Check the **Start Web Server** and **Start Web Server with ssl** options



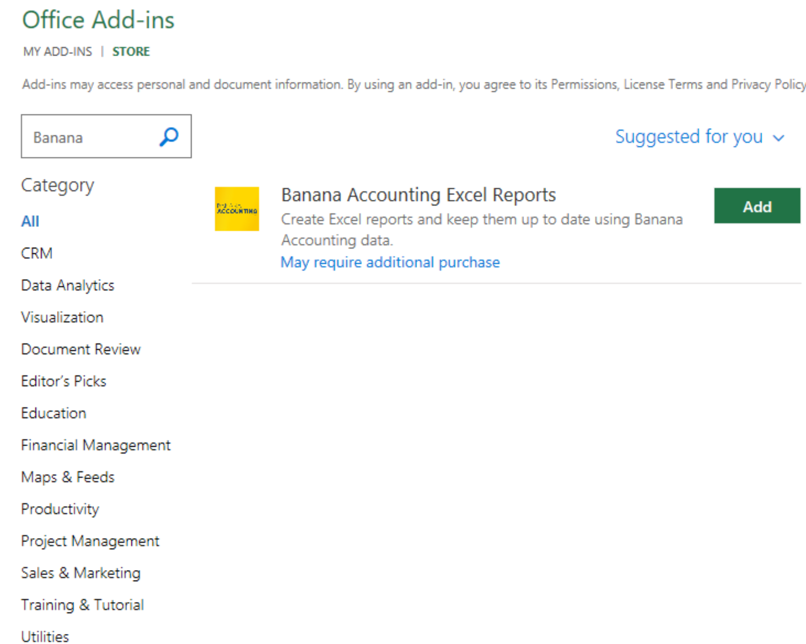
4. Click **Ok**

Load the Add-in

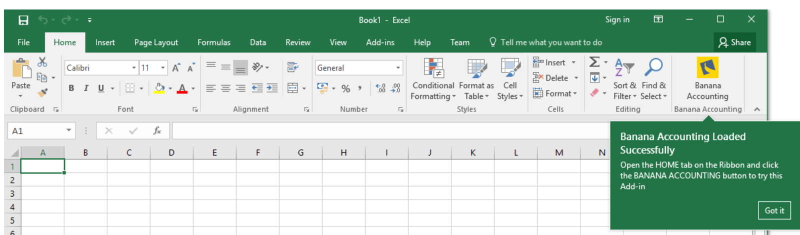
1. Open Microsoft Excel 2016
2. Click on **Insert tab**
3. Click on the **Store** icon to open the Office store



4. In the Office store page search for **Banana** add-in



5. Click on the **Add** button to add the Banana Accounting Excel Reports add-in
6. As soon as the add-in is added in Excel, on the **Home tab** of the main ribbon is loaded the Banana Accounting add-in command.

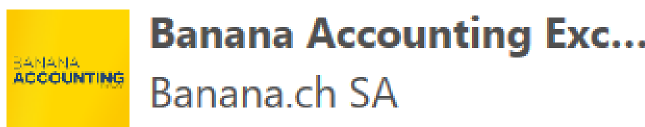


7. Click on the Banana Accounting icon to use the add-in

Once the add-in has been added from the Office store it is saved into My Add-ins section.
To load an add-in previously added from the office store:

1. Click on **Insert tab**
2. Click on the **My Add-ins** icon

3. Select the Banana Accounting add-in



4. Click on the **Add** button

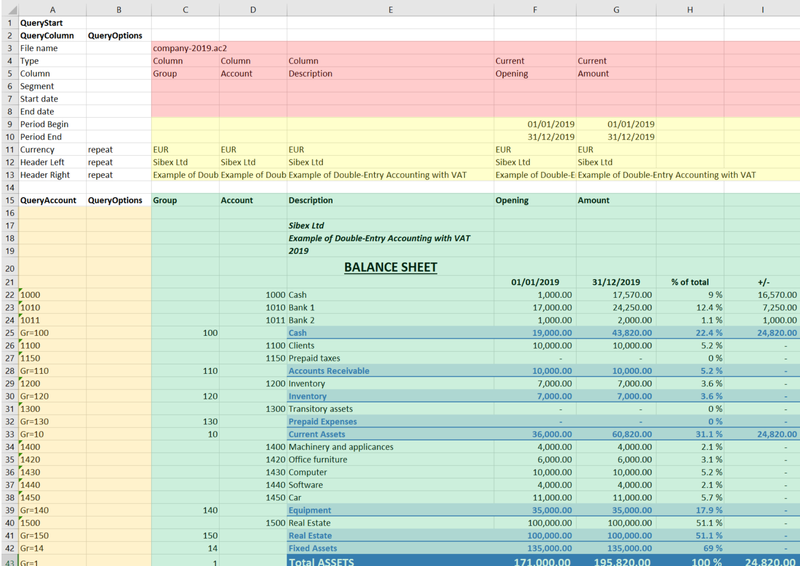
Documentation Excel Add-in

Introduction

With this add-in you can create Excel sheet that are filled with Banana Accounting data. Once you have added transactions to the accounting file you just need to click on the Update Button of the add-in and your spreadsheet content will be updated with the new data.
Your existing formatting and formula will be preserved.

1. **Create an Excel sheet with headers information**
   This information allows the add-in to retrieve data from Banana Accounting.
   There are information relative to the file, column and account or group to be retrieved.
   The add-in help you add the necessary information to retrieve the data.

2. **Click on the Update button**
   The add-in will retrieve the values from Banana Accounting software.
   It mantains the format or formula you enter.



	A	B	C	D	E	F	G	H	I
1	**QueryStart**								
2	**QueryColumn**	**QueryOptions**							
3	File name		company-2019.ac2						
4	Type		Column	Column	Column	Current	Current		
5	Column		Group	Account	Description	Opening	Amount		
6	Segment								
7	Start date								
8	End date								
9	Period Begin					01/01/2019	01/01/2019		
10	Period End					31/12/2019	31/12/2019		
11	Currency	repeat	EUR	EUR	EUR	EUR	EUR		
12	Header Left	repeat	Sibex Ltd	Sibex Ltd	Sibex Ltd	Sibex Ltd	Sibex Ltd		
13	Header Right	repeat	Example of Doub	Example of Doub	Example of Double-Entry Accounting with VAT	Example of Double-E	Example of Double-Entry Accounting with VAT		
14									
15	**QueryAccount**	**QueryOptions**	**Group**	**Account**	**Description**	**Opening**	**Amount**		
16									
17					***Sibex Ltd***				
18					***Example of Double-Entry Accounting with VAT***				
19					***2019***				
20					**BALANCE SHEET**				
21						**01/01/2019**	**31/12/2019**	**% of total**	**+/-**
22	1000			1000	Cash	1,000.00	17,570.00	9 %	16,570.00
23	1010			1010	Bank 1	17,000.00	24,250.00	12.4 %	7,250.00
24	1011			1011	Bank 2	1,000.00	2,000.00	1.1 %	1,000.00
25	Gr=100		100		**Cash**	**19,000.00**	**43,820.00**	**22.4 %**	**24,820.00**
26	1100			1100	Clients	10,000.00	10,000.00	5.2 %	-
27	1150			1150	Prepaid taxes	-	-	0 %	-
28	Gr=110		110		**Accounts Receivable**	**10,000.00**	**10,000.00**	**5.2 %**	**-**
29	1200			1200	Inventory	7,000.00	7,000.00	3.6 %	-
30	Gr=120		120		**Inventory**	**7,000.00**	**7,000.00**	**3.6 %**	**-**
31	1300			1300	Transitory assets	-	-	0 %	-
32	Gr=130		130		**Prepaid Expenses**	**-**	**-**	**0 %**	**-**
33	Gr=10		10		**Current Assets**	**36,000.00**	**60,820.00**	**31.1 %**	**24,820.00**
34	1400			1400	Machinery and applicances	4,000.00	4,000.00	2.1 %	-
35	1420			1420	Office furniture	6,000.00	6,000.00	3.1 %	-
36	1430			1430	Computer	10,000.00	10,000.00	5.2 %	-
37	1440			1440	Software	4,000.00	4,000.00	2.1 %	-
38	1450			1450	Car	11,000.00	11,000.00	5.7 %	-
39	Gr=140		140		**Equipment**	**35,000.00**	**35,000.00**	**17.9 %**	**-**
40	1500			1500	Real Estate	100,000.00	100,000.00	51.1 %	-
41	Gr=150		150		**Real Estate**	**100,000.00**	**100,000.00**	**51.1 %**	**-**
42	Gr=14		14		**Fixed Assets**	**135,000.00**	**135,000.00**	**69 %**	**-**
43	Gr=1		1		**Total ASSETS**	**171,000.00**	**195,820.00**	**100 %**	**24,820.00**

Example of a Balance sheet report created with the Banana Accounting Excel add-in

In the example above we can see:

- The data part
  Here is where the data is synchronized, based on the QueryAccount and QueryColumns.
    - Accounting data (Green)
      Filled with the information coming from Banana Accounting
    - Header data (Yellow)
- QueryColumns (Red)
  The file name, columns names and type to retrieve.
  If the column is empty no data in this column will be retrieved. You can use the columns to enter formula.
- QueryAccounts (Orange)
  The accounts or groups to retrieve.
  If the row is empty no data in this row will be retrieved. You can use the row for entering formula o text that is not overwritten.

By clicking on the update button the Data part is updated with the new data of the accounting file, and all the previously settings like fonts, colors, formulas will remain.

Download and installation

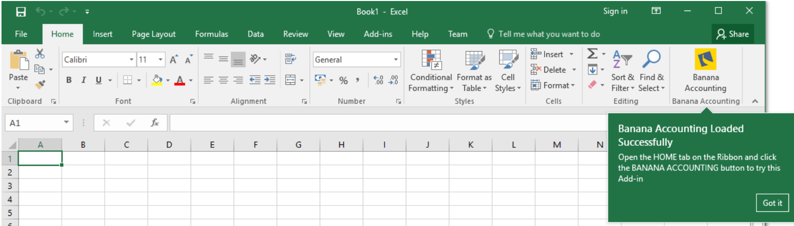
See documentation on how to Download and install the Add-in

Example files:

- File Excel already with columns setup.
- Banana Accounting file used for the example excel file.

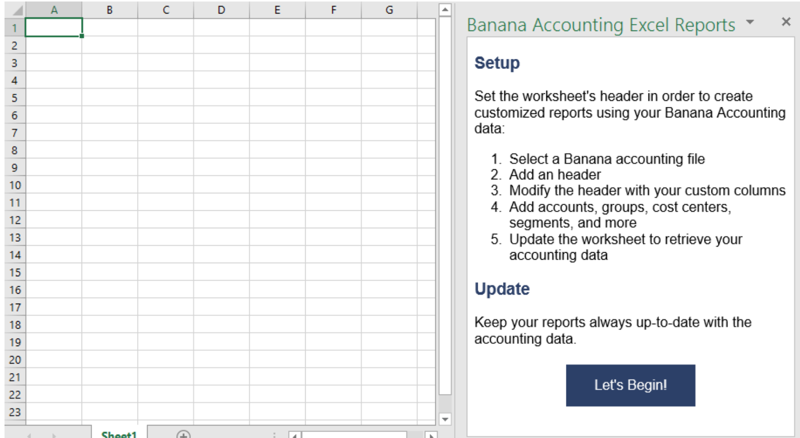
Add-in Command

As soon as the add-in is added in Excel, on the **Home tab** of the main ribbon is loaded the Banana Accounting add-in command.



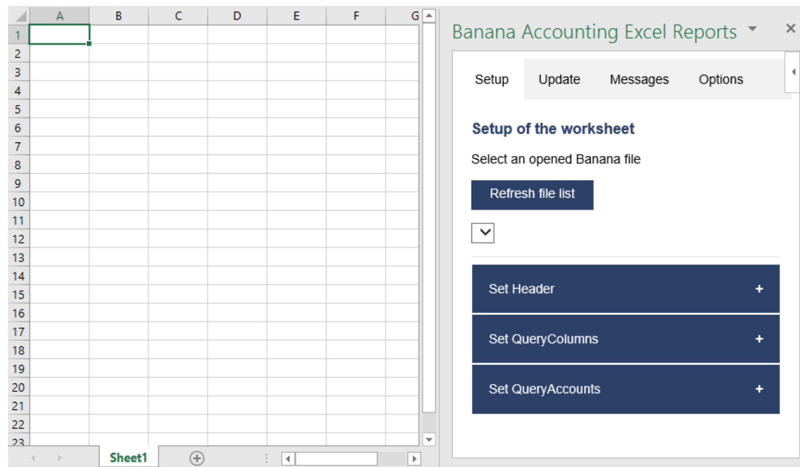
Banana Accounting Add-in command

When the Banana Accounting button is clicked, it loads the start screen of the add-in. The start screen provides additional information describing the functionalities of the add-in.



Banana Accounting Add-in start screen

Click on the **Let's Begin!** button to start using the Add-in.



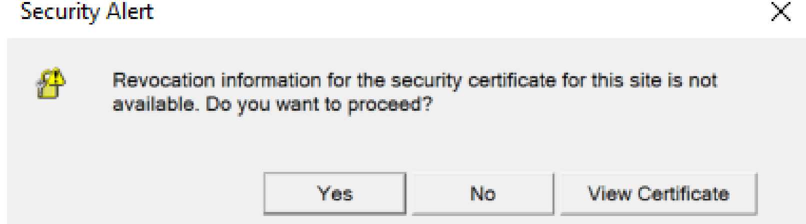
Banana Accounting Add-in setup screen

Security alert messages for Windows users

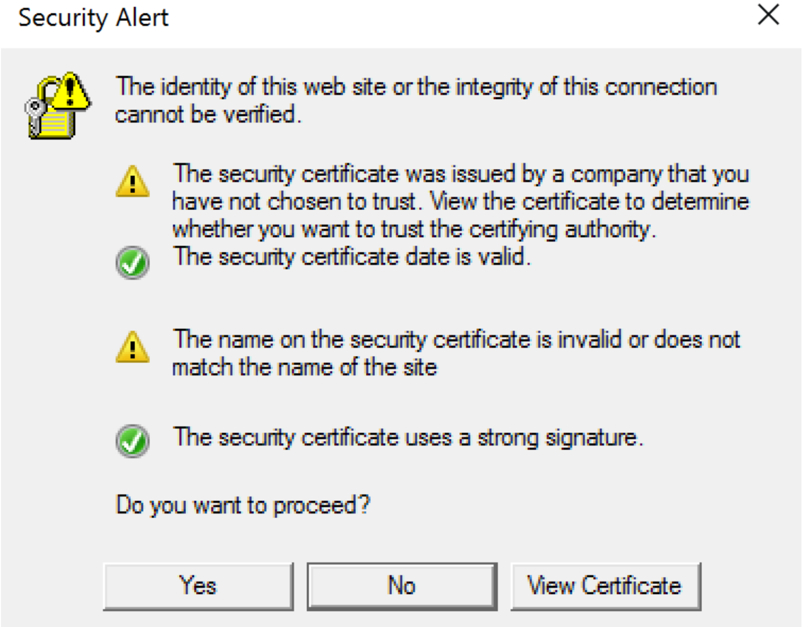
In order to properly establish a connection between the add-in and Banana Accounting web server, it is required to accept the Banana security certificate.

After the **Refresh file list** button has been clicked, securities alert dialogs like the following appear:

- The first security alert message is the following one, click on **Yes** to proceed:



- The second security alert message is the following one, click on **Yes** to proceed:



If the user clicks **Yes**, a connection between the add-in and the Banana Accounting web server is estabilished, and then it is possible to use the add-in. Otherwise, if the user clicks **No**, the add-in is loaded but none connection is established, and the add-in will not work.

If for some reason the security alert messages above do not appear, try to see the troubleshooting documentation.

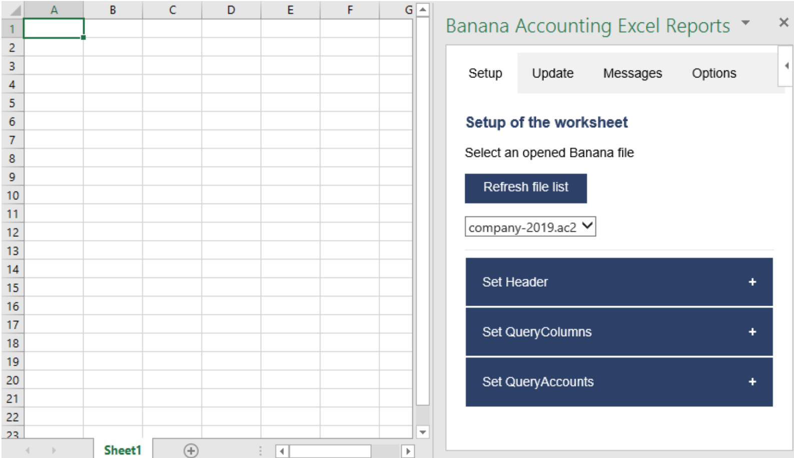
Add-in general overview

The add-in is a task pane add-in type. This means that the add-in is loaded in a pane on the right side of the Excel worksheet.

It is composed by three tabs, each of them has one specific task:

- The **Setup** tab contains all the tools needed to add information to your sheet so that the add-in can fill the data part with the accounting data. Typically it is used every time you want to create something new, like for example the very first time you use this add-in.
- The **Update** tab is used to update the content of the Excel worksheet with the accounting data. It is used after the header section and some accounts has been added.
- The **Messages** tab it's just a place where are displayed some messages about the add-in and the operations it does. For example when you update the sheet a message is displayed telling you that the update is completed.

- The **Options** tab is used to set some settings like the language and the server's Url.

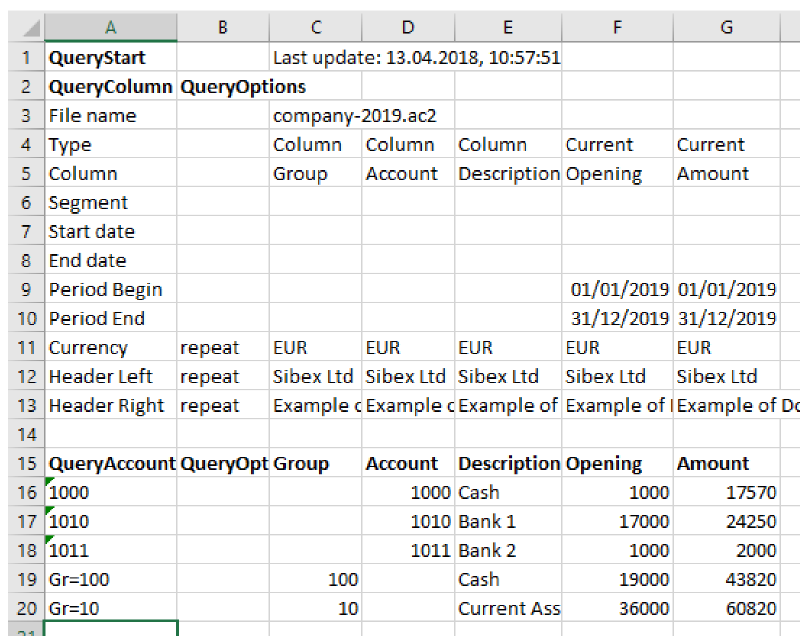


Banana Accounting task pane add-in

Update of the worksheet

The Update tab is composed only of one button: **Update current worksheet**.

When clicked, this will start the updating process of the current Excel worksheet. Combining the Header, QueryAccount and QueryOptions, the add-in retrieves all the data directly from the Banana Accounting and writes them in the Excel worksheet.



	A	B	C	D	E	F	G
1	**QueryStart**		Last update: 13.04.2018, 10:57:51				
2	**QueryColumn**	**QueryOptions**					
3	File name		company-2019.ac2				
4	Type		Column	Column	Column	Current	Current
5	Column		Group	Account	Description	Opening	Amount
6	Segment						
7	Start date						
8	End date						
9	Period Begin					01/01/2019	01/01/2019
10	Period End					31/12/2019	31/12/2019
11	Currency	repeat	EUR	EUR	EUR	EUR	EUR
12	Header Left	repeat	Sibex Ltd	Sibex Ltd	Sibex Ltd	Sibex Ltd	Sibex Ltd
13	Header Right	repeat	Example c	Example c	Example of	Example of	Example of Do
14							
15	**QueryAccount**	**QueryOpt**	**Group**	**Account**	**Description**	**Opening**	**Amount**
16	1000			1000	Cash	1000	17570
17	1010			1010	Bank 1	17000	24250
18	1011			1011	Bank 2	1000	2000
19	Gr=100		100		Cash	19000	43820
20	Gr=10		10		Current Ass	36000	60820
21							

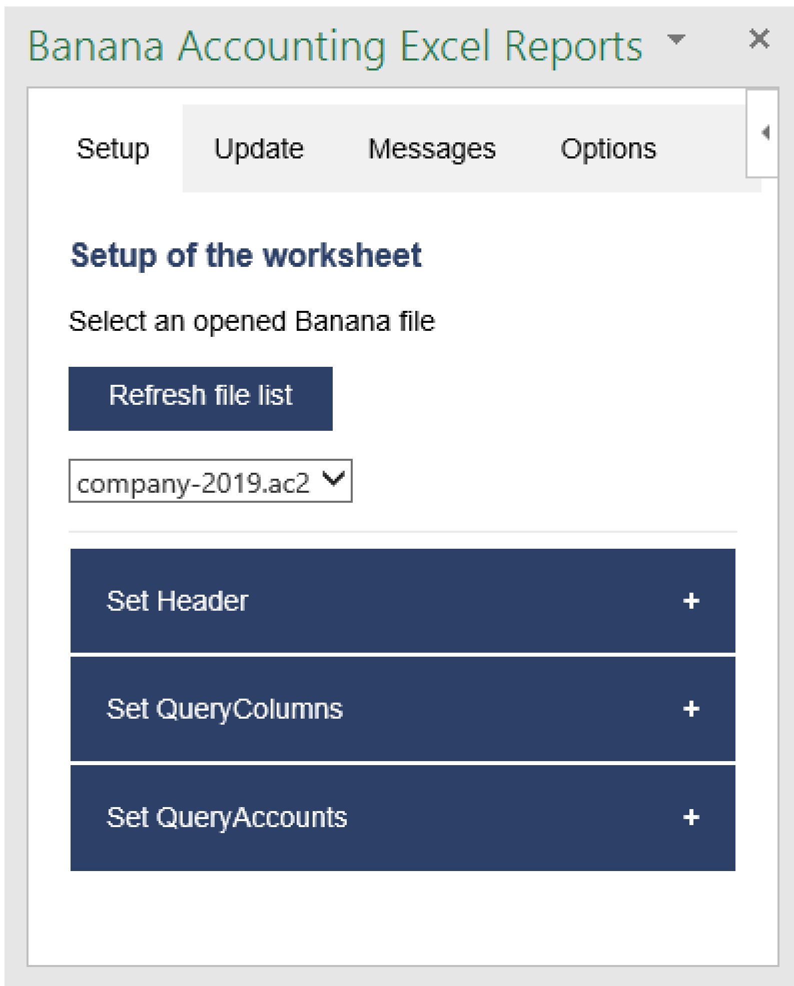
Retrieve data from Banana Accounting and update the worksheet

Setup of the worksheet

These features will add the information to the current worksheet necessary to retrieve data from Banana Accounting.

In the setup tab there are four sections:

- Accounting file selection
- Set Header
- Set QueryColumns
- Set QueryAccounts



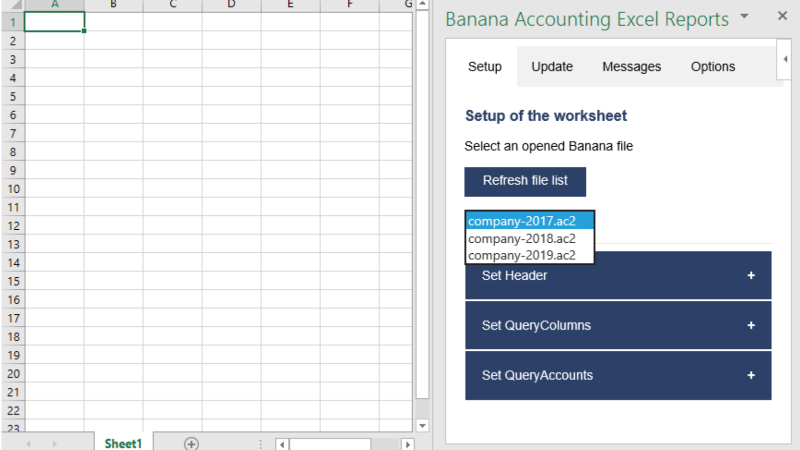
Setup of the worksheet tab

Select an opened Banana file

The first section of the setup page lists all the currently opened Banana Accounting file. Click the **Refresh file list** button and select the needed one and go to the next setup section.

If for some reason an accounting file is opened in Banana Accounting after the add-in is loaded, then this file doesn't appear in the list. In this case just click on **Refresh file list** button in order to recheck all the opened documents and recreate the list.

If nothing happens when the button is clicked (no file appears) and an error message is showed, please check the Troubleshooting page.



Example of file selection

Set Header

The second section of the setup page inserts, on the top of the current worksheet, the **header** that allows the user to insert information that will be used by the add-in to retrieve data from the accounting file.

Add an header

The first step is to select from the list a type of header. There are two options:

- **Predefined header with columns** to insert an header with default values for columns and options
- **Empty header** to insert a blank header

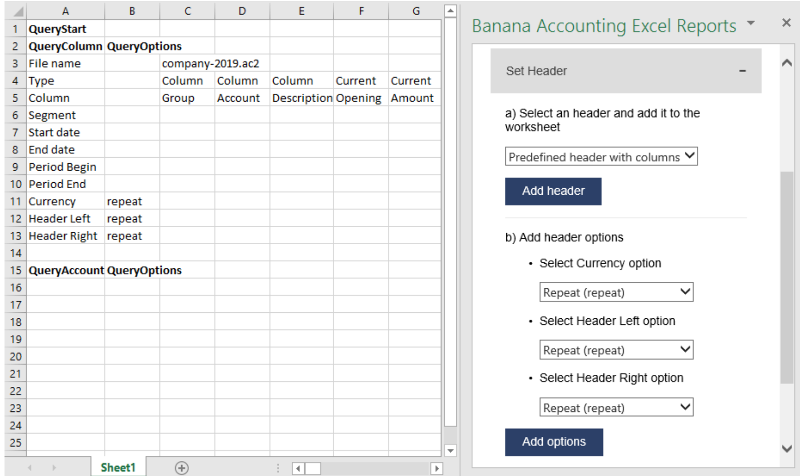
When the button **Add Header** is clicked, the selected type of header is inserted in the worksheet. It is then possible to modify by setting the QueryColumns and changing QueryOptions.

Add header options

The second step is to define some options for the **Currency**, **Header Left** and **Header Right** values using the QueryOptions column. The options are:

- **Repeat** to repeat the values in each column
- **Do not repeat** to avoid repeated values. Only when the file name changes the values are inserted again.

When the button **Add options** is clicked, the selected options will be inserted in the respective cells.



Example of predefined header

Set QueryColumns

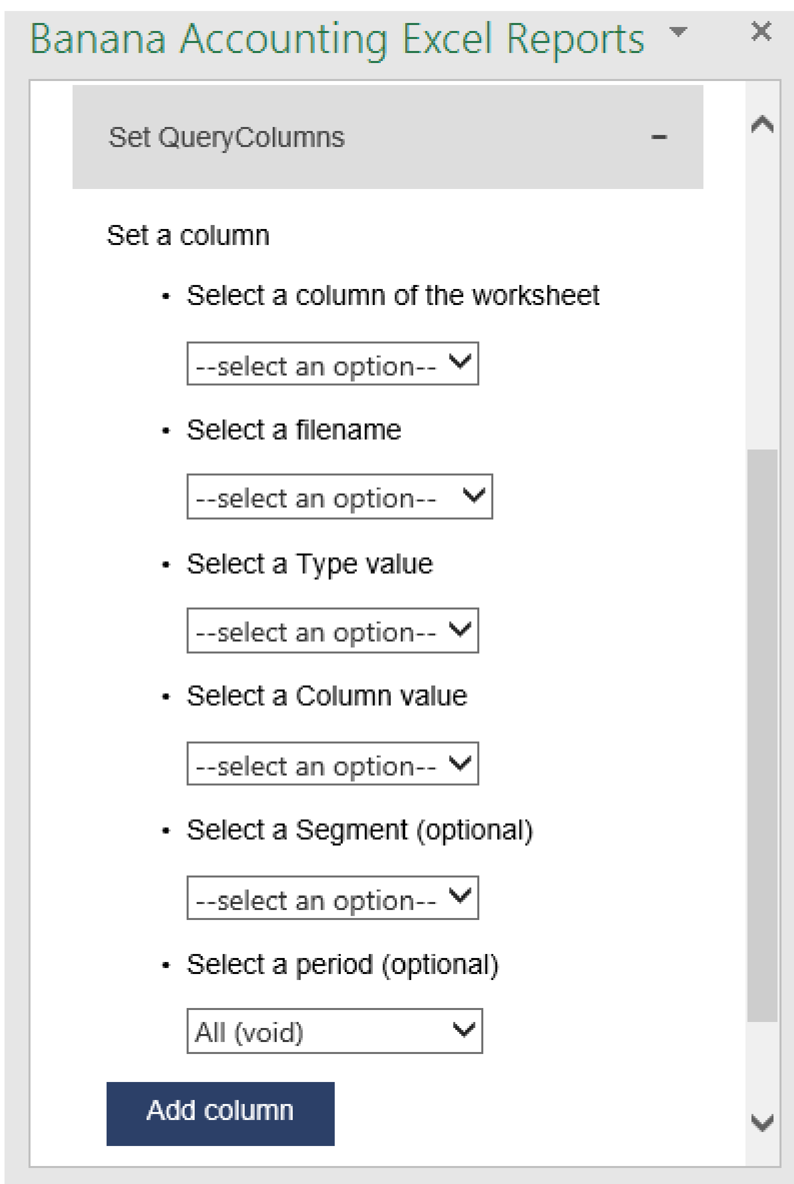
This section guides step by step the user to modify the header by adding QueryColumns to the worksheet.

The QueryColumns information allows the user to define exactly which data the add-in has to retrieve from the accounting file and in which column of the worksheet insert them.

Each QueryColumn consists of six information:

- The **Column of the worksheet** is used to define in which column of the worksheet all the QueryColumns values will be inserted.
- The **Accounting filename** is used to define the Accounting file to use when retrieving data.
- The **Type value** is used to define the type of data.
- The **Column value** is used to define the data for the given type.
- The **Segments (OPTIONAL)** is used to have a more detailed classification of the costs (**this is optional, if not specified none segments will be added**).
- The **Periods (OPTIONAL)** is used to define a period of the accounting (**this is optional, if not specified all accounting period will be automatically used**).

When the button **Add values to column** is clicked, all the information will be added automatically to the selected column of the worksheet.



Set QueryColumns section

Select a column of the worksheet

Use this to define in which column of the worksheet all the values of the QueryColumns are inserted. Possible values are:

- **Current selected** to use the colum of the cell selected on the worksheet (ex. if the cell D8 is selected, D column will be used).
- **C** ... **Z**

Remember that it is possible to use the columns from C to AZ, even if not all appear in the list.

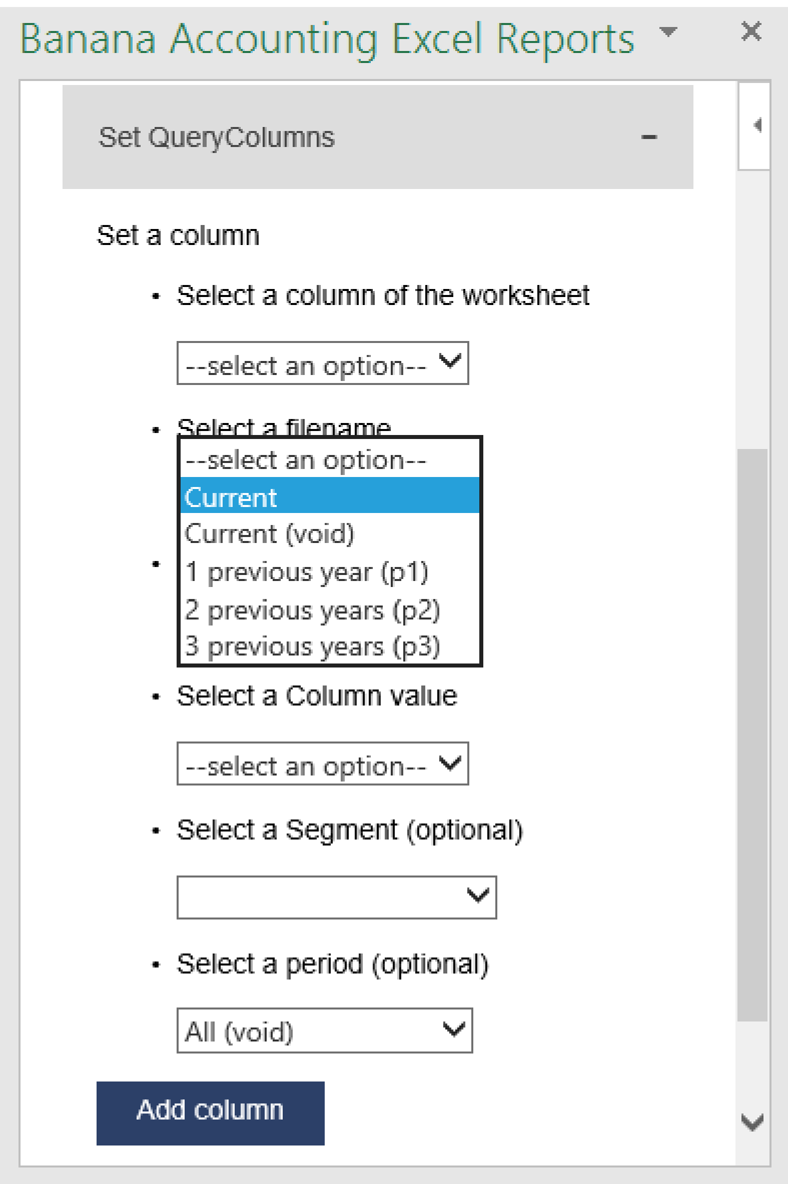
Select a filename

Use this to define the file name for a QueryColumn. When a file name is specified it is used until a new file name is inserted.

The possible values are:

- **Current** to use the selected file from the list on the top.

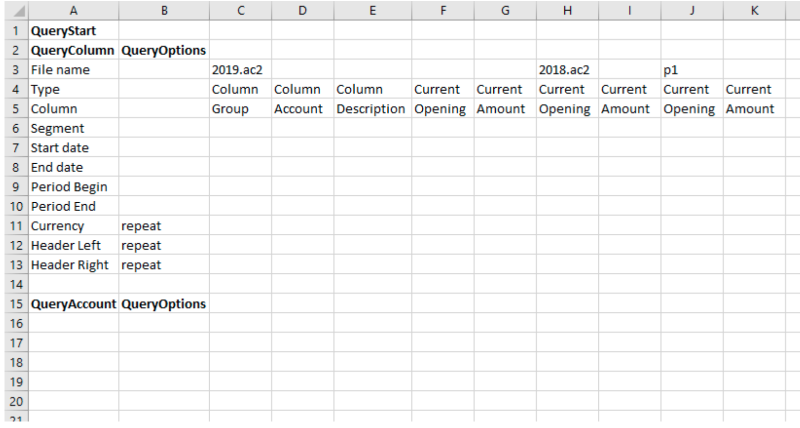
- **Current (void)** to use the previously inserted file but let the cell empty. It works only if in previous columns there is a specified file name.
- **1 previous year (p1)** to use the previous year file of the last file inserted (example: if current year is "2019.ac2", p1 refers to "2018.ac2")
- **2 previous years (p2)** to use two previous years file of the last file inserted(example: if current is "2019.ac2", p2 refers to "2017.ac2")
- **3 previous years (p3)** to use three previous years file of the last file inserted(example: if current is "2019.ac2", p3 refers to "2016.ac2")



Filename selection

Notes:

- remember to always open in Banana Accounting all the files specified in the header
- the p1, p2 and p3 abbreviations always refer to the last file specified in the header



	A	B	C	D	E	F	G	H	I	J	K
1	**QueryStart**										
2	**QueryColumn**	**QueryOptions**									
3	File name		2019.ac2					2018.ac2		p1	
4	Type		Column	Column	Column	Current	Current	Current	Current	Current	Current
5	Column		Group	Account	Description	Opening	Amount	Opening	Amount	Opening	Amount
6	Segment										
7	Start date										
8	End date										
9	Period Begin										
10	Period End										
11	Currency	repeat									
12	Header Left	repeat									
13	Header Right	repeat									
14											
15	**QueryAccount**	**QueryOptions**									
16											
17											
18											
19											
20											

Example of more file insertion

On the image above we can see there are three different files defined, each of them using different columns.

- Columns from C to G refer to the 2019.ac2 file
- Columns from H to I refer to the 2018.ac2 file
- Coumns from J to K refer to the 2017.ac2 file (p1 is the previous file of the last file inserted, in this case the 2018.ac2)

Select a Type and a Column value

Use them to define the data you want to retrieve from the accounting file.

- **Type** specify the type of data.
- **Column** specify the data for the given type.

The table below indicates for each **Type** of data which **Column** can be specified and so retrieved from the accounting (Not Case-Sensitive).

Possible Type-Column combinations

Type	Column
column	Group, Account, Description, Disable, FiscalNumber, BClass, Gr, Gr1, Gr2, Opening, Debit, Credit, Balance, Budget, BudgetDifference, Prior, PriorDifference, BudgetPrior, PeriodBegin, PeriodDebit, PeriodCredit, PeriodTotal, PeriodEnd, NamePrefix, FirstName, FamilyName, OrganisationName, Street, AddressExtra, POBox, PostalCode, Locality, Region, Country, CountryCode, Language, PhoneMain, PhoneMobile, Fax, EmailWork, Website, DateOfBirth, PaymentTermInDays, CreditLimit, MemberFee, BankName, BankIban, BankAccount, BankClearing, Code1
current	amount, amountcurrency, balance, balancecurrency, bclass, credit, creditcurrency, debit, debitcurrency, enddate, opening, openingcurrency, periodstring, rowcount, startdate, total, totalcurrency
budget	amount, amountcurrency, balance, balancecurrency, bclass, credit, creditcurrency, debit, debitcurrency, enddate, opening, openingcurrency, periodstring, rowcount, startdate, total, totalcurrency
columnvat	Group, VatCode, Description, Gr, Gr1, IsDue, AmountType, VatRate, VatRateOnGross, VatPercentNonDeductible, VatAccount
currentvat	taxable, amount, notdeductible, posted, rowcount

In the table below there are some examples of queries that can be used in the header to retrieve data from Banana Accounting:

Examples of queries

Type	Column	Segment	Start date	End date	RESULT
column	description				Return from the Accounts table the value of the column description for the account specified in the QueryAccount column
current	debit				Return the amount of debit transactions for all the accounting period for the account specified in the QueryAccount column
current	balance	:S1	01.01.2019	10.01.2019	Return the opening + debit-credit from the 01.01.2019 to 10.01.2019 for the account and segment specified in the QueryAccount column
current	total		M6		Return the difference between debit-credit for the 6th month for the account specified in the QueryAccount column
current	total		Q2		Return the difference between debit-credit for the second quarter for the account specified in the QueryAccount column
budget	opening		M12		Return the amount at the begining for the 12th month for the account specified in the QueryAccount column
columnvat	description				Return from the Vat Codes table the value of the column description for the vat code specified in the QueryAccount column
currentvat	taxable				Return the amount of the taxable column for the vat code specified in the QueryAccount column

Select a Segment (optional)

If the selected file has segments they will appear in the list.

Use this to define a segment to have a more detailed classification of the costs.

Select a period (optional)

Use this to define the accounting period that will be used to retrieve data from the accounting file.

Possible values are:

- **All (void)** to use all the accounting period
- **Custom date** to specify a Start date and End date (example: Start date "04.02.2019", End date "12.03.2019").
- **Month 1 (M1) ... Month 12 (M12)** to specify a single month (example: M1 for 1st month, M2 for 2nd month, etc.)
- **Quarter 1 (Q1) ... Quarter 4 (Q4)** to specify a single quarter (example: Q1 for the 1st quarter, period from 01.01 to 31.03)
- **Semester 1 (S1) ... Semester 2 (S2)** to specify a single semester (example: S2 for the 2nd semester, period from 01.07 to 31.12)
- **Year 1 (Y1) ... Year 10 (Y10)** to specify a single year (example: Y1 for the 1st year)

Set QueryAccounts

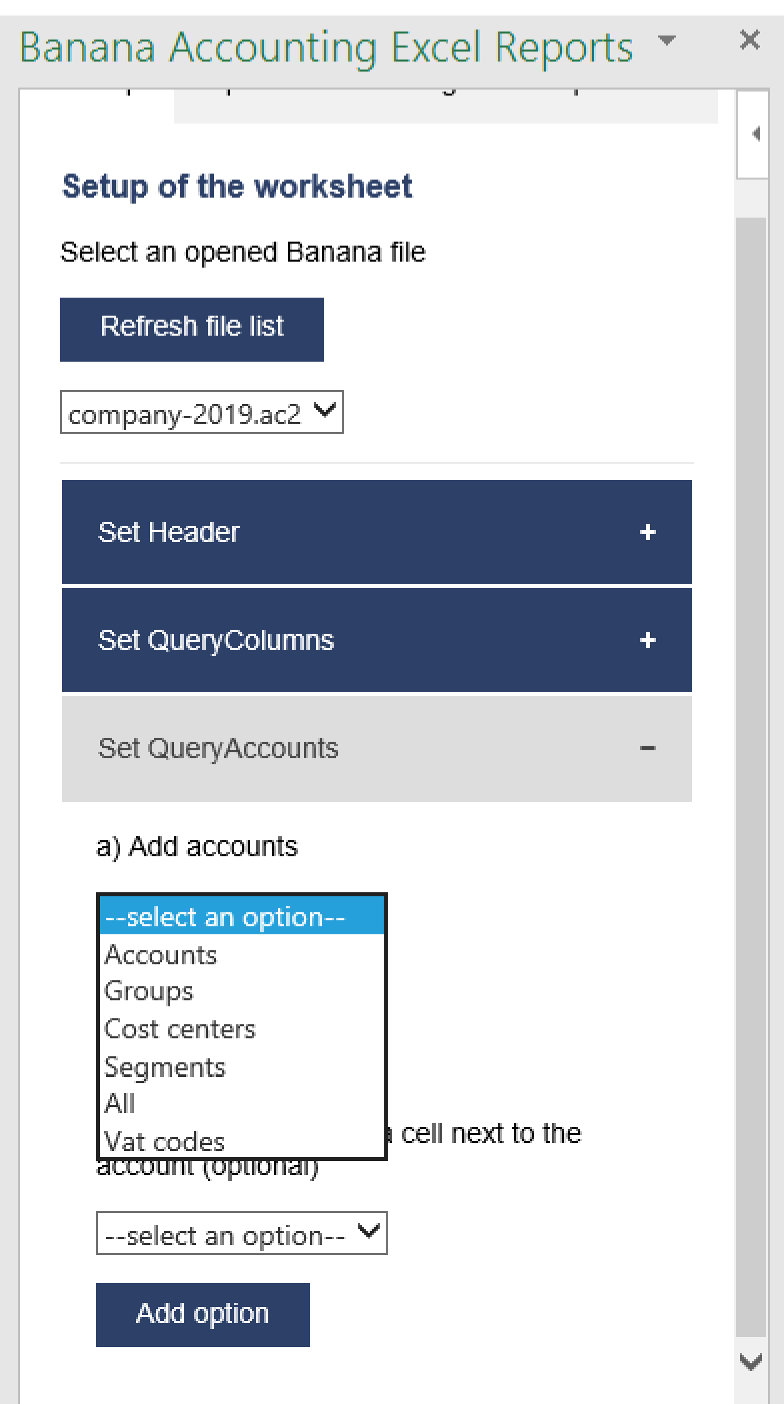
This section provides to insert:

- **QueryAccounts** to specify all the desired accounts, groups, cost centers, segments or vat codes that will be used with the data specified in the header to retrieve the accounting data.
- **QueryOptions (OPTIONAL)** to specify an option for a specific QueryAccount. Just select a cell next to the account and insert the option (**this is optional, if not specified none options will be added**).

Add accounts

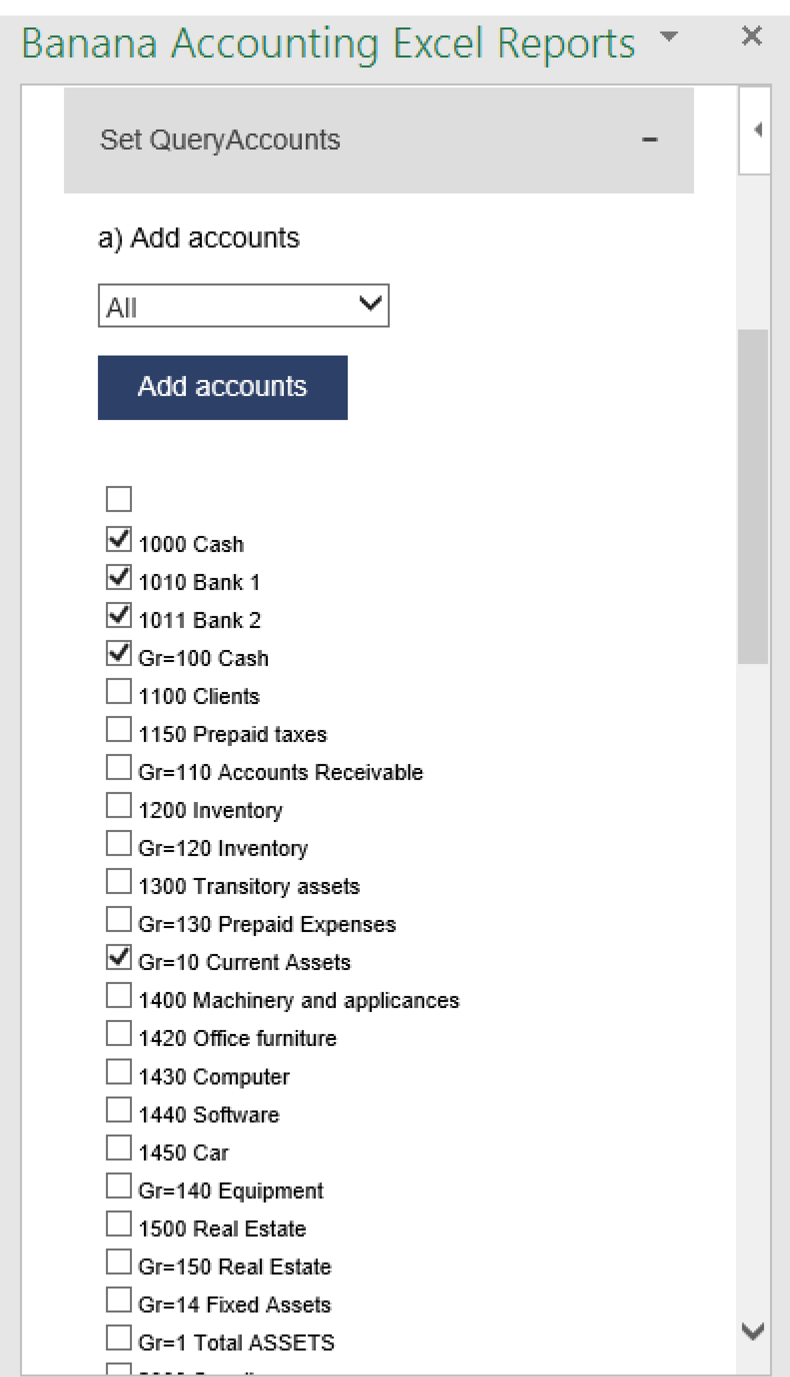
When an option is selected, the add-in loads the appropriate check box list with all the elements taken from the selected accounting file. It is possible to choose between six options:

- **Accounts** to load a list of all the accounts/categories codes taken from the table Accounts/Category of the accounting
- **Groups** to load a list of all the groups codes taken from the table Accounts/Category of the accounting
- **Cost centers** to load a list of all the cost centers codes taken from the table Accounts/Category of the accounting
- **Segments** to load a list of all the segments codes taken from the table Accounts/Category of the accounting
- **All** to load a list of all the accounts/categories, groups, cost centers and segments codes taken from the table Accounts/Category of the accounting
- **Vat codes** to load a list of all the VAT codes taken from the table VAT codes of the accounting



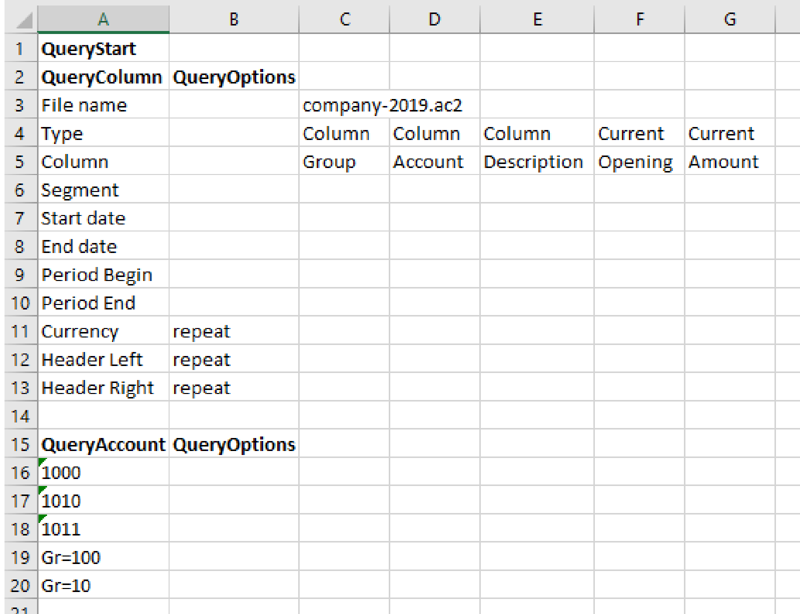
Type of account selection

For example, choosing the **All** option, the add-in loads a list containing all the accounts, groups, cost centers and segments respecting the order in which they appear in the accounting file.



Example of accounts and groups selection

After all desired elements has been checked, by clicking the **Add accounts** button will add them to the Excel worksheet under the QueryAccount starting from the selected cell. By default the add-in starts the insertion immediately after the QueryAccount title (row 16).



	A	B	C	D	E	F	G
1	QueryStart						
2	QueryColumn	QueryOptions					
3	File name		company-2019.ac2				
4	Type		Column	Column	Column	Current	Current
5	Column		Group	Account	Description	Opening	Amount
6	Segment						
7	Start date						
8	End date						
9	Period Begin						
10	Period End						
11	Currency	repeat					
12	Header Left	repeat					
13	Header Right	repeat					
14							
15	QueryAccount	QueryOptions					
16	1000						
17	1010						
18	1011						
19	Gr=100						
20	Gr=10						
21							

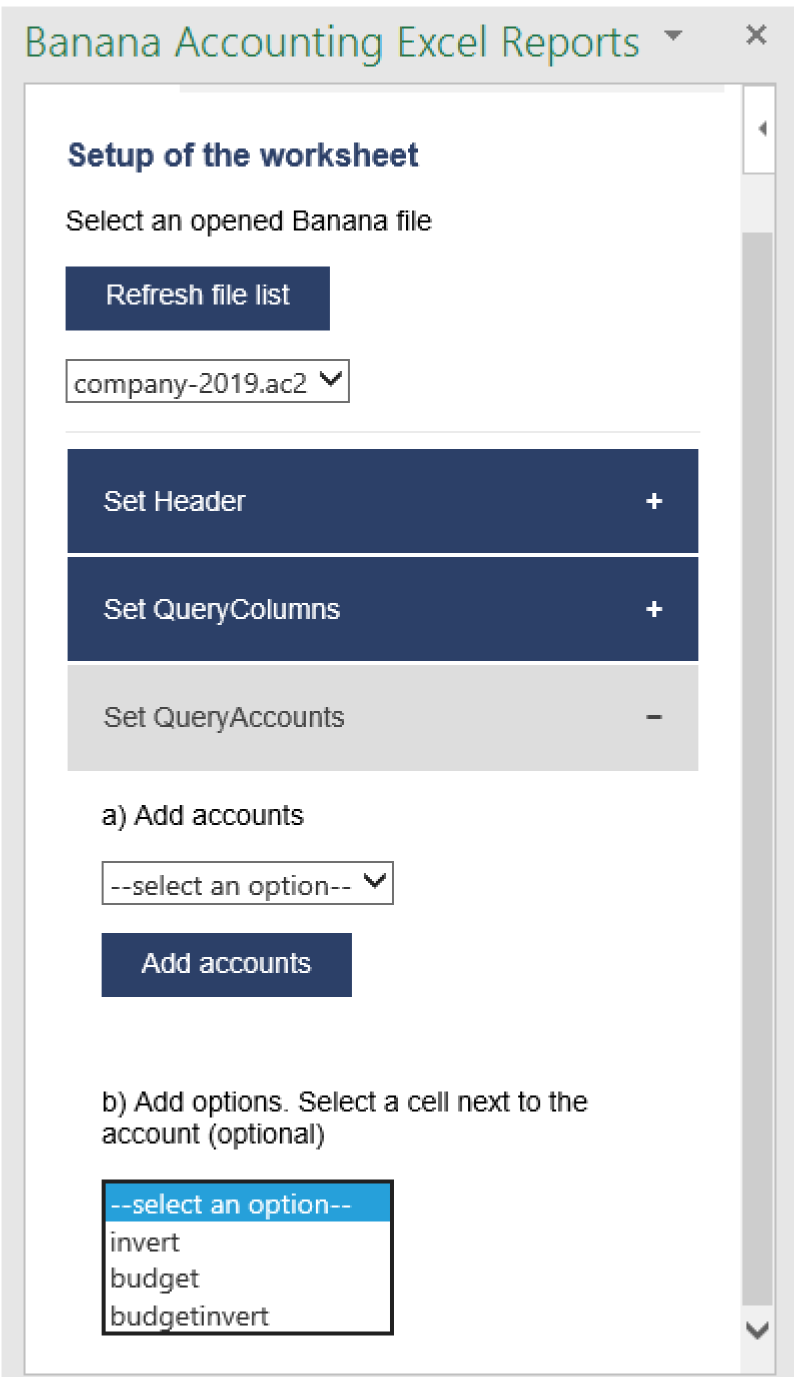
Add the selected accounts and groups to the Excel worksheet

Add option

The QueryOptions column is designed to add some options to the query that will retrieve the data from Banana Accounting. It is optional. If not used no options will be used.

The possible values are:

- **invert** to invert the sign of the current or budget balances.
- **budget** to get the budget balances (even if in the header are specified to use current balances).
- **budgetinvert** to get the budget balances (even if in the header are specified to use current balances) and also to invert the sign.



QueryAccounts options selection

Header settings

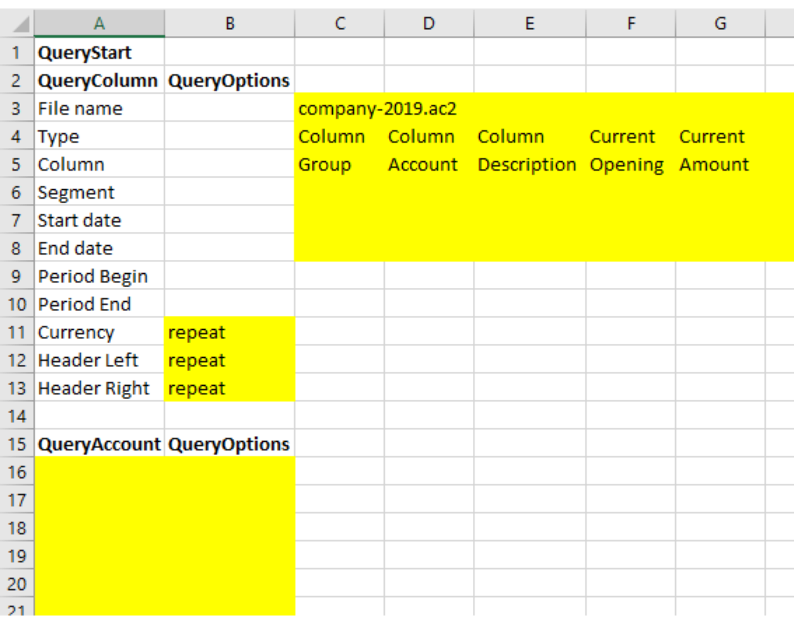
The purpose of the header is to let you to choose which data to import from Banana Accounting and on which columns in the Excel file to display them.
You must manually set column by column indicating, for each of them, the data that you want to import and display. It is possible to use the columns from **C** to **AZ**.

Notes:

- Do not add or delete rows in the header.
- Do not add or delete columns before the column B.
- From column C forward, it is possible to add or remove columns. Columns A (QueryColumn) and B (QueryOptions) must always exist.

- Added columns can also be empty.
- If columns from AA to AZ are used, **plese re-enter the file name at least on the AA column**, even if it is the same used in the previous column.

To better understand how exactly the header works and how to properly modify it, below there are some explanation about the most important things.



	A	B	C	D	E	F	G
1	QueryStart						
2	QueryColumn	QueryOptions					
3	File name		company-2019.ac2				
4	Type		Column	Column	Column	Current	Current
5	Column		Group	Account	Description	Opening	Amount
6	Segment						
7	Start date						
8	End date						
9	Period Begin						
10	Period End						
11	Currency	repeat					
12	Header Left	repeat					
13	Header Right	repeat					
14							
15	QueryAccount	QueryOptions					
16							
17							
18							
19							
20							
21							

Editable header parts

On the image above we highlighted in yellow all the header's parts that can be modified by adding information when creating a report.

Everything else will be automatically filled by the add-in when the **Update current worksheet** button is clicked.

Period Begin

A conversion of the start date to be easily read.
This is automatically filled for each column by the add-in when the worksheet is updated.

Period End

A conversion of the end date to be easily read.
This is automatically filled for each column by the add-in when the worksheet is updated.

Currency

The accounting basic currency.
This is automatically filled for each column by the add-in when the worksheet is updated.

Header Left

One of the information property of the accounting.
This is automatically filled for each column by the add-in when the worksheet is updated.

Header Right

One of the information property of the accounting.
This is automatically filled for each column by the add-in when the worksheet is updated.

QueryAccount

As already said, in this column are listed all the chosen accounts, each on a different row.

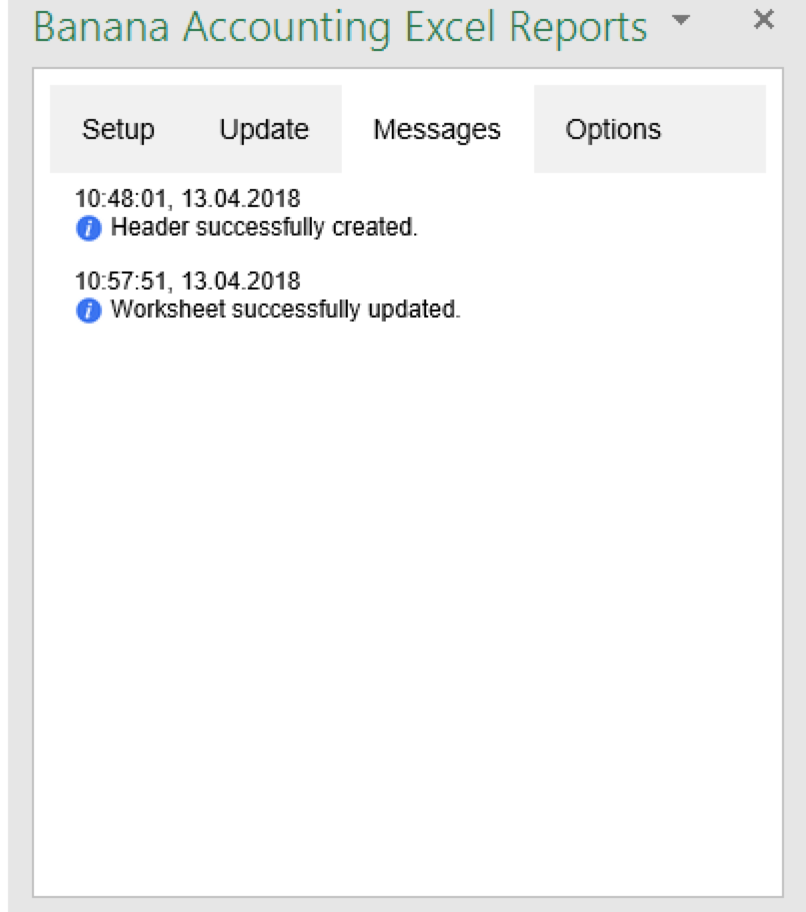
Instead of insert an account, is also possible to add a custom regroup using a particular accounting column.
The custom regroup QueryAccount syntax is **$column=value**, where:

- **$** indicates that a custom regroup is used.
- **column** is the Xml name of the column. It can be a user created column (for example "Abc") or a column that already exists in the accounting (for example the "Gr").
- **value** indicates the regroup.

If we insert something like "$Abc=1" in the QueryAccount cell, this means that the add-in takes and sums together all the accounts/groups balances that have the 1 value in the "Abc" column of the accounting.

Messages

The **Messages** tab shows some information about the add-in and the operations that it does.



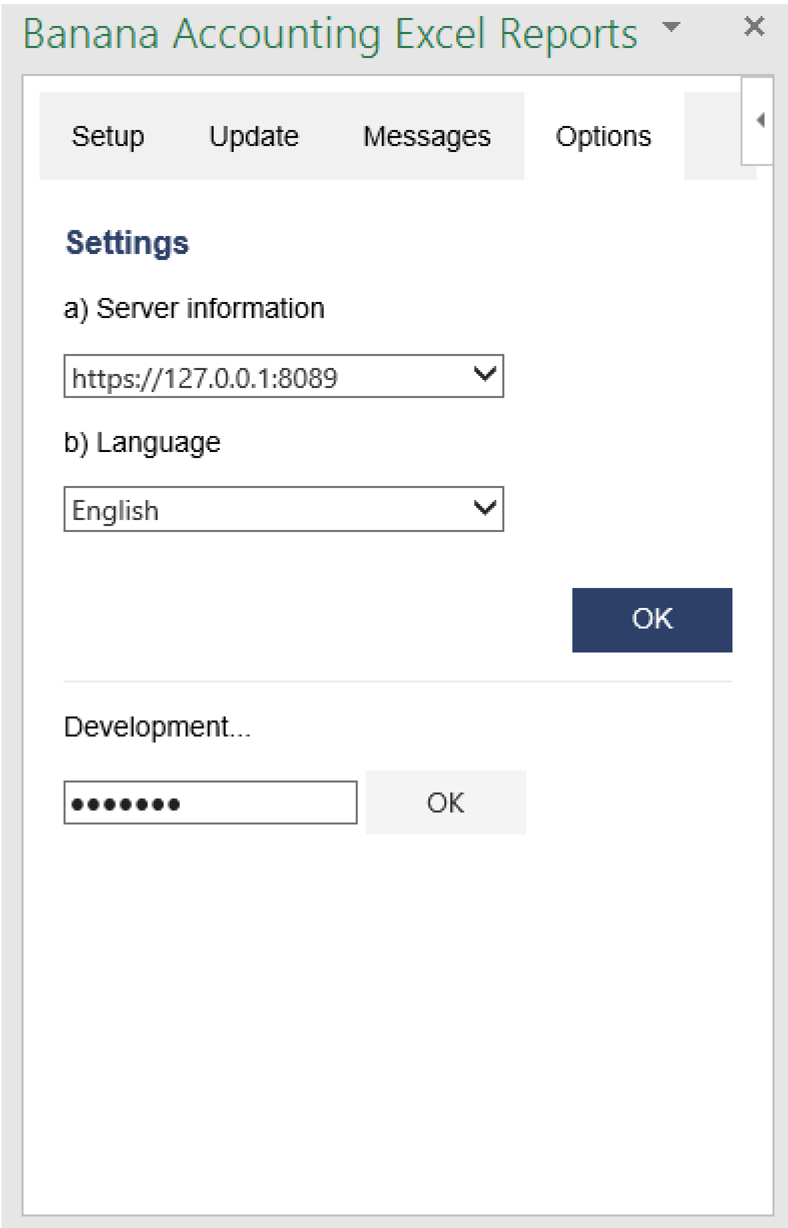
Example of messages

Settings

The Settings tab allows to change some settings of the add-in:

- the **Server information** allows to define the URL where Banana Accounting is hosted, to avoid to have Banana Accounting installed locally. By default it is defined the local Banana Accounting web server https://127.0.0.1:8089.
- the **Connection token** at the moment should be left empty.
- the **Language** to define the language of the Banana Excel Add-in. Available languages are english, french, german and italian.
- the **Development** is used only by developers for testing purposes, and users cannot access it.

Click on the Ok button and accept to reload the add-in in order to use the new settings. The settings are saved for future use of the add-in.



Settings tab

Release History

- 2017-06-12 First release
- 2017-07-07
  - Added Add-in Commands functionality.
  - Added a start screen that provides additional information describing the functionalities of the add-in.
  - Added the settings tab to allow the user to change the Port of the URL.
- 2017-09-29
  - Changed the name of the add-in to "Banana Accounting Excel Reports".
  - Changed some texts.
  - New add-in design.
  - Added new functionalities that allow the user to set and insert all the required information more easily.
  - Added localization language for english, french, german and italian.
- 2017-11-24
  - Added new functionality that allows to set the parameters for the connection.
- 2018-04-04
  - Settings options are now saved.
  - Changed the appearance of the error messages.
  - Changed some texts.
  - Other minor changes.

Troubleshooting

The first time the Banana Add-In for Excel connects to Banana Accounting it is necessary to accept the connections. Once it is accepted Excel can further retrieve data from Banana Accounting.

On Apple Mac, the system correctly accept the connection and the Add-in works just as expected.

On Windows the system does not always works as expected:

1. Some time the dialogs to accept the connection are not displayed.
2. The acceptance of the connection is not saved.

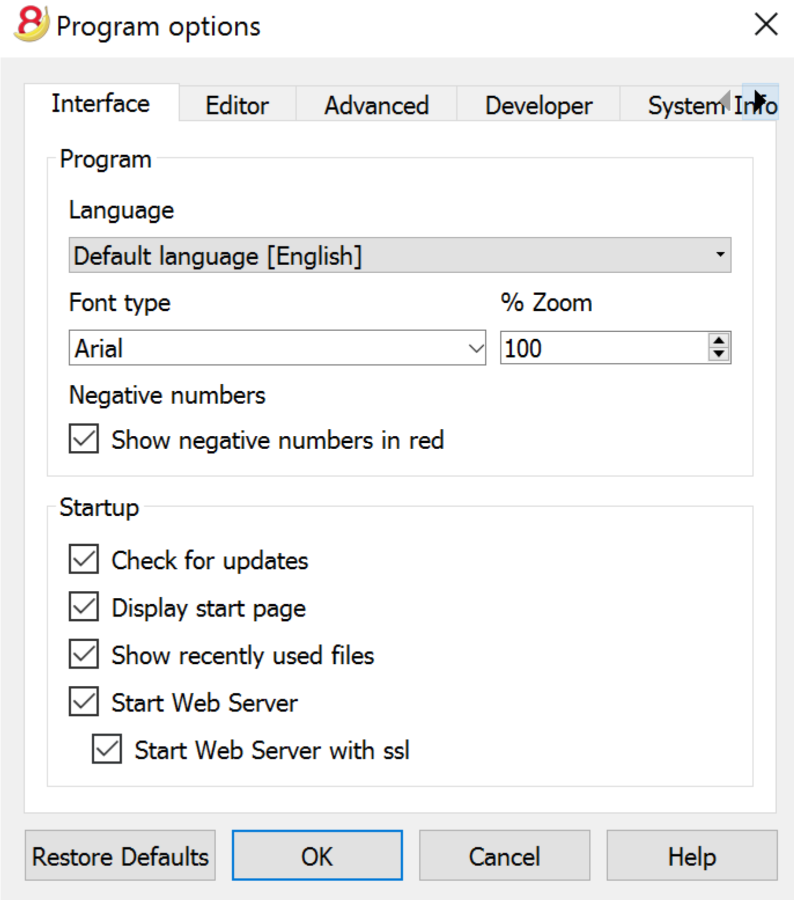
We have reported the bug to Microsoft. We are waiting for feedbacks.

In the mean time we do publish here a possible temporary fix. It require some technical knowledge and administrative permission.
We hope that Microsoft will fix the problem so that the Banana Add-in will be able to works under Windows the same as under Mac.

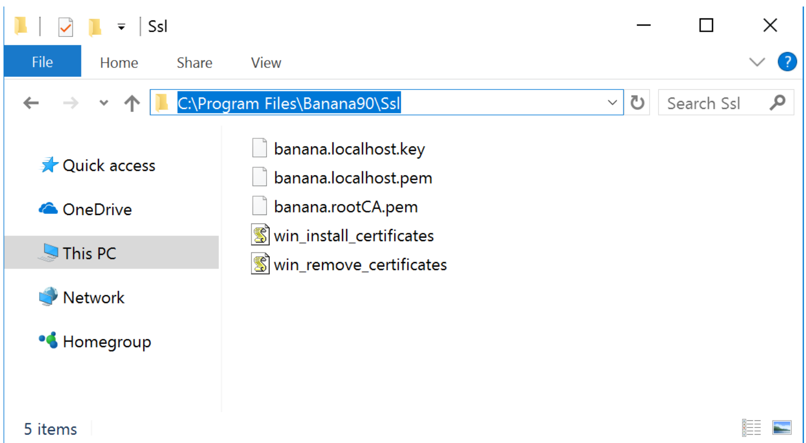
Troubleshooting for Windows

1. Download and install the latest version of **Banana Accounting 9** for Windows (**version 9.0.3 or more recent**).
2. Start Banana Accounting 9 web servers
   - Open Banana Accounting 9
   - Click on menu **Tools** -> **Program Options**
   - Select the tab **Interface**

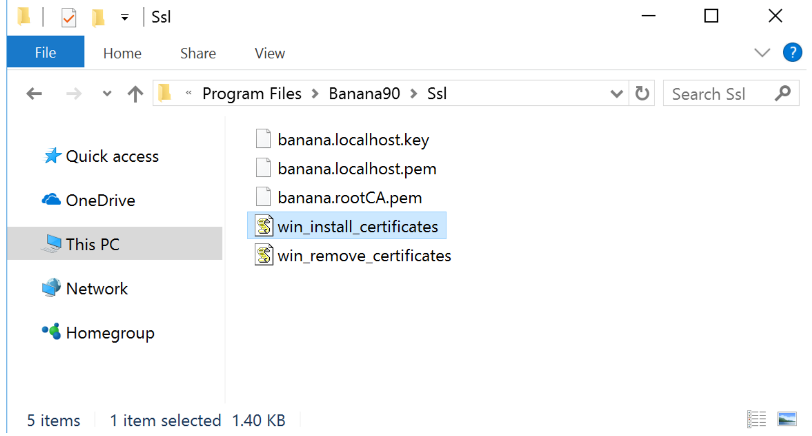
- Check the options **Start Web Server** and **Start Web Server with ssl**



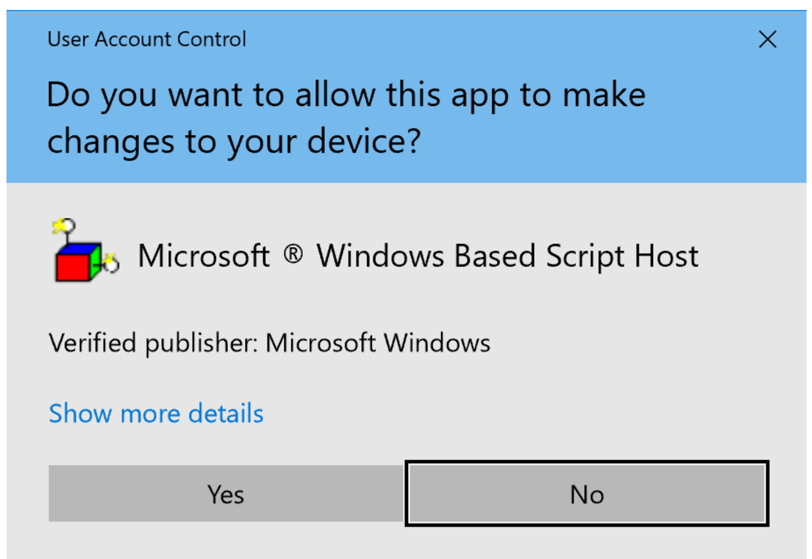
3. Install Banana.ch certificates:
    - Open the following Banana folder on your pc: **C:\Program Files\Banana90\Ssl**



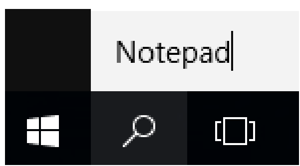
    - Double click on the file "**win_install_certificates**"



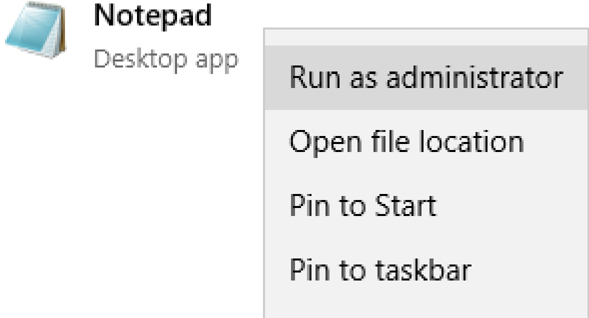
- Click **Yes**: this will install the Banana.ch certificates on your computer.



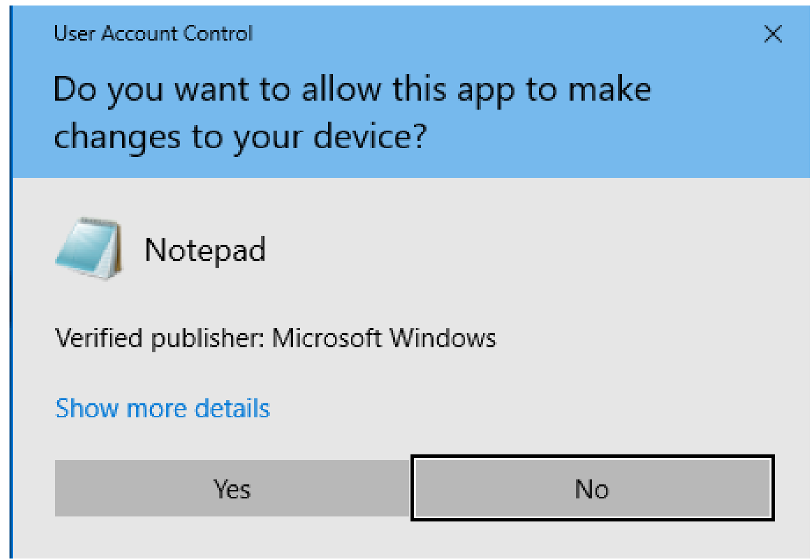
4. Edit the **hosts** file:
    - Press the Windows key and type **Notepad** in the search field



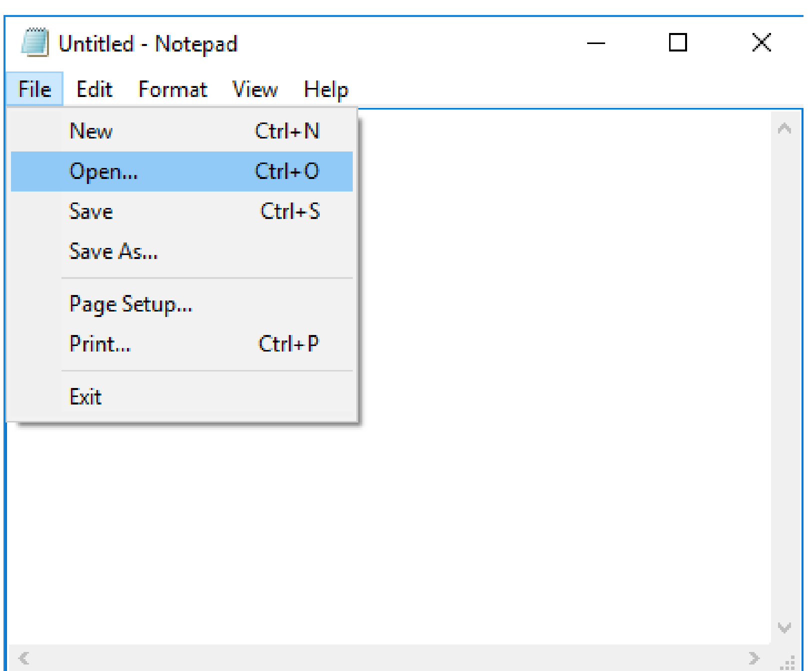
    - In the search results, **right-click** on Notepad and select **Run as administrator**



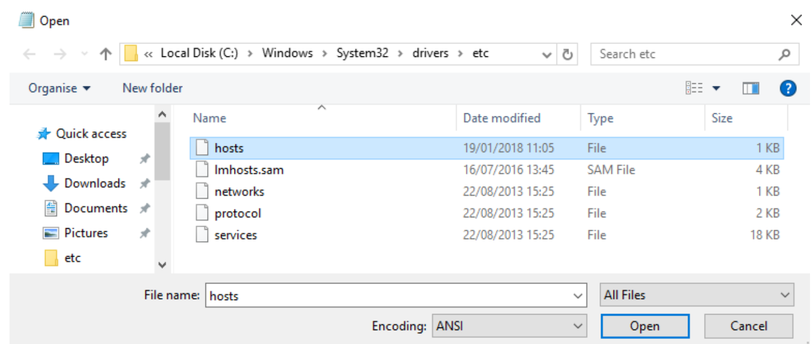
- Click **Yes**



- From Notepad, click on **File > open**



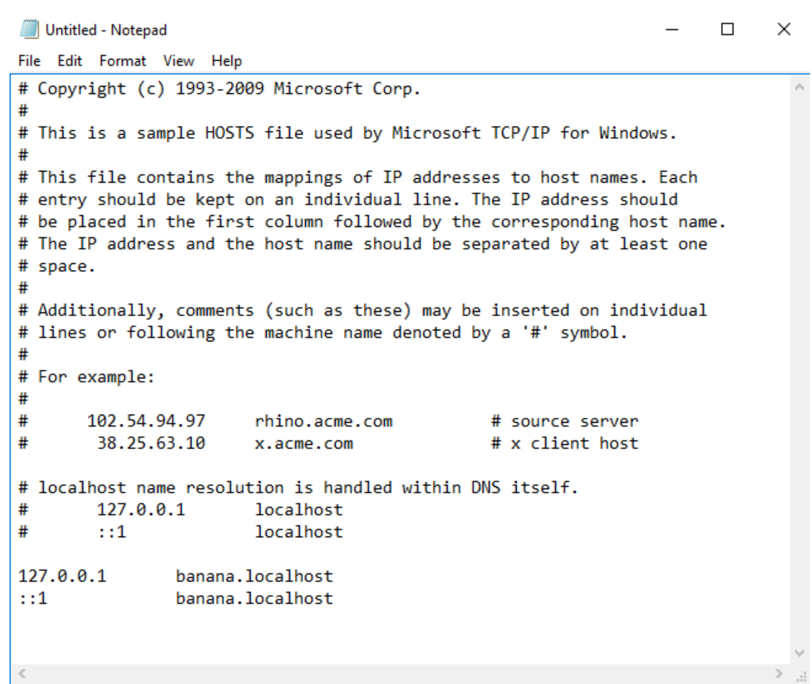
- Select the **hosts** file located in **C:\Windows\System32\drivers\etc** (if you don't see anything, on the bottom right select "All Files")



- Click on the **Open button** to open the file

- At the end of the file insert the following two lines:

127.0.0.1 banana.localhost
::1 banana.localhost



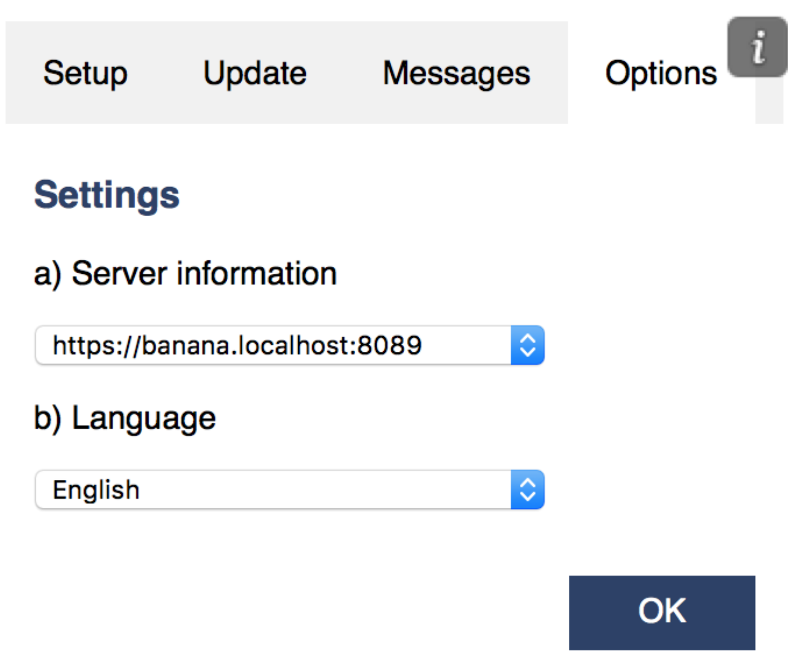

```
# Copyright (c) 1993-2009 Microsoft Corp.
#
# This is a sample HOSTS file used by Microsoft TCP/IP for Windows.
#
# This file contains the mappings of IP addresses to host names. Each
# entry should be kept on an individual line. The IP address should
# be placed in the first column followed by the corresponding host name.
# The IP address and the host name should be separated by at least one
# space.
#
# Additionally, comments (such as these) may be inserted on individual
# lines or following the machine name denoted by a '#' symbol.
#
# For example:
#
#      102.54.94.97     rhino.acme.com          # source server
#       38.25.63.10     x.acme.com              # x client host

# localhost name resolution is handled within DNS itself.
#       127.0.0.1       localhost
#       ::1             localhost

127.0.0.1       banana.localhost
::1             banana.localhost
```

Your file may be a little different. Do not delete or modify rows, just add these two lines at the end.

- Click **File > Save** to save your changes.
- Close the file.

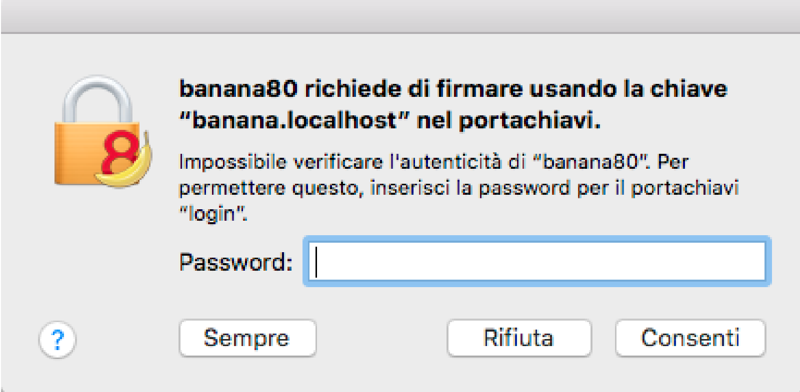
5. Change the server URL on the Add-in settings
   - Start Excel and Banana Accounting Add-in.
   - Click on the **Options** tab of the Add-in.
   - Select the server URL "**https://banana.localhost:8089**".

- Click **OK** to confirm and save the changes.
- On the **Setup** tab of the add-in **refresh the files list**.

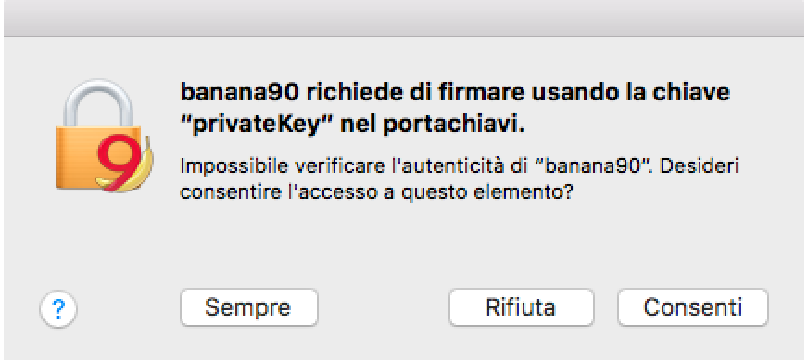


Troubleshooting for Mac

1. Download and install the [latest version of **Banana Accounting 9**] for Mac.
2. Start Banana Accounting 9 web servers
    - Open Banana Accounting 9
    - Click on menu **Tools** -> **Program Options**
    - Select the tab **Interface**
    - Check the options **Start Web Server** and **Start Web Server with ssl**
3. Open **Safari** and insert https://127.0.0.1:8089
4. When the dialog appears, insert your system password and click on always button



5. Start Excel 2016 and load the Add-in
6. Click on the refresh file list button
7. If a dialog window appears, click on always button on the left



Installation for developers

The steps below walk you through all the setup of the environment required to run the Banana Office Add-ins for Microsoft Office 2016.

Minimum requirements: Microsoft Office 2016 (Word, Excel, PowerPoint, Outlook).

Get Banana Accounting 9

1. Download Banana Accounting 9 for Windows or Mac.
2. Install it on your pc.

Activate Banana Accounting web server

1. Start Banana Accounting 9 Experimental.
2. On Menu bar click **Tools** -> **Program options...** -> select the **Interface** tab
3. Check the **Start Web Server** and **Start Web Server with ssl** options
4. Click **Ok**

Install the Manifest file

Each Office Add-in has its own manifest file. The manifest is an XML file that defines various settings, including description and links to all the add-in files.

Manifest file must be copied to a specific directory.

Manifest directory for Windows

On Windows you need to create a directory to save the manifest of the add-in.

The directory needs to be a shared directory.

1. Create a folder for the add-ins manifests on a network share:
    1. Create a folder on your local drive (for example, C:\Manifests).
    2. Right click on the folder, select **properties**.
    3. Click on **Sharing tab**.
    4. Click on **Advanced Sharing...**
    5. Check the **Share this folder** box.
    6. Click **Apply** and then **Ok**.
2. Tell Excel or Word to use the directory as trusted app catalog.

1. Launch Excel and open a blank spreadsheet.
2. Choose the **File tab**, and then choose **Options**.
3. Choose **Trust Center**, and then choose the **Trust Center Settings** button.
4. Choose **Trusted Add-in Catalogs**.
5. In the **Catalog URL box**, enter **the path to the network share you created**, and then choose **Add Catalog**.
   To see the path: right click on the shared folder -> Properties -> Sharing -> Network Path.
6. Select the **Show in Menu** check box, and then choose **OK**. A message appears to inform you that your settings will be applied the next time you start Office.
7. Close Excel and restart it.

Manifest directory for macOS

On Mac you need to create a folder to save the manifest of the add-in.

Go to one of the following folders where you'll save your add-in's manifest file. If the wef folder doesn't exist on your computer, create it.

- For Excel: /Users/<username>/Library/Containers/com.microsoft.Excel/Data/Documents/wef
- For Word: /Users/<username>/Library/Containers/com.microsoft.Word/Data/Documents/wef
- For PowerPoint: /Users/<username>/Library/Containers/com.microsoft.Powerpoint/Data/Documents/wef

where **<username>** is your name on the device.

Get the Office Add-in manifest

You can now download the manifest of the add-in you want to use.

- Download the XML BananaAccountingExcelManifest file.
- Copy the manifest to the manifest directory.

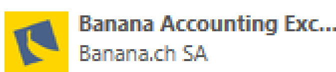
Load the Add-ins in Excel

Once all the setup and installations are done, it is possible to run and use the add-in.

1. Open Microsoft Excel 2016
2. Click on **Insert tab**
3. Click on the **Add-ins** button
4. Click on the **Shared folder**

   Office Add-ins

   MY ADD-INS | SHARED FOLDER | STORE

5. Select the Banana Accounting Add-in

6. Click Add

Other Resources

For more and detailed information about the developing of the Office Add-ins, please visit https://github.com/BananaAccounting/General/tree/master/OfficeAddIns.

Introduction to Excel 2016 Add-ins

Office 2016 Add-ins are extentions of Word, Excel, PowerPoint, and Outlook.
Add-ins are composed of:

- **Manifest file**
  An XML file that defines various settings, including description and links to all the add-in files.
  It is used by Word, Excel, PowerPoint, and Outlook to locate the Add-in resources.
  The manifest file can reside on a local directory or is published on the Office Store.
- **Webpage files**
  Files that compose the web app (HTML pages, JavaScript code and images).
  All the files need to reside on a web server.

Add-in Examples

These examples have been made available for programmers that want to create specialized add-ins to retrieve information from Banana Accounting.
You need to insall the add-ins on a web server.

- for Excel:
  - Account Card report to create an Excel worksheet with details of an account.
  - Retrieve Table report to create an Excel worksheet with a full table taken from the accounting.
- for Word:
  - Account Card report to create a Word document with details of an account.

ExcelSync functions

With ExcelSync your accounting data is available in Excel. No more need to copy and paste or to export and import.

You add new transactions and your Excel Sheets are instantly updated and calculated. For Apple/Mac this feature **is not available.**

ExcelSync uses VBA Macros. This technology has been replaced by the more recent Excel Add-in.
We invite you to use the Excel Report Add-in.

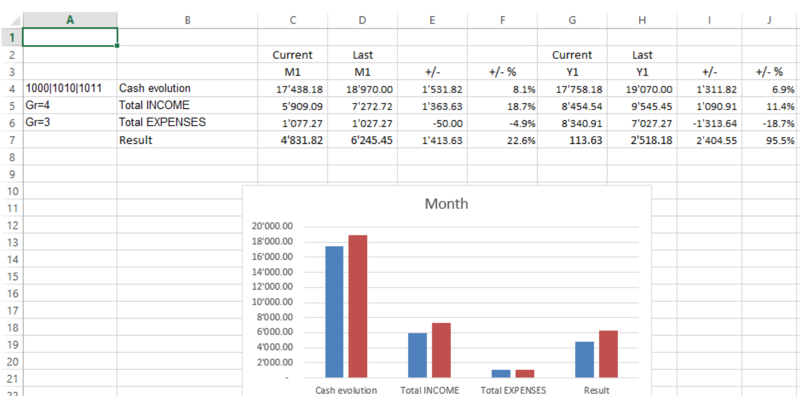
Example costs divided among co-owners or customers

The ExcelSync functions are used to retrieve from Banana in Excel the current accounts balances.
The costs are then divided among customers using normal Excel Formulas.
You could use the example to create a division of the apartment costs.

	A	B	C	D	E	F	G
1	**Sibex Ltd**						
2	**Costs Division by customers**						
3	Period start	1 Gennaio 2015					
4	Period end	31 Dicembre 2015					
5	Accounts		Total	Smith	Bianchi	xx	Muller
6		Percentage division	100%	20%	50%		30%
7	3220	Car expenses	-	-	-	-	-
8	3230	Other insurances	-	-	-	-	-
9	3240	Fees	-	-	-	-	-
10	3250	Electricity	-	-	-	-	-
11	3260	Office supplies	27.27	5.45	13.64	-	8.18
12	3270	Telephone, Fax	345.46	69.09	172.73	-	103.64
13	3280	Mailing fees	-	-	-	-	-
14	3290	Advertising	-	-	-	-	-
15		**Total**	**372.73**	**74.55**	**186.37**	-	**111.82**
16		Down payment	300.00	100.00	100.00		100.00
17		**Total Due**	**72.73**	**-25.45**	**86.37**	-	**11.82**
18	Rest distribution		-				
19							
20	Data to be entered by users						
21	Other cells are automatically calculated						

Example with the current and last year difference

In this example we take the data from the two years and create a graphic.



	A	B	C	D	E	F	G	H	I	J
1										
2			Current	Last			Current	Last		
3			M1	M1	+/-	+/- %	Y1	Y1	+/-	+/- %
4	1000|1010|1011	Cash evolution	17'438.18	18'970.00	1'531.82	8.1%	17'758.18	19'070.00	1'311.82	6.9%
5	Gr=4	Total INCOME	5'909.09	7'272.72	1'363.63	18.7%	8'454.54	9'545.45	1'090.91	11.4%
6	Gr=3	Total EXPENSES	1'077.27	1'027.27	-50.00	-4.9%	8'340.91	7'027.27	-1'313.64	-18.7%
7		Result	4'831.82	6'245.45	1'413.63	22.6%	113.63	2'518.18	2'404.55	95.5%

Example with segments subdivision

The amount of the segments are diveded also by segments.

Sibex Ltd						
Period start	1 Gennaio 2015					
Period end	31 Dicembre 2015					
Cost per segments						
Account	Gr=3					
		Segments Level 2				
		Not Assigned	Sales	Administratio	R&D	
Level 1		{}	SA	AD	RD	**Total**
{}	Not Assigned	-	-	-	-	-
LU	Lugano	-	4.818,18	1.050,00	-	**5.868,18**
NY	New York	-	-	-	2.100,00	**2.100,00**
HK	Hong Kong	-	-	372,73	-	**372,73**
	Total	-	**4.818,18**	**1.422,73**	**2.100,00**	**8.340,91**
Formula example: BAmount(file0;"Gr=3:LU:SA")						
{} Not assigned to check if all accounts have a segment						

Introduction to Banana ExcelSync User defined functions

Introduction

ExcelSync are Excel User defined functions that allow to synchronize in real time your Excel spreadsheet with the data from Banana Accounting.
You update you accounting file, adding new transactions, and instantly you get your Excel Sheets updated.

Excel has the ability to integrate documents and data that are made available throught the internet protocol. Banana includes a web server, and a RESTful API, that can be accessed through http protocol. ExcelSync uses the Banana integrated web server to retrive data on real time.

Using Excel formula

Banana ExcelSync are functions, with the name that start witht the "B", that you can use within the cell to retrive accounting data.
Here some example:

```
// return the opening balance of the account 1000 for all the period
=BOpening("1000")
// return the description of the account 1000
=BAccountDescription("1000")
// return the end balance of the group 10
=BBalance("Gr=10")
// return the opening of the account 5000 for the period 3. month
=BOpening("2000", "2017-03-01/2017-03-31")
// return the total debit minus credit of the account 5000 for 3. month of
the year
=BTotal("5000", "M3")
// return the total debit minus credit of the group 50 for 3. quarter of the
year
=BTotal("Gr=50", "Q3")
```

The advantage of the Excel Sync functions :

- You can dynamically retrieve the take data from Banana Accounting.
- No more need to retype data in Excel (or import, copy and paste)

- When the accounting file is changed, the spreadsheet is populated with the new values
- Easy to use formulas that let you calculate values for periods and create powerful spreadsheets for evaluating, presenting accounting data or creating graphics.

Tecnical details

Banana ExcelSync are Excel User defined functions (UDF), small Visual Basic Programs that extend Excel allowing to insert formula within the cell.

- Banana ExcelSync requires a recent version of Excel, and due to the Excel Mac limitations works only on Windows versions.
- In order to use the ExcelSync UDF you need an Excel file with the extension *.xlsm.
- The Banana ExcelSync UDF are provided according the Apache License (open source software. See: /www.apache.org/licenses/LICENSE-2.0
- Development and latest version of the function are available on github.com/BananaAccounting/General/
- Banana ExcelSync UDF make use of the Banana web server.
- You can extend the ExcelSync by adding other functionalities.

For more information on the formula used see the Banana API regarding the Accounting functions

Using the examples

1. **Download the Excel spreadsheet with examples files.**
2. Unzip the content
3. Start Banana Accounting
4. Activate the Webserver (**Tools** -> **Program options** -> **Interface** -> **Start web server)**
5. Open the Banana accounting files "company_2019.ac2" and "company_2020.ac2"
6. Open the "BananaSync.xlsm" file and activate the Macro
   If the macro are automatically disabled by Excel you should change your macro security setting
   Eventually follow this instructions to show the developer tab in the ribbon
7. Recalculate the Spreadsheet with the Macro “RecalculateAll” (Ctrl+R)

Excel does not react

If you open a file and Banana or the Banana Web Server are not running, Excel will wait until it can contact the Banana Web server.

Start the Banana and the Banana Web Server.

How to create your spreadsheet

- Save as the "BananaSync.xlsm" file with another name
- Open your accounting files in Banana Accounting
- In your Excel spreadsheet, replace the file name (yellow highlighted cells) with your accounting file name
- Change the spreadsheet according to your needs
- Recalculate with the "Recalculate" button or the "Ctrl+R" shortcut

Functions use

Argument file name

Most ExcelSync functions require, as first parameters, a name of a Banana Accounting file.

- The file must be openened in Banana.
- You use only the file name without the directory.

DO NOT use the file name directly in the fuctions. Instead use a reference to a cell, that contains the file name.

- You can use the same spreadsheed also for differrent years. You only need to change the file name in on cell.
- If Banana Accounting is not open of the Banana webserver is not active you don't have to wait.

The best way is the one used in the example file.

- The file name of the current year is taken from the cell named "File0" .
- The cell File0 contains a function =BFileName(DisableConnection).
  This function checks if the file is open in Banana.
  - If the file is not open the content of the cell is set to an empty string.
    The other Banana Sync functions will not make any call to Banana, to retrive data.
  - If the file is open it will insert the name of the file.
- The cell B6 contain the name of the file to be used. **Insert the file name in cell B6. =BFileNameF(File0, DisableConnection).**
- The file name of the current year is taken from the cell named "File0" .
- The file name of the last year is taken from the cell named "File1" .

Argument period

Many functions use the optional argument period. This can be:

- An empty string. The start and end date of the accounting are used.
- A start date and end date in the form of yyyy-mm-dd/yyyy-mm-dd
  example “2015-01-01/2015-01-31”
  In order to create a period from two Excel dates use the function BCreatePeriod.
- An abbreviation
  With the abbreviation you can easily use the same spreadsheet for accounting file of different periods.
  The start and the end date will be determined based on the date of the accounting file
  - M + the month number M1, M2, ..
  - Q + the quarter number Q1, Q2,
  - Y + the year number Y1, Y2, ...

BananaSync Functions description

Most function are available

- Without the parameter FileName.
  In this case the File0 (Current Year) is used
- With the paramenter FileName.

- The function is the same but end with "F"

BAccountDescription(account[, column]) and BAccountDescriptionF(fileName, account[, column])

Retrieve the account *description* of the specified account or group.
With *argument* column you can indicate to retrieve another column instead of the Description column.
Examples:

```
=BAccountDescription("1000")          // Description of account 1000 current year
=BAccountDescription("Gr=10")         // Description of Group 10 current year
=BAccountDescription("1000", "Gr1")   // Contet of column "Gr1" relative to the account 1000 current year
// Last year
=BAccountDescriptionF(File1, "1000") // Desctiption of account 1000
=BAccountDescriptionF(File1, "Gr=10")       // Description of Group 10
```

BAmount(account, [,period]) and BAmountF(fileName, account, [,period])

Retrieve the normalized amount based on the BClass.
Only work for double entry accounting only. For Income and expenses accounting use BBalance or BTotal.

- for accounts of BClass 1 or 2 it return the balance (value at a specific instant).
- for accounts of BClass 3 or 4 it return the total (value for the duration).
- For accounts of BClass 2 and 4 the amount is inverted.

You can use this functions also with groups provided you assign a BClass also to a group.

BBalance(account [, period]) and BBalanceF(fileName account [, period])

Retrieve the Balance at the end of the period of the indicate account, cost center, groups, segments
The BBalance result is the sum of the BOpening + BTotal
It is used for retrieving accounting data for the Balance Sheet accounts (Assets, Liabilities)

- Single account number ("1000")
- Several accounts summed toghether.
  Enter the accounts numbers separated by the character "|" ("1000|1001).
- You can specify normal accounts, cost centers or segments.
- You can also use wild cards and also use "Gr=" followed by the accounting group.
- For more information see the Javascript function description for currentBalance
- Example

```
BBalance( "1000")              // Balance of account 1000
BBalance( "1000|1010")         // Balance of account 1000 and 1010 are summed together
BBalance( "10*|20*")           // All account that start with 10 or with 20 are summed toghether
BBalance( "Gr=10")             // Group 10
BBalance( "Gr=10| Gr=20")      // Group 10 or  29
```

```
BBalance( ".P1")                // Cost center .P1
BBalance( ";C01|;C02")          // Cost center ;C01 and C2
BBalance( ":S1|S2")             // Segment :S1  and :S2
BBalance( "1000:S1:T1")         // Account 1000 with segment :S1 or ::T1
BBalance( "1000:{}")            // Account 1000 with segment not assigned
BBalance( "1000:S1|S2:T1|T2")  // Account 1000 with segment :S1 or ::S2 and
::T1 and ::T
BBalance( "1000&&JCC1=P1")      // Account 1000 and cost center .P1
// Last year
BBalanceF(File1, "1000")         // Balance of account 1000 (last year)
BBalanceF(File1, "1000|1010")    // Balance of account 1000 and 1010 are
summed together (last year)
```

BBalanceGet(account, cmd, valueName [,period]) and BBalanceGetF(fileName, account, cmd, valueName [,period])

This function allows to easily access all other data made available by the REST API as “balance”, “budget”
Examples:

```
=BAmount( “1000”, “balance”, “currencyamount”)
=BAmount( “1000”, “balance”, “count”)
=BAmount( “1000”, “balance”, “debit”)
// Last year
=BAmount( File0, “1000”, “budget”, “debit”)
```

BBudgetAmount(account [, period]) and BBudgetAmountF(fileName account [, period])

Same as BAmount but use the budget data instead of the accounting data.

BBudgetBalance(account [, period]) and BBudgetBalanceF(fileName account [, period])

Same as BBalance but use the budget data instead of the accounting data.

BBudgetInterest(account, interestRate [, period]) and BBudgetInterestF(filename, account, interestRate [, period])

Same as BInterest but use the budget data instead of the accounting data.

BBudgetOpening(account [, period]) and BBudgetOpeningF(fileName account [, period])

Same as the BOpening but use the budget data instead of the accounting data.

BBudgetTotal(account [, period]) and BBudgetTotalF(fileName account [, period])

Same as the BTotal but use the budget data instead of the accounting data.

BCellAmount(table, rowColumn, column) and BCellAmountF(fileName, table, rowColumn, column)

Retrieve the content of a table cell as an amount.
Examples:

```
=BCellAmount("Accounts", 2, "Opening")
=BCellAmount("Accounts", "Account=1000", "Balance")
=BCellAmount("Accounts", "Group=10", "Balance")
// Last year
=BCellAmountF(File1, "Accounts", 2, "Opening")
```

BCellValue(table, rowColumn, column) and BCellValueF(fileName, table, rowColumn, column)

Retrieve the content of a table cell as a text.
Examples:

```
=BCellValue("Accounts", 2, "Description")
=BCellValue("Accounts", "Account=1000", "Description")
=BCellValue("Accounts", "Group=10", "Description")
// Last year
=BCellValueF(File1, "Accounts", 2, "Description")
```

BCreatePeriod(startDate, endDate)

Take two cell dates and create a string period
=BCreatePeriod(D4, D5)

BDate(isoDate)

Convert an Iso Date to an Excel date.

BFileName(fileName [, disable connection])

Return the FileName or an empty string if there is no connection with the web server or if the file is not correct.
If the value of disableConnection is not void the function returns an empty string.
Use the cells that contain the result of this function as the file name parameter when using the other functions. If Banana is not open only one query is made and Excel will not wait for a long time.

BFunctionsVersion()

Return the version of the function in the date format.

BInfo(sectionXml, idXml) and BInfoF(fileName, sectionXml, idXml)

Retrieve information regarding the file properties.
Examples:

```
=BInfo("Base", "HeaderLeft")
=BInfo("Base", "DateLastSaved")
=BInfo("AccountingDataBase", "OpeningDate")
=BInfo("AccountingDataBase", "BasicCurrency")
// Last year
=BInfoF( File1, "Base", "HeaderLeft")
```

BInterest(account, interestRate [, period]) and BInterestF(filename, account, interestRate [, period])

Calculate the interest for this account for the specified period

account can be any account as specified in BBalance

interestRate in percentage

- > 0 calculate the interest on the debit amounts
- < 0 calculate the interest on the credit amount

BOpening(account [period]) and BOpeningF(filename, account [period])

Retrieve the Balance for balance of period start for the indicated account.

BQuery(fileName, query)

Return the result of a free defined query.
Examples:

```
=BQuery(File0;"startperiod?M1”)
=BQuery(File0;"startperiod?M1”)
```

BTotal(account [,period]) and BTotalF(filename, account [,period])

Retrieve the movement for the period.
Should be used to retrieve the data for the Profit and Loss accounts (Cost and Revenues).

BVatBalance(vatCode, vatValue [, period]) and BVatBalanceF(filename, vatCode, vatValue [, period])

Return a value regarding the specified VatCode (or multiple VatCodes).
“vatValue” can be “taxable”, “amount”, “notdeductible”, “posted”
Examples:

```
=BVatBalance(“V10”, “taxable”)
=BVatBalance(“V10|V20”, “posted”)
//Last year
=BVatBalanceF( File0, “V10”, “taxable”)
```

Additional function explanation

The retrieve the exact content of the cells

If you wanto to retrive the content of a cell you can use:

- BCellValue
  The content of a cell, useful for text.
- BCellAmount
  The content of a cell is converted to a number so that you can use it for calculation.
  With this you will retrive the exact content of a column "Balance" for the row where Account is 1000.
  If the Balance is credit the amount is negative.

```
=BCellAmount(File0, “Accounts”, “Account=1000”, “Balance”)
```

Accounting Period calculation

You have different formula that allow to retrieve the amount.

- **BBalance**.
  This is equivalent to the above. It retrieve the Balance of the whole accounting period.
  But BBalance allow you to use also a period.
  As a period you can use the date being, date end of an abbreviation. M3 means the first month of the accounting period.
  If you use abbreviation instead of date your sheet will automatically adapt to file of different year.

```
BBalance( "1000")  //Balance end of year
BBalance( "1000", "2017-03-01", "2017-03-31")  //Balance end of March
BBalance( "1000", M3);   // Balance and of March if accounting period start
on 1. of January
```

- **BTotal**
  It retrieve the total movement (Debit - Credit) for the period.
  Use BTotal to the amount for income and expenss account.
  Cedit amounts are retrieved as negative numbers.

```
BTotal( "1000")  //Total movement end of year
BBalance( "1000", "2017-03-01", "2017-03-31")  //Total end of March
BBalance( "1000", M3);   // Total and of March if accounting period start on
1. of January
```

- **BAmount**
  BAmount put the sign in positive based on the BClass of the account.
  The amount retrieved depend on the BClass of the account or the group.
  For Balance accounts (bclass 1 and 1) retrieve the Balance.
  For Income and expenses accounts (bclass 3 and 4) retrieve the Total.
  It also invert the sign in case of BClass 2 and 4.
  So if you use BAmount for the Account revenues (BClass 4) you will have the total sales for the period in positive.

Your are free to use the most appropriate function.

- BAccountDescription.
  It is the same as GetCellValue but it deal automatically with accounts or groups.
  Is usefull to retrieve the description of an account or group, in combination with BBalance, BTotal or BAmount.

=BAccountDescription("1000")   //Retrieve the column Description of the account 1000
=BAccountDescription("1000", "Notes") //Retrieve the column Notes of the account 1000
=BAccountDescription("Gr=10")   //Retrieve the column Descrition of the group 10

Recalculate

The automatic recalculation does not update the data from the accounting file.
In order to have the data updated it is necessary to call the macro RecalculateAll() that call the method Application.CalculateFullRebuild

The example files contain a button “Recalculate” that call the macro RecalculateAll.

Banana host name and port

Web server data is retrieved from “localhost:8081”

You can specify a different host by entering a value in a cell named “BananaHostName”

Modify the functions or add your owns

Functions are defined in the Visual Basic module “Banana”.
If you add your function it would be better to add to your module.

To access the Visual Basic Macro Functionalities you should activate the macro.

In order to see and edit the functions your nedd to [show the Developer tab in the Excel ribbon.]()

Use a new version of the Banana functions

In order to see and edit the functions your nedd to [show the Developer tab in the Excel ribbon.]()

- Download on your computer the latest version
- Open your file in Excel
- Open the file "BananaSync.xlsm" in Excel
- Go to the Developer Tab
- Click on "Visual Basic"
- Copy the content of the "BananaSync.xmls - Banana (Code)"
- Paste the content in the Modules->Banana of your file.

Compatibility

Banana ExcelSync functions have been tested with Excel 2013 and 2016 for Windows.

Excel for Mac is not ready yet.

In Excel for Mac is not possible to call the http.
Any contribute to solve this problem is welcome.

Release History

- 2014-07-24 First release
- 2015-02-28 Updated for new version with new functionalities
- 2015-05-12 Call to webserver now require v1
- 2015-05-12 Development moved to github
- 2015-05-25 Changed BAmoount function to use BClass
- 2015-10-04 Added BDate function
- 2015-11-12 Renamend ExcelSync
- 2015-11-28
    - Added example for cost division
    - Started working on Mac support
- 2016-04-26 Added BCellAmount
- 2016-04-28 Fixes in some case rounding amount to zero

- 2017-02-01 New Version 2 (Functions without the file parameters)

How to report a bug

Send us full details of the issue, giving as much detail as possible, this can include:

- For bugs:
    - Steps to Reproduce:
      Minimized, easy-to-follow steps that will trigger the described problem. Include any special setup steps;
    - Actual Results:
      What the application did after performing the above steps;
    - Expected Results:
      What the application should have done, if there was no bug;
    - The accounting file:
      An example of an accounting file where the problem occur;
    - The system informations
      The dialog system informations display the information to attach to the request;
    - The system event log
      If he application has stopped working or disappears attach a copy of the system event log related to the crash;

- For feature requests:
    - A description of what you would like to achieve, and why.
      A user story is an effective way of conveying this;

You can submit your issue through our contact form.

Rare Cases

Computers configurations can be quite different.
Banana Accounting is known to work seamleassy on most system, but in rare cases there may be some problems.

If you are experiencing a problem starting or using the application follow these steps:

- Install the latest version of the application
- Install the latest version of the graphics driver:
  Visit the manufacturer website, download the driver file and install it
- Install the latest version of the printer driver.
  Visit the manufacturer website, download the driver file and install it
- Consult the troubleshooting pages

Bug reporting

- Report the bug

Rare cases troubleshooting pages

Banana9 hangs on startup on a system with two monitors
Banana9 hangs on startup with the message "LoadLibrary failed with error XX: Wrong parameter."
Banana9 hangs on startup with the message "dll is missing" or "error 0xc000007b"
Banana9 hangs some seconds after the main page of the program is showed
Banana9 hangs when the Open File Dialog or the Save File Dialog is opened
Banana9 hangs when trying to print

Banana9 hangs on startup on a system with two monitors

Problem

Banana 9 stops working on startup.

System

The computer has an ATI Dispaly card with two attached monitors.

Affected OS: Windows 10, Windows 8, Windows 7

Solution

- Remove (just an update is not enough) and install again the ATI driver;

- Try to start the application on the AMD window / monitor.

- Look for more solutions at: https://bugreports.qt.io/browse/QTBUG-50371

Resources

- Qt bug reports: https://bugreports.qt.io/browse/QTBUG-50371

product:
Banana 8

Banana9 hangs on startup with the message "LoadLibrary failed with error XX: Wrong parameter."

Problem

Banana 9 stops working on startup. After the crash the message "LoadLibrary failed with error XX: Wrong parameter." is displayed.

System

The system is an old notebook where windows 10 was installed afterwards.
The notebook's discrete graphic card is not officially supported by windows 10.

Affected OS: Windows 10

Solution

- Update the graphic card driver

or

- Rename in folder C:\Windows\System32 the file "atig6pxx.dll" to "atig6pxx.dll.bak"

Banana9 hangs on startup with the message "dll is missing" or "error 0xc000007b"

Problem

Banana Accounting 9 does not start and a message stating one of the following texts appears:

- The program can't start because a dll is missing from your computer.
- **The application was unable to start correctly** accompanied by an error code (**0xc000007b**)

System

Usually this error occurs on Computers running Windows 7 or Vista, sometimes it also occurs on Windows 8 and 8.1.

Cause

Probably your system is not up to date.

Solution

Run Windows update and reinstall Microsoft Visual C++ redistributable

1. Run windows update
    1. Select Start > Control Panel > System and Security > Windows Update.
    2. Select "Check for updates." Wait as the Windows Update tool scans checks for updates that you have not installed
    3. If you see a message that says, "Updates are available for your computer" or "Install updates for your computer," click the button labeled "Install updates."
    4. Wait as the updates install. Restart your computer when prompted to complete the updating process.
2. Reinstall Microsoft Visual C++
    1. Open **Programs and Features**
    2. **Uninstall** all items with the name of "Microsoft Visual C++ 2*** Redistributable"
    3. Download the latest Microsoft Visual C++ redistributable packages
        1. Click here for 32 bit systems
    4. Run the file vcredist.exe that you just downloaded
3. If the problem persists, force Windows Update to install missing components:
    1. Go to this Microsoft Support webpage
    2. Scroll down in the "Method 2" section.
    3. Download and install the package for your Operative System

Banana9 hangs some seconds after the main page of the program is showed

Problem

Banana 9 stops working a few seconds after the main page of the program is showed.

System

The system is running on a Acer notebook

Affected machines: Acer aspire es 17
Affected OS: Windows 10, Windows 8

Solution

- Open in Explorer the folder C:\Program files\Banana90

- Create a new folder 'backup_dll'

- Move the files 'libeay32.dll' and 'ssleay32.dll' to the folder 'backup_dll'

- Restart the application

Banana9 hangs when the Open File Dialog or the Save File Dialog is opened

Problem

Banana 9 stops working as soon an Open File Dialog or a Save File Dialog is opened.

System

The system is running on a Dell computer or notebook.
The software Dell Backup and Recovery (from Softthink) is installed and running on the system.

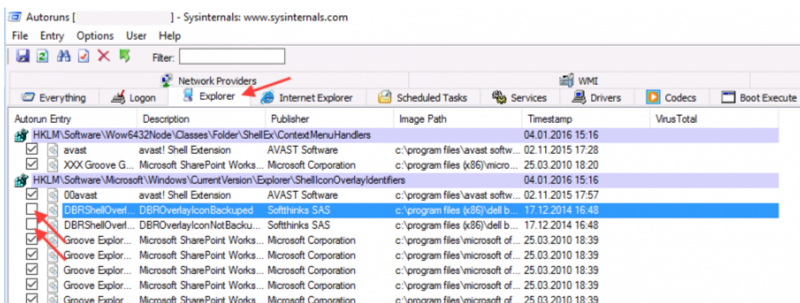
Affected OS: Windows 10, Windows 8, Windows 7
Affected systems: Dell computer

Solution A (recommended)

- Update Dell Backup and Recovery tools to version 1.9.2.8 or greather
- Check Dell User's Guides and Support pages on how to update your system

Solution B

- Download the [Autoruns for Windows - TechNet - Microsoft] utility;
- Expand the file Autoruns.zip;
- Run the application Autoruns.exe as Administrator;
- Select the "Explorer" tab;
- Deselect **all** "DBRShellOverlay" entries from Softthinks SAS Publisher



- Close Autoruns;

- Restart Banana 9.

Notes

- In one case, after the changes in the registry, the customer had some problems running Internet explorer.
  Those problems could be resolved by restoring the changes in the registry.
- The number of "DBRShellOverlay" entries to be disable can be on some systems more than two, those entries have to be searched in all sections listed in the tab Explorer.

Resources

- Dell Forum page "Backup and Recovery causing applications using Qt5 DLLs to crash": http://en.community.dell.com/support-forums/software-os/f/3526/t/19634253

Banana9 hangs when trying to print

Problem

Banana 9 hangs the first or second time the print dialog is opened.

System

The 'Devices and Printers' configuration contains printers no longer attached to the system.

Affected OS: Windows 10, Windows 8, Windows 7

Solution

- Open Start and select 'Devices and Printers'

- Remove all printers no longer used or attached to the system