CETENNIAL COLLEGE
CO-OPRATIVE EDUCATION

WEEKLY USER USAGE REPORT AUTO CREATION AND UPLOADING

Royal Bank of Canada
315 Front Street West
Toronto, Ontario

Prepared by:

Xiao Ping Cai
5th Semester
Computer Systems Technology – Networking (Co-op)
January 6, 2012

416-839-6065

315 Front Street West
Toronto, Ontario
M5V 3A4

May 12, 2011

Ms. Melanie Richardson
Manager, Co-op Education and Employment Resources
Centennial College
P.O. Box 631, Station A
Scarborough, Ontario
M1K 5E9

Dear Ms. Richardson:

This is my second Co-op work term report which outlines my work at Royal Bank of Canada from September to December 2011.

Royal Bank of Canada (RBC) is Canada’s largest bank as measured by assets and market capitalization. It provides personal and commercial banking, wealth management services, insurance, corporate and investment banking and transaction processing services on a global basis.

This report named “Weekly User Usage Report Auto Creation And Uploading”, articulates my activities while working with Windows Terminal Server Team at the RBC. Chris Gilbert is Team Leader of Windows Terminal Server Team. Vino Balachandra is Technical Systems Analyst in this Team.

One of the responsibilities of the Windows Terminal Server Team is to create Weekly User Usage Report and upload the report to SharePoint every week. In this work term, Vino taught me to do the report and I was responsible for this report. In his method, the report was done manually, and it took 2 hours. In order to improve the efficiency, I developed a program to do the report. All the report can be done automatically, and it took only 7 minutes.

This report has been prepared and written by me and has not received any previous credit at this or any other institution. I would like to thank Mr. Gilbert and Mr. Balachandra for their help in this report.

Sincerely,

Xiao Ping Cai

ID 300-591-463

TABLE OF CONTENTS

SUMMARY

This is my second Co-op work term. I worked at the Windows Terminal Server Team in Royal Bank of Canada from September to December 2011.

One of the responsibilities of the Windows Terminal Server Team is to create Weekly User Usage Report and upload the report to SharePoint every week.

The current method is manually creating and transferring Weekly User Usage Report to SharePoint. It takes two hours to do the report.

The purpose of this report is to develop a program to create and transfer transferring Weekly User Usage Report.

I have developed the program and given it to Vino Balachandra.

CONCLUSIONS

The current method is manually creating and transferring Weekly User Usage Report to SharePoint. It is manual work. The procedures are very complex and are easy to make mistakes. It will take a person 2 hours to do it every week.

My program can create and transfer Weekly User Usage Reports automatically in seven minutes. It can save human resource. It is very simple and more accurate.

RECOMMENDATIONS

The current method of producing and transferring Weekly User Usage Reports can be replaced by program.

The program has passed the test in the test SharePoint site.

1 INTRODUCTION

1.1 Background

This is my second Co-op work term report which outlines my work at Royal Bank of Canada from September to December 2012.

Royal Bank of Canada (RBC) is Canada's largest bank as measured by assets and market capitalization. It provides personal and commercial banking, wealth management services, insurance, corporate and investment banking and transaction processing services on a global basis.

I worked at Windows Terminal Server Team in RBC. My report manager is Chris Gilbert. He is Team Leader of Windows Terminal Server Team. Chris asked Vino Balachandra to guide my daily work.

Vino is Technical Systems Analyst. One of the jobs he guided me to do is creating and transferring Weekly User Usage Report to the SharePoint website. Weekly scheduled reports are run on Terminal Servers to keep track of the number of users logging on during various times of the day. These reports are used as a gauge to determine if servers are overloaded. The info from these reports is pulled into an Excel Pivot Table which then makes PDF charts. The process is very complex. It will take a person two hours.

I studied the development in SharePoint, Office and Web application. Finally, I developed the program to create and transfer Weekly User Usage Report to SharePoint. I have tested in the test directory in the SharePoint.

1.2 Purpose of Report

The purpose of this report is to develop a program to create and transfer Weekly User Usage Report automatically. The current method is manually creating and transferring Weekly User Usage Report to SharePoint and it will take a person two hours to do it every week. My program can do it automatically in seven minutes.

2 CURRENT OPERATING PROCEDURES

2.1 Procedures

The Excel file, Excel spreadsheets, and data files are located at \\s3w01998\G$, so you will need to map a network to there. You must map this to the Z: drive for a script to work.

Reports are scheduled by the Asset Tag/Hostname of your computer. So, if you log onto your computer and schedule these reports to run on Saturday, and then you log onto a different computer to check the scheduled runs, you won't see them. You have to go back to your computer to see them.

First step is to check if there is at least one scheduled "Run" waiting in the queue. Go to this link to check- http://s3w01694/scripts/rsasqrep.pl?folder=user

If there is at least one scheduled run left, then skip steps 1, 2, and 3 below and start the Retrieving Reports section of this doc.

Request NTSMF Reports to run on every Saturday for next 10 weeks in advance (Steps 1, 2, and 3 below).

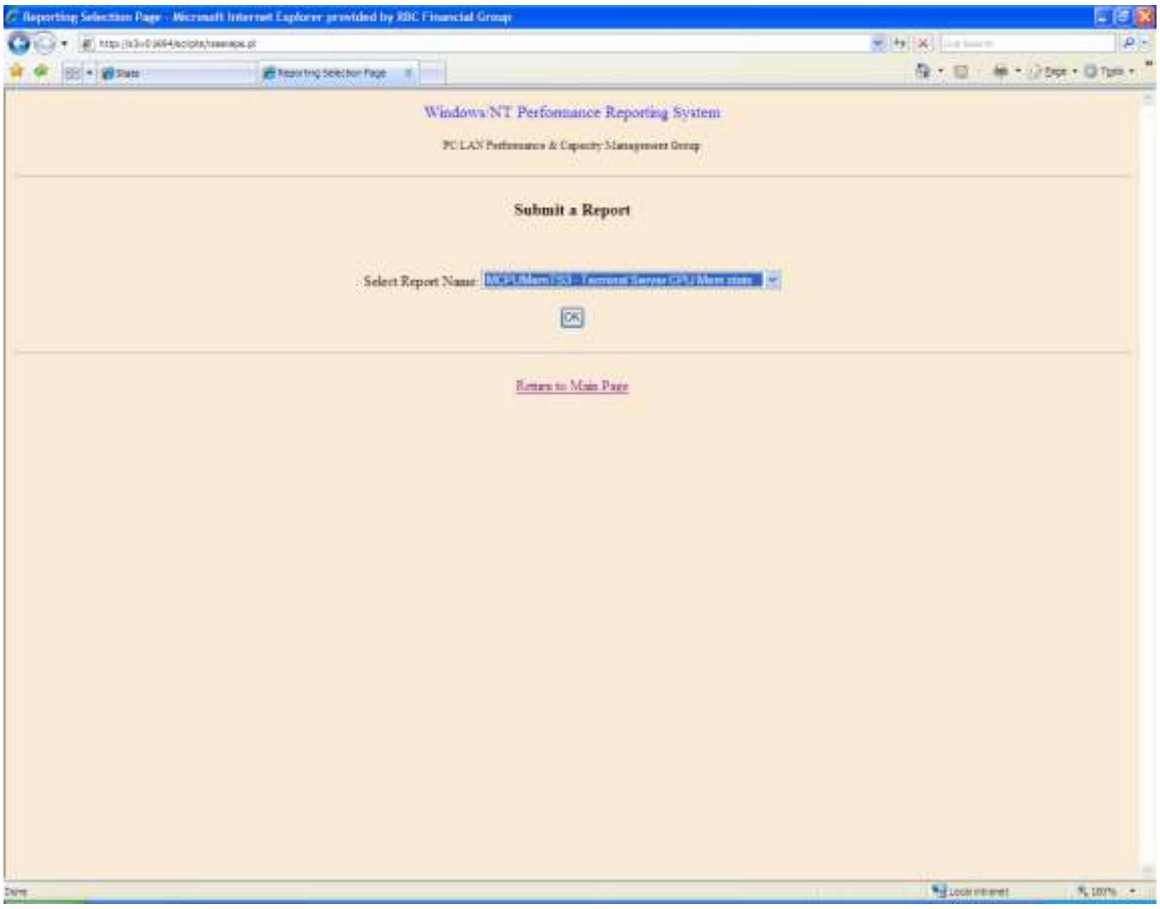
Scheduling Reports

Step #1:

NTSMF Web Reporting Site
Go to NTSMF web site http://www.itnet.fg.rbc.com/perfcapacity/cid-4903.html
Click “Web Reporting” at left side of the web page, and then select “Submit a Report” on next page.

Step #2:

Select “MCPUMemTS3” from the following page


Reporting Selection Page - Microsoft Internet Explorer provided by RBC Financial Group

Windows/NT Performance Reporting System

PC LAN Performance & Capacity Management Group

Submit a Report

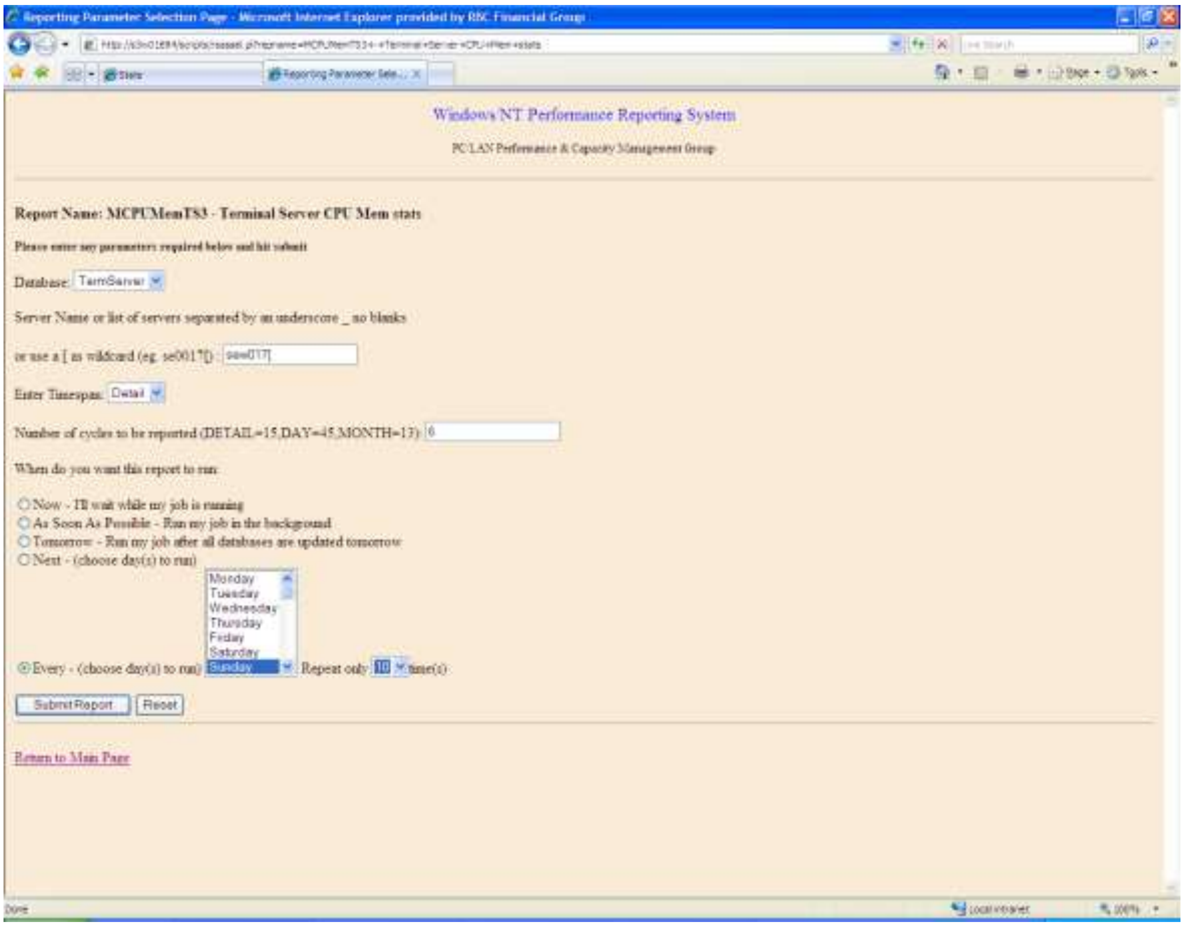
Select Report Name: MCPUMemTS3 - Terminal Server CPU/Mem stats

OK

Return to Main Page

Step #3:

Request data based on range of TS server name, details see screenshot below. Type in the following server names, one at a time, including the "[" wildcard character, don't include the ",”and click submit after each one. The naming convention of the Terminal Servers, is to have 8 characters. Server names, such as sew20016, have 8 characters, and therefore will not need the wildcard "[" character at the end. The number of cycles to be reported, is referring to the number of days to report, so input '6', for six days.



Reporting Parameter Selection Page - Microsoft Internet Explorer provided by RBC Financial Group

Windows NT Performance Reporting System

PC LAN Performance & Capacity Management Group

Report Name: MCPUMemTS3 - Terminal Server CPU Mem stats

Please enter any parameters required below and hit submit

Database: TermServer

Server Name or list of servers separated by an underscore _ no blanks

or use a [as wildcard (eg. se0017[): sew017[

Enter Timespan: Detail

Number of cycles to be reported (DETAIL=15,DAY=45,MONTH=13): 6

When do you want this report to run:

- Now - I'll wait while my job is running
- As Soon As Possible - Run my job in the background
- Tomorrow - Run my job after all databases are updated tomorrow
- Next - (choose day(s) to run)
- [✓ Every - (choose day(s) to run)] Sunday Repeat only 10 time(s)

Servers to Schedule

These are the servers required to report upon.

Branch:
s3w017[, s3w018[

NonBranch:
sew017[, sew0612[, sew30[, Sew4905[, sew95[, sew950[, s3w01744, s3w01745, s3w019[, s3w039[, S3W06[, s3w07[, se0017[, se0019[, se101[, se4000[, sew018[, sew019[, sew06[, sew20016, sew21030, USCAR[, USOLP[, VUSOLP[

Retrieving Reports / Saving Raw Data

This is to be completed every Monday morning. The raw reports generated on Sunday need to be saved to folders found in s3w01998. After all of the reports are saved in their proper folder, a script needs to be run that pulls the information from the multiple files into a larger .csv file.

1. Create User Usage Report every Monday before 10AM.
2. Retrieve the reports from http://s3w01694/rsasmain.htm and select "Browse your existing Reports". Click "MCPUMEMTS3" on the next page near the bottom.
3. Save each report of the SCHEDULED SERVERS into the report folder, see NTSMF\NTSMFRaw folder under \\s3w01998\g$.

For Branch Servers s3w017, s3w018, save here:
\\s3w01998\G$\2k3Dev\NTSMFBranch\NTSMFRaw
For Non-Branch servers, save here:
\\s3w01998\G$\2k3Dev\NTSMF\NTSMFRaw

You will be prompted to Login to server \\s3w01998\G$:
It should be SAIMaple/LAN ID

For Branch info – contact Wendy Solway.

4. Combine all data into one big CSV file, by running the .bat files in Step #5 below (see RawDataAll.csv file under \2k3dev\NTSMF\csv\.)
5. \\s3w01998\G$\2k3dev\NTSMF\Script\copycsv.bat
   \\s3w01998\G$\2k3Dev\NTSMFBranch\Script\copycsv.bat

Create PDF copies of Reports

Once all the data has been pulled into the large .csv file, it's ready to be displayed in graph form and printed off into PDF. The excel files that will generate the graphs are found in NTSMFBranch\Excel. Open up the file and refresh the pivot table found on the second tab "Pivot". (Right click on a cell and hit the "Refresh Data" option. This will take a minute.) Once refreshed go to the first tab "Chart1". Here you must change the date to the week you want to report on. Also change the top left portion of the chart to decide what group of servers and what shift to generate a graph on. The print to PDF and save in the NTSMF\Report.

1. Refresh the pivot tables (by right clicking the DATE column) in TSUserUsageV1.xls under \\s3w01998\G$\2k3dev\NTSMF\Excel, and the TSUserUsage.xls located in \\s3w01998\G$\2k3dev\NTSMFBRANCH\Excel . (If a z: drive mapping error occurs, disconnect any z: network drive mappings on your computer, and re-map z: to \\s3w01998\G$.   (A one time configuration needs to be done to create pivot table based on the 3 CSV files, do not change the CSV file names.)
2. Filter the dates on the Chart Sheets in both TSUserUsage.xls files, to include only 5 working days of the week.
3. Print Charts into PDF format and save to 2k3dev\NTSMF\report folder, by doing the following:

TSUserUsageV1.xls located in \\s3w01998\G$\NTSMFBranch\Excel, print to PDF for Branch DRP SymCor, All shifts, Branch Mobile, All shifts, and Branch SDF Dev Offshore, All shifts.

Branch DRP Symcor Capacity Chart.pdf
Branch Mobile Capacity chart.pdf
Branch SDF Dev Offshore Capacity chart.pdf
RefreshData

TSUserUsageV1.xls located in \\s3w01998\G$\NTSMF\Excel, print to PDF for NON-Branch Chart 1,(Shifts 1/P and 2/N) and NON Branch Chart 2, (Shifts 1/P and 2/N), and NON Branch DRP, ALL shifts. Print to PDF for Other Non-Branch (Shifts 1/P and 2/N, and All Shifts). Print to PDF for Opal (Shifts 1/P and 2/N). Print to PDF RBC Insurance BPO, All Shifts.

Non Branch Chart 1 HO:

Non Branch Chart1, P	Non Branch Capacity chart 1 shift 1- HO.pdf
Non Branch Chart1, N	Non Branch Capacity chart 1 shift 2 - HO.pdf

Platform: 6
Head Office/Trav 18 Servers-Max 637
HO Pandemic Servers – Max 550(?HO Pandemic – 11 Servers – Max 550)
HO OCC VM -2 Servers-Max 105
Citrix Head Office-4 Servers
Compass Citrix TS-8 Servers-Max 250
Head Office/Travel DRP-14 servers-Max 630

No Branch Chart 1:

Non Branch Chart1, P	Non Branch Capacity chart 1 shift 1.pdf

Non Branch Chart1, N	Non Branch Capacity chart 1 shift 2.pdf

Platform:8

eGL 2003-10 Servers-Max 250

GLB-17 Servers-Max 578

GPB Jersey DRP-7 servers-Max 245

RBC Centura-5 servers-Max 150

GPLUS-3 servers-Max 180

Home & Auto -6 servers-Max 500

RBTT/FBW-3 servers

PH&N-3 servers

RBC CM/Investments International:

RBC CM Investments, All	RBC CM Investments International All Shifts.pdf
RBC CM Investments, P	RBC CM Investments International Shift 1.pdf
RBC CM Investments, N	RBC CM Investments International Shift 2.pdf

Platform:4

RBC US Wealth Management-19 Servers-Max 760

RBC CM Asia-Pacific-4 servsers-Max 160

RBC CM London-16 Servers-Max 400

RBC CM NY -21 Servers-Max 525

RBC CM/Investments :

RBC CM Investments, All	RBC CM Investments International All Shifts.pdf
RBC CM Investments, P	RBC CM Investments International Shift 1.pdf
RBC CM Investments, N	RBC CM Investments International Shift 2.pdf

Platform:3

RBC US Remote Access – 13 servers – Max 440

RBC CM Branch – 22 servers – Max 570

RBC Investment-Remote Access – 9 servers – Max 225

Others

Platform:all

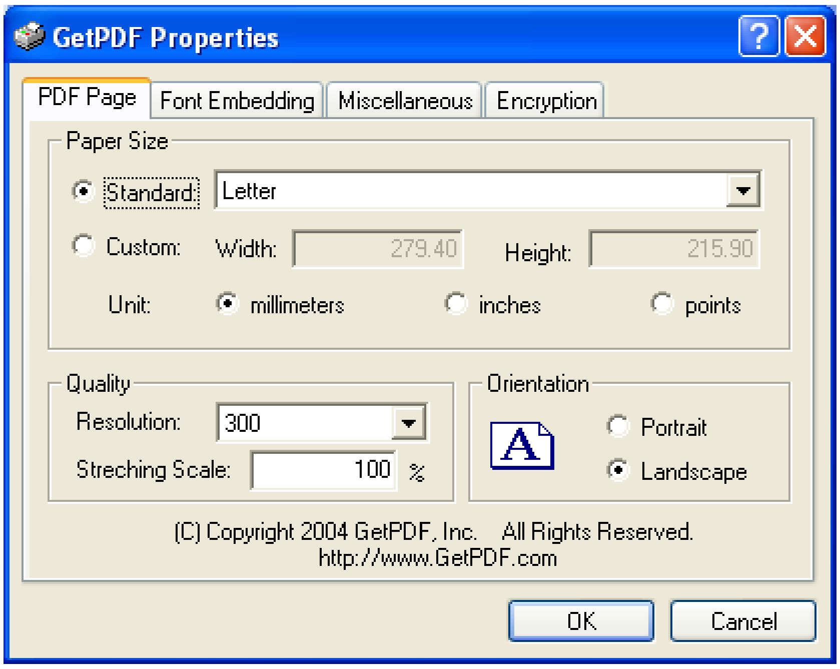
RBC Insurance BPO, P	RBC BPO Morning Shifts.pdf
RBC Insurance BPO, N	RBC BPO Night Shifts.pdf
RBC CM/Investments, P	RBC CM Investments Shift 1.pdf
RBC CM/Investments, N	RBC CM Investments Shift 2.pdf
RBC CM/Investments, All	RBC CM Investments All Shifts.pdf

TSUserUsageCMChartV1.xls located in \\s3w01998\G$\NTSMF\Excel, print to PDF for RBC CM Investments (Shift 1/P and 2/N, and All shifts)

RBC CM Investments All Shifts.pdf

RBC CM Investments Shift 1.pdf

RBC CM Investments Shift 2.pdf
(When printing to PDF, choose Paper Size Standard to Letter and Orientation to Landscape)


GetPDF Properties

Paper Size
[✓ Standard:] Letter
Custom: Width: 279.40 Height: 215.90
Unit: [✓ millimeters] inches points

Quality
Resolution: 300
Streching Scale: 100 %

Orientation
Portrait [✓ Landscape]

(C) Copyright 2004 GetPDF, Inc. All Rights Reserved.
http://www.GetPDF.com

Publish User Usage Report every Monday morning.

After creating the PDF files, put them up on the SharePoint server at this location.
http://rbcportal.fg.rbc.com/ykl0/cas/F12_Terminal%20Server/Server%20Based%20Computing%20Group/Stats/Forms/AllItems.aspx?RootFolder=%2Fykl0%2Fcas%2FF12%5FTerminal%20Server%2FServer%20Based%20Computing%20Group%2FStats%2FWeekly%20and%20Daily%20User%20Stats%2FYear%202009

(http://rbcportal.fg.rbc.com/ykl0/cas/F12_Terminal%20Server/Server%20Based%20Computing%20Group/Stats/Forms/AllItems.aspx?RootFolder=%2fykl0%2fcas%2fF12%5fTerminal%20Server%2fServer%20Based%20Computing%20Group%2fStats%2fWeekly%20and%20Daily%20User%20Stats%2fYear%202011&FolderCTID=&View=%7b139A2B74%2d8BFC%2d4C45%2dB9F9%2d5E8C2F8C6D8B%7d)

Make a folder specifying today’s date (or if you happen to be doing this a bit later than Monday, specify the date that the report should have been completed to be consistent with previous folders. This will cause less confusion.) And post all the PDF files into it.

2.2 Working Systems

The local workstation operating system is Windows XP. The remote server operating system is

Windows Server 2003. The SharePoint Server is Windows SharePoint Services 3.0. The Weekly User Usage Report is PDF document.

2.3 User Interfacing

Because the current process is manual, there is no special user interface for the current Weekly User Usage Report. All the user interfaces are common Application user interface. For example, when operate in the NTSMF web site, the user interface is browser’s common style; when operate in the SharePoint, the user interface is the SharePoint website style; when produce the Weekly User Usage Report in local workstation, the user interface is Excel style.

3 ANALYSIS OF CURRENT PROBLEMS

3.1 Procedures

The current procedures problems are the following:

1 The procedures are all manual work. It occupies a lot of human resource. It takes a person two hours to produce and transfer the Weekly User Usage Report every day.

2 The procedures are very complex. It will take a long time to train a new staff to do the procedures.

3 It is easy to make mistakes because all the procedures are manual work and complex.

3.2 Working Systems

There are too many applications in the working systems for producing and transferring Weekly User Usage Reports. It includes browser, Excel and SharePoint.

3.3 User Interfacing

There are too many user interfaces in the working systems. There are 3 different user interface styles. It includes browser, Excel and SharePoint.

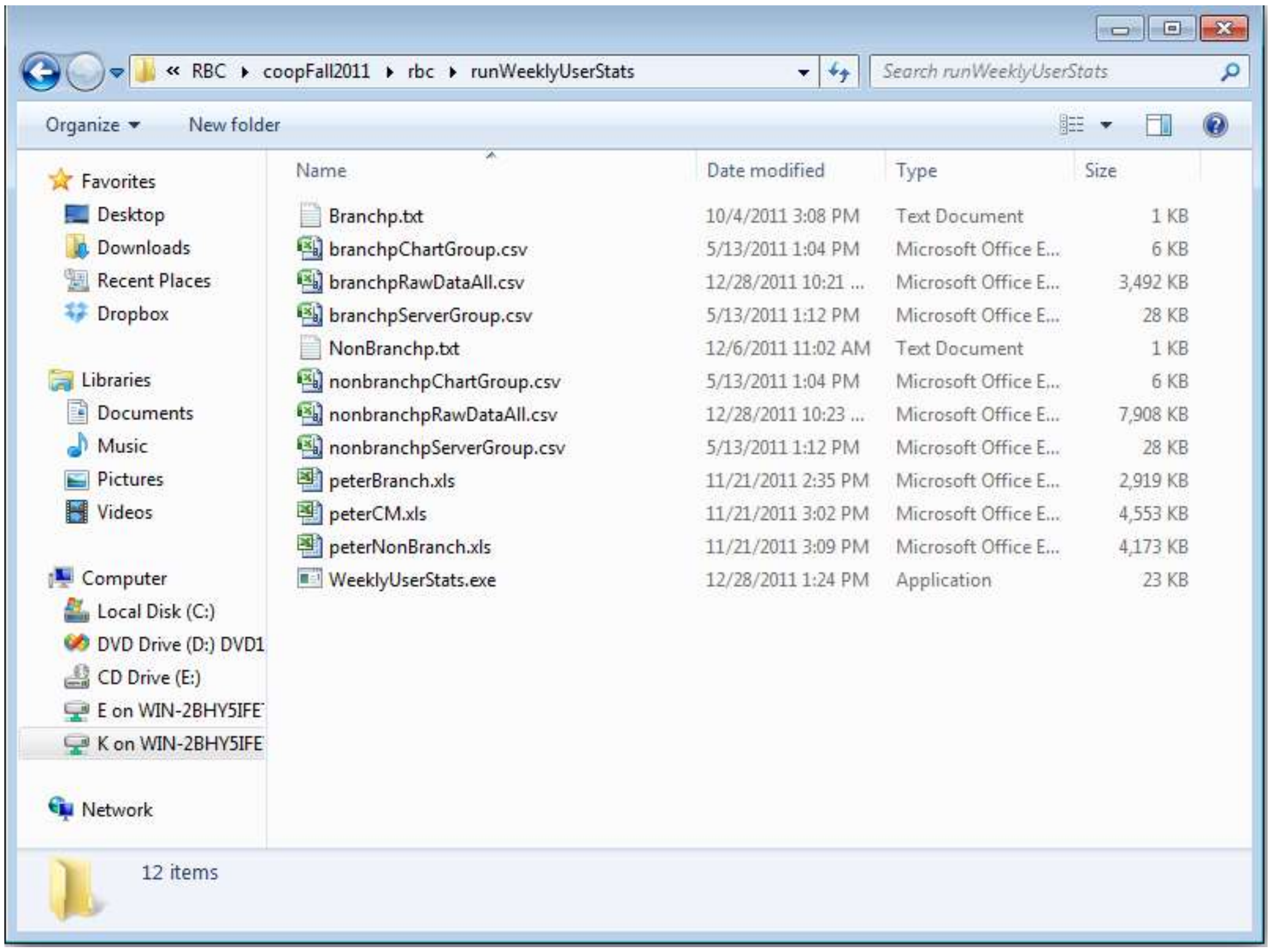
4 ANALYSIS OF ALTERNATIVE SOLUTIONS

4.1 Procedures

The alternative solution is developing the program to produce and transfer the Weekly User Usage Reports. The procedures are very simple. If the program is installed, all the procedures can be done automatically, and you only need double click the program.

So the following only introduce the installation procedures:

1 Create dir "c:\runWeeklyUserUsageReport", copy the following files to this directory:
Branchp.txt, branchpChartGroup.csv, branchpRawDataAll.csv, branchpServerGroup.csv, NonBranchp.txt, nonbranchpChartGroup.csv, nonbranchpRawDataAll.csv, peterBranch.xls, peterCM.xls, peterNonBranch.xls, WeeklyUserStats.exe



Name	Date modified	Type	Size
Branchp.txt	10/4/2011 3:08 PM	Text Document	1 KB
branchpChartGroup.csv	5/13/2011 1:04 PM	Microsoft Office E...	6 KB
branchpRawDataAll.csv	12/28/2011 10:21 ...	Microsoft Office E...	3,492 KB
branchpServerGroup.csv	5/13/2011 1:12 PM	Microsoft Office E...	28 KB
NonBranchp.txt	12/6/2011 11:02 AM	Text Document	1 KB
nonbranchpChartGroup.csv	5/13/2011 1:04 PM	Microsoft Office E...	6 KB
nonbranchpRawDataAll.csv	12/28/2011 10:23 ...	Microsoft Office E...	7,908 KB
nonbranchpServerGroup.csv	5/13/2011 1:12 PM	Microsoft Office E...	28 KB
peterBranch.xls	11/21/2011 2:35 PM	Microsoft Office E...	2,919 KB
peterCM.xls	11/21/2011 3:02 PM	Microsoft Office E...	4,553 KB
peterNonBranch.xls	11/21/2011 3:09 PM	Microsoft Office E...	4,173 KB
WeeklyUserStats.exe	12/28/2011 1:24 PM	Application	23 KB

2 Double click WeeklyUserStats.exe:

4.2 Development Environment

Language: C#

Platform: Visual Studo Team 2008 SP1,

Source Code: Program.cs, Helper.cs

4.3 Working Systems

The working systems are the same like current systems.

The local workstation operating system is Windows XP. The remote server operating system is Windows Server 2003. The SharePoint Server is Windows SharePoint Services 3.0. The Weekly User Usage Report is PDF document.

4.4 User Interfacing

The user interface is dos command. It is only used for monitoring the whole process. It needs no user input and any manual work.

5 ANALYSIS OF RECOMMENDED SOLUTIONS

The recommended solution is to develop program to produce and transfer Weekly User Usage Report.

This solution has the following advantages:

1 It can save human resource. Once it is installed, there is no additional manual work to do. Comparing the current solution, it will save two hours per week.

2 It is very quick. It only takes seven minutes to complete the whole procedures. Comparing the current solution, it takes two hours to do the same work.

3 It is very simple. Except the installation, all the procedures can be done automatically. It only

needs to train how to install the program in the first time, and no training for weekly work.

4 It is more accurate. All the procedures are done by computer, no manual work, so there is no manual mistake in the procedures.

Source Code List:

1 Pogram.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO;
using System.Web;
using System.Net;
using Helper;
using System.Text.RegularExpressions;
using System.Threading;

namespace WeeklyUserStats
{
    class Program
    {
        //project diretory
        const String projDir = @"C:\WeeklyUserStats";
        //const                    string                    baseUrl                    =
"http://rbcportal.fg.rbc.com/ykl0/cas/F12_Terminal%20Server/Server%20Based%20Computing%20Group/Stats/t
est/Weekly and Daily User Stats";
        const                    string                    baseUrl                    =
"http://rbcportal.fg.rbc.com/ykl0/cas/F12_Terminal%20Server/Server%20Based%20Computing%20Group/Stats/
Weekly and Daily User Stats";
        const String outfilep = "_submitReport";
        const                    String                    prefixSubmitUrl1                    =
"http://s3w01694/scripts/rsassubm.pl?repname=MCPUMemTS3&MXG=YES&ex2k=2000&db=TermServer&ts=Det
ail&runwhen=Now&times=1&wnts=";
        const String prefixSubmitUrt2 = "&wdt=";
        const String prefixDownUrl1 = @"http://s3w01694/reports/s7mv9433\MCPUMemTS3/";
        const String prefixDownUrl2 = @"/MCPUMemTS3_E2K.csv";

        static void Main(string[] args)
        {

            string result = "";
            string localDir = "";
            string dateDir;
            dateDir = DateTime.Now.ToString("MMM dd");
```

```
try
{
    // Create an instance of StreamReader to read from a file.
    // The using statement also closes the StreamReader.

    String branchp;

    Console.WriteLine("Start creating Weekly User Usage Report...");

    branchp = "Branchp";
    Console.WriteLine("    copying Branch Report files...");
    result = submitReportTxt(branchp);
    if (result != "")
    {
        Console.WriteLine(result);
    }
    else
    {
        Console.WriteLine("    Branch Report files copied");
    }

    branchp = "NonBranchp";
    Console.WriteLine("    copying Non Branch Report files...");
    result = submitReportTxt(branchp);
    if (result != "")
    {
        Console.WriteLine(result);
    }
    else
    {
        Console.WriteLine("    Non Branch Report files copied");
    }

    Console.WriteLine("    All Weekly User Usage Report files copied.\n");


    ///// ////////////////////////////////////////////////
    // Get pivot data
    string oFile = "";
    string outputFile = "";
    string currentPath = Directory.GetCurrentDirectory();
    string inputPath = "";
    string[] inputfilePaths;
    string output = "";
```

```
        Console.WriteLine("Getting chart data...");

        output = "SYSTEM*NAME " + "," +
            "Date " + "," +
            "ACTIVE*SESSIONS " + "," +
            "SHIFT*OF*START ";

        inputPath = currentPath + @"\Branchp\";
        inputfilePaths = Directory.GetFiles(inputPath, "*.csv");
        oFile = "branchpRawDataAll.csv";
        outputFile = currentPath + @"\" + oFile;
        Helper.PivotUtil.getLastWeekPivotData(outputFile, inputfilePaths, output);

        inputPath = currentPath + @"\NonBranchp\";
        inputfilePaths = Directory.GetFiles(inputPath, "*.csv");
        oFile = "nonbranchpRawDataAll.csv";
        outputFile = currentPath + @"\" + oFile;
        Helper.PivotUtil.getLastWeekPivotData(outputFile, inputfilePaths, output);

        Console.WriteLine("Finished chart data");
        ////////////////////////////////////////////////////
    }
    catch (Exception e)
    {
        // Let the user know waht went wrong.
        Console.WriteLine("The file could not be read:");
        Console.WriteLine(e.Message);
        result = e.Message;
    }

    // Run the macros.
    string branchpExcel = "";
    string macro = "";
    string[] copyFiles ={
                        "branchpChartGroup.csv",
                        "branchpServerGroup.csv",
                        "branchpRawDataAll.csv",
                        "peterBranch.xls",
                        "nonBranchpChartGroup.csv",
                        "nonBranchpServerGroup.csv",
                        "nonbranchpRawDataAll.csv",
                        "peterNonBranch.xls",
                        "peterCM.xls"
                        };

DirectoryInfo dir;
string[] newDirs = {
                            projDir,
                            projDir+@"\reportBranch",
                            projDir+@"\reportNonBranch"
                        };
try
{
    foreach (string newDir in newDirs)
    {
        //newDir = projDir;
        dir = new DirectoryInfo(newDir);
        if (Directory.Exists(newDir))
        {
            dir.Delete(true);
        }
        dir = Directory.CreateDirectory(newDir);
    }
}
catch (Exception e)
{
    Console.WriteLine("The process failed: {0}", e.ToString());
}


//To copy a file to another location and
//overwrite the destination file if it already exists.
foreach (var copyFile in copyFiles)
{
    File.Copy(copyFile, projDir + @"\" + copyFile, true);
}

//branchpExcel = projDir + @"\" + "peterBranch.xls";
//macro = "BranchAll";
//Helper.ExcelUtil.runExcelMacro(branchpExcel, macro);

//branchpExcel = projDir + @"\" + "peterNonBranch.xls";
//macro = "nonBranchAll";
//Helper.ExcelUtil.runExcelMacro(branchpExcel, macro);

//branchpExcel = projDir + @"\" + "peterCM";
//macro = "CMAll";
```

```
        //Helper.ExcelUtil.runExcelMacro(branchpExcel, macro);

        Dictionary<string, string> macroDic = new Dictionary<string, string>();
        macroDic.Add(projDir + @"\" + "peterBranch.xls", "BranchAll");
        macroDic.Add(projDir + @"\" + "peterNonBranch.xls", "nonBranchAll");
        macroDic.Add(projDir + @"\" + "peterCM", "CMAll");
        Helper.ExcelUtil.runExcelMacro(macroDic);



        ////////////////////////////////////////////////////////////////////////////
        //// upload to sharepoint
        ////////////////////////////////////////////////////////////////////////////
        string upUrl = Helper.WebUtil.createUploadDir(baseUrl);
        //string localDir;
        localDir = projDir + @"\reportBranch";
        Helper.WebUtil.uploadDir(upUrl, localDir, "*.pdf");
        localDir = projDir + @"\reportNonBranch";
        Helper.WebUtil.uploadDir(upUrl, localDir, "*.pdf");


    }

    static string HttpGet(string uri)
    {
        Stream stream;
        StreamReader reader;
        string response = null;
        WebClient webClient = new WebClient();

        using (webClient)
        {
            try
            {
                // open and read from the supplied URI
                stream = webClient.OpenRead(uri);
                reader = new StreamReader(stream);
                response = reader.ReadToEnd();
            }
            catch (WebException ex)
            {
                if (ex.Response is HttpWebResponse)
                {
                    switch (((HttpWebResponse)ex.Response).StatusCode)
                    {
                        case HttpStatusCode.NotFound:
                            response = "Not Found";
                            break;
                        default:
                            throw ex;
                    }
                }
            }
        }
        return response;
    }


    public static string submitReportTxt(String branchp)
    {
        String fileBranchp;
        fileBranchp = branchp + ".txt";
        String line = "";
        String myUrl = "";
        string submitResponse = "";
        string result = "";
        string jobid = "";
        string path = @".\" + branchp;
        string downUrl = "";
        string fileDown = "";
        int wdt = 0;
        DateTime lastMonday;
        DateTime lastFriday;

        Helper.DateUtil.GetLastWeekWorkDates(out lastMonday, out lastFriday);
        wdt = DateTime.Now.Subtract(lastMonday).Days + 1;

        // Delete old files
        if (Directory.Exists(path))
        Directory.Delete(path, true);
        // Added on 2011/12/06

        using (StreamReader sr = new StreamReader(fileBranchp))
        {
            // Read and display lines from the file until the end of
            // the file is reached.
            while ((line = sr.ReadLine()) != null)
            {
                    myUrl = prefixSubmitUrl1 + HttpUtility.UrlEncode(line) + prefixSubmitUrt2 +
wdt.ToString();
                    submitResponse = HttpGet(myUrl);
                    if (!submitResponse.Contains("1 file(s) copied."))
                    {
                        result += "error in " + fileBranchp;
                    }
                    else
                    {
                        RegexUtil.regex = new Regex(@"&jobid=([A-Za-z0-9_\[]+)&lvl=");
                        jobid = RegexUtil.MatchKey(submitResponse);
                        // Determing whether the directory exists.
                        if (!Directory.Exists(path))
                        {
                            DirectoryInfo di = Directory.CreateDirectory(path);
                        }
                        downUrl = prefixDownUrl1 + jobid + prefixDownUrl2;
                        fileDown = path + @"\MCPUMemTS3_E2K_" + line + ".csv";
                        //
                        //while (!Helper.WebUtil.isRemoteFileExists(downUrl))
                        //{
                        //      ////
                        //      Thread.Sleep(30000);
                        //      ////Added on 2011/12/12 to solve cannot find file.
                        //}
                        Thread.Sleep(200000);
                        //
                        Helper.WebUtil.lDownload(downUrl, fileDown);
                        Console.WriteLine("          " + line + " copied;");
                    }
                }
                return result;
            }
        }
    }
}
```

2 Helper.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Text.RegularExpressions;
using System.Net;
using System.IO;
using Excel = Microsoft.Office.Interop.Excel;
using System.Diagnostics;
using System.ComponentModel;
using System.Runtime.InteropServices;
using System.Threading;

namespace Helper
{
    //public static class DateUtil
    public class DateUtil
    {
        public static void GetLastWeekWorkDates(out DateTime stDate, out DateTime endDate)
        {
            double offset = 0;
            switch (DateTime.Today.DayOfWeek)
            {
                case DayOfWeek.Monday:
                    offset = 0;
                    break;
                case DayOfWeek.Tuesday:
                    offset = -1;
                    break;
                case DayOfWeek.Wednesday:
                    offset = -2;
                    break;
                case DayOfWeek.Thursday:
                    offset = -3;
                    break;
                case DayOfWeek.Friday:
                    offset = -4;
                    break;
                case DayOfWeek.Saturday:
                    offset = -5;
                    break;
                case DayOfWeek.Sunday:
```

```
                offset = -6;
                break;
        }
        int year, month, day;
        stDate = DateTime.Today.AddDays(-7 + offset);
        year = stDate.Year;
        month = stDate.Month;
        day = stDate.Day;
        stDate = new DateTime(year, month, day);
        endDate = DateTime.Today.AddDays(-2 + offset);
        year = endDate.Year;
        month = endDate.Month;
        day = endDate.Day;
        endDate = new DateTime(year, month, day);


}

public static string GetMonthName(DateTime dt, bool abbrev)
{
        if (abbrev) return dt.ToString("MMM");
        return dt.ToString("MMMM");
}

public static string pivotDate(DateTime dt)
{
        string pd = String.Format("{0:ddMMMyy}", dt);
        pd = pd.ToUpper();
        return pd;
}

public static int GetMonthNumber(string shortMonth)
{
        int mn = 0;
        switch (shortMonth)
        {
                case "JAN":
                        mn = 1;
                        break;
                case "FEB":
                        mn = 2;
                        break;
                case "MAR":
                        mn = 3;
```

```
                    break;
                case "APR":
                    mn = 4;
                    break;
                case "MAY":
                    mn = 5;
                    break;
                case "JUN":
                    mn = 6;
                    break;
                case "JUL":
                    mn = 7;
                    break;
                case "AUG":
                    mn = 8;
                    break;
                case "SEP":
                    mn = 9;
                    break;
                case "OCT":
                    mn = 10;
                    break;
                case "NOV":
                    mn = 11;
                    break;
                case "DEC":
                    mn = 12;
                    break;
                default:
                    throw new System.InvalidOperationException("Invalid month");
            }
            return mn;
        }
    }

    public static class RegexUtil
    {

        static public Regex regex { get; set; }

        static public string MatchKey(string input)
        {
            Match match = regex.Match(input);
            if (match.Success)
            {
                return match.Groups[1].Value;
            }
            else
            {
                return null;
            }
        }

    }

    public class WebUtil
    {
        static string HttpGet(string uri)
        {
            Stream stream;
            StreamReader reader;
            string response = null;
            WebClient webClient = new WebClient();

            using (webClient)
            {
                try
                {
                    // open and read from the supplied URI
                    stream = webClient.OpenRead(uri);
                    reader = new StreamReader(stream);
                    response = reader.ReadToEnd();
                }
                catch (WebException ex)
                {
                    if (ex.Response is HttpWebResponse)
                    {
                        switch (((HttpWebResponse)ex.Response).StatusCode)
                        {
                            case HttpStatusCode.NotFound:
                                response = "Not Found";
                                break;
                            default:
                                throw ex;
                        }
                    }
                }
            }
            return response;
        }

        //public static string submitReportTxt(String branchp)
        //{
        //      String fileBranchp;
        //      fileBranchp = branchp + ".txt";
        //      String line = "";
        //      String myUrl = "";
        //      string submitResponse = "";
        //      string result = "";
        //      string jobid = "";
        //      string path = @".\" + branchp;
        //      string downUrl = "";
        //      string fileDown = "";
        //      int wdt = 0;
        //      DateTime lastMonday;
        //      DateTime lastFriday;

        //      Helper.DateUtil.GetLastWeekWorkDates(out lastMonday, out lastFriday);
        //      wdt = DateTime.Now.Subtract(lastMonday).Days + 1;

        //      using (StreamReader sr = new StreamReader(fileBranchp))
        //      {
        //          // Read and display lines from the file until the end of
        //          // the file is reached.
        //          while ((line = sr.ReadLine()) != null)
        //          {
        //                  myUrl = prefixSubmitUrl1 + HttpUtility.UrlEncode(line) + prefixSubmitUrt2 +
wdt.ToString();
        //              submitResponse = HttpGet(myUrl);
        //              if (!submitResponse.Contains("1 file(s) copied."))
        //              {
        //                  result += "error in " + fileBranchp;
        //              }
        //              else
        //              {
        //                  RegexUtil.regex = new Regex(@"&jobid=([A-Za-z0-9_\[]+)&lvl=");
        //                  jobid = RegexUtil.MatchKey(submitResponse);
        //                  // Determing whether the directory exists.
        //                  if (!Directory.Exists(path))
        //                  {
        //                      DirectoryInfo di = Directory.CreateDirectory(path);
        //                  }
        //                          downUrl = prefixDownUrl1 + jobid + prefixDownUrl2;
        //                          fileDown = path + @"\MCPUMemTS3_E2K_" + line + ".csv";
        //                          Helper.WebUtil.lDownload(downUrl, fileDown);
        //                          Console.WriteLine("          " + line + " copied;");
        //                     }
        //               }
        //               return result;
        //         }
        //}

        public static string createUploadDir(string baseUrl)
        {
            string newfolderUrl = "";
            string yearUrl = "";
            DateTime lastMonday;
            DateTime lastSaturday;
            DateTime lastFriday;
            Helper.DateUtil.GetLastWeekWorkDates(out lastMonday, out lastSaturday);
            lastFriday = lastSaturday.AddDays(-1);
            yearUrl = baseUrl + @"/Year " + lastMonday.ToString("yyyy");
            Helper.WebUtil.CreateFolder(yearUrl);
            if (lastMonday.Month == lastFriday.Month)
            {
                newfolderUrl   =   yearUrl   +   @"/"   +   lastMonday.ToString("MMM   dd")   +   "   -   "   +
lastFriday.ToString("dd");
            }
            else
            {
                newfolderUrl   =   yearUrl   +   @"/"   +   lastMonday.ToString("MMM   dd")   +   "   -   "   +
lastFriday.ToString("MMMdd");
            }

            Helper.WebUtil.CreateFolder(newfolderUrl);
            return newfolderUrl;
        }


        public static void wssCreatFolder(string wssParentDir, string wssSubDir)
        {
            string wssNewFolder = wssParentDir + wssSubDir;
            HttpWebRequest request = (System.Net.HttpWebRequest)HttpWebRequest.Create(wssNewFolder);
            request.Credentials = CredentialCache.DefaultCredentials;
            request.Method = "MKCOL";
            HttpWebResponse response = (System.Net.HttpWebResponse)request.GetResponse();
        response.Close();
}

static public bool Download(string downUrl, string fileDown)
{
        HttpWebRequest webRequest;
        HttpWebResponse webResponse;
        Stream strResponse;
        FileStream strLocal;

        using (WebClient wcDownload = new WebClient())
        {
                try
                {
                        // Create a request to the file we are downloading
                        webRequest = (HttpWebRequest)WebRequest.Create(downUrl);
                        // Retrieve the response from the server
                        webResponse = (HttpWebResponse)webRequest.GetResponse();
                        // Ask the server for the file size and store it
                        Int64 fileSize = webResponse.ContentLength;
                        // Open the URL for download
                        //wcDownload.DownloadFile(downUrl, fileDown);
                        strResponse = wcDownload.OpenRead(downUrl);


                        // Create a new file stream where we will be saving the data (local drive)
                        strLocal = new FileStream(fileDown, FileMode.Create, FileAccess.Write, FileShare.None);
                        // It will store the current number of bytes we retrieved from the server
                        int bytesSize = 0;
                        // A buffer for storing and writing the data retrieved from the sever
                        byte[] downBuffer = new byte[16384];
                        // Loop through the buffer until the buffer is empty
                        while ((bytesSize = strResponse.Read(downBuffer, 0, downBuffer.Length)) > 0)
                        {
                                // Write the data from the buffer to the local hard drive
                                strLocal.Write(downBuffer, 0, bytesSize);
                        }
                        // Check file size
                        FileInfo f = new FileInfo(fileDown);
                        long downFileSize = f.Length;
                        if (downFileSize < fileSize)
                        {
                                return false;
                        }
                else
                {
                    return true;
                }
            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);
                return false;
            }
        }
    }

    static public bool lDownload(string downUrl, string outputFilePath)
    {
        bool bResult;
        const int BUFFER_SIZE = 16 * 1024;
        using (var outputFileStream = File.Create(outputFilePath, BUFFER_SIZE))
        {
            try
            {
                //var req = WebRequest.Create(downUrl);
                HttpWebRequest req = WebRequest.Create(downUrl) as HttpWebRequest;
                req.KeepAlive = false;
                //req.Timeout = 5000;
                //req.Proxy = null;
                //req.ServicePoint.ConnectionLeaseTimeout = 5000;
                //req.ServicePoint.MaxIdleTime = 5000;

                ////
                //Thread.Sleep(30000);
                req.Timeout = 30000;


                ////Added on 2011/12/12 to solve cannot find file.
                try
                {
                    //using (WebResponse response = req.GetResponse())
                    using (var response = req.GetResponse())
                    {
                        //while (response.ContentLength < 1000000)
                        //{
                        //      Thread.Sleep(30000);
                        //}
```

```
                                using (var responseStream = response.GetResponseStream())
                                {
                                    var buffer = new byte[BUFFER_SIZE];
                                    int bytesRead;
                                    do
                                    {
                                        bytesRead = responseStream.Read(buffer, 0, BUFFER_SIZE);
                                        outputFileStream.Write(buffer, 0, bytesRead);
                                    } while (bytesRead > 0);
                                    responseStream.Flush();
                                    responseStream.Close();
                                }
                                response.Close();
                                //req.Abort();
                            }
                            //return true;
                            bResult = true;
                        }
                        catch (WebException ex)
                        {
                            Console.WriteLine(ex.Message);
                            //return false;
                            bResult = false;
                        }
                        finally
                        {
                            req.Abort();
                            //req = null;
                            //GC.Collect();
                        }
                        return bResult;
                    }
                    catch (Exception e)
                    {
                        Console.WriteLine(e.Message);
                        //return false;
                        bResult = false;
                        return bResult;
                    }
                }

            }
```

```
public static string uploadDir(string upUrl, string LocalDir, string extension)
{
        string upFileUrl;
        string returnString;

        //if (isRemoteFileExists(upUrl))
        //{
        //        returnString = "Files was uploaded before";
        //        Console.WriteLine(returnString);
        //        return returnString;
        //}

        //if (!Directory.Exists(LocalDir))
        //{
        //        returnString = "Files was uploaded before";
        //        Console.WriteLine(returnString);
        //        return returnString;
        //}

        Console.WriteLine(LocalDir + " uploading...");
        using (WebClient client = new WebClient())
        {
                try
                {
                        client.Credentials = CredentialCache.DefaultCredentials;
                        string[] filePaths = Directory.GetFiles(LocalDir, extension);
                        foreach (string filePath in filePaths)
                        {
                                upFileUrl = upUrl + @"/" + Path.GetFileName(filePath);
                                //if (isRemoteFileExists(upFileUrl))
                                //{
                                //        returnString = "File was uploaded before";
                                //        Console.WriteLine(returnString);
                                //        return returnString;
                                //}
                                if (isRemoteFileExists(upFileUrl))
                                {
                                        Console.WriteLine(Path.GetFileName(filePath) + " is already exist");
                                }
                                else
                                {
                                        //while (!isRemoteFileExists(upFileUrl))
                                        do
                                        {
```

```
                                        try
                                        {
                                            client.UploadFile(upFileUrl, "PUT", filePath);
                                        }
                                        catch (Exception ex)
                                        {

                                            //Console.WriteLine(ex.Message);
                                            Thread.Sleep(10000);
                                        }
                                    } while (!isRemoteFileExists(upFileUrl));

                                }
                            }
                            returnString = LocalDir + " uploaded";
                            Console.WriteLine(returnString);
                            return returnString;
                        }
                        catch (Exception ex)
                        {
                            returnString = ex.Message;
                            Console.WriteLine(returnString);
                            return returnString;
                        }
                    }
                }

                public static bool isRemoteFileExists(string url)
                {
                    try
                    {
                        HttpWebRequest request = WebRequest.Create(url) as HttpWebRequest;
                        request.Credentials = CredentialCache.DefaultCredentials;
                        request.Method = "HEAD";
                        HttpWebResponse response = request.GetResponse() as HttpWebResponse;
                        return (response.StatusCode == HttpStatusCode.OK);
                    }
                    catch (WebException ex)
                    {
                        //LogMessage = ex.Message + " \r\n";
                        //Console.WriteLine(ex.Message);
                        return false;
                    }
                    catch (Exception ex)
```

```
            {
                //LogMessage = ex.Message + " \r\n";
                //Console.WriteLine(ex.Message);
                return false;
            }
        }

        public static bool Move(string oldUrl, string newUrl)
        {
            string result = "";
            string webUrl = GetWebURL(oldUrl);
            oldUrl = oldUrl.Substring(webUrl.Length + 1);
            newUrl = newUrl.Substring(webUrl.Length + 1);
            return Move(webUrl, oldUrl, newUrl, out result);
        }

        public static bool Move(string webUrl, string oldUrl, string newUrl, out string result)
        {
            EnsureFolders(webUrl, newUrl);
            string renameOption = "findbacklinks";
            string putOption = "overwrite,createdir,migrationsemantics";
            bool doCopy = false;
            string method =
"method=move+document%3a12.0.0.6545&service_name=%2f&oldUrl={0}&newUrl={1}&url_list=[]&rename_opt
ion={2}&put_option={3}&docopy={4}";
            method = String.Format(method, oldUrl, newUrl, renameOption, putOption,
doCopy.ToString().ToLower());
            try
            {
                using (WebClient webClient = new WebClient())
                {
                    webClient.Credentials = CredentialCache.DefaultCredentials;
                    webClient.Headers.Add(
                        "Content-Type",
                        "application/x-vermeer-urlencoded");
                    webClient.Headers.Add(
                        "X-Vermeer-Content-Type",
                        "application/x-vermeer-urlencoded");
                    result = Encoding.UTF8.GetString(
                        webClient.UploadData(
                        webUrl + "/_vti_bin/_vti_aut/author.dll",
                        "POST",
                        Encoding.UTF8.GetBytes(method)));
                    if (result.IndexOf("\n<p>message=successfully") < 0)
```

```
                        throw new Exception(result);
                    }
                }
                catch (Exception ex)
                {
                    result = ex.Message;
                    return false;
                }
                return true;
            }

            public static void EnsureFolders(string rootUrl, string folderUrl)
            {
                StringBuilder sb = new StringBuilder(rootUrl.TrimEnd('/'));
                string[] segments = folderUrl.Split('/');
                for (int i = 0; i < segments.Length - 1; i++)
                {
                    sb.Append("/");
                    sb.Append(segments[i]);
                    CreateFolder(sb.ToString());
                }
            }

            public static bool CreateFolder(string folderURL)
            {
                // Check if folderUrl exist
                if (isRemoteFileExists(folderURL))
                {
                    return false;
                }

                try
                {
                    WebRequest request = WebRequest.Create(folderURL);
                    request.Credentials = CredentialCache.DefaultCredentials;
                    request.Method = "MKCOL";
                    WebResponse response = request.GetResponse();
                    response.Close();
                    return true;
                }
                catch (WebException ex)
                {
                    Console.WriteLine(ex.Message);
                    return false;
```

```
        }
    }


    public static string GetWebURL(string url)
    {
        try
        {
            url = url.Substring(0, url.LastIndexOf("/"));
            try
            {
                using (WebClient webClient = new WebClient())
                {
                    webClient.Credentials = CredentialCache.DefaultCredentials;
                    webClient.Headers.Add(
                        "Content-Type",
                        "application/x-vermeer-urlencoded");
                    webClient.Headers.Add(
                        "X-Vermeer-Content-Type",
                        "application/x-vermeer-urlencoded");
                    byte[] data = Encoding.UTF8.GetBytes(
                        "method=open+service%3a12.0.0.6545&service_name=%2f");
                    string result = Encoding.UTF8.GetString(
                        webClient.UploadData(url + "/_vti_bin/_vti_aut/author.dll", "POST", data));
                    if (result.IndexOf("\n<li>status=327684") == -1)
                        return url;
                    throw new Exception();
                }
            }
            catch
            {
                return GetWebURL(url);
            }
        }
        catch
        {
            return null;
        }
    }

}

public class TextUtil
{
        static public bool selectDate(DateTime dt)
        {
            return false;
        }

        static public string FindInFile(string inputFile, string spattern)
        {
            string input = "";
            string output = "";
            //Regex pattern = new Regex(@"spattern");
            //MatchCollection matches; // = pattern.Matches("This hopefully will pick up 1Bob9error1 as a 
name");
            //Console.WriteLine(matches[0].Groups["name"]);
            using (StreamReader sr = File.OpenText(inputFile))
            {
                while ((input = sr.ReadLine()) != null)
                {
                    if (input.Contains(spattern))
                    {
                        output = spattern;
                        return output;
                    }

                }
            }
            return output;
        }
    }

    public class PivotUtil
    {
                public static void getLastWeekPivotData(string outputFile, string[] inputfilePaths, string 
output)
        {
            //Console.WriteLine("\nGetting NonBranch pivot data...");

            if (File.Exists(outputFile))
            {
                try
                {
                    File.Delete(outputFile);
                }
                catch (IOException e)
                {
```

```
                Console.WriteLine(e.Message);
            }
        }
        using (StreamWriter sw = File.CreateText(outputFile))
        {

            //output = "SYSTEM*NAME " + "," +
            //      "Date " + "," +
            //      "ACTIVE*SESSIONS " + "," +
            //      "SHIFT*OF*START ";
            sw.WriteLine(output);
        }

        foreach (var filePath in inputfilePaths)
        {
            //Console.WriteLine("    getting pivot data from " + filePath);
            //inputFile = filePath;
            Helper.PivotUtil.getPivotData(filePath, outputFile);
        }
        //Console.WriteLine("Finished NonBranch pivot data");
    }

    public static void getPivotData(string inputFile, string outputFile)
    {
        if (!File.Exists(inputFile))
        {
            Console.WriteLine("{0} does not exist.", inputFile);
            return;
        }
        //string output = "";
        //if (File.Exists(outputFile))
        //{
        //      try
        //      {
        //          File.Delete(outputFile);
        //      }
        //      catch (IOException e)
        //      {
        //          Console.WriteLine(e.Message);
        //      }
        //}
        //using (StreamWriter sw = File.CreateText(outputFile))
        //{
        //      output = "SYSTEM*NAME " + "," +
            //              "Date " + "," +
            //              "ACTIVE*SESSIONS " + "," +
            //              "SHIFT*OF*START ";
            //      sw.WriteLine(output);
            //}

            using (StreamWriter sw = File.AppendText(outputFile))
            {
                using (StreamReader sr = File.OpenText(inputFile))
                {
                    string input = "";
                    //string output = "";
                    //string                                    pattern                                    =
@"^\w+\s,(\w+\s),[0-3][0-9][A-Z]{3}201[1-9]:\d{2}:\d{2}:\d{2}.\d{3}.+TermServer  ,([0123]\d[A-Z]{3}[1][1-9]\s).+[0-
3][0-9][A-Z]{3}201[1-9]\s,(\d+\s),\d+\s,\d+\s,([A-Z])$";
                    string                                    pattern                                    =
@"\w+\s,(\w+\s),[0-3][0-9][A-Z]{3}201[1-9]:\d{2}:\d{2}:\d{2}.\d{3}.+TermServer  ,([0123]\d[A-Z]{3}[1][1-9]\s).+[0-3
][0-9][A-Z]{3}201[1-9]\s,(\d+\s),\d+\s,\d+\s,([A-Z])";
                    int year = 0;
                    int month = 0;
                    int day = 0;
                    DateTime dt;
                    DateTime lastMonday;
                    DateTime lastSatuday;
                    Helper.DateUtil.GetLastWeekWorkDates(out lastMonday, out lastSatuday);
                    //TextWriter tw = new StreamWriter(outputFile);
                    string output = "";

                    while ((input = sr.ReadLine()) != null)
                    {
                        //if (i == 0)
                        //{
                        //      output = "SYSTEM*NAME " + "," +
                        //              "Date " + "," +
                        //              "ACTIVE*SESSIONS " + "," +
                        //              "SHIFT*OF*START ";

                        //      tw.WriteLine(output);
                        //}
                        //i++;
                        MatchCollection matches = Regex.Matches(input, pattern);
                        if (matches.Count == 0)
                        {
                            //Console.WriteLine("No match");
```

```
                                    }
                                    else
                                    {
                                        foreach (Match match in matches)
                                        {
                                            if (match.Success)
                                            {
                                                //Console.WriteLine(match.Groups[1].Value  +  match.Groups[2].Value
+ match.Groups[3].Value);
                                                output = match.Groups[2].Value;
                                                int.TryParse(output.Substring(0, 2), out day);
                                                month = Helper.DateUtil.GetMonthNumber(output.Substring(2, 3));
                                                int.TryParse(output.Substring(5, 2), out year);
                                                //int.TryParse(match.Groups[1].Value, out day);
                                                //month                                                          =
Helper.DateUtil.GetMonthNumber(match.Groups[2].Value);
                                                //int.TryParse(match.Groups[3].Value, out year);
                                                year += 2000;
                                                //Console.WriteLine(year.ToString()  +  "  "  +  month.ToString()  +  "  "  +
date.ToString());
                                                dt = new DateTime(year, month, day);
                                                //Console.WriteLine(dt.ToShortDateString());
                                                //Console.WriteLine(lastMonday.ToLongDateString());
                                                //Console.WriteLine(lastSatuday.ToLongDateString());
                                                if      (((DateTime.Compare(dt,      lastMonday)      >=      0)      &&
(DateTime.Compare(dt, lastSatuday) < 0)))
                                                {
                                                    //Console.WriteLine(dt.ToLongDateString());
                                                    output = match.Groups[1].Value + "," + match.Groups[2].Value +
"," + match.Groups[3].Value + "," + match.Groups[4].Value;
                                                    //tw.WriteLine(output);
                                                    sw.WriteLine(output);
                                                    //sw.Flush();
                                                }
                                            }
                                        }

                                    }
                                }
                                //tw.Close();
                                sw.Flush();
                            }
                        }
                    }
```

```
    }

    public class ExcelUtil
    {
        public static void runExcelMacroPrint(string branchpExcel, string macro, string printFile)
        {
            object oMissing = System.Reflection.Missing.Value;
            // Create an instance of Microsoft Excel, make it visible
            Excel.ApplicationClass oExcel = new Microsoft.Office.Interop.Excel.ApplicationClass();
            oExcel.Visible = true;
            Excel.Workbooks oBooks = oExcel.Workbooks;
            Excel._Workbook oBook = null;
            oBook = oBooks.Open(branchpExcel, oMissing, oMissing,
                oMissing, oMissing, oMissing, oMissing, oMissing, oMissing,
                oMissing, oMissing, oMissing, oMissing, oMissing, oMissing);
            //RunMacro(oExcel, new Object[] { macro});
            oExcel.GetType().InvokeMember("Run",
                System.Reflection.BindingFlags.Default |
                System.Reflection.BindingFlags.InvokeMethod,
                null, oExcel, new Object[] { macro });

            ///////////////////////////////////////////////////////////////////
            // print
            //string printFile = "";
            try
            {
                oExcel.Charts.PrintOut(1, 1, 1, false, "Microsoft Office Document Image Writer on Ne01:", true,
true, printFile);
                //oExcel.Charts.PrintOut(1, 1, 1, false, "GetPDF on Ne01:", true, true, printFile);
            }
            catch (Exception ex)
            {
                Console.WriteLine(ex.Message);
            }
            ///////////////////////////////////////////////////////////////////

            // Quit Excel and clean up.
            int hWnd = oExcel.Application.Hwnd;
            oBook.Close(false, oMissing, oMissing);
            System.Runtime.InteropServices.Marshal.ReleaseComObject(oBook);
            oBook = null;
            System.Runtime.InteropServices.Marshal.ReleaseComObject(oBooks);
            oBooks = null;
            oExcel.Quit();
            System.Runtime.InteropServices.Marshal.ReleaseComObject(oExcel);
            oExcel = null;
            GC.Collect();
            TryKillProcessByMainWindowHwnd(hWnd);
        }

        public static void runExcelMacro(string branchpExcel, string macro)
        {
            object oMissing = System.Reflection.Missing.Value;
            // Create an instance of Microsoft Excel, make it visible
            Excel.ApplicationClass oExcel = new Microsoft.Office.Interop.Excel.ApplicationClass();
            oExcel.Visible = true;
            Excel.Workbooks oBooks = oExcel.Workbooks;
            Excel._Workbook oBook = null;
            oBook = oBooks.Open(branchpExcel, oMissing, oMissing,
                oMissing, oMissing, oMissing, oMissing, oMissing, oMissing,
                oMissing, oMissing, oMissing, oMissing, oMissing, oMissing);
            //RunMacro(oExcel, new Object[] { macro});
            oExcel.GetType().InvokeMember("Run",
                System.Reflection.BindingFlags.Default |
                System.Reflection.BindingFlags.InvokeMethod,
                null, oExcel, new Object[] { macro });

            // Quit Excel and clean up.
            int hWnd = oExcel.Application.Hwnd;
            oBook.Close(false, oMissing, oMissing);
            System.Runtime.InteropServices.Marshal.ReleaseComObject(oBook);
            oBook = null;
            System.Runtime.InteropServices.Marshal.ReleaseComObject(oBooks);
            oBooks = null;
            oExcel.Quit();
            System.Runtime.InteropServices.Marshal.ReleaseComObject(oExcel);
            oExcel = null;
            GC.Collect();
            TryKillProcessByMainWindowHwnd(hWnd);
        }

        [DllImport("user32.dll")]
        private static extern uint GetWindowThreadProcessId(IntPtr hWnd, out uint lpdwProcessId);

        /// <summary> Tries to find and kill process by hWnd to the main window of the process.</summary>
        /// <param name="hWnd">Handle to the main window of the process.</param>
        /// <returns>True if process was found and killed. False if process was not found by hWnd or if it could
not be killed.</returns>
```

```
        public static bool TryKillProcessByMainWindowHwnd(int hWnd)
        {
            uint processID;
            GetWindowThreadProcessId((IntPtr)hWnd, out processID);
            if (processID == 0) return false;
            try
            {
                Process.GetProcessById((int)processID).Kill();
            }
            catch (ArgumentException)
            {
                return false;
            }
            catch (Win32Exception)
            {
                return false;
            }
            catch (NotSupportedException)
            {
                return false;
            }
            catch (InvalidOperationException)
            {
                return false;
            }
            return true;
        }

        /// <summary> Finds and kills process by hWnd to the main window of the process.</summary>
        /// <param name="hWnd">Handle to the main window of the process.</param>
        /// <exception cref="ArgumentException">
        /// Thrown when process is not found by the hWnd parameter (the process is not running).
        /// The identifier of the process might be expired.
        /// </exception>
        /// <exception cref="Win32Exception">See Process.Kill() exceptions documentation.</exception>
        ///     <exception     cref="NotSupportedException">See     Process.Kill()     exceptions
documentation.</exception>
        ///     <exception     cref="InvalidOperationException">See     Process.Kill()     exceptions
documentation.</exception>
        public static void KillProcessByMainWindowHwnd(int hWnd)
        {
            uint processID;
            GetWindowThreadProcessId((IntPtr)hWnd, out processID);
            if (processID == 0)
```

```
                throw new ArgumentException("Process has not been found by the given main window
handle.", "hWnd");
            Process.GetProcessById((int)processID).Kill();
        }



        public static void runExcelMacro(Dictionary<string, string> macroDic)
        {
            object oMissing = System.Reflection.Missing.Value;
            // Create an instance of Microsoft Excel, make it visible
            Excel.ApplicationClass oExcel = new Microsoft.Office.Interop.Excel.ApplicationClass();
            oExcel.Visible = true;
            Excel.Workbooks oBooks = oExcel.Workbooks;
            Excel._Workbook oBook = null;
            foreach (var macrodic in macroDic)
            {
                oBook = oBooks.Open(macrodic.Key, oMissing, oMissing,
                    oMissing, oMissing, oMissing, oMissing, oMissing, oMissing,
                    oMissing, oMissing, oMissing, oMissing, oMissing, oMissing);
                //RunMacro(oExcel, new Object[] { macro});
                oExcel.GetType().InvokeMember("Run",
                    System.Reflection.BindingFlags.Default |
                    System.Reflection.BindingFlags.InvokeMethod,
                    null, oExcel, new Object[] { macrodic.Value });
            }

            // Quit Excel and clean up.
            int hWnd = oExcel.Application.Hwnd;
            oBook.Close(false, oMissing, oMissing);
            System.Runtime.InteropServices.Marshal.ReleaseComObject(oBook);
            oBook = null;
            System.Runtime.InteropServices.Marshal.ReleaseComObject(oBooks);
            oBooks = null;
            oExcel.Quit();
            System.Runtime.InteropServices.Marshal.ReleaseComObject(oExcel);
            oExcel = null;
            GC.Collect();
            TryKillProcessByMainWindowHwnd(hWnd);

        }
    }
}
```