CDR retrieval and fraudulent voice traffic monitoring with Excel VBA database connection to billing / radius server

Emin Gabrielyan  
2013-11-01

# Table of contents

[1. Table of contents 1](#_Toc371088498)

[2. Introduction 1](#_Toc371088499)

[3. The Excel file worksheets 2](#_Toc371088500)

[4. Worksheet parameters 3](#_Toc371088501)

[5. Calculating the statistics 8](#_Toc371088502)

[6. Calculating the rates 11](#_Toc371088503)

[7. Chart samples 18](#_Toc371088504)

[8. VBA script 21](#_Toc371088505)

[9. Named ranges 59](#_Toc371088506)

[10. Recent functionalities 60](#_Toc371088507)

[11. Installation 66](#_Toc371088508)

[12. References 66](#_Toc371088509)

[13. Link log 67](#_Toc371088510)

[14. Acronyms 68](#_Toc371088511)

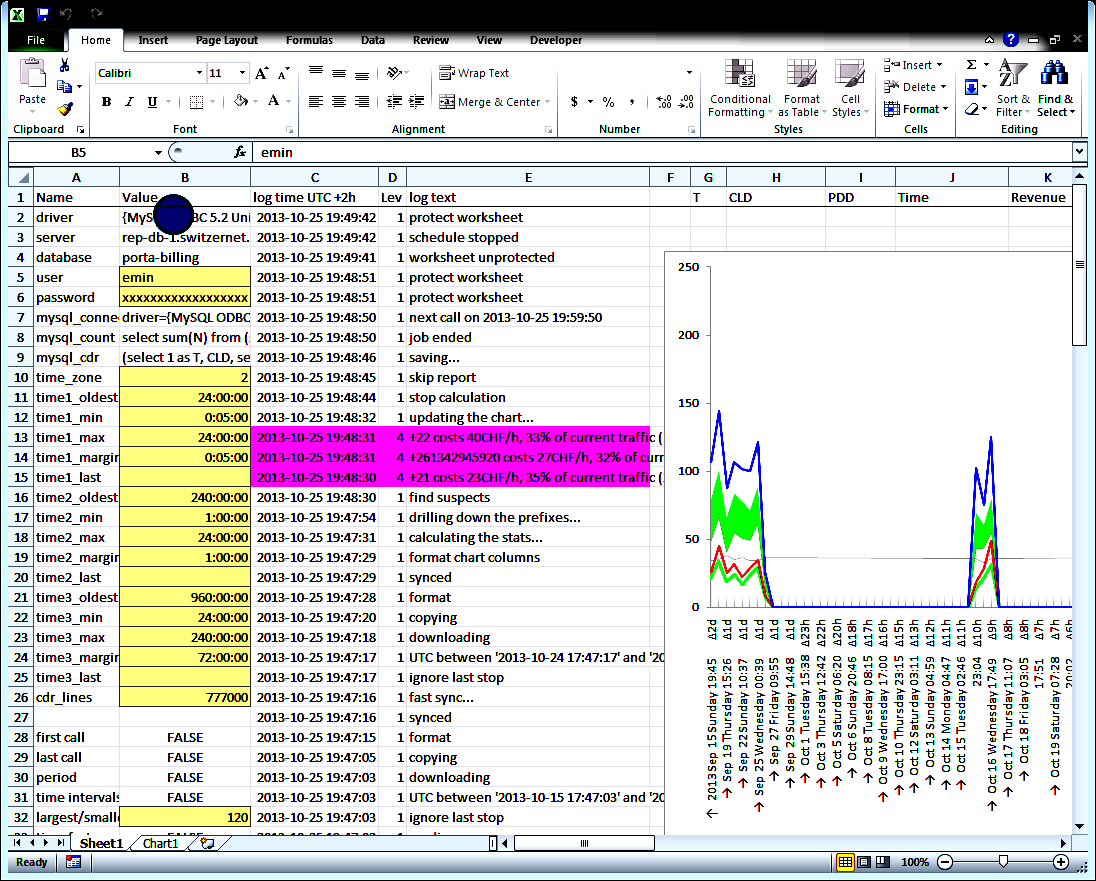
[15. Legal 69](#_Toc371088512)

# Introduction

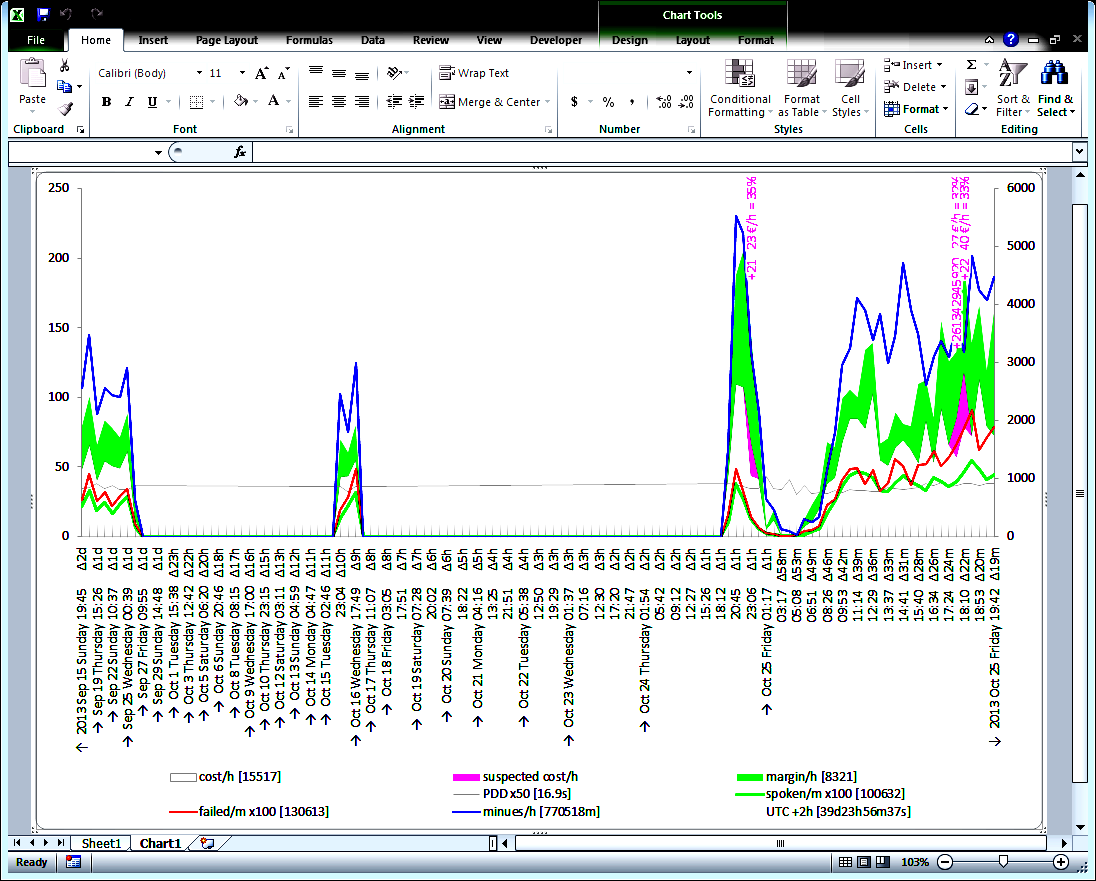
We present a real-time monitoring application replicating from the voice billing RADIUS database server the flow of the call records and continuously analyzing it for fraudulent patterns. The entire application fits in a single Microsoft Excel file and is tested on processing rates of about a million records per ten-minute intervals. The connection to the billing database radius server relies on the ADODB module of the Excel VBA. Whenever a suspicious activity is detected a graphical chart highlighting the fraudulent cost patterns is emailed to a list of recipients defined by the user. Both, the priority level and the recipient list of the outgoing emails change automatically as a function of the level of the suspect detected in incoming traffic patterns. Highly suspicious patterns suggesting fraudulent activities result also in the generation and sending of Excel files with concerned CDR portions where all suspicious call records are highlighted. The application also sends a normal priority heartbeat messages twice a day. The email transmission is based on the CDO module of the Excel VBA. The pattern analysis is carried out by drilling down and finding the longest heavily used phone prefixes. We do not rely on a table of predefined destinations. The prefixes are discovered dynamically by analyzing only the call records and can turn out to be very long and specific when representing fraud scenarios involving national and international value added numbers. A suspicious prefix discovered by the algorithm may not exist at all in the table of destinations and must be created for the further blockage.

# The Excel file worksheets

The Excel workbook consists of two sheets. The first sheet contains the parameters and the data, including CDR downloaded from the billing and the statistics computed with Excel formulas. The first two columns of the data sheet are reserved for the parameters.



The second sheet contains only the chart.



# Worksheet parameters

In this section we describe all worksheet parameters occupying the first two columns of the data worksheet. All parameters used by Excel VBA subroutines as well as by statistics and chart construction formulas are under the column B. Column A contains only a title for the user. The cells highlighted in yellow are input parameters expected from the user.

|  |  |  |  |
| --- | --- | --- | --- |
|  | A | B | **Comments** |
| 1 | **Name** | **Value** |  |
| 2 | driver | {MySQL ODBC 5.2 Unicode Driver} |  |
| 3 | server | rep-db-1.switzernet.com |  |
| 4 | database | porta-billing |  |
| 5 | user | emin |  |
| 6 | password | xxxxxxxxxxxxxxxxxx |  |
| 7 | mysql\_connect | xxxxxxxxxxxxxxxxxx | **=A2&"="&B2&"; "&A3&"="&B3&"; "&A4&"="&B4&"; "&A5&"="&B5&"; "&A6&"="&B6&"; option=3"** |
| 8 | mysql\_count | select sum(N) from (select count(1) as N from CDR\_Vendors where disconnect\_time [between] union all select count(2) from CDR\_Vendors\_Failed where connect\_time [between]) as T | Template for the MySQL request counting the calls within a specified period. |
| 9 | mysql\_cdr | (select 1 as T, CLD, setup\_time, disconnect\_time as time, revenue, charged\_amount, account\_id, charged\_quantity from CDR\_Vendors where disconnect\_time [between]) union all (select 2, CLD, setup\_time, connect\_time, null, null, account\_id, null from CDR\_Vendors\_Failed where connect\_time [between]) order by time | Template for the MySQL request retrieving the CDR of a specified period. |
| 10 | time\_zone | 1 | The time zone in hours |
| 11 |  |  |  |
| 12 | timing | fast | A part of the [timing rhythms] named range |
| 13 | oldest | 24:00:00 |
| 14 | min | 0:05:00 |
| 15 | max | 24:00:00 |
| 16 | margin | 0:05:00 |
| 17 | UTC last |  |
| 18 |  |  |  |
| 19 | timing | medium | A part of the [timing rhythms] named range |
| 20 | oldest | 240:00:00 |
| 21 | min | 1:00:00 |
| 22 | max | 24:00:00 |
| 23 | margin | 1:00:00 |
| 24 | UTC last |  |
| 25 |  |  |  |
| 26 | timing | slow | A part of the [timing rhythms] named range |
| 27 | oldest | 960:00:00 |
| 28 | min | 24:00:00 |
| 29 | max | 240:00:00 |
| 30 | margin | 72:00:00 |
| 31 | UTC last |  |
| 32 |  |  |  |
| 33 | cdr\_lines | 777000 |  |
| 34 |  |  |  |
| 35 | first call | FALSE | **=IF(calc1,MIN(INDEX(cdr,,4)))** |
| 36 | last call | FALSE | **=IF(calc1,MAX(INDEX(cdr,,4)))** |
| 37 | period | FALSE | **=IF(calc1,MAX(B36-B35,TIME(0,1,1)))** |
| 38 | time intervals | FALSE | **=IF(calc1,MAX(P:P))**  Number of intervals in the chart |
| 39 | largest/smallest | 90 | Ratio between the largest (the oldest) and shortest (the most recent intervals) |
| 40 | time factor | FALSE | **=IF(calc1,B39^(1/B38))**  Factor between two adjacent intervals |
| 41 | smallest | FALSE | **=IF(calc1,B37\*(B40-1)/(B39\*B40-1))**  Computing the smallest interval as a function of the period, time factor, and the largest to smallest factor. See the formulas in the previous publications (see in the references section). |
| 42 | largest | FALSE | **=IF(calc1,B39\*B41)**  The largest interval is the factor of the largest over the smallest interval multiplied by the duration of the smallest interval |
| 43 | spoken | 10 | The duration of a call in seconds to be considered as a valid conversation. All calls below this duration are considered as failed. |
| 44 | drilling cost/h | 77 | The minimal hourly cost of the interval for prefix drilling. If the interval’s hourly cost does not reach this level no prefix drilling is carried out. |
| 45 | drill down until | 30.0% | A longer prefix is searched as long as the prefix continues to represent this percentage of the interval’s cost (or above). |
| 46 | shortest prefix | 1 | The length of the prefix is set 1, meaning that all lengths of prefixes are considered. |
| 47 | suspect\_factor | 22 | The heaviest prefix will be considered suspicious if within the examined interval its weight (its hourly cost) is at least 22 times more than its usual average hourly cost measured over the entire observation period. |
| 48 | suspect\_new |  | If new suspects are found (in the most recent interval) their alert message is recorded here. |
| 49 | PDD factor | 50 | In order to scale the PDD curve together with the traffic curve (the curve of terminated minutes per hour) the PDD values are multiplied by this factor. The PDD header is accompanied with “x50” prefix to show this factor to the user. |
| 50 | Call factor | 100 | In order to scale the call rate (number of failed or answered calls per minute) curve with the traffic curve, the call rates are multiplied by this factor. The factor is reflected in the call rate headers. |
| 51 | calc1 | FALSE | Almost all formulas in the worksheet use this cell. Only if the value of this cell equal to true, the formulas will carry out the rest of their calculations. The value of this cell is set to false by the macro before downloading data. It is set to true before the generation of the chart data and is again set back to false after the chart data generation is complete and the chart data values are copied. |
| 52 | calc2 | TRUE | The value of this cell is used by drilling formulas. The drilling formulas will be activated only if the values of both this and the previous cells are true. |
| 53 | max labels | 70 | The maximal number of cells to visualize on the chart’s horizontal time axis. |
| 54 | label every | FALSE | **=IF(calc1,CEILING(B38/B53,1))**  Computing the number of time-axis points per one label. |
| 55 |  |  |  |
| 56 | smtp\_debug | FALSE | If this value is true the emails will be sent only to a single address, the first email in the SMTP TO range. |
| 57 | smtp\_to | Emin Gabrielyan <emin.gabrielyan@gmail.com> | The range with all email addresses of the to-field |
| 58 | smtp\_to | Emin Gabrielyan <emin.gabrielyan@switzernet.com> |
| 59 | smtp\_to | Nicolas Bondier <nicolas.bondier@switzernet.com> |
| 60 | smtp\_to | Elen Virabyan <elen.virabyan@intarnet.com> |
| 61 | smtp\_to | Sujatha Nampally <sujatha.nampally@switzernet.com> |
| 62 | smtp\_to |  |
| 63 | smtp\_to |  |
| 64 | smtp\_to |  |
| 65 | smtp\_to |  |
| 66 | smtp\_cc | xxxxxxxxxxxxxxxxxx | The range of all cc-fields (the email addresses to be used in case of fraud alerts) |
| 67 | smtp\_cc | xxxxxxxxxxxxxxxxxx |
| 68 | smtp\_cc | xxxxxxxxxxxxxxxxxx |
| 69 | smtp\_cc | xxxxxxxxxxxxxxxxxx |
| 70 | smtp\_cc | xxxxxxxxxxxxxxxxxx |
| 71 | smtp\_cc | xxxxxxxxxxxxxxxxxx |
| 72 | smtp\_cc |  |
| 73 | smtp\_cc |  |
| 74 | smtp\_cc |  |
| 75 | smtp\_cc |  |
| 76 | smtp\_bcc | xxxxxxxxxxxxxxxxxx | The range of all bcc-fields (the additional email addresses used in case of fraud alerts) |
| 77 | smtp\_bcc | xxxxxxxxxxxxxxxxxx |
| 78 | smtp\_bcc | xxxxxxxxxxxxxxxxxx |
| 79 | smtp\_bcc | xxxxxxxxxxxxxxxxxx |
| 80 | smtp\_bcc | xxxxxxxxxxxxxxxxxx |
| 81 | smtp\_bcc | xxxxxxxxxxxxxxxxxx |
| 82 | smtp\_bcc | xxxxxxxxxxxxxxxxxx |
| 83 | smtp\_bcc | xxxxxxxxxxxxxxxxxx |
| 84 | smtp\_bcc |  |
| 85 | smtp\_bcc |  |
| 86 | smtp\_bcc |  |
| 87 | smtp\_bcc |  |
| 88 | smtp\_receipt | Emin Gabrielyan <emin.gabrielyan@switzernet.com> | The email address where the fraud alert return receipts must be sent |
| 89 | smtp\_subject | [1'dap'1 cost monitor] | The email routing tag to be added to the subject line of all outgoing emails |
| 90 |  |  |  |
| 91 | smtp server | smtp.mail.yahoo.com | A part of multiple-area range [smtp\_accounts] |
| 92 | smtp port | 25 |
| 93 | smtp authenticate | 1 |
| 94 | smtp ssl | TRUE |
| 95 | smtp user | d9a.monitor@yahoo.com |
| 96 | smtp password | xxxxxxxxxxxxxxxxxx |
| 97 | smtp from | d9a Monitor Yahoo.com <d9a.monitor@yahoo.com> |
| 98 |  |  |  |
| 99 | smtp server | smtp.googlemail.com | A part of multiple-area range [smtp\_accounts] |
| 100 | smtp port | 25 |
| 101 | smtp authenticate | 1 |
| 102 | smtp ssl | TRUE |
| 103 | smtp user | d9a.monitor@gmail.com |
| 104 | smtp password | xxxxxxxxxxxxxxxxxx |
| 105 | smtp from | d9a Monitor Gmail.com <d9a.monitor@gmail.com> |
| 106 |  |  |  |
| 107 | smtp server | smtp.switzernet.com | A part of multiple-area range [smtp\_accounts] |
| 108 | smtp port | 587 |
| 109 | smtp authenticate | 1 |
| 110 | smtp ssl | FALSE |
| 111 | smtp user | d9a.monitor@smtp.switzernet.com |
| 112 | smtp password | xxxxxxxxxxxxxxxxxx |
| 113 | smtp from | d9a Monitor Switzernet.com <d9a.monitor@switzernet.com> |
| 114 |  |  |  |
| 115 | schedule\_next | 2013-10-25 19:59:50 | The next time the periodic procedure is scheduled by the application |
| 116 | schedule\_running | FALSE | True if a periodic procedure is scheduled by the application |
| 117 | schedule\_interval | 00:11:00 | The time interval (11 minutes) of the next call of the periodic procedure. |
| 118 | schedule\_reported | 2013-10-25 14:00:05 | The time of the last successfully sent email. |
| 119 | schedule\_after | 11:00:00 | The time (11 hours) of the next periodic report after the successfully sent last email. |
| 120 |  |  |  |
| 121 | log\_rows | 9000 | The maximum number of log calls |
| 122 | log\_index | 2 | The current location of the log index |

# Calculating the statistics

Statistics and drilling is carried out in columns from P to AE. Below we show the formulas and provide comments for all columns of the range.

Here are the first five columns of the index of the interval, the start time, the duration of the interval, and the two columns of time criteria of the interval.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | P | Q | R | S | T |
| 1 |  |  |  |  |  |
| 2 | **=0** | **=B35** | **=B42** | FALSE | FALSE |
| 3 | FALSE | FALSE | FALSE | FALSE | FALSE |
| **=IF(calc1, P2+1)** | **=IF(calc1, Q2+R2)** | **=IF(calc1, R2/$B$40)** | **=IF(calc1, ">=" & Q3)** | **=IF(calc1, IF(P3=MAX(P:P), ">0", "<"&(Q3+R3)))** |
| Interval index | The start time of the interval | The duration of the interval. The initial duration is equal to the largest interval (see row 2) and each next duration is equal to the previous interval divided by the time factor. | The criterion text for the lower bound of the interval | The text of the criterion for the upper bound of the interval. Note that the lower bound is used inclusively and the upper bound exclusively. For the last interval the upper bound is skipped (replaced by “>0” string that will always return true). This is to not exclude the very last record from the statistics. |

Here are the columns computing the numbers of all calls, of answered calls, and of spoken conversations for each interval.

|  |  |  |  |
| --- | --- | --- | --- |
|  | U | V | W |
| 1 | count | answered | spoken |
| 2 | FALSE | FALSE | FALSE |
| 3 | FALSE | FALSE | FALSE |
| **=IF(calc1, COUNTIFS( INDEX(cdr,,4), $S3, INDEX(cdr,,4), $T3))** | **=IF(calc1, COUNTIFS( INDEX(cdr,,1), "=1", INDEX(cdr,,4), $S3, INDEX(cdr,,4), $T3))** | **=IF(calc1, COUNTIFS(INDEX(cdr,,1), "=1", INDEX(cdr,,4), $S3,INDEX(cdr,,4), $T3, INDEX(cdr,,8), ">="&$B$43))** |
| Compute only if calculation is allowed. Counts all rows where the 4th column of CDR (i.e. the time values) meets the condition of the $S column (i.e. the lower bound of the period) and of the $T column (i.e. the upper bound, non-inclusively). The result is the number of records in the interval. | Like in the previous column, but with an additional condition on the values of the first column of CDR range. The result is the number of records in the interval with the record source table identification equal to 1 (i.e. counting only the answered calls). | Like in the previous column, but with yet another condition on the 8th column of CDR (i.e. on the duration of calls). Counting all calls of the interval, such that the connection status is answered but additionally, also if the duration of calls is more than the value of the $B$43 cell, which is defined in the parameters’ table as the minimal duration of the call to be considered as a successful phone conversation (and is equal to 10 seconds in our sample Excel file). |

Below are the columns in which we compute the sum of all PDD, the total revenue, cost, and duration corresponding to each interval.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | X | Y | Z | AA |
| 1 | PDD | Revenue | Cost | Duration |
| 2 | FALSE | FALSE | FALSE | FALSE |
| 3 | FALSE | FALSE | FALSE | FALSE |
| **=IF(calc1, SUMIFS( INDEX(cdr,,3), INDEX(cdr,,1), "=1", INDEX(cdr,,4), $S3, INDEX(cdr,,4), $T3))** | **=IF(calc1, SUMIFS( INDEX(cdr,,5), INDEX(cdr,,1), "=1", INDEX(cdr,,4), $S3, INDEX(cdr,,4), $T3))** | **=IF(calc1, SUMIFS( INDEX(cdr,,6), INDEX(cdr,,1), "=1", INDEX(cdr,,4), $S3, INDEX(cdr,,4), $T3))** | **=IF(calc1, SUMIFS( INDEX(cdr,,8), INDEX(cdr,,1), "=1", INDEX(cdr,,4), $S3, INDEX(cdr,,4), $T3))** |
| Proceed with the rest, only if the calculation of statistics is allowed. Sum all values of PDD in CDR range, if the status of the call is equal to answered, and if the time values (column 4) are above the lower bound (criterion of S:S column) and below the upper bound (criterion of T:T column). | Similarly to the previous formula sum all revenue values (column 5 of the CDR range) of the interval for all answered calls. | Similarly to the previous, but sum the values of the column 6, which is the cost column in the CDR range. | Sum the values of the 8th column in the CDR range, which is the duration of calls represented in seconds. |

The last four columns contain the deepest prefix, its cost, the search algorithm’s log, and the factor of the prefix’s hourly cost within the interval over the average hourly cost of the prefix (over the entire period of time).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | AB | AC | AD | AE |
| 1 | Deepest Prefix | Deepest Cost | Deepest Log | Factor |
| 2 | FALSE | FALSE | FALSE | FALSE |
| 3 | FALSE | FALSE | FALSE | FALSE |
| **{=IF(AND(calc1,calc2), IF($Z3>$B$44\*($R3\*24), deepest(INDEX(cdr,,6), INDEX(cdr,,1),"=1", INDEX(cdr,,2), "", $Z3, $B$45\*$Z3, INDEX(cdr,,4), $S3, $T3), ""))}** | | | **=IF(AND(calc1,calc2), IF(LEN($AB3) >= $B$46, ROUND(AC3/(R3\*24), 4) / ROUND( SUMIF (INDEX(cdr,,2), "="&AB3&"\*", INDEX(cdr,,6)) / ($B$37\*24), 4) ))** |
| This is an array formula fulfilling simultaneously all three cells in a row. The formula must be therefore entered with Ctrl-Shift-Enter stroke (while all three cells in a row are selected). If entered correctly, the formula bar will show the formula surrounded by curved parenthesis. All calculations in these three cells will be skipped except both calc1 and calc2 worksheet variables are set to true. The VBA macro activates the calculation of these columns when the calculations of statistics (carried out in the previous cells) are complete. The next condition compares the hourly cost of the current interval, i.e. the value of the cost (in the Z column) divided by the interval duration (in the R column) is compared against the value of $B$44 cell. The value of $B$44 is entitled as “drill cost/h” in the table of parameters and is equal to 77 CHF/h. If the current hourly cost of the interval is below the drill cost, we do nothing and we return an empty string “”. Now if the calc2 is allowed and the cost of the interval exceeds the critical limit (defined in $B$44), we call the deepest array function. This is a UDF defined by ourselves (see the section introducing the VBA macro functions and subroutines). The UDF takes as the 1st argument the column of the cost values (the 6th column in CDR); the 2nd and 3rd arguments are the column of the call status (answered or failed) and its criterion (“=1” selects only answered calls). The 4th argument is the column of the called numbers, the 5th argument is the base prefix (if equal to an empty string “” all called numbers must be examined without exception), the 6th argument (the value of Z column) is the base cost of the interval, and the 7th argument is the base cost of the interval multiplied by parameter $B$45, which is defined above (see the section of worksheet parameters) under title “drill down until” and is equal to 30% in our sample. The 7th argument determines the stop-condition of the sub-prefix exploration (digging/drilling-down) process. The last three arguments (the 8th, 9th and 10th one) are being given to the deepest function for bounding its activity within the current interval (defined by the criterions of the S and T columns). The deepest function finds and returns us the longest prefix that costs above the **$B$45** percentage (drill down until) of the interval’s total cost. It gives the prefix string to the first cell, to the second cell the exact cost of the prefix within the interval is given, and to the third cell a log text generated by the search algorithm (which is rather a commentary than a tangible value and is empty most of the time). | | | If the calc2 is allowed (i.e. the prefix digging is allowed), if the prefix length is not less the minimal prefix length defined as 1 in the parameters table (see the section of the worksheet parameters) then compute the ratio of the prefix’s hourly cost within the current interval over the same prefix’s hourly cost within the entire observation period. The hourly cost of the prefix within the entire observation period is computed by dividing the total cost of the prefix in CDR columns over the full duration of CDR set computed in **$B$37** cell of parameters range (entitled as “period”, see worksheet parameters). |

In this section we complete the interval statistics and prefix digging. In the next section we describe the calculation of the hourly rates of the cost and revenue values and the construction of the input data of the chart.

# Calculating the rates

Based on the previously obtained per-interval statistics described in the previous section, we can now generate the input data for the chart. When visualizing the statistics on the chart, we cannot use the absolute values per interval as the interval widths are changing exponentially and therefore the totals per interval (for instance the total of costs per interval) will also differ by a very important factor. To align the values, we display the rates instead of totals. For the cost, revenue, and the conversation duration, we display the hourly rates. For the failed and answered calls we display the call rate per minute. For PDD, we obviously display the average value per call.

The time axis labels and the etiquettes of the suspicious prefixes are constructed here as well. Columns from [AG] to [AN] described in the three tables below serve only to the construction of these tables.

In the following three columns we calculate the middle of each interval in the local time zone and the type of the time-axis label (the first label, the last label, a time label, and a day change label).

|  |  |  |  |
| --- | --- | --- | --- |
|  | AG | AH | AI |
| 1 |  |  |  |
| 2 | FALSE | FALSE | FALSE |
| 3 | FALSE | FALSE | FALSE |
| **=IF(calc1, $Q3+$R3/2 + time\_zone/24)** | **=IF(calc1, IF(ISNUMBER(AW3), INT(AG3), ""))** | **=IF(calc1, IF( ISNUMBER(AW3), IF( COUNT(AW$1:AW3) = 1, 1, IF( COUNT(AW$1:AW3) < COUNT(AW:AW), IF(AH3 = MAX(AH$1:AH2), 2, 3), 4))))** |
| As shown in the previous section, column Q contains the start time of the interval (in the UTC time-zone) and column R the interval’s duration. Here we compute the middle of the current interval in the current time zone. | AW column represents the time axis and its value is numerical (and is equal to 0) if the time-axis label must be shown. Otherwise, its value is equal to Not-Available Excel value. See for the formulas of the AW column in the last table of this section. Here, if the time axis label must be show, we display the date value (of the middle of the interval in the local time-zone) without the time. | If the time-axis label must be displayed (see the AW column) return a value from 1 to 4 depending on the following. If it is the first visible label, return 1. If it is a middle visible time axis label, then, if the visible label’s date is not changed (see the previous AH column), return 2, if the visible date value is new, return 3. Finally, if it is the last visible label, return 4. The previous column which contains the truncated date values (without time component) only for the visible label positions permits us to determine whether the new label represents a new date or not. If the label represents a new date we will show on the chart the full date string, otherwise, during the day, we display only the time value (without repeating the date string for each label). |

In the following three columns we are calculating the part of the time-axis label showing the width of the interval (i.e. the delta value).

|  |  |  |  |
| --- | --- | --- | --- |
|  | AJ | AK | AL |
| 1 |  |  |  |
| 2 | FALSE | FALSE | FALSE |
| 3 | FALSE | FALSE | FALSE |
| **=IF(calc1, "Δ" & IF( $R3>1, TEXT($R3, "0\d"), IF( $R3>1/24, TEXT($R3 \* 24, "0\h"), IF( $R3>1/(24\*60), TEXT($R3 \* 24 \* 60, "0\m"), TEXT($R3 \* 24 \* 60 \* 60, "0\s") ))))** | **=IF(calc1, LEN(AJ3))** | **=IF(calc1, REPT(" ",3\*( MAX(AK:AK) - AK3 + 1) ) & AJ3)** |
| The string is preceded by the delta sign. The delta value is displayed as a rounded day, rounded hour, rounded minute, or as a rounded second, depending whether the value is more than 1 day, more than 1 hour, more than 1 minute, or less than 1 minute. | This column contains the lengths of the strings obtained in the previous column. | Here we add the extra spaces (3 spaces per missing character) to align the labels to an equal visual width. |

In the following two columns we obtain the two final tables, one for displaying the suspicious prefixes, the other one for labeling the time axis values.

|  |  |  |
| --- | --- | --- |
|  | AM | AN |
| 1 | prefix | time |
| 2 | FALSE | FALSE |
| 3 | FALSE | FALSE |
| **=IF(calc1, IF( AND(calc2,AP3), "+" & $AB3 &" " & ROUND(AP3,0) & " €/h = " & TEXT(AC3/Z3, "0%"), ""))** | **=IF(calc1, IF( AI3, CHOOSE(AI3, "↑ " & TEXT( $B$35 + time\_zone/24, "yyyy mmm d dddd hh:mm") & AL3, TEXT(AG3, "hh:mm") & AL3, "→ " & TEXT(AG3, "mmm d dddd hh:mm") & AL3, "↓ " & TEXT( $B$36 + time\_zone/24, "yyyy mmm d dddd hh:mm") &AL3),""))** |
| If the prefix digging calculation is on and if there is a suspected cost (see the AP column), display the prefix (see the AB column in the previous section) preceded by the “+” sign, show the hourly cost of the prefix and finally show the relative weight of the prefix cost with respect to the total cost of the interval (see the AC and Z columns in the previous section). | Here we obtain the time axis label. If this is a visible position, depending on the value of the AI column (1 for the very first label, 2 for a same-day middle label, 3 for a new-day middle label, and 4 for the very last label), show the first call time **$B$35** (see the parameters of the worksheet) in the local time-zone, show the time of the middle of the current interval, show the date and time of the current interval’s middle, and finally, show the last call time **$B$36** in the local time-zone. Note that middle time-values will be marked on the horizontal axis by vertical arrows perpendicular to the horizontal axis indicating that the value being shown is the middle of the interval, the left (first) interval will have an arrow pointing to the left, indicating that the time value corresponds to the left edge of the time axis, and the right (last) interval of the horizontal axis will be accompanied with a right arrow indicating that the value corresponds to the right edge of the time axis. Only the first and last intervals will carry the year information. |

The following table shows the results of the formulas of the [AN] column.

|  |
| --- |
| Time |
| ↑ 2013 Sep 21 Saturday 15:54 Δ1d |
|  |
| → Sep 25 Wednesday 06:35 Δ1d |
|  |
| → Sep 27 Friday 22:26 Δ1d |
|  |
| → Sep 30 Monday 09:41 Δ1d |
|  |
| → Oct 2 Wednesday 16:38 Δ1d |
|  |
| → Oct 4 Friday 19:37 Δ1d |
|  |
| → Oct 6 Sunday 18:55 Δ23h |
|  |
| → Oct 8 Tuesday 14:49 Δ21h |
|  |
| → Oct 10 Thursday 07:31 Δ20h |
|  |
| → Oct 11 Friday 21:18 Δ18h |
|  |
| → Oct 13 Sunday 08:20 Δ17h |
|  |
| → Oct 14 Monday 16:51 Δ16h |
|  |
| → Oct 15 Tuesday 23:01 Δ15h |
|  |
| → Oct 17 Thursday 03:00 Δ13h |
|  |
| → Oct 18 Friday 04:57 Δ12h |
|  |
| → Oct 19 Saturday 05:02 Δ12h |
|  |
| → Oct 20 Sunday 03:23 Δ11h |
|  |
| → Oct 21 Monday 00:07 Δ10h |
|  |
| 19:21 Δ9h |
|  |
| → Oct 22 Tuesday 13:11 Δ9h |
|  |
| → Oct 23 Wednesday 05:45 Δ8h |
|  |
| 21:06 Δ7h |
|  |
| → Oct 24 Thursday 11:21 Δ7h |
|  |
| → Oct 25 Friday 00:34 Δ6h |
|  |
| 12:50 Δ6h |
|  |
| → Oct 26 Saturday 00:13 Δ5h |
|  |
| 10:46 Δ5h |
|  |
| 20:33 Δ5h |
|  |
| → Oct 27 Sunday 05:39 Δ4h |
|  |
| 14:04 Δ4h |
|  |
| 21:53 Δ4h |
|  |
| → Oct 28 Monday 05:09 Δ3h |
|  |
| 11:53 Δ3h |
|  |
| 18:07 Δ3h |
|  |
| 23:55 Δ3h |
|  |
| → Oct 29 Tuesday 05:17 Δ3h |
|  |
| 10:16 Δ2h |
|  |
| 14:54 Δ2h |
|  |
| 19:11 Δ2h |
|  |
| 23:10 Δ2h |
|  |
| → Oct 30 Wednesday 02:52 Δ2h |
|  |
| 06:18 Δ2h |
|  |
| 09:28 Δ2h |
|  |
| 12:25 Δ1h |
|  |
| 15:10 Δ1h |
|  |
| 17:42 Δ1h |
|  |
| 20:03 Δ1h |
|  |
| 22:14 Δ1h |
|  |
| → Oct 31 Thursday 00:16 Δ59m |
|  |
| 02:09 Δ54m |
|  |
| 03:53 Δ50m |
|  |
| 05:31 Δ47m |
|  |
| 07:01 Δ43m |
|  |
| 08:24 Δ40m |
|  |
| 09:42 Δ37m |
|  |
| 10:54 Δ35m |
|  |
| 12:01 Δ32m |
|  |
| 13:03 Δ30m |
|  |
| 14:00 Δ28m |
|  |
| 14:53 Δ26m |
|  |
| ↓ 2013 Oct 31 Thursday 15:55 Δ24m |

In the next three columns we prepare the label strings of the cost, suspicious prefix cost, and margin areas as well as their values. Under column AR we compute the positions of the suspicious prefixes on the chart.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | AO | AP | AQ | AR |
| 1 | FALSE | suspected cost/h | FALSE |  |
| **=IF(calc1, "cost/h [" & TEXT ( SUM(Z:Z), "0" ) & "]")** |  | **=IF(calc1, "margin/h [" & TEXT( SUM(Y:Y) - SUM(Z:Z), "0") & "]")** | **=IF( suspect\_new = "", "", IFERROR( "new " & LEFT( suspect\_new, SEARCH (" ", suspect\_new) -1 ), ""))** |
| The label of the cost per hour followed by the total cost over the entire observation column surrounded by square parenthesis. See the Z column in the previous section of the calculation of the statistics. | The cost of the suspected prefix, if any, will appear under this column. Otherwise, the values in this column will be equal to zero. | The label of the margin per hour is accompanied by the total absolute margin over the entire observation period. See the columns Y and Z for the revenue and the cost in the previous section of the statistics | If the VBA script set a flag of a fresh suspect, display the prefix of the fresh suspect as a title of this curve. The prefix or phone number of the current suspect will appear in the legend of the chart. |
| 2 | FALSE | FALSE | FALSE | FALSE |
| 3 | FALSE | FALSE | FALSE | FALSE |
| **=IF(calc1, ($Z3-IF(calc2, IF(AE3, IF( $AE3 > $B$47, $AC3,0),0),0))/ (24\*$R3))** | **=IF(calc1, IF(calc2, IF( AE3, IF( $AE3 > $B$47, $AC3, 0), 0), 0) / (24\*$R3) )** | **=IF(calc1, ($Y3-$Z3)/ (24\*$R3))** | **=IF(calc1, IF(AP3, MAX( OFFSET (AO3, -1,0,3,1) ) + MAX( OFFSET (AP3, -1,0,3,1) ) + MAX( OFFSET (AQ3, -1,0,3,1), 0 ), NA()))** |
| Remove from the cost (Z column) the cost of the suspected prefix (see the AC column) if a prefix is found (the value of the AE column is numerical and is not false) and if the factor AE is more than **$B$47** which is the minimal suspect factor (see the worksheet parameters). Divide the result of the subtraction by the interval duration (column R) in order to obtain the hourly rate of the cost (without suspicious prefixes). | The hourly rate of the suspicious prefix is computed here. If the value of AE column is numerical (i.e. a prefix is found), and if its value is more than the minimal factor defined under **$B$47** parameter, then take the cost of the suspicious prefix from the column AC. Otherwise suspicious cost is zero. Divide the cost by the duration of the interval (column R) to obtain the hourly rate of the suspicious cost. | The hourly margin rate is the revenue of the interval (column Y) minus the cost of the interval (column Z) divided by the duration of interval in hours. | Here is the location of the prefix on the chart. This curve is invisible, only its labels are visible. For the points with the suspicious cost value (column AP) equal to zero, the value of the curve is equal to Not-Available Excel value, meaning that the point will be skipped on the chart and no label will be displayed. Labels will be displayed only for the intervals where suspicious prefixes are detected. The label is taken from the above-presented column AM (in this section). The label is positioned above the chart areas. The risk of the overlapping with the chart areas is minimized by taking the sum of the maximums of the surrounding values for cost, suspicious cost, and margin. |

The next three columns represent the average PDD and the rates of spoken and failed calls per minute.

|  |  |  |  |
| --- | --- | --- | --- |
|  | AS | AT | AU |
| 1 | FALSE | FALSE | FALSE |
| **=IF(calc1, "PDD x" & $B$49 & " [" & TEXT ( SUMIF (INDEX(cdr,,1), "=1", INDEX(cdr,,3)) / COUNTIF (INDEX(cdr,,1), "=1") / 1000, "0.0" ) & "s]")** | **=IF(calc1, "spoken/m x" & $B$50 & " [" & SUM(W:W) & "]")** | **=IF(calc1, "failed/m x" & $B$50 & " [" & (SUM(U:U)-SUM(W:W)) & "]")** |
| The label PDD shows its display factor **$B$49** (its value is set to 50) introduced for aligning the PDD curve to the same order of grandeur with the values of the traffic minutes per hour. In the square parenthesis we display the average value of PDD over the entire observation period. It is the sum of all PDD values (for answered calls only) divided to the number of answered calls. The milliseconds are converted to seconds. | Spoken calls are accompanied by a display factor **$B$50** for the visual purposes (its value is equal to 100). The total number of spoken calls over the entire period of the chart is displayed within square parenthesis. | The same factor of **$B$50** is used also for the failed calls. Within the square parenthesis is the total number of failed calls which is the total number of records (see column U) minus the number of answered calls (see column W). |
| 2 | FALSE | FALSE | FALSE |
| 3 | FALSE | FALSE | FALSE |
| **=IF(calc1, IF(V3, X3/V3/1000 \* $B$49, NA()))** | **=IF(calc1, $W3/($R3\*24\*60) \* $B$50)** | **=IF(calc1, ($U3-$W3)/ ($R3\*24\*60) \* $B$50)** |
| If the number of answered calls (column V) is equal to 0, then return the special not-available value skipping the points on the chart. Otherwise the average PDD is the value of column X divided by the value of column V. | Here we calculate the number of spoken calls divided by the duration of the interval in minutes. The rate is multiplied by factor **$B$50** for visualization purposes. | The total number of records in the interval minus the spoken call divided by the duration of the interval gives us the rate of the failed calls. A high rate of the failed calls suggests a network or vendor problem. |

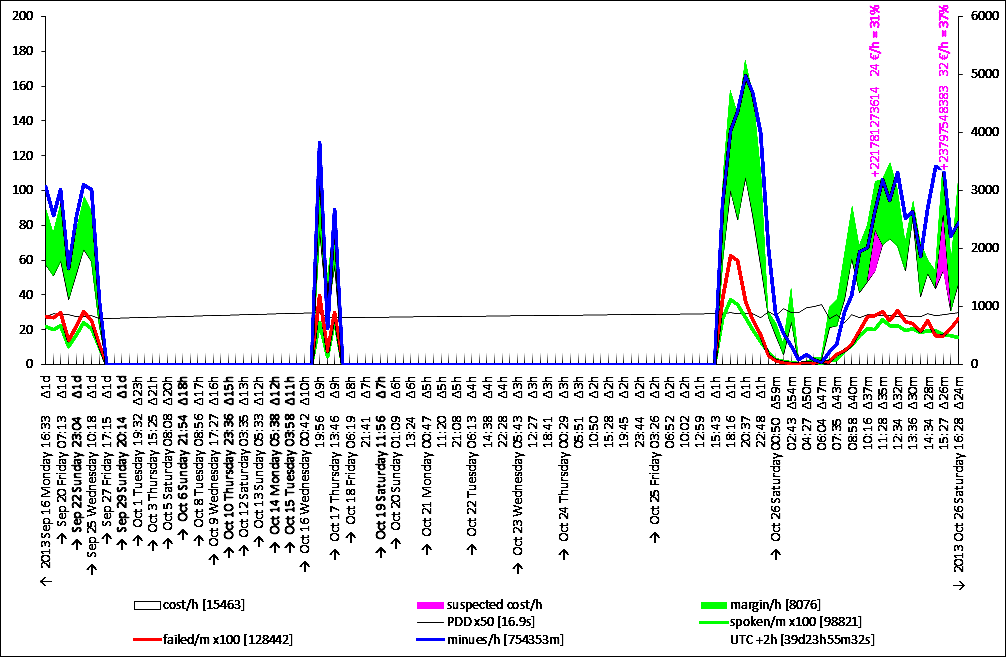
Column AV represents the traffic measured in minutes per hour and the last column represents the time axis. We do not use the horizontal axis of the Excel chart, instead we have an invisible time-curve at the level zero permitting us to control the density of the time labels and chose the format of the labels as a function of the previous set of visible labels (e.g. we can display the time in the full format when the day of visible labels changes and display the time in a short format otherwise; see column AN of this section).

|  |  |  |
| --- | --- | --- |
|  | AV | AW |
| 1 | FALSE | FALSE |
| **=IF(calc1, "minutes/h [" & TEXT( SUM(AA:AA)/60, "0") & "m]")** | **=IF(calc1, "UTC " & TEXT(time\_zone, "+0;-0;+0") & "h [" & INT($B$37) & "d" &TEXT($B$37-INT($B$37), "[h]\hm\ms\s") & "]")** |
| Showing the total number of minutes during the entire observation period next to the label of the traffic in minutes per hour | Display in the legend the time zone and the duration of the entire time axis in days, hours, minutes, and seconds. See **$B$37** in the section of the worksheet parameters. |
| 2 | FALSE | FALSE |
| 3 | FALSE | FALSE |
| **=IF(calc1, $AA3/($R3\*24)/ 60)** | **=IF(calc1, IF(MOD(ROW()-1,$B$54), 0,NA()))** |
| The hourly rate of the traffic in minutes | The density of the visible labels is computed as a function of **$B$54** parameter (which is in its turn computed as a function of the input parameter of the max number of labels to show) |

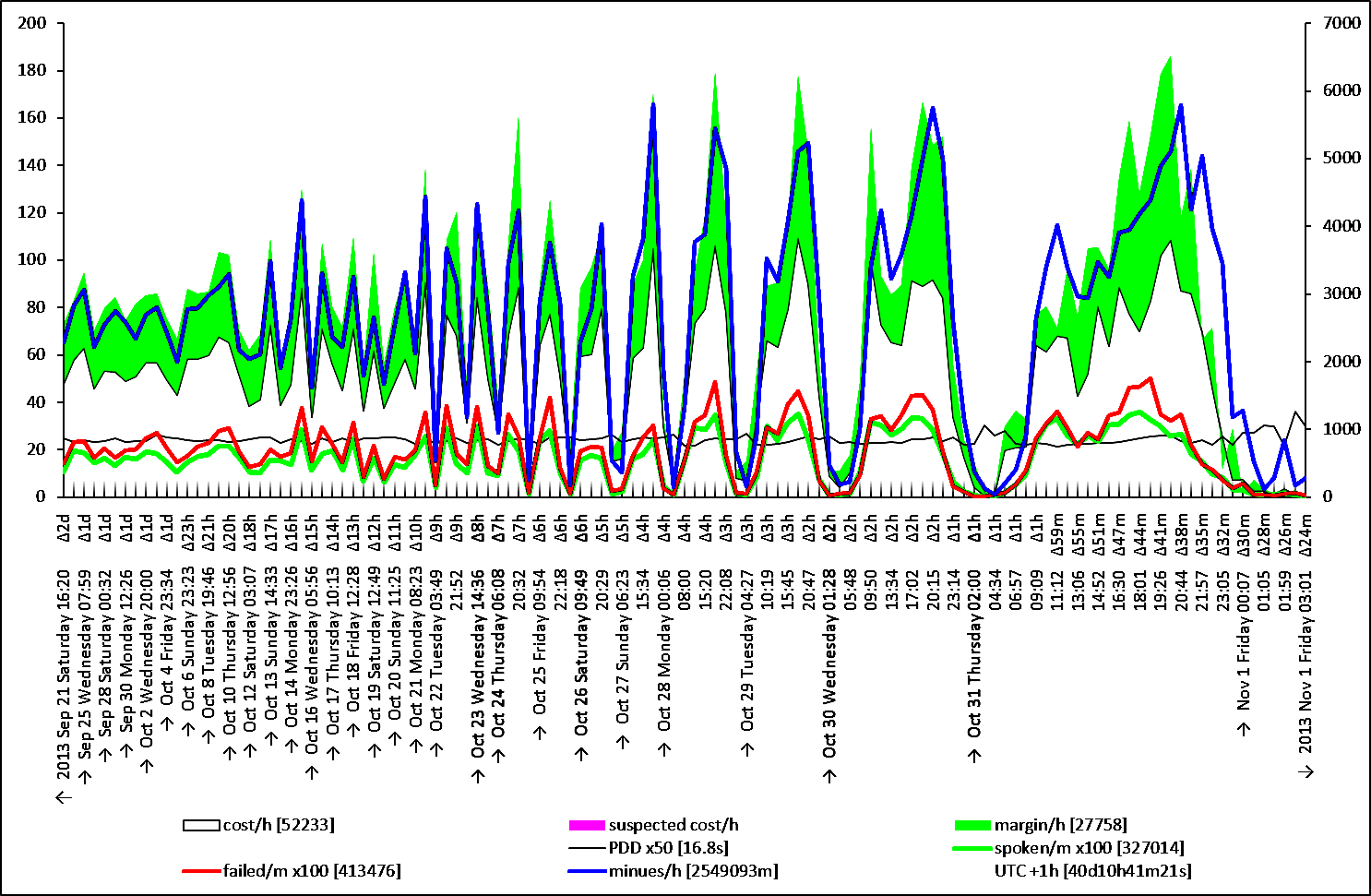
In the next section you can see several chart samples based on the above-presented chart range columns.

# Chart samples

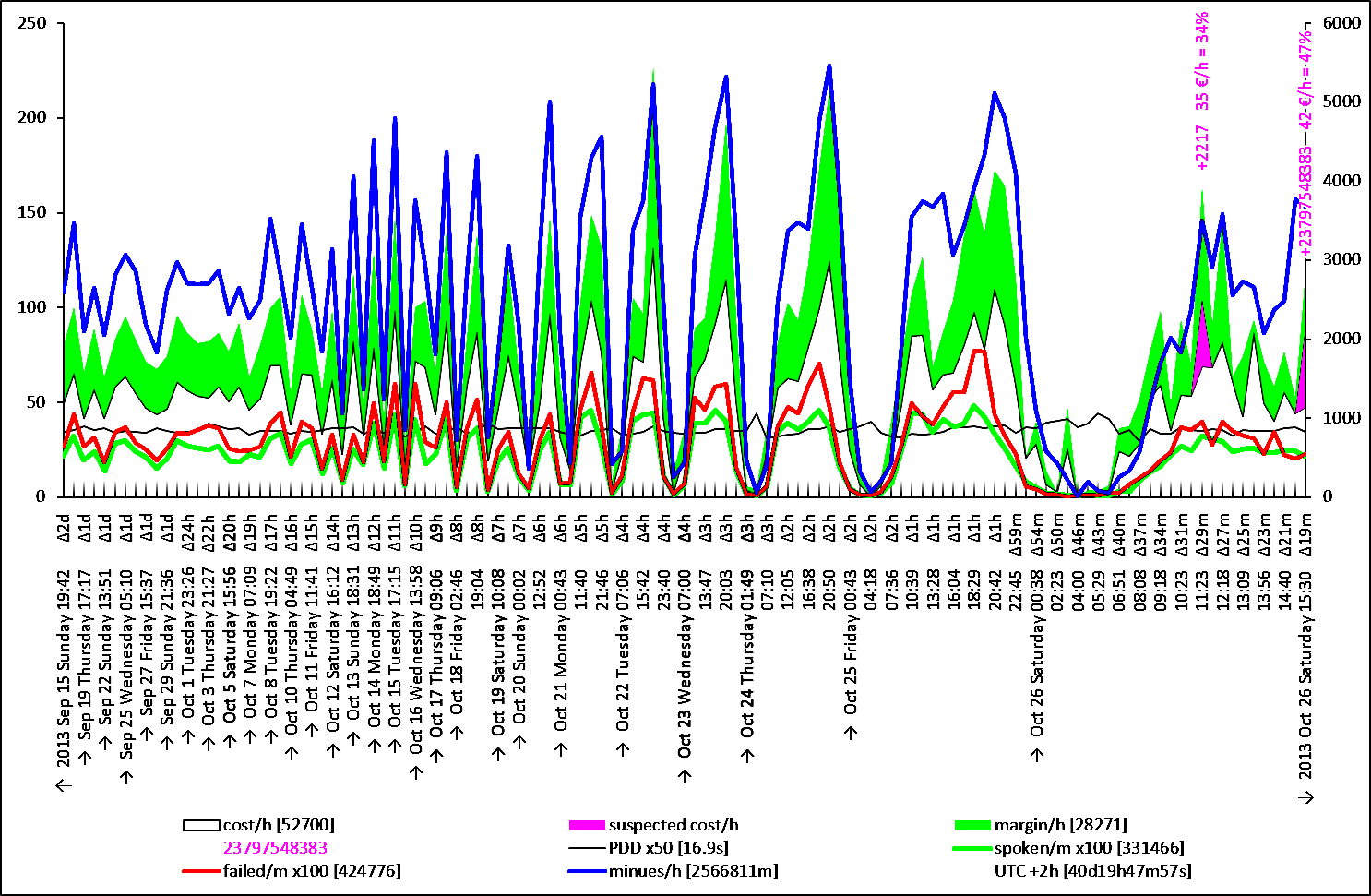
The following is an image of the chart with to areas containing suspicious prefixes (or phone numbers). We can see on the chart the three synchronization rhythms separated by the gaps. After a few cycles of periodic execution, the application downloads the missing calls and fulfills the gaps.



Below is a chart (with no suspicious interval detected). The properties of the exponentially growing time intervals are observed visually. The largest to smallest interval ratio is equal to 90 in this example. The most left interval is of the duration of about 2 days while the most recent interval is of the duration of 24 minutes. On the right side of the chart the days are identified by clearly separated waves with peak times and off-peak times. The largest space is taken by the current day, but when moving toward the left, the days are melting down to the overall average level of the traffic.



Here is another sample with a fully synced timescale containing two intervals with suspicious prefixes.



# VBA script

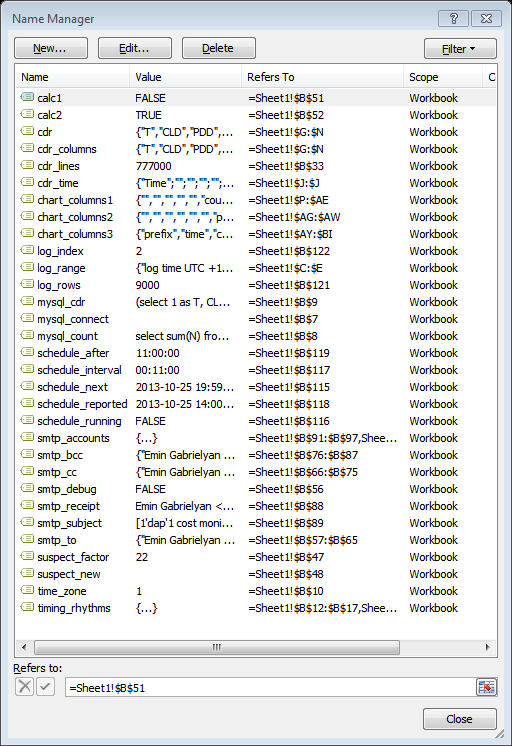
The entire script, all subroutines and functions of the VBA macro is presented and commented below.

|  |  |
| --- | --- |
| Code | Comments |
| Option Explicit | All variables must be declared |
| Sub Oval3\_Click()    Schedule    End Sub | The sole user interface to activate or deactivate the background periodic script |
| Sub shdata()  ThisWorkbook.Activate  ThisWorkbook.Sheets(1).Activate  range("log\_range").Cells(2, 3).Activate    End Sub | This subroutine activates the data sheet and in it, activates the most recent log record.  For an unknown reason, CopyFromRecordset method of a range (see below in the code) generates an error when the active worksheet is not the one where the range belongs to. |
| Sub shchart()    ThisWorkbook.Activate  ThisWorkbook.Sheets(2).Activate    End Sub | Activates the chart sheet. |
| Sub datalock()    ThisWorkbook.Activate  ThisWorkbook.Sheets(1).Unprotect  vba\_out "protect worksheet"  ThisWorkbook.Sheets(1).Protect password:=""  ThisWorkbook.Sheets(2).Protect password:=""    End Sub | The worksheet is protected after the execution of the script. In the Excel file, only the input cells of the user are unlocked. |
| Sub datafree()    ThisWorkbook.Activate  ThisWorkbook.Sheets(1).Unprotect  vba\_out "worksheet unprotected"    End Sub | The first thing the script does upon a periodic execution is unprotecting the data worksheet for bringing there the CDR and changing global variables stored in data worksheet cells. |
|  |  |
| Sub Unusual()    Dim stats As range  Dim interval As range  Dim start As range  Dim delta As range  Dim cost As range  Dim prefix As range  Dim subcost As range  Dim drilllog As range  Dim factor As range  Dim exam As range | This procedure examines the columns of interval statistics that are computed with Excel formulas based on the values of CDR columns retrieved from the billing.  Range [stats] is the entire set of statistics columns. Range [interval] will refer to the column of incremental numeric indexes of intervals. Column [start] will refer to the start time values of intervals. The next column, [delta], is the duration of the interval. The next column contains the total cost within each interval. The prefix column contains the heaviest prefix within the interval computed with a UDF (see below). The next column contains the cost of the heaviest prefix and the next one a search log string generated by UDF (if any). Column factor contains the ratios of the hourly rate of the heaviest prefix within the current interval over the average rate of the same prefix during the entire observation period. Range exam will represent a subset of the cells of factor column to be examined. |
| Dim suspect\_factor As Long  Dim time\_zone As Integer  Dim lastinterval As Integer  Dim alert As String  Dim suspect\_new As range | Suspect factor is a user defined parameter read from the worksheet. In our example its value is equal to 22. The heaviest used prefix of the interval will be considered suspicious only if its hourly rate within the interval exceeds its overall average hourly rate (over the entire period of observation) by a factor greater than the suspect factor.  Time zone is the number of hours the local time is away from the UTC. Recall that all time values in the CDR columns are in UTC.  Last interval is the index of the last interval.  Alert is a string with a digest description of the alert including the prefix, its hourly cost, etc.  Suspect new is a cell in the worksheet where this procedure writes the alert if it occurred in the last, i.e. the most recent interval. If this procedure writes something in the suspect new cell, then the next subroutine will send a high priority email to all recipients with a warning.  Note: in the new version the types of suspect factors and time zones are changed to double. |
| Dim iexam As range  Dim iinterval As Integer  Dim istart As Date  Dim idelta As Date  Dim icost As Double  Dim iprefix As String  Dim isubcost As Double  Dim idrilllog As String | When looping through the rows of the statistic columns, the first variable will refer to the current cell in factor column and the other variables on will store the values of each corresponding column. |
| ThisWorkbook.Activate    Set stats = range("chart\_columns1")  Set interval = stats.Columns(1)  Set start = stats.Columns(2)  Set delta = stats.Columns(3)  Set cost = stats.Columns(11)  Set prefix = stats.Columns(13)  Set subcost = stats.Columns(14)  Set drilllog = stats.Columns(15)  Set factor = stats.Columns(16) | Now assigning the columns. |
| suspect\_factor = range("suspect\_factor").Value  time\_zone = range("time\_zone").Value  lastinterval = WorksheetFunction.Max(interval)  Set suspect\_new = range("suspect\_new") | Reading the values of the suspect factor and of the time zone. Computing the last interval index. Referring to the cell where new suspect text must be written, if detected. |
| On Error Resume Next  Set exam = factor.SpecialCells( \_  Type:=xlCellTypeFormulas, \_  Value:=xlNumbers)  If Err.Number <> 0 Then Set exam = Nothing  On Error GoTo 0    If exam Is Nothing Then  vba\_out "no unusual prefix"  Exit Sub  End If | The values of factor column are computed by Excel formulas whenever a heavily used prefix is found. Otherwise the value of factor column is not numerical. From the range of all cells of factor column we obtain the subset of cells containing the numerical values. If such a subset is empty an error will be generated, which we catch here in order to assign the value of Nothing to the subset [exam]. If there is nothing to examine we quit the subroutine. |
| suspect\_new.Value = ""  suspect\_new.Interior.ColorIndex = xlNone | Before examining the intervals, emptying the old value of the cell of new suspects. |
| For Each iexam In exam.Cells  If iexam.Value > suspect\_factor Then  iinterval = Intersect(iexam.EntireRow, interval).Value  istart = Intersect(iexam.EntireRow, start).Value \_  + time\_zone / 24  idelta = Intersect(iexam.EntireRow, delta).Value  icost = Intersect(iexam.EntireRow, cost).Value  iprefix = Intersect(iexam.EntireRow, prefix).Value  isubcost = Intersect(iexam.EntireRow, subcost).Value  idrilllog = Intersect(iexam.EntireRow, drilllog).Value  If idrilllog <> "" Then idrilllog = " (" & idrilllog & ")"  alert = "" \_  & "+" & iprefix & " costs " \_  & Format(isubcost / (idelta \* 24), "0") & "CHF/h, " \_  & Format(isubcost / icost, "0%") & " of current traffic (" \_  & Format(isubcost, "0.0") & "/" \_  & Format(icost, "0") & "), and " \_  & Format(iexam.Value, "0") & "x of own average within " \_  & WorksheetFunction.Text(idelta, "[h]\hm\ms\s") & " from " \_  & Format(istart, "yyyy-mm-dd hh:mm:ss") \_  & idrilllog \_  & ""  vba\_alert alert  If iinterval = lastinterval Then  vba\_alert "alert in the last interval"  suspect\_new.Value = alert  suspect\_new.Interior.Color = RGB(255, 0, 255)  End If  End If  Next iexam    End Sub | For each cell of factor column with a numerical value (i.e. for each cell of exam subset), if the measured value is greater suspect factor, we assign the values of interval, start time, interval duration, interval cost, heavies prefix, its cost, and drilling log to the corresponding variables. An alert text is generated. The alert text for each suspicious interval contains the prefix, the hourly cost of the prefix within the current interval, the factor by which the prefix usage in the current interval is higher than its own average, the time and the duration of the interval, and the drilling log if any.  If the alert occurred in the last, i.e. the most recent interval, the value of the cell of new suspects is updated so the macro can take further actions and alert the recipients by email about a possible fraudulent phone traffic (for example to the value added international phone numbers). |
|  |  |
| Function savelog() As String    Dim log\_range As range  Dim firstlog As Date  Dim lastlog As Date  Dim logname As String  Dim logWB As Workbook    ThisWorkbook.Activate    vba\_out "creating a log file"    Set log\_range = range("log\_range")    firstlog = WorksheetFunction.Min(log\_range.Columns(1))  lastlog = WorksheetFunction.Max(log\_range.Columns(1))  logname = Format(firstlog, "yymmdd'hhmmss") & "\_" \_  & Format(lastlog, "yymmdd'hhmmss")    range(log\_range.Rows(1), \_  Intersect(log\_range, \_  log\_range.Columns(1) \_  .Find(what:="\*", SearchDirection:=xlPrevious) \_  .EntireRow)).Copy    Set logWB = Workbooks.Add    With logWB.Sheets(1).Cells(1)  .PasteSpecial Paste:=xlPasteColumnWidths  .PasteSpecial xlPasteValues, , False, False  .PasteSpecial xlPasteFormats, , False, False  End With    logname = ThisWorkbook.Path & "\" \_  & ThisWorkbook.Name & "\_" \_  & logname & "\_log.xlsx"    Application.DisplayAlerts = False  logWB.SaveAs logname  Application.DisplayAlerts = True    logWB.Close    vba\_out "log file created"    savelog = logname    End Function | This function generates a log file and returns its full path as a string. This function is used by the email sending procedure (see below).  Log range is a three column range in the first worksheet of the workbook. First log is to be the date and time value of the earliest log record. The last log is the time of the most recent log record. The log name, the name of the log file, is a string which is computed as a function of the first and last log times. The workbook type variable refers to the log file to be created.  Computing the first and last log records times using worksheet functions of the Excel application. Then computing the log file name.  Computing the sub range of the log range containing data. For that purpose we take the area located between the first and the last used rows of the log range. We copy the range into memory.  We create a new workbook.  We paste the values and the formats into the worksheet of the new workbook.  We add to the log name the full path of our Excel file and we prefix the log name with the name of the operational Excel file.  The log file is saved and its full name is returned as a result of this function. |
| Sub Job()    Dim try As Integer  Dim tries As Integer  Dim schedule\_reported As range  Dim schedule\_after As Date  Dim suspect\_new As String    ThisWorkbook.Activate    Set schedule\_reported = range("schedule\_reported")  schedule\_after = range("schedule\_after").Value    vba\_out "job started"    Traffic  suspect\_new = range("suspect\_new").Value    If Len(suspect\_new) > 0 Then  vba\_out "report that " & suspect\_new  tries = 5  ElseIf Now > schedule\_reported.Value + schedule\_after Then  vba\_out "scheduled report"  tries = 2  Else  vba\_out "skip report"  tries = 0  End If    For try = 1 To tries  If Chart\_Email Then  With schedule\_reported  .Value = Now  .NumberFormat = "yyyy-mm-dd hh:mm:ss"  End With  vba\_clr  Exit For  End If  vba\_out "waiting.."  Application.Wait (Now + TimeValue("0:00:15"))  Next try    Save\_Excel  vba\_out "job ended"    End Sub | This is the main subroutine called periodically (in our example every 11 minutes).  The tries integer variable determines how many times the email transmission attempt must be carried out (until succeeded). Its value is zero when no email must be sent. The schedule reported is a reference to the cell where the last successful email transmission date is stored. The scheduled reports are sent 11 hours (in our example) after the last successful transmission. Exceptions are cases when the cell of new suspects contains an alert.  Set a reference to the cell of the last email report time. Read the value of the time after which the next scheduled report must be sent.  Call subroutine Traffic which is retrieving the data from the billing / radius server and carrying out the statistics. Read the value of the new suspect cell after the execution of the Traffic subroutine.  If the new suspect is in the last interval set a high number of email transmission attempts. Otherwise, if it is the time of the scheduled report set a low number of the email transmission attempts (if the email is not sent at this cycle it will be sent at the next cycle). If there is no suspicious activity and is too early to send a report, set the email transmission attempts number to zero, i.e. no email must be sent.  Then we have the loop statement with the try integer from 1 to the values of tries. Chart email function returns true if the email is sent successfully. In that case we quit the loop. Before quitting we set the value of the successful report to the current time. If the email is not sent successfully, we wait for 15 seconds.  Note: in the new version of the Excel file (refer to the file provided for the downloading) the waiting time is increasing exponentially (see the TCP Friendly protocol). The waiting time doubles upon each unsuccessful attempt.  We save the Excel file and quit the job. |
| Sub Save\_Excel()    vba\_out "saving..."  On Error Resume Next  ThisWorkbook.Save  If Err.Number <> 0 Then  vba\_err "saving error " \_  & Err.Number & " : " & Err.Description  End If  Err.Clear  On Error GoTo 0    End Sub | This subroutine saves the running Excel file, the file which contains the code. At each periodic execution of the macro (and retrieval of new CDR) we call this subroutine to save the Excel file. |
| Sub Schedule()    Dim schedule\_running As range  Dim schedule\_next As range    ThisWorkbook.Activate    datafree    Set schedule\_running = range("schedule\_running")  Set schedule\_next = range("schedule\_next")    schedule\_running.Value = Not (schedule\_running.Value)    If (schedule\_running.Value) Then  vba\_out "schedule started"  Periodic  Else  On Error GoTo Error\_Handler\_1  Application.OnTime schedule\_next.Value, "Periodic", , False  On Error GoTo 0  vba\_out "schedule stopped"  End If  Cleanup:  datalock    Exit Sub  Error\_Handler\_1:  vba\_err "nothing to stop"  MsgBox "nothing to stop"  On Error GoTo 0  Err.Clear  Resume Cleanup    End Sub | This subroutine is called when the user clicks on the oval shape located in the data worksheet (it is the unique user interface designated for starting or stopping the scheduler).  The range variable [schedule running] refers to the cell with a Boolean value indicating whether the background periodic job execution is running or not. The range [schedule next] refers to the cell with the next launch time of the background periodic task.  First we unprotect the data worksheet.  We set references to the cells with the Boolean (indicating whether the schedule is running or not) and the next execution time of the task (making sense if the Boolean value is set to true).  Change the running status to the opposite.  If now, the running status is true, launch the periodic procedure.  If the running status is now false, it means the scheduler must stop and the scheduled execution of the periodic task must be cancelled. For cancelling the scheduled subroutine, we need its name and its execution time. Here we read the next execution time from the data worksheet. If cancellation is failed, it means there was nothing scheduled to stop. Such a situation can occur if you close the Excel file and reopen it while the scheduler was running.  We protect the worksheets before quitting, which is needed when stopping the periodic task, as in the cause of activation, the periodic task itself protects the worksheets before quitting. |
| Sub Periodic()    Dim schedule\_next As range  Dim schedule\_interval As Date    ThisWorkbook.Activate    datafree    shdata    Set schedule\_next = range("schedule\_next")  schedule\_interval = range("schedule\_interval").Value    Job    schedule\_next.Value = Now() + schedule\_interval  schedule\_next.NumberFormat = "yyyy-mm-dd hh:mm:ss"  vba\_out "next call on " \_  & Format(schedule\_next.Value, "yyyy-mm-dd hh:mm:ss")  Application.OnTime schedule\_next.Value, "Periodic"    shchart    datalock    End Sub | This is the periodic subroutine that calls the main job procedure (see above) and schedules its own execution after the period specified in the [schedule interval] cell.  Range variable [schedule next] refers to the cell where we store the time this subroutine will be called by the application the next time.  First we unprotect the data worksheet.  Then we activate the data worksheet. By default the chart sheet is active, but during the execution of the job we activate the data worksheet so the user can follow the activity log.  Setting the reference to the [schedule next] cell and reading the value of the [schedule interval].  Calling the main job subroutine.  After the job subroutine execution is complete setting the new value of [schedule next] cell. Telling the excel application to run this subroutine at the time stored in [schedule next] cell.  Switching now to the chart sheet.  Protecting the worksheets. |
|  |  |
| Function Chart\_Email()    Dim chart\_path As String  Dim chart\_name As String  Dim filename As String  Dim smtp\_account As range | This function generates the PNG image of the chart, randomly selects one of the SMTP accounts available for transmission and calls the SMTP transmission function in order to email the chart image. This function return true if the transmission of email is successful. |
| ThisWorkbook.Activate    chart\_path = Application.ActiveWorkbook.Path & "\" \_  & Application.ThisWorkbook.Name \_  & "\_Charts"    If Len(Dir(chart\_path, vbDirectory)) = 0 Then  MkDir chart\_path  End If | Computing the folder name where the charts must be stored.  If the folder does not exists yet, create it. |
| chart\_name = \_  Format( \_  WorksheetFunction.Min(range("cdr\_time")) \_  + range("time\_zone").Value / 24, \_  "yymmdd'hhmmss") & "\_" & \_  Format( \_  WorksheetFunction.Max(range("cdr\_time")) \_  + range("time\_zone").Value / 24, \_  "yymmdd'hhmmss") & "\_" & \_  Format( \_  Now, \_  "yymmdd'hhmmss") & ".png"    filename = chart\_path & "\" & chart\_name | Computing the current chart’s PNG file name. The filename is formed by the first CDR calls date and time, the last CDR calls date and time, and the current time. The CDR times are converted into the local time zone.  Full path filename of the chart is obtained. |
| vba\_out "exporting to " & chart\_name  If False Then  Worksheets(1).ChartObjects(1).Activate  ActiveChart.Export filename  Else  Charts(1).Export filename  End If | Exporting the chart image into the PNG file. In our case we export the chart object represented by the chart sheet instead of exporting the chart located in the data worksheet. |
| Select Case Int(Rnd \* 3) + 1  Case 1  Set smtp\_account = range("smtp\_account1")  Case 2  Set smtp\_account = range("smtp\_account2")  Case 3  Set smtp\_account = range("smtp\_account3")  End Select | Set the SMTP account range to one of the three user defined SMTP accounts for transmission of email. We use three accounts in our example, a google.com account, a switzernet.com account, and a yahoo.com account. In case of a failure of transmission there is a greater chance of a successful retransmission with more than one account.  Note: in the new version of this script (refer to the excel file available for downloading), the number of accounts is not fixed. Instead of using multiple named ranges we use a single range with multiple areas. User can define this single range with any given number of areas. Each area must contain seven rows with credentials of the SMTP account. |
| vba\_out "emailing..."  If SMTP( \_  filename, \_  smtp\_account) Then  vba\_out "email sent"  Chart\_Email = True  Else  vba\_err "failed to email"  Chart\_Email = False  End If    End Function | Parsing the SMTP account selection and the filename to the email transmission function described below. If the email transmission is unsuccessful we quit the function with the false return value. This permits the main job subroutine to attempt another retransmission of the email. |
|  |  |
| Function SMTP(ByVal filename As String, ByVal smtp\_account) As Boolean    Dim iMsg As Object    Dim server As String  Dim port As Long  Dim authenticate As Integer  Dim ssl As Boolean  Dim user As String  Dim password As String  Dim from As String | This function receives the filename with the SMTP account information area as arguments and returns true if it succeeds to email the file.  We need to add Microsoft CDO for Windows 2000 library reference in VBA project (with Tools / References).  The values of server, port, authentication, SSL, user name, password, and the from-field, declared here, are retrieved from the account range area parsed to this function as the second argument. |
| Dim smtp\_to As range  Dim tofield As range  Dim recipients As String  Dim copies As String  Dim bcc As String | The email addresses are stored in ranges with a single email address per cell. For each field (To, CC, and BCC), we need all addresses of the field separated with a semicolon in a single string. |
| Dim logfile As String | This is the name of the log file to be attached to the email each time we send the chart. It permits us to clear the log records of the main file upon each successful transmission of email. |
| ThisWorkbook.Activate    With smtp\_account  If \_  .Areas.Count <> 1 Or \_  .Columns.Count <> 1 Or \_  .Rows.Count <> 7 Then    vba\_err "smtp account error"  SMTP = False  Exit Function    End If  End With | Verifying the dimensions of the area parsed to this function that must contain the credentials of the SMTP account to be used for the transmission. |
| server = smtp\_account.Rows(1).Value  port = smtp\_account.Rows(2).Value  authenticate = smtp\_account.Rows(3).Value  ssl = smtp\_account.Rows(4).Value  user = smtp\_account.Rows(5).Value  password = smtp\_account.Rows(6).Value  from = smtp\_account.Rows(7).Value | If the previous validation is passed, we collect the seven values necessary for transmission of the email (SMTP server, user name, password, etc.). |
| Set iMsg = CreateObject("CDO.Message")    iMsg.Configuration.Load -1    With iMsg.Configuration.Fields  .Item(cdoSendUsingMethod) = 2  .Item(cdoSMTPServer) = server  .Item(cdoSMTPServerPort) = port  .Item(cdoSMTPAuthenticate) = authenticate  .Item(cdoSMTPUseSSL) = ssl  .Item(cdoSendUserName) = user  .Item(cdoSendPassword) = password  .Update  End With | Creating the CDO message object. SMTP account configuration parameters as well as the email parameters (the recipients, body, and the attachments) are all defined in this object.  We initialize the configuration object (with its method load) of the CDO message object.  Then we define the fields of configuration object of the CDO message object. The first field (send using method) is set to transmission via the SMTP protocol (if its value equal to 1 the email is dropped to the pickup directory of the mail server, a proprietary Microsoft’s protocol). |
| Set smtp\_to = range("smtp\_to")  recipients = ""  For Each tofield In smtp\_to.Cells  If tofield.Value <> "" Then  If recipients <> "" Then  recipients = recipients & ";"  End If  recipients = recipients & tofield  End If  Next tofield    If recipients = "" Then  vba\_err "no recipients"  SMTP = False  Exit Function  End If | Building the string of the semicolon separated recipients to be associated with TO field. |
| copies = ""  For Each tofield In range("smtp\_cc").Cells  If tofield.Value <> "" Then  If copies <> "" Then copies = copies & ";"  copies = copies & tofield  End If  Next tofield | Building the string of the semicolon separated recipients for the CC field. The recipients of the SMTP CC range are used by this function only for the urgent transmissions (i.e. when a fraudulent traffic is suspected). |
| bcc = ""  For Each tofield In range("smtp\_bcc").Cells  If tofield.Value <> "" Then  If bcc <> "" Then bcc = bcc & ";"  bcc = bcc & tofield  End If  Next tofield | Building the BCC field from the range of cells containing the individual BCC addresses. Similarly to the CC addresses the BCC addresses are used uniquely in case of the high priority transmissions. In BCC cells user must enter the private email addresses of the recipients, who must be alerted, but wish to hide their private addresses from the other recipients. |
| If range("smtp\_debug").Value Then  vba\_warn "single recipient when debugging"  recipients = smtp\_to.Rows(1).Value  If recipients = "" Then  vba\_err "recipient field is empty"  SMTP = False  Exit Function  End If  copies = ""  bcc = ""  End If | When the debugging mode is active, only one single recipient is used. All other recipients provided in the data worksheet are ignored in the debugging mode. The recipient used in the debugging mode is the first line of the range of the TO fields. |
| With iMsg  .To = recipients  .from = from  .TextBody = "" | We now set the properties of the CDO message object. Here are the properties which are common for the normal and urgent messages. The from-field changes from the account to account. |
| If range("suspect\_new").Value = "" Then  .CC = ""  .bcc = ""  .Subject = range("smtp\_subject").Value \_  & " " & Dir(filename)  .htmlbody = ""  Else | This is in case there is no suspected activity in the most recent time interval. The subject is set to the filename of the attached image (the filename contains the start and the stop time of the period) without the full path. The subject is preceded by a constant tag used for the routing of the emails (in our case the IMAP project name). |
| .Subject = range("smtp\_subject").Value \_  & " " & range("suspect\_new").Value    vba\_out "highest priority email"  With .Fields  .Item(cdoImportance) = cdoHigh  .Item(cdoPriority) = cdoPriorityUrgent  .Item("urn:schemas:mailheader:X-MSMail-Priority") = "High"  .Item("urn:schemas:mailheader:X-Priority") = "1 (Highest)"  .Update  End With    vba\_out "return receipt"  With .Fields  .Item("urn:schemas:mailheader:disposition-notification-to") = \_  range("smtp\_receipt").Value  .Item("urn:schemas:mailheader:return-receipt-to") = \_  range("smtp\_receipt").Value  .Update  End With    vba\_out "more recipients"  .CC = copies  .bcc = bcc    .htmlbody = "<b>Attention required!</b>"  End If | If currently a suspicious activity is going on, the subject contains the alert message (which includes the prefix, the hourly rate, etc.). We also tag the email with the highest importance and the urgent priority. Microsoft Exchange, Outlook, and all other alternative standards are employed for this purpose.  When it is about a suspicious traffic alert, a return receipt is also required from the recipients. The email address where the receipts must be sent is provided in a cell of the data worksheet.  All, the CC, the BCC addresses are used when it is an urgent transmission (the CC and BCC addresses are not used for periodic reports). |
| .addattachment filename  End With | Attaching the chart’s PNG file. |
| logfile = savelog    With iMsg  .addattachment logfile  On Error GoTo Error\_Handler\_1  .Send  On Error GoTo 0  End With    vba\_out "delete the log"    If Dir(logfile) <> "" Then  SetAttr logfile, vbNormal  Kill logfile  End If    SMTP = True | Generating the log file (see the function above). Attaching the generated log file and sending the email. If error occurs upon the email transmission, the function ends with the false result.  Deleting the log file.  Returning a true value, indicating that the email is sent successfully. |
| Cleanup:    Exit Function    Error\_Handler\_1:  vba\_err server & " error " \_  & Err.Number & " : " \_  & Replace(Err.Description, vbCrLf, "\_")  On Error GoTo 0  Err.Clear  SMTP = False  Resume Cleanup    End Function | In case of email transmission failure, logging the server name and the error message, setting the return value to false and quitting the function. |
|  |  |
| Sub Traffic()    Dim chart\_columns1 As range  Dim chart\_columns2 As range  Dim chart\_columns3 As range  Dim chart\_width2 As Long  Dim chart\_width3 As Long    ThisWorkbook.Activate    range("calc1").Value = False  range("calc2").Value = False    CDR\_Replicate  CDR\_Fit    Set chart\_columns1 = range("chart\_columns1")  Set chart\_columns2 = range("chart\_columns2")  Set chart\_columns3 = range("chart\_columns3")  chart\_width2 = chart\_columns2.Columns.Count  chart\_width3 = chart\_columns3.Columns.Count    vba\_out "format chart columns"  With chart\_columns1  .NumberFormat = "General"  .Columns(2).NumberFormat = "yy-mm-dd hh:mm:ss.000"  .Columns(3).NumberFormat = "[h]:mm:ss.000"  End With    vba\_out "calculating the stats..."  range("calc1").Value = True    vba\_out "drilling down the prefixes..."  range("calc2").Value = True    vba\_out "find suspects"  Unusual    vba\_out "updating the chart..."  With chart\_columns2  With range(.Columns(chart\_width2 + 1 - chart\_width3), .Columns(chart\_width2))  chart\_columns3.Value = .Value  .Interior.ColorIndex = xlNone  End With  End With    vba\_out "stop calculation"  range("calc1").Value = False    End Sub | This is the first subroutine called from the main periodic Job subroutine. This subroutine does all tasks with an exception of the transmission of email. It replicates the CDR from the remote data base, it controls the calculation of statistics, it calls the subroutine of the unusual traffic detection, and it provides a copy of the calculated stats to the chart.  Range chart columns 1 corresponds to summary statistics per interval, i.e. the total numbers of calls, of answered calls, of successful conversations, of the revenue, cost, duration per interval. Within this range we also have columns for drilling down to the most heavily used prefix and computing its cost per interval. Range chart columns 2, prepares the data for the chart. Here we build (with Excel formulas) the labels of the time axis as well as of the heavily used prefixes. The usages (cost, revenue, numbers of calls, and minutes) are converted into rates (hourly or minutely). Range chart columns 3, contains only the values computed in range chart columns 2, and represents an input data for the chart. The chart therefore does not undergo changes during the replication and computation, until its input data is updated by the VBA script.  All Excel worksheet formulas computing the statistics are designed to do nothing if the value of calc1 is false. The more advanced excel formulas detecting the heaviest prefixes do nothing except if both values of calc2 and calc1 are true. By setting the values of these cells to false at the beginning of this subroutine we ensure that there will be no worksheet calculations during the arrival of CDR. Worksheet calculations during the arrival of data can increase the processing time from a couple of seconds or minutes to an hour or more (we are referring here to several hundreds of thousands of records). This simple technique, embedding all Excel worksheet formulas into an if-statement depending on the value of calc1 or calc2 on one hand, and setting the values of calc1 and calc2 to false before replication and changing their values back to true after replication on the other hand, permits us to save hours of useless calculations of the application.  CDR Replicate subroutine is carrying out synchronizations in various places of the timescale (fresh data sync combined with the old data verification and re-sync). This subroutine is taking care of the consistency of the local data and that it is up to date. CDR Fit subroutine is deleting top rows if the maximal number of records in the local worksheet is exceeded. This is a user defined data and in our example is equal to 777000. Note that the intelligent management of the excel worksheet formula activity permits us to work with such large data sets in a reasonable time.  Excel VBA function copying from record sets has an incomprehensible side effect; it changes the formatting adjacent columns if these columns contain date and time values (this bug is discussed in various discussion lists on the web). After CDR replication we therefore reformat the values of range chart columns 1.  By setting the value of calc1 to true, we first calculate the overall statistics.  By setting the value of cell calc2 to true, we then find out the heavily used prefixes. Formulas activating on calc2 use the data already calculated by formulas relying on calc1. Our measurements show that the sum of calculation times is equal to the calculation time under the scenario when Excel does the entire job in one go. Therefore, by splitting down the calculation into two phases we do not scarify the performance. However, it is important to organize the formulas in a way such that the second phase does not trigger the recalculation of any element already computed in the first phase.  Subroutine Unusual creates a log of suspected utilization and sets the flag of the new suspect if suspicious activity is detected in the current interval.  Once all calculations and logging are complete, we update the chart by copying the new interval stats into the chart area.  Now we can stop the calculations, and all values of statistics computed by the Excel formulas will be lost (except the values already copied to the chart range). When calculations are stopped the Excel worksheet is lightweight and though it still has enormous number of data records, the user can work in the worksheet without any heavy worksheet update delays. |
| Function deepest( \_  ByVal costs As range, \_  ByVal answered As range, \_  ByVal answered\_criteria As String, \_  ByVal CLD As range, \_  ByVal this\_pref As String, \_  ByVal this\_cost As Double, \_  ByVal drill\_until As Double, \_  ByVal time As range, \_  ByVal time\_criteria1 As String, \_  ByVal time\_criteria2 As String \_  ) As Variant()    Dim deepest\_pref As String  Dim deepest\_cost As Double  Dim drill\_log As String  Dim deepest\_array(1 To 3) As Variant    drill\_log = ""    drill \_  costs, \_  answered, \_  answered\_criteria, \_  CLD, \_  this\_pref, \_  this\_cost, \_  deepest\_pref, \_  deepest\_cost, \_  drill\_until, \_  drill\_log, \_  time, \_  time\_criteria1, \_  time\_criteria2    deepest\_array(1) = deepest\_pref  deepest\_array(2) = deepest\_cost  deepest\_array(3) = drill\_log  deepest = deepest\_array    End Function | This is a UDF (not to confound with an UFO, see the acronyms at the end of this document). It takes as arguments the columns in CDR raw data (the columns of costs, answered status, CLD, and time), the answered and time criteria as string (to define the boundaries of the interval), the sub cost of the total interval cost a prefix should weight in order to continue the drilling toward a more specific prefix.  This is a UDF to be used in the excel worksheet and moreover, it is an array function. It returns three values occupying a row of 3 adjacent cells. Therefore when typing this function in the Excel worksheet, three cells in a row must be selected and instead of hitting on Enter, you must press Ctrl-Shift-Enter in order to apply the single array formula to the three selected cells (each taking one of the three return values of a single function call). As each of the three return values has a different type (string for the prefix, double for the cost, and string for the log text), the array has a special type, called Variant which can hold values belonging to several Excel types.  Calling the drill subroutine which is doing the real job. It receives all input variables and returns the deepest prefix, the deepest cost, and the drill log variables parsed as 7th, 8th, and 10th arguments. These arguments are parsed as references in contrast to all other arguments that are parsed as values. So the three local variables defined in this UDF will be modified by the drill subroutine. We will take this variable and assign them to the array of three elements of type Variant. This array will be returned by the UDF.  Next we describe the arguments this UDF receives and parses to drill subroutine in the order and more specifically. The 1st argument is the column of costs in the raw CDR range. The 2nd one is the column of the call statuses or of the billing table of the answered or failed calls (in our example we use porta-billing which stores the answered and failed calls in separate tables). The 3rd one is the criterion (a text string) for the selection of the answered calls. The 4th argument is the CLD (the called number), the 5th argument specifies a specific prefix you wish to drill down and it must be an empty string when all possible prefixes are subject of examination, the 6th argument is the cost of the specific prefix and is the entire cost of the interval if the 5th previous argument is an empty string. The 7th argument is parsed by reference and is used to hold the deepest prefix found and returned by drill subroutine, and the 8th argument, also parsed by reference, holds the cost of the deepest prefix (returned in 7th argument) returned by drill subroutine. The 9th argument is the lower bound of the cost of a drilled prefix. The drilling stops if the prefix costs less than this sub cost. If the cost of the prefix is still high the value of the 9th argument the subroutine will continue to search more specific (i.e. longer) prefixes until the cost of the longer prefix hits this lower bound. The 10th argument is parsed by reference and holds a log text returned by drill subroutine. The 11th argument is the column of times in raw CDR range, and the 12th and 13th arguments are the boundary criteria for time interval. |
|  |  |
| Sub drill( \_  ByVal costs As range, \_  ByVal answered As range, \_  ByVal answered\_criteria As String, \_  ByVal CLD As range, \_  ByVal this\_pref As String, \_  ByVal this\_cost As Double, \_  ByRef deepest\_pref As String, \_  ByRef deepest\_cost As Double, \_  ByVal drill\_until As Double, \_  ByRef drill\_log As String, \_  ByVal time As range, \_  ByVal time\_criteria1 As String, \_  ByVal time\_criteria2 As String \_  ) | This subroutine must be the most complex one in the macro. It does the job of finding the most specific prefixes with the heaviest cost. The digging is carried out recursively.  The 13 arguments described in comments of the UDF above are recalled here.  The column of costs in CDR range  The column of the status (answered/failed)  The criterion string for selection of answered calls  The called number column  The base prefix to dig (must be empty at the root of call)  The cost of the base prefix being parsed (the entire cost of the interval of the base prefix is an empty string)  The return value of the deepest prefix (expected back by reference)  The cost of the deepest prefix (expected back by reference)  The cost until which the prefix must be deepened (the deeper/longer is the prefix the less is the cost and this value is the stop condition for digging)  The log information returned by the subroutine  The time column in CDR range  The lower bound condition string for the time column (beginning of the interval)  The upper bound condition string for the time column (end of the interval) |
| Dim digit\_pref As Integer  Dim digit\_cost As Double | Examine [this prefix] with an additional digit from 0 to 9.  The digit cost is the cost of [this prefix] extended by an additional digit. There are 10 costs to be examined for each of 10 additional digits. |
| Dim deep\_pref As String  Dim deep\_cost As Double | Here are two local variables in the current instance of the function (recall that the function is called recursively so at each level of the recursion will be present a different instance of these two local variables). For each digit, we ask the drill subroutine to return (into these two local variables) the longest continuation that [this prefix] extended by the digit will lead to. |
| deepest\_pref = "" | This is the value to return to the previous instance. Here variable [deepest prefix] refers to the local variable [deep prefix] of the previous instance. We set it to empty string. If among ten digits no digit exceeds the minimal cost [drill until], this variable will remain empty, telling to the previous instance of the function that no further digging is possible. |
| For digit\_pref = 0 To 9    digit\_cost = WorksheetFunction.SumIfs( \_  costs, \_  answered, \_  answered\_criteria, \_  CLD, \_  "=" & this\_pref & digit\_pref & "\*", \_  time, \_  time\_criteria1, \_  time, time\_criteria2) | For each of the ten digits examine the cost of [this prefix] extended by that digit. |
| If digit\_cost > 0 And digit\_cost > drill\_until Then    drill \_  costs, \_  answered, \_  answered\_criteria, \_  CLD, \_  this\_pref & digit\_pref, \_  digit\_cost, \_  deep\_pref, \_  deep\_cost, \_  drill\_until, \_  drill\_log, \_  time, \_  time\_criteria1, \_  time\_criteria2 | If an appendix digit (an appendix to [this prefix]) is found such that the cost of the (more specific) prefix is still above the limit (of [drill until]), then it means already that first of all the [deepest prefix] will not remain empty.  We now do a self-call to drill subroutine with the base prefix equal to [this prefix] extended by [digit prefix]. The subroutine will return us whether there is a further continuation of this story. The tail, if exists, will be returned into local variable [deep prefix]. |
| If Len(digit\_pref & deep\_pref) > Len(deepest\_pref) Then  deepest\_pref = digit\_pref & deep\_pref  deepest\_cost = deep\_cost | We already have at least one digit in addition to [this prefix]. If [deepest prefix] was empty, then we replace it with the digit [digit prefix] eventually followed by the additional tail [deep prefix], which is empty if no tail available.  If [deepest prefix] was already a non-empty string, then we will see if our current digit with its tail, lead us to a longer length than the current [deepest prefix] has. If so, we replace the [deepest prefix] by the longer one just found. |
| ElseIf Len(digit\_pref & deep\_pref) = Len(deepest\_pref) \_  And deep\_cost > deepest\_cost Then  If drill\_log <> "" Then drill\_log = drill\_log & ","  drill\_log = drill\_log & "+" \_  & digit\_pref & deep\_pref & "@" & Round(deep\_cost, 1) & "/" \_  & "+" & deepest\_pref & "@" & Round(deepest\_cost, 1)  deepest\_pref = digit\_pref & deep\_pref  deepest\_cost = deep\_cost  End If    End If | If the lengths are identical we will have a look at the costs and will take the one which is the costlier.  Here is the only place where the drill subroutine updates its log string, it is when the prefixes have identical lengths and a more costly one shall replace the less costly. |
| Next digit\_pref    If deepest\_pref = "" Then deepest\_cost = this\_cost    End Sub | If the value of [deepest prefix] remained empty, it means that each of the ten examined digits brought the cost below the [drill until] limit. It means that [this prefix] cannot be any longer if its cost has to remain above [drill until] level. |
|  |  |
| Sub CDR\_Fit()    Dim cdr\_columns As range  Dim time\_column As range  Dim time\_values As range  Dim cdr\_lines As Long    ThisWorkbook.Activate    cdr\_lines = range("cdr\_lines").Value  Set cdr\_columns = range("cdr\_columns")  Set time\_column = range("cdr\_time")    On Error Resume Next  Set time\_values = time\_column \_  .SpecialCells( \_  Type:=xlCellTypeConstants, \_  Value:=xlNumbers \_  )  If Err.Number <> 0 Then  vba\_out "no data found"  On Error GoTo 0  Exit Sub  End If  On Error GoTo 0    With time\_values  If .Areas.Count <> 1 \_  Or .Columns.Count <> 1 Then  vba\_err "invalid time column"  Exit Sub  End If  End With    With time\_values  If .Rows.Count > cdr\_lines Then  With Intersect(range(.Rows(1), \_  .Rows(.Rows.Count - cdr\_lines)) \_  .EntireRow, cdr\_columns)  vba\_out "deleting " & .Rows.Count & " top rows"  .Delete shift:=xlUp  End With  End If  End With    End Sub | This subroutine is called after the replication of CDR. After arrival of new records we check here whether the authorized number of local records is exceeded. If so, we remove the top rows in order to fit CDR in authorized limits.  Range [CDR columns] refers to all columns the CDR occupies in the local worksheet.  Range [time column] refers only to the CDR column with the time values.  Range [time values] refers to the subset of [time column] containing valid time values (the header and the empty cells are excluded).  Reading the value of the number of authorized records. Assigning the range variables to the CDR columns and to the time column CDR.  Computing the range of the time values, the range which contains only the values (without headers and empty cells). Method Special Cells will result into an error, if there are no matching cells. We catch the error and return from the subroutine if there are no time values yet (when no CDR is downloaded at all).  In case of a fragmentation, we generate an error and we exit. The number of areas in the obtained range must be equal to 1, otherwise it signifies that there are holes in the time column.  The job is done here when the number of rows in the [time values] range exceeds the number of authorized lines. We compute the range of the top rows to delete and we apply the delete method with the shift argument set to a value indicating that after the deletion the rows below must be shifted up. The range of the top rows of CDR columns to delete is computed by intersecting the entire top rows with the CDR columns. |
| Sub CDR\_Replicate()    Dim conn1 As ADODB.Connection    ThisWorkbook.Activate    Set conn1 = New ADODB.Connection    On Error GoTo Handler1  conn1.Open range("mysql\_connect").Value  On Error GoTo 0    vba\_out "slow sync..."  CDR\_Sync conn1, range("timing\_slow")    vba\_out "medium sync..."  CDR\_Sync conn1, range("timing\_medium")    vba\_out "fast sync..."  CDR\_Sync conn1, range("timing\_fast")    conn1.Close    Cleanup:    Set conn1 = Nothing    Exit Sub    Handler1:  vba\_err "mysql open error " \_  & Err.Number & " : " & Err.Description  On Error GoTo 0  Err.Clear  Resume Cleanup    End Sub | This subroutine opens a connection to the database server. String [MySQL connect] contains the connection information which includes the driver name, the database server name, and the database name on the server. The username and the password are also in this string. The string is read from the data worksheet. Once the connection is opened it is shared by CDR synchronization procedures called from this subroutine.  This subroutine carries out the CDR Synchronizations at the various places of the timescale. In three areas of the data worksheet we provide three different replication rhythms. The rhythm is defined by the gap with the current time. The gap (or margin) indicates until which limit before the current time, the CDR must be synced. The larger is the gap the bigger is the delay of the arrival of the fresh records to the local worksheet. The database to which we connect is a replication of the billing radius server and has its own sync delays. By fixing a larger gap, we minimize the risk of missed call records because we wait until the replicated database receives all call records for the time interval.  The three CDR synchronization instances carried out at different rhythms, start at different points on the time scale, and will eventually overlap. The overlapping is managed by comparing the local number of call records with the remote number of call records (for time intervals being synced). The downloading does not take place when no missing records are encountered.  The two other parameters defining the rhythm of the synchronization is the minimal and the maximal time periods of synchronization. The minimal time period is a value which does not permit to proceed with sync if the time from the last synced record until the current time minus the gap is not sufficiently large. The larger is the value of the minimal period the less frequently the synchronization will take place.  In our example we have a slow synch rhythm that starts 40 days prior the current time with a minimal synch period equal to 24 hours and with the gap equal to 72 hours. That means that the synchronization edge cannot approach to the current time closer than a 3-days distance and that when it hits this limit, the synchronization cannot be carried out more frequently than once per day.  Then we defined the medium synch rhythm that starts 10 days prior the current time with a minimal synch period equal to 1 hour and with the gap equal to 1 hour. This sync rhythm will respect a 1-hour distance with the current time and when hits this limit will not be executed more frequently than every 1 hour.  When the slow and medium syncs overlap they will not re-download the data if records are already downloaded and the local and remote numbers of calls match. However if the medium sync missed some records the slow sync can download the missing lines. The medium sync may miss records if the replication server encounters problems lasting more than 1 hour.  Finally we defined the fast sync which initially starts 1 hour before the current time and can reach a 5-minute limit before the current time. If records belonging to a 5-minute old interval are not yet arrived to the replication server, the fast sync will miss them, but the medium sync will catch up within an hour or so.  Replication will not start and the subroutine will end if an error occurs while opening the MySQL connection. The next attempt will take place at the following periodic call of the procedure.  Note: the new version of VBA macro (see the Excel file available for downloading) does not contain a fixed number of ranges for slow, medium, and fast sync rhythms. Instead we have a single multi-area range called [timing rhythms]. This multiple area range is defined by the user and can contain from one (not recommended) to as many rhythm-areas as needed. |
|  |  |
| Sub CDR\_Sync( \_  conn1 As ADODB.Connection, \_  timing As range)    Dim time\_oldest As Date  Dim time\_min As Date  Dim time\_max As Date  Dim time\_margin As Date  Dim time\_last As range | The subroutine receives the opened MySQL connection descriptor and the range of parameters determining the rhythm of the synchronization. There are five parameters in the range. The first parameter is the default oldest time starting from which the synchronization shall start. It is a time value to be subtracted from the current time. This parameter is used when the last synchronization is not available, e.g. when the application is launched the first time. The oldest time is used also when the last synchronization is too old. Variable [time last] is declared as range as we will need not only to read its value but also update its value in the data worksheet. |
| Dim time\_utc As Date  Dim time\_after As Date  Dim time\_stop As Date | Value of [time UTC] is the current time in the UTC time zone. The value of [time after] is the previous stop time, after which the calls must be retrieved. Variable [time stop] is the time until which (inclusively) the records of calls must be downloaded. |
| ThisWorkbook.Activate    With timing  If .Areas.Count <> 1 Then  vba\_err "invalid timing range"  Exit Sub  End If  If .Columns.Count <> 1 Or .Rows.Count <> 5 Then  vba\_err "invalid timing input"  Exit Sub  End If  End With | Verifying the properties of the range of parameters defining the rhythm. |
| time\_oldest = timing.Rows(1).Value  time\_min = timing.Rows(2).Value  time\_max = timing.Rows(3).Value  time\_margin = timing.Rows(4).Value  Set time\_last = timing.Rows(5) | Assigning the values to the parameters and a reference to the cell of the last synced call time. |
| time\_utc = Now - range("time\_zone") \* TimeValue("1:00:00")  time\_utc = DateValue(Format(time\_utc, "yyyy-mm-dd")) \_  + 1 \* TimeValue(Format(time\_utc, "hh:mm:ss")) | Computing the UTC time and rounding it to a second (seems it is not necessary as the “now” function of VBA does not return milliseconds anyway in contrast to the same worksheet function). The database contains the time rounded to seconds and for the time selection we use the “between” statement of MySQL with inclusive properties for the lower and upper bounds. For two adjacent intervals in the MySQL request we set the start value of the second interval to the second following the stop value of the previous interval. Therefore making sure that in our application we are dealing with integer seconds is important (it does not mean that seconds are stored as integers in Excel). |
| If time\_utc - time\_last.Value > time\_oldest Then  vba\_out "ignore last stop"  time\_after = time\_utc - time\_oldest  Else  time\_after = time\_last.Value  End If | Computing after which time the calls must be downloaded. If the last stop value is not too far in the past we will download after the last sync time. |
| If time\_after + time\_min < time\_utc - time\_margin Then    time\_stop = time\_after + \_  WorksheetFunction.Min(time\_max, \_  time\_utc - time\_margin - time\_after)    If CDR\_Update( \_  conn1, \_  range("cdr\_columns"), \_  range("cdr\_time"), \_  time\_after + TimeValue("0:00:01"), \_  time\_stop) Then    time\_last.Value = time\_stop    Else    vba\_err "CDR update failed"    End If    Else  vba\_out "skip sync"  End If    End Sub | If the minimal CDR retrieval period does not exceed the security (margin) limit of the current time, then we shall proceed with the sync. Otherwise we must skip the sync until the next time when there will be sufficient time lap between the last sync and the current time.  Stop time is computed as the time until the security margin of the current time, but such that the period does not exceeds the max time.  Calling the CDR update function which returns true if the update is successful. Note that the start time value is the next second after the last stop value. The new value of the last stop is set to the new stop time if the CDR update was successful. |
|  |  |
| Function CDR\_Download( \_  ByVal radius As ADODB.Connection, \_  ByVal mysql\_cdr As String, \_  ByVal cdr\_columns As range, \_  ByVal time\_column As range, \_  ByVal wherefrom As range, \_  ByVal space As Long \_  ) As Boolean | CDR Update function is called by CDR Sync function. CDR Update determines the location in the local worksheet where the new data must be inserted and calls this function, where the actual downloading takes place.  We receive as arguments, the opened connection descriptor, the string containing the MySQL request retrieving the CDR, CDR columns and time column ranges in the local worksheet, the location of the cell in the time column starting from where the data must be copied, and finally the amount of lines to be inserted in the worksheet before copying the downloaded data. |
| Dim CDR\_set As ADODB.Recordset    ThisWorkbook.Activate    vba\_out "downloading"  On Error GoTo Handler1  Set CDR\_set = radius.Execute(mysql\_cdr)  On Error GoTo 0 | Parsing the request to the server. If error occurs, we quit function with the false return value indicating that the downloading did not succeed. |
| Set wherefrom = Intersect(cdr\_columns, wherefrom.EntireRow)    If space = 0 Then  vba\_out "copying"  shdata  On Error GoTo Handler2  wherefrom.CopyFromRecordset CDR\_set  On Error GoTo 0  ElseIf space > 0 Then  vba\_out "inserting"  range(wherefrom, wherefrom.Offset(space - 1)) \_  .insert shift:=xlDown  vba\_out "copying"  shdata  On Error GoTo Handler2  wherefrom.Offset(-space).CopyFromRecordset CDR\_set  On Error GoTo 0  End If    CDR\_set.Close  Set CDR\_set = Nothing    vba\_out "format"  cdr\_columns.NumberFormat = "General"  time\_column.NumberFormat = "yy-mm-dd hh:mm:ss"  vba\_out "synced"    CDR\_Download = True    Cleanup:    Exit Function    Handler1:  vba\_err "download query error " \_  & Err.Number & " : " \_  & Err.Description  On Error GoTo 0  Err.Clear  CDR\_Download = False  Resume Cleanup    Handler2:  vba\_err "transfer error " \_  & Err.Number & " : " \_  & Err.Description  On Error GoTo 0  Err.Clear  CDR\_Download = False  Defragment  Resume Cleanup    End Function | Extending the cell starting from where the CDR must be copied to a row of CDR columns.  If there is no additional space to make copy the data retrieved from the server and stored in (or rather represented by) the record set object. If an error occurs while copying, exit from the function with the false value. The network transfer is initiated by the execution of the MySQL request. The limited buffer size and the flow control mechanism can stall the network transfer at the initial phase. The network transfer will be fully completed when the copy from record set operation is completed. Therefore the network errors may occur not only when executing the MySQL request but also when copying the data from the record set. The errors from both origins must be caught and trigger a return from this function with the false value.  If space is needed, insert it. When inserting rows the cell starting from where the data must be copied is shifting down. We compute the right location by compensating this shift with a negative offset.  Space is inserted when CDR is downloaded in the middle of the timescale (i.e. we already have downloads after the period being downloaded) and the number of lines to be inserted corresponds to the number of missing records observed when comparing the number of calls on the remote server with the number of calls in the local worksheet.  Before copying the data from record set we obligatorily activate the data worksheet. Excel produces an error otherwise (but the data is being actually copied).  Destroying the record set.  Due to a bug, Excel changes the formatting of the time column every time a record set is copied; the reason why we reformat the CDR columns after the copy is complete.  Function returns the true value when no error occurred.  The error handler 2 catches the errors occurring while copying from the record set. If a large empty space is inserted in the middle of the CDR columns and the copying/transfer is interrupted, there is a risk of an empty space, i.e. of a discontinuity, in CDR columns. The rest of the macro cannot operate properly if CDR columns are fragmented. Therefore when copy/transfer error occurs we control the local CDR columns with the Defragment subroutine. |
|  |  |
| Sub Defragment()    Dim cdr\_columns As range  Dim time\_column As range  Dim time\_values As range  Dim upper As range  Dim lower As range    ThisWorkbook.Activate    Set cdr\_columns = range("cdr\_columns")  Set time\_column = range("cdr\_time")    On Error Resume Next  Set time\_values = time\_column \_  .SpecialCells( \_  Type:=xlCellTypeConstants, \_  Value:=xlNumbers \_  )  If Err.Number <> 0 Then  Err.Clear  On Error GoTo 0  vba\_err "no time values to defragment"  Exit Sub  End If  On Error GoTo 0    If time\_values.Areas.Count = 1 Then  vba\_out "time values are continuous"  Exit Sub  End If    With time\_values  If .Areas(2).Row > .Areas(1).Row Then  Set upper = .Areas(1)  Set lower = .Areas(2)  Else  Set upper = .Areas(2)  Set lower = .Areas(1)  End If  End With    With Intersect(cdr\_columns, range( \_  upper.Rows(upper.Rows.Count).Offset(1, 0), \_  lower.Rows(1).Offset(-1, 0)).EntireRow)    vba\_out "Gap of " & .Rows.Count & " rows"  .Delete shift:=xlUp    End With    Defragment    End Sub | Here we create a range of time values, a subset of the time column range.  If the time values range is fragmented at least into two pieces then we remove the gap between the first two pieces and call the Defragment subroutine again (recursively) to see if there are other gaps. |
|  |  |
| Function CDR\_Update( \_  ByVal conn1 As ADODB.Connection, \_  ByVal cdr\_columns As range, \_  ByVal time\_column As range, \_  ByVal time\_start As Date, \_  ByVal time\_stop As Date) As Boolean | This function is called by CDR Sync subroutine. It receives as arguments the start and the stop times. It returns to the CDR Sync true if it successfully updates the local worksheet records in the range between start and stop times. |
| Dim xsql\_count As String  Dim xsql\_cdr As String  Dim between As String  Dim mysql\_count As String  Dim mysql\_cdr As String  Dim rs1 As ADODB.Recordset | The first string is the template of the MySQL request for counting the records. The template becomes executable when it is edited and a valid period between start and stop times is specified. The second string is the template of the MySQL request for requesting the actual CDR. The third string is the text defining the range between the start and stop times. The next two strings are valid MySQL records corresponding to the two samples and to the value of the “between” string. |
| Dim wherefrom As range  Dim shiftdown As Boolean  Dim localcount As Long  Dim remotecount As Long | The first variable represents the cell where from the copy of CDR corresponding to the start/stop range must be copied. The second variable informs whether an insertion of rows is necessary. The two last variables correspond to the number of records found locally and on the server for the period of time between the start and stop values. |
| ThisWorkbook.Activate    xsql\_count = range("mysql\_count").Value  xsql\_cdr = range("mysql\_cdr").Value | Reading the templates of MySQL requests for counting calls and retrieving CDR. |
| period\_location \_  time\_column, \_  time\_start, \_  time\_stop, \_  wherefrom, shiftdown, localcount | We call a subroutine which takes the start and stop times of a period, analyzes the time column in the local data worksheet, and returns us three arguments (parsed as references), which are the place where from the CDR for this period must be copied, whether insertion of rows is required, and the number of calls found in the local worksheet. |
| CDR\_Update = False    If Not wherefrom Is Nothing Then    Set wherefrom = Intersect(cdr\_columns, wherefrom.EntireRow)    between = "between " \_  & Format(time\_start, "'yyyy-mm-dd hh:mm:ss'") \_  & " and " \_  & Format(time\_stop, "'yyyy-mm-dd hh:mm:ss'")    vba\_out "UTC " & between    mysql\_count = Replace(xsql\_count, \_  "[between]", between)    mysql\_cdr = Replace(xsql\_cdr, \_  "[between]", between)    If Not shiftdown Then    CDR\_Update = CDR\_Download(conn1, mysql\_cdr, \_  cdr\_columns, time\_column, \_  wherefrom, 0) | The range returned by the previous subroutine is empty only in case of serious errors, such as a disorder in CDR columns. If the range where from the copy must start is a valid position, continue with the communication with the remote server.  Compute the “between” condition string of MySQL requests.  Compute the MySQL strings retrieving the counts and the CDR for the period in question.  If CDR for our period must be copied from the last location without any insertion, proceed directly with the retrieval of CDR without counting records on the remote server.  Set the return value of this function equal to the return value of the CDR Update function being called. |
| Else    On Error GoTo Handler1  Set rs1 = conn1.Execute(mysql\_count)  On Error GoTo 0  rs1.MoveFirst  remotecount = rs1.Fields(0).Value  rs1.Close  Set rs1 = Nothing    If remotecount > localcount Then    vba\_warn "missing " \_  & (remotecount - localcount) \_  & " " & between & " UTC"  CDR\_Update = CDR\_Download(conn1, mysql\_cdr, \_  cdr\_columns, time\_column, \_  wherefrom, remotecount - localcount)    Else    vba\_out "no missing records"  CDR\_Update = True    End If    End If    Else    CDR\_Update = False    End If    Cleanup:    Exit Function    Handler1:  vba\_err "counting error " \_  & Err.Number & " : " \_  & Err.Description  On Error GoTo 0  Err.Clear  CDR\_Update = False  Resume Cleanup    End Function | If location found in local worksheet is in the middle of CDR records and therefore requires insertion figure out first the number of calls of the period on the remote server.  If the remote number of calls is greater than the local number of calls, update the calls of the period by parsing to the CDR Update function the difference in the number of calls to be inserted in the local worksheet before downloading the calls.  In principle the remote number of calls can be less than the local number when the cleanup procedures of the server delete old records. We do not update the local worksheet in such case as we have our own cleanup procedure (CDR Fit).  We had troubles counting the number of records in the record set. Whether it is a problem of the driver or a bug in Excel, the record count method does not return a valid number neither the last record condition can be detected when looping through the records of the record set. CDR Update function therefore cannot discover the number of remote records without parsing a preliminary SQL request counting the records explicitly.  If an error occurred while counting the records the function ends with a false return value. |
|  |  |
| Sub period\_location(ByVal time\_column As range, \_  ByVal time\_start As Date, ByVal time\_stop As Date, \_  ByRef wherefrom As range, \_  ByRef shiftdown As Boolean, \_  ByRef matched As Long) | This subroutine finds in the time column of CDR for a period bounded by the start and stop input values the location (the row in the column) where the period belongs. It also returns the number of local records belonging to the period.  The last three arguments are parsed by reference and are used by this subroutine to return the results. |
| ThisWorkbook.Activate    Dim time\_values As range  Dim first As range  Dim last As range  Dim i\_time As range  Dim match As range | Range [time values] will refer to the cells with time values of the time column (no headers or empty cells). The [first] is a reference to the first cell with time value in the time column and the [last] is a reference to the last cell with a time value.  Range [match] represents all cells falling into the period in question (defined by start and stop time values). |
| On Error Resume Next  Set time\_values = time\_column \_  .SpecialCells( \_  Type:=xlCellTypeConstants, \_  Value:=xlNumbers \_  )  If Err.Number <> 0 Then  Set wherefrom = time\_column \_  .Find(what:="\*", \_  SearchDirection:=xlPrevious) \_  .Offset(1, 0)  If wherefrom Is Nothing Then \_  Set wherefrom = time\_column.Cells(1)  shiftdown = False  matched = 0  Err.Clear  On Error GoTo 0  Exit Sub  End If  On Error GoTo 0 | Compute the subset of time column consisting of cells with time values only. An empty subset generates an error and we catch it. In case the subset is empty (no data downloaded yet), we must return the first location after the header. If even no header is found, we return the first row of the column. Shifting down flag is obviously set to false and the number of local records to zero. We quit the subroutine. |
| If time\_values.Areas.Count <> 1 Then  Set wherefrom = Nothing  shiftdown = 0  matched = 0  vba\_err "fragmented range of time values"  Exit Sub  End If    If time\_values.Columns.Count <> 1 Then  Set wherefrom = Nothing  shiftdown = 0  matched = 0  vba\_err "time values in multiple columns"  Exit Sub  End If | We return the value of Nothing in the where from range in case of errors. |
| With time\_values  Set last = .Rows(.Rows.Count)  End With    If time\_start > last.Value Then  Set wherefrom = last.Offset(1, 0)  shiftdown = False  matched = 0  Exit Sub  End If    Set first = time\_values.Rows(1)    If time\_stop < first.Value Then  Set wherefrom = first  shiftdown = True  matched = 0  Exit Sub  End If | If the period lies after the last local CDR record then return the next position after the last record and quit.  If the period lies before the first local CDR record then return the position of the first record and tell the calling function that for adding records insertion of rows is required. |
| Set wherefrom = Nothing  shiftdown = True  matched = 0    Set match = Nothing    For Each i\_time In time\_values.Cells  If i\_time.Value >= time\_start Then  If wherefrom Is Nothing Then \_  Set wherefrom = i\_time  If i\_time.Value <= time\_stop Then  matched = matched + 1  If match Is Nothing Then  Set match = i\_time  Else  Set match = Union(match, i\_time)  End If  Else  Exit For  End If  End If  Next i\_time | Go through each cells time values. The first cell with a value more than or equal to the start-time is the location where the CDR of the period belong to. For all cells with a value more than or equal to the start-time check if the value is less than or equal to the stop-time. If not quit the loop. If yes increment the counter of matched calls and join all such cells into a range. |
| If Not match Is Nothing Then  If match.Areas.Count <> 1 Then  vba\_err "disorder in time values"  Set wherefrom = Nothing  End If  End If    End Sub | Check if the local cells belonging to the period of time are forming a continuous area in the worksheet. If not we have a serious error and must quit with the location reference equal to Nothing. |
|  |  |
| Sub vba\_out(ByVal message As String)    vba\_msg message, 1    End Sub | Logging a standard message |
| Sub vba\_warn(ByVal message As String)    vba\_msg message, 2    End Sub | Logging a warning message |
| Sub vba\_err(ByVal message As String)    vba\_msg message, 3    End Sub | Logging an error message |
| Sub vba\_alert(ByVal message As String)    vba\_msg message, 4    End Sub | Logging a fraud alert |
| Sub vba\_clr()    Dim log\_range As range  Dim log\_index As range    ThisWorkbook.Activate    Set log\_range = range("log\_range")  Set log\_index = range("log\_index")    With log\_range  With range(.Rows(2), .Rows(.Rows.Count))  .Clear  .Interior.ColorIndex = xlNone  End With  .Columns(1).NumberFormat = "yyyy-mm-dd hh:mm:ss"  End With    log\_index.Value = 2    End Sub | The logging “screen” occupies two columns in the data worksheet. This procedure cleans the content of the logging screen. |
| Sub vba\_msg( \_  ByVal message As String, \_  ByVal priority As Integer)    If True Then  vba\_msg1 message, priority  Else  vba\_msg2 message, priority  End If    End Sub | We have two scrolling direction of the logging screen. One of the subroutines prints the new messages on the top row and scrolls the rest down and the other one prints the new messages at the bottom and rotates when it reaches the last row. |
| Sub vba\_msg1( \_  ByVal message As String, \_  ByVal priority As Integer)    Dim log\_range As range  Dim log\_rows As Long  Dim log\_bottom As range  Dim log\_src As range  Dim log\_dst As range  Dim i\_cell As range    ThisWorkbook.Activate    Set log\_range = range("log\_range")  log\_rows = range("log\_rows").Value    Set log\_bottom = log\_range.Columns(1) \_  .Find(what:="\*", SearchDirection:=xlPrevious)    If Not log\_bottom.Row > 1 Then \_  Set log\_bottom = log\_range.Cells(2, 1)    If Not log\_bottom.Row < log\_rows Then \_  Set log\_bottom = log\_bottom.Offset(-1, 0)    Set log\_bottom = Intersect( \_  log\_range, log\_bottom.EntireRow)    Set log\_src = range(log\_range.Rows(2), log\_bottom)  Set log\_dst = log\_src.Offset(1, 0)    log\_dst.Value = log\_src.Value    log\_range.Cells(2, 1).Value = Now  log\_range.Cells(2, 2).Value = priority  log\_range.Cells(2, 3).Value = message    For Each i\_cell In Union(log\_src, log\_dst) \_  .Columns(2).Cells  With Intersect(i\_cell.EntireRow, \_  Union(log\_src, log\_dst)).Interior  Select Case i\_cell.Value  Case 1  .ColorIndex = xlNone  Case 2  .Color = RGB(255, 192, 0)  Case 3  .Color = RGB(255, 0, 0)  Case 4  .Color = RGB(255, 0, 255)  Case Else  .Color = RGB(240, 240, 240)  End Select  End With  Next i\_cell    End Sub | This procedure prints the new messages on the top row and the log is therefore in the inverse chronological order.  Define a range [log\_src]. It is the log space (without the header row) until the last log record, except the last log record already reached the last allowed log line. In that case the [log\_src] range stops at the row before to the last. This is to be able to shift down the range by one position without exceeding the log limit.  Define the range of [log\_dst] which is the [log\_src] range shifted down by one position.  Copy the values from the source to the destination, and all will be shifted down by one position (except the formatting).  The first log row now can carry the message (accompanied with the current time and the message priority).  Now associate the colors to the entire new log range according to the message priorities.  The standard messages are transparent.  The warning messages have an orange background color.  The error messages have red background color.  The fraud alerts are with pink color. |
| Sub vba\_msg2( \_  ByVal message As String, \_  ByVal priority As Integer)    Dim log\_range As range  Dim log\_rows As Long  Dim log\_index As range  Dim i\_row As Long    ThisWorkbook.Activate    Set log\_range = range("log\_range")  log\_rows = range("log\_rows").Value  Set log\_index = range("log\_index")  i\_row = log\_index.Value    log\_range.Cells(i\_row, 1).Value = Now  log\_range.Cells(i\_row, 2).Value = priority  log\_range.Cells(i\_row, 3).Value = message  With log\_range.Rows(i\_row).Interior  Select Case priority  Case 1  .ColorIndex = xlNone  Case 2  .Color = RGB(255, 192, 0)  Case 3  .Color = RGB(255, 0, 0)  Case 4  .Color = RGB(255, 0, 255)  Case Else  .Color = RGB(240, 240, 240)  End Select  End With    If i\_row < log\_rows Then  i\_row = i\_row + 1  Else  i\_row = 2  End If    log\_index.Value = i\_row    End Sub | In this version of the logging is in the chronological order. In the excel file provided for downloading we use the previous, inverse-chronological version. |

Note that in the published Excel file, the VBA script did undergo to slight modifications and improvements.

# Named ranges

For a reference the capture below shows all named ranges used in the Excel file. Named ranges allow a more reliable sharing of cells between the worksheet and VBA. With named ranges, no need to modify the VBA macro when the location of a cell is changed in contrast to the case when VBA access the cell or range using its address (with column and row headers).



Go to Formulas / Name Manager to access this window. Some ranges will not correspond to the script presented above. For the most recent version, refer to the script of the Excel file attached to this publication. For example the named ranges [smtp\_account1], [smtp\_account2], and [smtp\_account3] are replaced in the new version by a single multiple-area named range [smtp\_accounts]. Similarly, the timing ranges are replaced by a single [timing\_rhythms] multiple-area range.

# Recent functionalities

Additional functionality is added in the recent version. Brief description of these functions is given in the table below. The existing code is also modified in several places. Refer to the script of the Excel file of the installation section.

|  |  |
| --- | --- |
| Code | Comments |
| Sub Clear\_Unused()    Dim params As range  Dim chart3 As range  Dim all As range  Dim icol As range  Dim ilast As range  Dim iblank As range    ThisWorkbook.Activate    vba\_out "clearing blank cells"    Set params = range("param\_columns")  Set chart3 = range("chart\_columns3")    Set all = range(params.Columns(1), chart3.Columns(chart3.Columns.Count))    range(all.Columns(all.Columns.Count).Offset(0, 1), \_  all.Rows(1).EntireRow.Columns( all.Rows(1).EntireRow.Columns.Count ).EntireColumn).Clear  vba\_out "right columns cleared"    For Each icol In all.Columns  Set ilast = icol.Find(what:="\*", SearchDirection:=xlPrevious)  If ilast Is Nothing Then  Set iblank = icol.Rows(1)  Else  Set iblank = ilast.Offset(1)  End If  range(iblank, iblank.EntireColumn.Rows( iblank.EntireColumn.Rows.Count)).Clear  Next icol  vba\_out "bottom rows cleared"    End Sub | If the Excel application considers that the empty cells of the bottom rows are used, it does will not allow inserting (with the shifting down option) of chunks of cells or of rows above the used cells that may move the used cell beyond the limits of the spreadsheet.  Insertion of CDR rows occurs when the synchronization subroutines detect missing calls within the timescale.  In this subroutine we clear all unused cells outside of the range of parameters, log records, CDR, statistic columns, and the chart columns.  First we define a range that covers all used columns.  We then clear all columns on the right, after the last used by our script.  Then for each column we clear the cells laying below the last position. |
| Sub Tail\_Suspect( \_  ByVal length As Date, \_  ByVal minhcost As Double, \_  ByVal wpref As Double, \_  ByVal fpref As Double)  Dim cdr As range  Dim first\_call As Date  Dim last\_call As Date  Dim start\_from As Date  Dim time\_criterion1 As String  Dim time\_criterion2 As String  Dim cost As Double  Dim hcost As Double  Dim aprefix() As Variant  Dim overall As Double    Dim cdrfile As String  Dim alert As String    ThisWorkbook.Activate    range("tail\_attach,tail\_text").Value = ""    Set cdr = range("cdr\_columns")  first\_call = WorksheetFunction.Min(cdr.Columns(4))  last\_call = WorksheetFunction.Max(cdr.Columns(4))  start\_from = last\_call - length  time\_criterion1 = ">=" & WorksheetFunction.Text(start\_from, "General")  time\_criterion2 = "<=" \_  & (WorksheetFunction.Text(last\_call, "General") + 0.5 \* 1 / 24 / 3600)    vba\_out "exam tail " & WorksheetFunction.Text(length, "[h]:mm:ss")    cost = WorksheetFunction.SumIfs( \_  cdr.Columns(6), \_  cdr.Columns(4), time\_criterion1, \_  cdr.Columns(4), time\_criterion2)    hcost = cost / (length \* 24)    If hcost < minhcost Then  vba\_out Format(hcost, "0.0") & " CHF/h is too low"  Exit Sub  End If    aprefix = deepest( \_  cdr.Columns(6), \_  cdr.Columns(1), "=1", \_  cdr.Columns(2), "", cost, wpref \* cost, \_  cdr.Columns(4), time\_criterion1, time\_criterion2)    If aprefix(1) = "" Then  vba\_out "no prefix found"  Exit Sub  End If    vba\_out "prefix +" & aprefix(1)  vba\_out "costs " & Round(aprefix(2), 3)  If aprefix(3) <> "" Then vba\_out "log " & aprefix(3)    overall = WorksheetFunction.SumIfs( \_  cdr.Columns(6), \_  cdr.Columns(2), "=" & aprefix(1) & "\*")    vba\_out "overall " & Round(overall, 1)    If (aprefix(2) / (length \* 24)) \_  / (overall / ((last\_call - first\_call) \* 24)) \_  < fpref Then    vba\_out "low factor " & Round(aprefix(2) / (length \* 24), 1) & " / " \_  & Round(overall / ((last\_call - first\_call) \* 24), 1)  Exit Sub    End If    cdrfile = savecdr(start\_from, last\_call, aprefix(1))  alert = "" \_  & "+" & aprefix(1) \_  & " costs " & Round(aprefix(2), 1) & "CHF = " \_  & Round(aprefix(2) / (length \* 24), 1) & "CHF/h within last " \_  & Round(length \* 24 \* 60, 1) & " minutes whereas " \_  & "+" & aprefix(1) \_  & " costs " & Round(overall, 0) & "CHF = " \_  & Round(overall / ((last\_call - first\_call) \* 24), 3) & "CHF/h over full period of " \_  & WorksheetFunction.Text(last\_call - first\_call, "[h]\hm\ms\s") \_  & ""    range("tail\_attach").Value = cdrfile  range("tail\_text").Value = alert    End Sub | In this subroutine we analyze the tail of CDR columns. The duration of the tail is defined in data the worksheet. This duration does not depend on the chart intervals and is an input parameter defined by the user. It is equal to 45 minutes in our sample Excel file.  We compute the total cost of the interval.  Then we compute the hourly cost.  If the hourly cost is below the limit defined in the data worksheet (111 CHF/h in our sample), we do not analyze further the interval.  If the hourly cost is high, we drill for the longest prefix costing more than the input weight defined in the data worksheet (20% in our case).  If no prefix is found (taking more than 20% of the total cost) we quit the subroutine.  If a heavily used prefix found, we compute its average hourly rate over the entire period of time axis. If the hourly rate of the prefix in last 45 minutes exceeds the overall hourly rate (throughout the entire observation period) by a factor less than 33 times (an input parameter defined in the data worksheet), then quit the subroutine.  If we are still in the subroutine, we generate the CDR file for the recent period (45 minutes in our case).  A warning message is also generated briefly describing the nature of the alert. |
| Function savecdr( \_  ByVal tstart As Date, \_  ByVal tstop As Date, \_  ByVal prefix As String) As String    Dim cdr As range  Dim wherefrom As range  Dim shiftdown As Boolean  Dim matched As Long    Dim cdrWB As Workbook  Dim i As Long    ThisWorkbook.Activate    Set cdr = range("cdr\_columns")    period\_location cdr.Columns(4), tstart, tstop, \_  wherefrom, shiftdown, matched    If matched > 0 Then    vba\_out "exporting CDR"    cdr.Rows(1).Copy    Set cdrWB = Workbooks.Add  With cdrWB.Sheets(1).Cells(1)  .PasteSpecial Paste:=xlPasteColumnWidths  .PasteSpecial xlPasteValues, , False, False  .PasteSpecial xlPasteFormats, , False, False  End With    Intersect(cdr, \_  range(wherefrom, wherefrom.Offset(matched - 1)).EntireRow).Copy  With cdrWB.Sheets(1).Cells(2, 1)  .PasteSpecial xlPasteValues, , False, False  End With    With cdrWB.Sheets(1)  With range(.Columns(1), .Columns(cdr.Columns.Count))  .NumberFormat = "General"  .Columns(4).NumberFormat = "yyyy-mm-dd hh:mm:ss"  For i = 1 To matched  With .Cells(1 + i, 2)  If Left(.Value, Len(prefix)) = prefix Then  .Interior.Color = RGB(255, 0, 255)  .Offset(0, 4).Interior.Color = RGB(255, 0, 255)  End If  End With  Next i  End With  With ActiveWindow  .FreezePanes = False  .ScrollRow = 1  .ScrollColumn = 1  .FreezePanes = True  End With  Union(.Cells(1, 2), .Cells(1, 6)).Select  End With    savecdr = ThisWorkbook.Path & "\" \_  & ThisWorkbook.Name & "\_" \_  & Format(tstart, "yymmdd'hhmmss") & "\_" \_  & Format(tstop, "yymmdd'hhmmss") & "\_" \_  & Format(Now, "yymmdd'hhmmss") \_  & "\_CDR+" \_  & prefix \_  & ".xlsx"    Application.DisplayAlerts = False  cdrWB.SaveAs savecdr  Application.DisplayAlerts = True    cdrWB.Close    vba\_out "exported to " \_  & Dir(savecdr)    savecdr = Dir(savecdr)    Else  savecdr = ""  End If    End Function | This function generates the CDR file and returns its name to the calling subroutine.  The headers are copied into the new file.  Then all call records of the period are copied.  Then we go through all records in the CDR file and we highlight the records matching the suspicious prefix.  We freeze the header row in the CDR file being generated.  The filename consists of the time of the first record, the time of the last record, the file creation time, and the suspicious prefix. |

The current version of the macro takes 1659 lines

# Installation

[Download](download) the most recent version of the Excel file (for demo usage).

Fulfill your database connection credentials which include the server name, username, and password.

Fulfill your outgoing email connection credentials.

Fulfill the lists of recipients.

The MySQL requests are compatible with PORTA-billing database structure; customize the requests in the worksheet for your database structure if different.

Click on the circle on the data worksheet to launch the scheduler.

The MS-Word version of [this document](index11.docx) is also available.

# References

Connecting Excel to a remote MySQL server  
<http://www.switzernet.com/3/public/130715-excel-to-remote-mysql/>

Properties of Excel to remote MySQL connection  
<http://www.switzernet.com/3/public/130715-excel-mysql-connections/>

Creating a vendor cost on-line monitoring chart  
<http://switzernet.com/3/public/130716-vendor-cost-monitor/>

Retrieval of hourly cost revenue and traffic  
<http://switzernet.com/3/public/130723_cost_revenue_and_traffic_excel_mysql/>

Incremental retrieval and visualization with Excel MySQL connector   
<http://www.unappel.ch/2/public/130807-excel-vba-mysql-CDR_Vendors/>

Traffic cost and revenue monitoring  
<http://switzernet.com/3/public/130915-mysql2html2excel2chart2smtp/>

Voice traffic real-time cost and revenue monitoring with Excel  
<http://switzernet.com/3/public/131003-Excel-ADODB-CDO-traffic-reports/>

Porta-Billing  
<http://portaone.com/portabilling/>

# Link log

How to calculate Sum of Count in MySQL  
<http://stackoverflow.com/questions/6792431/how-to-calculate-sum-of-count-in-mysql>

Array function in Excel VBA  
<http://stackoverflow.com/questions/10290591/array-function-in-excel-vba>

Sending mail from Excel with CDO  
<http://www.rondebruin.nl/win/s1/cdo.htm>

CDO + Set Priority Level  
<http://www.access-programmers.co.uk/forums/showthread.php?t=191354>

CDO / Envelope Elements / Importance  
<http://msdn.microsoft.com/en-us/library/gg671973(v=exchg.80).aspx>

Sending email with VBA  
<http://www.cpearson.com/excel/EMail.aspx>

Dictionary, Encyclopedia and Thesaurus  
<http://acronyms.thefreedictionary.com/>

# Acronyms

AAA, Authentication, Authorization, and Accounting

ADO, ActiveX Data Object

ADODB, ActiveX Data Objects Database

ALOC, Average Length of Call

BASIC, Beginner's All-Purpose Symbolic Instruction Code

BCC, Blind Carbon Copy

CC, Carbon Copy

CDO, Collaboration Data Objects

CDO, Collaboration Data Objects

CDR, Call Data Records

CLD, Called Line

CLI, Calling Line Identification

IMAP, Internet Message Access Protocol

MySQL, My Structured Query Language

PDD, Post Dial Delay

PNG, Portable Network Graphics

RADIUS, Remote Authentication Dial-In User Server/Service

SMTP, Simple Mail Transfer Protocol

SQL, Structured Query Language

SSL, Secure Sockets Layer

TCP, Transmission Control Protocol

UDF, User Defined Functions

UFO, Unidentified Flying Object (has nothing to do with this document)

UTC, Coordinated Universal Time

VBA, Visual BASIC for Applications

# Legal

Copyright © 2013 by Emin Gabrielyan and Switzernet.com

END OF THIS DOCUMENT