Calculate Import/Export runtimes

Jochen Hein

$Id: runtime.xml,v 1.2 2002/08/05 19:10:34 jhein Exp $

Table of Contents

Import/Export runtimes

When you do an OS/DB migration with the SAP tools (**R3SETUP** and internally **R3load**), you can easily get runtimes longer than a day. If you happen to test the migration, you can analyze the export and import logs with this small tool. You get a runtime for each table class, so you know which **R3load** process must be started first (the longest running processes), so that you get the shortest runtime possible. For information how this has to be done, see SAP note xxx.

Analyzing the logs manually is cumbersome and, especially if you had to restart the import a couple of times, not easy. This script reads all the logfiles, stores all filenames, start and endtimes in an CSV-file. You can import the file in Excel and you are ready. To generate the CSV-file, cd into you export or import directory and call:

```
awk -f runtime.awk SAP*.log > runtime.csv
```

What we have done is reading all the **R3load** logs, stored the start and end time of each run (including restarts). Store the output in a file `runtime.csv` and import that in Excel. All you have to do is formating the computed columns as date/time. If you have restarted the import, you can add all the partial runtimes into a sum and use that for your export and import planning.

If you have any questions or comments, feel free to contact me at `<jochen@jochen.org>`. You may find an updated version on http://www.jochen.org/.

The code

The code has been written in AWK, a Unix scripting language, that is well suited for maniputating text files. As you will see, nothing magic hase been done here.

The main code of the program is structured as follows:

```
Main Listing

#!/usr/bin/awk -f
# $Id$
# This program is released under the GNU General Public License
# You may find updates under http://www.jochen.org/
Convert Column in Excel column
Compute the runtime by Excel formula
Initialize variables
Format date and time for Excel
Store a start time for this run
Store the end time for this run
finally, end the file
```

The **R3load** log files contain the date and time in an easily sorted format. Excel (and other spread sheets) import these as Numbers, which is not quite what we want. We convert a given date string (e.g. 20020731152423) and mange it into an ISO like format (e.g. 2002-07-31 15:24:23). When Excel is reading that from an CSV-file it does exactly what we want.

```
Format date and time for Excel =

# convert the date and time for Excel
function format_date(dat)
{
  return substr(dat,1,4) "-" substr(dat,5,2)  "-" substr(dat,7,2) " " \
         substr(dat,9,2) ":" substr(dat,11,2) ":" substr(dat,13,2);
}
```

Just to be sure that the above code does what we want it to do, we'll add some testcases and examples here. This gives us finally a testsuite to avoid regressions.

```
Format date and time for Excel =
```

```
function check_format_date()
{
}
```

Excel has no numbers for columns, but uses letters. So we have to convert the column number we want to reference into the Excel format. So column "1" is "A", "26" is "Z", "27" is "AA" and so on. This function works for small numbers of columns, and might be rewritten in the future. On the other hand, if you restarted the import a lot of times, your runtime calculation will be screwed anyway, so better get you import running in one go.

The algorithm is as follows...

```
Convert Column in Excel column =

# Calculate the column name
function column(col)
{
  col = col - 1;
  rem = col % 26;
  div = col / 26;

  if ( div >= 1) {
    ret = sprintf("%c",64+div);
  }

  ret2 = sprintf("%c", 65+rem);
  return ret ret2
}
```

Again, to be sure, we need to create some test cases.

```
testcases for column =

function check_column(col, expect)
{
  result = column(col);
  if ( result != expect ) {
    printf "Error: col %s, exp %s, result %s\n", col, expect, result;
  }
}
```

```
function check_columns()
{
  check_column(1, "A");
  check_column(2, "B");
  check_column(26, "Z");
  check_column(27, "AA");
  check_column(28, "AB");
}

function check_format()
{
}

function check_regression()
{
  check_columns();
  check_format();
}
```

```
Compute the runtime by Excel formula =

# fill Excel cell with the runtime formula (end time - start time)
function compute_runtime(run)
{
  return "=" column(run*3) filenum+1 "-" column(run*3-1) filenum+1;
}
```

Before anything else, we initialize some variables that help us to track files, restarts and that we finally found a start and end time (still, when you manually changed the logs or some weired crashed happend we will be screwed anyway).

```
Initialize variables =

BEGIN {
      if ( regression_test == 1 ) { check_regression(); exit; }
      ok=0; run=0; file = ""; filenum=0; start=0; }
```

When we find a start time in the log, we store that as a potential start time. This may be a correct start time, but that can only be checked later. If we find a new start time, but no end time, we only use the latest time. This may invalidate all computations, but it is the best we can do. Remember, if the **R3load** crashed, you have other problems to worry about.

If we started to read a new log file, we finish the last line (with CR/LF to get a DOS file). We add one to the count of files, so we can generate the right references for Excel formulas later on. We store the new filename so we can repeat the check for the next logfile...

```
Store a start time for this run =

/^#START:/ {
        if ( file != FILENAME ) {
                run = 0;
                filenum = filenum + 1;
                printf "\r\n\"%s\";", FILENAME;
                file = FILENAME;
        }
        ok = 1;
        maybe_start = format_date($2);
        };
```

Finally, we may find the end time of a **R3load** run. This is only valid for computations, if we had a start time earlier (if we have one, the variable ok hold a "1". Ok, we write the latest start time (stored on maybe_start), a ";", the formatted end time and the Excel formula to compute the runtime for this file and restart. maybe_start is initialized, so we start anew.

The detection of start and end time in the logs are not bullet-proof, but work for runs with no restarts and simple restarts pretty well. If your logfiles are beyonf repair, retry your export or import to get usful times.

```
Store the end time for this run =

/^#STOP:/ {
        run = run + 1;
        if ( ok == 1 ) {
                ok = 0;
                printf "\"%s\";", maybe_start; maybe_start = "";
                printf "\"%s\";\"%s\";", format_date($2), compute_runtime(run); };
        };
```

The program is finished, only on thing remains: The CSV file must be ended with a newline, so Excel is not overly confused. Happy hacking.

```
finally, end the file =

END { print "\r\n"; }
```