Project: "Masscap.udc"

Metric	Value	Description
CountDeclClass	0	Number of classes.
CountDeclFile	1	Number of files.
CountDeclFileCode	1	Number of code files.
CountDeclFileHeader	0	Number of header files.
CountDeclFunction	11	Number of functions.
CountLine	564	Number of all lines. [aka NL]
CountLineBlank	110	Number of blank lines. [aka BLOC]
CountLineCode	346	Number of lines containing source code. [aka LOC]
CountLineCodeDecl	70	Number of lines containing declarative source code.
CountLineCodeExe	236	Number of lines containing executable source code.
CountLineComment	172	Number of lines containing comment. [aka CLOC]
CountLineInactive	0	Number of inactive lines.
CountLinePreprocessor	16	Number of preprocessor lines.
CountSemicolon	249	Number of semicolons.
CountStmt	297	Number of statements.
CountStmtDecl	70	Number of declarative statements.
CountStmtEmpty	0	Number of empty statements.
CountStmtExe	227	Number of executable statements.
RatioCommentToCode	0.50	Ratio of comment lines to code lines.

```
// masscap.cpp : Defines the entry point for the console application.
//

/*  Ken Black 4/28/99

   This program reads a time series of concentrations at each nodal point of extraction and
   injection wells.  This information is generated with the postproc program written by Steve Andreson.
   This program computes mass capture statistics from the concentration information.
   This program is applicable for all box models.


argv[1] = post proc output file
argv[2] = modified well file - time must be added to the number of nodes header for each stress period
argv[3] = output file name

Updated on 9/30/04 to be in sync with post-proc output.

The number of lines being read in the well_hist.sum file has been changed from 10 to 13.
You still need to add the initial mass tag at the beginning, and this comes from the concentration.sum file
from post proc.

Here is the batch file to run this program:

m:\data\c\source\masscap\debug\masscap wellhist.sum c2003.wtm c2003mc.out

The contents of wellhist.sum

im 494.13  <- This has to to be manually added to this file
con="c2003.CON  9/29/2004 10:49:04 PM 482014400 bytes"
btn="c2003.btn  1/11/2000 8:36:10 PM 36677596 bytes"
bas="c2003.bas  9/29/2004 8:19:40 PM 29420590 bytes"
bcf="c2003.bcf  9/29/2001 8:24:24 PM 28906226 bytes"
no mask file
sit="c2003.sit  9/30/2004 2:57:28 PM 3898 bytes"
easting=1959840
northing=298040
rotation=0.0
NMonitor=421
NTime=51.0
         Well ID     Easting    Northing   Elevation   Col  Row  Lay
        Time       Value (head or conc)
            AM20   1976501.7    305564.9      -185.0    81  206   27
      0.0000     0.00000
    356.2500     0.00000

The contents of c2003.wtm

       421         0  <- starting time, this has only one stress period
         1       185        30  -1657.36      0.00 AM06
         2       185        30  -8233.36      0.00 AM06
         3       185        30  -8179.89      0.00 AM06
         4       185        30  -8233.35      0.00 AM06
  ....
```

```


*/


#include <stdio.h>
#include <stddef.h>
#include <stdlib.h>
#include <stdarg.h>
#include <string.h>
#include <malloc.h>
#include <dos.h>
#include <float.h>
#include <math.h>


/*****************************Preprocessor Directives ************************************************/

#define mass_convert   28.3/1000000   /* convert ug/l*ft^3/day to g/day */
#define max_time_steps 160   /* maximum number of time steps */
#define max_nodes      10000 /* maximum number of observation (nodal) points - this is not equal to the
                               number of wells since each well usually has more than one nodal point */
#define max_well       500 /* maximum number of unique wells */
#define blank_lines     13 /* the number of lines to read at the beginning of the well_hist.sum */
#define true 1
#define false 0

/***************************** Data Structure Definitions **********************************************/

struct concentration {  /* these are the nodal concentrations, flow rates, etc over time */
    char wellid[20];                 /* MMR well id       */
    float easting;                   /* ft MA state plane */
    float northing;                  /* ft MA state plane */
    float elevation;                 /* ft msl            */
    float time[max_time_steps[1]];      /* days              */
    float conc[max_time_steps[1]];      /* parts per billion */
    float flow_rate[max_time_steps[1]]; /* flow in ft^3/day  */
        float nodal_mass_flux[max_time_steps[1]]; /* flow in ft^3/day  */
        float nodal_mass_removed[max_time_steps[1]]; /* total mass removed at the node at end of simulation */
        float total_water_pumped[max_time_steps[1]]; /* total flow removed from the node */
        int row;                         /* row number        */
        int col;                         /* column number     */
        int lay;                         /* layer number      */
    };
struct concentration[2] wellconc[max_nodes[3]];

struct wellflow {  /* these are the nodal flows which can be time varying (read from modified *.wel file) */
        int   row[max_nodes[3]];       /* row number        */
        int   col[max_nodes[3]];       /* column number     */
        int   lay[max_nodes[3]];       /* layer number      */
        int   numwell;                /* num wells per ts  */
    float time;                  /* days              */
    float flow_rate[max_nodes[3]]; /* flow ft3/day      */
```

Footnotes:
1: *masscap.cpp:73*
2: *masscap.cpp:83*
3: *masscap.cpp:74*

```
    };
struct wellflow[1] nodeflow[max_time_steps[2]];

struct welltotal {  /* these are the totals for each well */
        int nodes_per_well;/* number of nodes per well */
    float total_mass;  /* well accumulative mass recovery     */
    float total_pump;  /* total volume pumped from the well   */
        float gal_gram;    /* total gallons removed per gram recovered */
        float inf_conc;    /* average influent concentration */
        float total_flow;  /* total flow in ft^3/day for the well */
        float nod_conc;    /* conc*flow at each node */
    };
struct welltotal[3] wellsum[max_well[4]];

/************************************** Global Variables  **************************************/

FILE *summary_file;
char input_file1[120],input_file2[120],input_file3[120];
int num_stress_periods,total_nodes,total_time_steps,unique_wells,tpt; /* tpt = time pointer */
float initial_mass;

/************************************** function declarations **********************************/


void Abort[5](char*mess);
void Abort_nodes[6](int);
void Abort_times[7](int);
void Abort_ini_mass[8](void);
void Abort_well[9](int);
void accumulate_well_data[10](void);
void link_flow_conc[11](void);
int  main[12](int,char **);
void process_nodal_concentration_data[13](void);
void read_concentration_data[14](void);
void read_flow_data[15](void);


/**************************************  Main ***************************************************/

int main(int argc,char*argv[]){
  strcpy(input_file1[16],argv[17][1]);  /* post proc output file */
  strcpy(input_file2[18],argv[17][2]);  /* modified well file - time added to each stress period */
  strcpy(input_file3[19],argv[17][3]);  /* output file name */
  read_concentration_data[14]();
  read_flow_data[15]();
  link_flow_conc[11]();

  process_nodal_concentration_data[13]();
  accumulate_well_data[10]();
  return(1);

}
```

Footnotes:
1: *masscap.cpp:100*
2: *masscap.cpp:73*
3: *masscap.cpp:110*
4: *masscap.cpp:76*
5: *masscap.cpp:162*
6: *masscap.cpp:183*
7: *masscap.cpp:194*
8: *masscap.cpp:205*
9: *masscap.cpp:172*
10: *masscap.cpp:214*
11: *masscap.cpp:348*
12: *masscap.cpp:146*
13: *masscap.cpp:296*
14: *masscap.cpp:391*
15: *masscap.cpp:509*
16: *masscap.cpp:124*
17: *masscap.cpp:146*
18: *masscap.cpp:124*
19: *masscap.cpp:124*

```
/**************************************  Abort ***************************************************/

void Abort(char*mess)                       /* write message and exit */
  {
  fprintf(stderr,"\nfile open error: %s\n",mess[1]);
  fprintf(summary_file[2],"\nfile open error: %s\n",mess[1]);
  fcloseall();
  exit(1);
  }

/**************************************  Abort_well ***************************************************/

void Abort_well(int numwell)                       /* write message and exit */
  {
  fprintf(summary_file[2],"\nMaximum number of wells is: %d\n",max_well[3]);
  fprintf(summary_file[2],"\nCurrent number of wells is: %d\n",numwell[4]);
  fprintf(stderr,"\nMaximum number of wells is: %d\n",max_well[3]);
  fprintf(stderr,"\nCurrent number of wells is: %d\n",numwell[4]);
  fcloseall();
  exit(1);
  }
/**************************************  Abort_nodes ***************************************************/

void Abort_nodes(int numnodes)                      /* write message and exit */
  {
  fprintf(summary_file[2],"\nMaximum number of nodes is: %d\n",max_nodes[5]);
  fprintf(summary_file[2],"\nCurrent number of nodes is: %d\n",numnodes[6]);
  fprintf(stderr,"\nMaximum number of nodes is: %d\n",max_nodes[5]);
  fprintf(stderr,"\nCurrent number of nodes is: %d\n",numnodes[6]);
  fcloseall();
  exit(1);
  }
/**************************************  Abort_times ***************************************************/

void Abort_times(int numtimes)                       /* write message and exit */
  {
  fprintf(summary_file[2],"\nMaximum number of timesteps is: %d\n",max_time_steps[7]);
  fprintf(summary_file[2],"\nCurrent number of timesteps is: %d\n",numtimes[8]);
  fprintf(stderr,"\nMaximum number of timesteps is: %d\n",max_time_steps[7]);
  fprintf(stderr,"\nCurrent number of timesteps is: %d\n",numtimes[8]);
  fcloseall();
  exit(1);
  }
/**************************************  Abort_times ***************************************************/

void Abort_ini_mass(void)                       /* write message and exit */
  {
  fprintf(summary_file[2],"\nInitial mass has not been added to first line of: %s\n",input_file1[9]);
  fprintf(stderr,"\nInitial mass has not been added to first line of: %s\n",input_file1[9]);
  fcloseall();
  exit(1);
  }
/**************************************  accumulate_well_data ***************************************************/
```

Footnotes:
1: *masscap.cpp:162*
2: *masscap.cpp:123*
3: *masscap.cpp:76*
4: *masscap.cpp:172*
5: *masscap.cpp:74*
6: *masscap.cpp:183*
7: *masscap.cpp:73*
8: *masscap.cpp:194*
9: *masscap.cpp:124*

```

void accumulate_well_data(void){
int j,k,n,local_node_counter,global_node_number,gnpt;

/* calculate nodal mass flux for each well for each time step */
fprintf(summary_file[1],"\n*******************************************************************************\n");
fprintf(summary_file[1],"Table 4. Summary of contaminant recovery for each well over time\n\n");
fprintf(summary_file[1],"Accumulative Mass Removed - mass_rem (kg) \n");
fprintf(summary_file[1],"Percent Initial Mass = perc init mass - this is calculated based on dissolved mass+adsorbed mass\n");

fprintf(summary_file[1],"Total Water Pumped from Well = h20_pump (millions of gallons)\n");
fprintf(summary_file[1],"Total Flow Rate = tot_flow (gal/min) from each extraction/injection well\n");
fprintf(summary_file[1],"Gallons Pumped Per Gram Recovered = gallon per gram (gal/gm)\n");
fprintf(summary_file[1],"Computed Influent Concentration = inf_conc (ppb)\n");
fprintf(summary_file[1],"Initial Mass Read From File = %10.2f (kg)\n\n",initial_mass[2]);

fprintf(summary_file[1],"well    easting   northing      time   mass_rem  perc_init   h20_pump   tot_flow     gallon   inf_co
nc\n");
fprintf(summary_file[1],"id       (ft)       (ft)       (yr)       (kg)    mass      (mil_gal)   (gpm)      per_gram      (ppb)
\n");

global_node_number[3]=0;  /* this moves the array pointer from 0 to number of observation nodes */

for(k[4]=0;k[4]<unique_wells[5];k[4]++){
   wellsum[6][k[4]].total_mass[7]=0;
   wellsum[6][k[4]].total_pump[8]=0;
   wellsum[6][k[4]].gal_gram[9]=0;
   wellsum[6][k[4]].inf_conc[10]=0;

    for(j[11]=0;j[11]<total_time_steps[12];j[11]++){
            local_node_counter[13]=0;
            for(n[14]=0;n[14]<wellsum[6][k[4]].nodes_per_well[15];n[14]++){
             gnpt[16]=global_node_number[3]+local_node_counter[13]; /* global node pointer */
                 if(j[11]==0){
                  /* initialize totals */
           wellsum[6][k[4]].total_mass[7]+=(wellconc[17][gnpt[16]].nodal_mass_removed[18][j[11]]);
           wellsum[6][k[4]].total_pump[8]+=(wellconc[17][gnpt[16]].total_water_pumped[19][j[11]]);
           wellsum[6][k[4]].total_flow[20]+=(wellconc[17][gnpt[16]].flow_rate[21][j[11]]);
           wellsum[6][k[4]].nod_conc[22]+=(wellconc[17][gnpt[16]].conc[23][j[11]]*wellconc[17][gnpt[16]].flow_rate[21][j[11]]);
                 }
                 else {
                  /* since nodal mass removed is stored as an accumulative total, I must subtract the previous value
 */
           wellsum[6][k[4]].total_mass[7]+=(wellconc[17][gnpt[16]].nodal_mass_removed[18][j[11]]-wellconc[17][gnpt[16]].nodal_mass_removed[18][j[11]-1
]);
           wellsum[6][k[4]].total_pump[8]+=(wellconc[17][gnpt[16]].total_water_pumped[19][j[11]]-wellconc[17][gnpt[16]].total_water_pumped[19][j[11]-1
]);
           wellsum[6][k[4]].total_flow[20]+=(wellconc[17][gnpt[16]].flow_rate[21][j[11]]-wellconc[17][gnpt[16]].flow_rate[21][j[11]-1]);
           wellsum[6][k[4]].nod_conc[22]+=(wellconc[17][gnpt[16]].conc[23][j[11]]*  wellconc[17][gnpt[16]].flow_rate[21][j[11]])-
                     (wellconc[17][gnpt[16]].conc[23][j[11]-1]*wellconc[17][gnpt[16]].flow_rate[21][j[11]-1]);}



      local_node_counter[13]++;}
```

Footnotes:
1: *masscap.cpp:123*
2: *masscap.cpp:126*
3: *masscap.cpp:215*
4: *masscap.cpp:215*
5: *masscap.cpp:125*
6: *masscap.cpp:119*
7: *masscap.cpp:112*
8: *masscap.cpp:113*
9: *masscap.cpp:114*
10: *masscap.cpp:115*
11: *masscap.cpp:215*
12: *masscap.cpp:125*
13: *masscap.cpp:215*
14: *masscap.cpp:215*
15: *masscap.cpp:111*
16: *masscap.cpp:215*
17: *masscap.cpp:98*
18: *masscap.cpp:92*
19: *masscap.cpp:93*
20: *masscap.cpp:116*
21: *masscap.cpp:90*
22: *masscap.cpp:117*
23: *masscap.cpp:89*

```

         wellsum[1][k[2]].gal_gram[3]=wellsum[1][k[2]].total_pump[4]/wellsum[1][k[2]].total_mass[5];
         if(wellsum[1][k[2]].gal_gram[3]>150000||wellsum[1][k[2]].total_pump[4]<10) /* this is an injection well - set value to 0, or beginning of pumping */
                         wellsum[1][k[2]].gal_gram[3]=0;

                 /* compute average influent concentration only if well is pumping*/
                 if(wellsum[1][k[2]].total_flow[6]>(float)0.)
                 wellsum[1][k[2]].inf_conc[7]=wellsum[1][k[2]].nod_conc[8]/wellsum[1][k[2]].total_flow[6];
         else
                 wellsum[1][k[2]].inf_conc[7]=0.0;
         if(wellsum[1][k[2]].inf_conc[7]<0.00001) /* less than detection */
                         wellsum[1][k[2]].inf_conc[7]=0;

                 fprintf(summary_file[9],"%s %10.2f %10.2f %10.2f %10.4f %10.2f %10.1f %10.2f %10.1f %10.5f\n",
                 wellconc[10][global_node_number[11]].wellid[12],
                 wellconc[10][global_node_number[11]].easting[13],
                 wellconc[10][global_node_number[11]].northing[14],
                 wellconc[10][global_node_number[11]].time[15][j[16]],
                 wellsum[1][k[2]].total_mass[5]/1000,    /* convert to kg */
         wellsum[1][k[2]].total_mass[5]/1000/initial_mass[17]*100,   /* convert to kg,divide by initial mass, convert to percent */
                 wellsum[1][k[2]].total_pump[4]/1000000, /* convert to million gallons */
                 wellsum[1][k[2]].total_flow[6]/1440*7.48, /* flow rate in gallons/minute */
                 wellsum[1][k[2]].gal_gram[3],           /* gallons pumped per gram contaminant recovered */
                 wellsum[1][k[2]].inf_conc[7]);          /* average influent concentration */



            }
         global_node_number[11]+=local_node_counter[18];

    /* fprintf(summary_file,"\n");*/}

}

/************************* process_nodal_concentration_data ****************************************************/

void process_nodal_concentration_data(void){
int j,k;


/* calculate nodal mass flux for each well for each time step */
fprintf(summary_file[9],"\n*******************************************************************************\n");
fprintf(summary_file[9],"Table 3. Summary of nodal concentration data and accumulative mass removal over time \n\n");
fprintf(summary_file[9],"well    easting   northing  elevation       time     concen  flow_rate  nod_mflux   mass_rem   h20_pump\n");
fprintf(summary_file[9],"id       (ft)       (ft)       (ft)       (yr)     (ug/l)   (ft^3/d)    (g/day)        (g)   (thousand_gal)\n");

for(k[19]=0;k[19]<total_nodes[20];k[19]++){


   for(j[21]=0;j[21]<total_time_steps[22];j[21]++){
```

Footnotes:
1: *masscap.cpp:119*
2: *masscap.cpp:215*
3: *masscap.cpp:114*
4: *masscap.cpp:113*
5: *masscap.cpp:112*
6: *masscap.cpp:116*
7: *masscap.cpp:115*
8: *masscap.cpp:117*
9: *masscap.cpp:123*
10: *masscap.cpp:98*
11: *masscap.cpp:215*
12: *masscap.cpp:84*
13: *masscap.cpp:85*
14: *masscap.cpp:86*
15: *masscap.cpp:88*
16: *masscap.cpp:215*
17: *masscap.cpp:126*
18: *masscap.cpp:215*
19: *masscap.cpp:297*
20: *masscap.cpp:125*
21: *masscap.cpp:297*
22: *masscap.cpp:125*

```
       wellconc[k].nodal_mass_removed[j]=0;
       wellconc[k].total_water_pumped[j]=0;
       wellconc[k].nodal_mass_flux[j]=(float)fabs((double)(wellconc[k].conc[j]*wellconc[k].flow_rate[j]*(float)mass_convert));

           if(j>=1){ /* sum up mass removed at each node and total water pumped */

       wellconc[k].nodal_mass_removed[j]=(wellconc[k].nodal_mass_flux[j-1]+wellconc[k].nodal_mass_flux[j])/(float)2.
                                              *(wellconc[k].time[j]-wellconc[k].time[j-1])*(float)365.25+
                                                                          wellconc[k].nodal_mass_removed[j-1];

       wellconc[k].total_water_pumped[j]=(float)(fabs((double)(wellconc[k].flow_rate[j-1]+wellconc[k].flow_rate[j]))/(float)2.
                                              *(wellconc[k].time[j]-wellconc[k].time[j-1])*(float)365.25*(float)7.48+
                                                                            wellconc[k].total_water_pumped[j-1]);}


         fprintf(summary_file,"%s %10.2f %10.2f %10.2f %10.2f %10.2f %10.2f %10.2f %10.2f %10.0f\n",
                 wellconc[k].wellid,
                 wellconc[k].easting,
                 wellconc[k].northing,
                 wellconc[k].elevation,
                 wellconc[k].time[j],
                 wellconc[k].conc[j],
                 wellconc[k].flow_rate[j],
                 fabs((double)wellconc[k].nodal_mass_flux[j]),
                 wellconc[k].nodal_mass_removed[j],
                 wellconc[k].total_water_pumped[j]/1000 /* convert to thousand gallons */);
            }

    /* fprintf(summary_file,"\n");*/}



}


/***************************************  link_flow_conc ****************************************************/
/* this routine finds the flow rate that is applicable for every concentration time step stored in the file */

void link_flow_conc(void ){
int j,k,n,tm,tpt,wellfound,nsp1; /* tpt = time pointer */
float cur_time;

nsp1=num_stress_periods+1;

for(k=0;k<total_nodes;k++){ /* process every observation node */

            for(n=0;n<total_time_steps;n++){ /* for each time step, find the appropriate flow */
        cur_time=wellconc[k].time[n]; /* initialize time */

                   for(tm=0;tm<nsp1;tm++){  /* time look-up: check if same time or less than next time */
                           if(fabs((double)(cur_time-(nodeflow[tm].time/(float)365.25)))<0.01){  /* exact time match */
```

Footnotes:
1: masscap.cpp:98
2: masscap.cpp:297
3: masscap.cpp:92
4: masscap.cpp:297
5: masscap.cpp:93
6: masscap.cpp:91
7: masscap.cpp:89
8: masscap.cpp:90
9: masscap.cpp:72
10: masscap.cpp:88
11: masscap.cpp:123
12: masscap.cpp:84
13: masscap.cpp:85
14: masscap.cpp:86
15: masscap.cpp:87
16: masscap.cpp:349
17: masscap.cpp:125
18: masscap.cpp:349
19: masscap.cpp:125
20: masscap.cpp:349
21: masscap.cpp:125
22: masscap.cpp:350
23: masscap.cpp:349
24: masscap.cpp:108
25: masscap.cpp:105

```
                                    tpt[1]=tm[2];
                                  /*   fprintf(summary_file,"Exact Time Match: cur time = %6.2f node time =%6.2f and timeptr = %d \n",cur_time,(nodeflow[tpt].time/365.25),tpt);*/}
                                  else if((cur_time[3]>(nodeflow[4][tm[2]].time[5]/(float)365.25))&&(cur_time[3]<(nodeflow[4][tm[2]+1].time[5]/(float)365.25))){ /* in between time intervals */
                                  tpt[1]=tm[2];
                                  /*      fprintf(summary_file,"cur time = %6.2f node time1 =%6.2f node time2 =%6.2f and timeptr = %d \n",cur_time,(nodeflow[tm].time/365.25),(nodeflow[tm+1].time/365.25),tm); */}

                    } /* end time step loop */

                    for(j[6]=0;j[6]<nodeflow[4][tpt[1]].numwell[7];j[6]++){ /* process from 0 to number of wells to match i,j,k at that time step */
                 wellfound[8]=false;
                         if((nodeflow[4][tpt[1]].row[9][j[6]]==wellconc[10][k[11]].row[12])&&(nodeflow[4][tpt[1]].col[13][j[6]]==wellconc[10][k[11]].col[14])&&(nodeflow[4][tpt[1]].lay[15][j[6]]==wellconc[10][k[11]].lay[16])){ /* match found */
                         wellconc[10][k[11]].flow_rate[17][n[18]]=nodeflow[4][tpt[1]].flow_rate[19][j[6]];
                           fprintf(summary_file[20],"flow rate = %10.2f for well %19s at time %6.2f yr at row=%3d col=%3d layer=%2d\n",nodeflow[4][tpt[1]].flow_rate[19][j[6]],wellconc[10][k[11]].wellid[21],cur_time[3],wellconc[10][k[11]].row[12],wellconc[10][k[11]].col[14],wellconc[10][k[11]].lay[16]);
                                  wellfound[8]=true;
                                  break;}

                   } /* end j loop */
                         if(!wellfound[8]){ /* no entry found for well at this time - set value to 0 */
                  wellconc[10][k[11]].flow_rate[17][n[18]]=0.;
                                  fprintf(summary_file[20],"Warning: no entry found for well  %19s at time %6.2f yr so rate is being set to: \n",wellconc[10][k[11]].wellid[21],cur_time[3]);
                                  fprintf(summary_file[20],"flow rate = %10.2f for well %19s at time %6.2f yr at row=%3d col=%3d layer=%2d\n",wellconc[10][k[11]].flow_rate[17][n[18]],wellconc[10][k[11]].wellid[21],cur_time[3],wellconc[10][k[11]].row[12],wellconc[10][k[11]].col[14],wellconc[10][k[11]].lay[16]); }


                 } /* end stress period loop */
             } /* end k loop */
} /* end function */


/**************************************  read_concentration_data ***************************************************/

void read_concentration_data(void){

  FILE *file;
  char bufr[120],dummystr[25];
  int i,n,k,npw;
  float maxelev,minelev;

  /* open input data summary file */

  if((summary_file[20]=fopen(input_file3[22],"wt"))==NULL){
    printf("can't open the transport data summary ouput file");
    Abort[23](input_file3[22]);}
```

Footnotes:
1: *masscap.cpp:349*
2: *masscap.cpp:349*
3: *masscap.cpp:350*
4: *masscap.cpp:108*
5: *masscap.cpp:105*
6: *masscap.cpp:349*
7: *masscap.cpp:104*
8: *masscap.cpp:349*
9: *masscap.cpp:101*
10: *masscap.cpp:98*
11: *masscap.cpp:349*
12: *masscap.cpp:94*
13: *masscap.cpp:102*
14: *masscap.cpp:95*
15: *masscap.cpp:103*
16: *masscap.cpp:96*
17: *masscap.cpp:90*
18: *masscap.cpp:349*
19: *masscap.cpp:106*
20: *masscap.cpp:123*
21: *masscap.cpp:84*
22: *masscap.cpp:124*
23: *masscap.cpp:162*

```
403:   else
404:           printf("\nData summary output file %s is open!\n",input_file3[1]);
405:
406:   /* open concentration file - need to use input_file for filename after debugging */
407:
408:   if((file[2]=fopen(input_file1[3],"rt"))==NULL){
409:     printf("can't open the nodal concentration file");
410:     Abort[4](input_file1[3]);}
411:   else
412:           printf("\nConcentration file %s from post-proc is open!\n",input_file1[3]);
413:
414:   fprintf(summary_file[5],"\nThis transport output file is : %s",input_file3[1]);
415:   fprintf(summary_file[5],"\nThe concentration data file is: %s",input_file1[3]);
416:   fprintf(summary_file[5],"\nThe nodal flow data file is   : %s\n",input_file2[6]);
417:
418:   fprintf(summary_file[5],"\n********************** CURRENT PROGRAM LIMITS *****************************************\n");
419:   fprintf(summary_file[5],"\nMaximum Number of Wells       : %d\n",max_well[7]);
420:   fprintf(summary_file[5],"\nMaximum Number of Well Nodes  : %d\n",max_nodes[8]);
421:   fprintf(summary_file[5],"\nMaximum Number of Time Steps  : %d\n",max_time_steps[9]);
422:   fprintf(summary_file[5],"\nThe source code is masscap.cpp and can be recompiled for larger arrays\n\n");
423:
424:   fprintf(summary_file[5],"\n*****************************************************************************\n");
425:   fprintf(summary_file[5],"Table 1. Summary of well field obtained from the concentration file: %s\n\n",input_file1[3]);
426:   n[10]=0;
427:   npw[11]=1; /* nodes per well */
428:   minelev[12]=100000;
429:   maxelev[13]=-100000;
430:   unique_wells[14]=0;
431:
432:   /* read the initial mass from the file */
433:
434:   fgets(bufr[15],sizeof(bufr[15]),file[2]);
435:     if(sscanf(bufr[15],"%s%f",dummystr[16],&initial_mass[17])!=2)
436:                 Abort_ini_mass[18]();
437:
438:
439:   /* read a set number of dummy lines at the top of the wellhist.sum - the initial stuff in this file */
440:   for (i[19]=0;i[19]<blank_lines[20];i[19]++)
441:      fgets(bufr[15],sizeof(bufr[15]),file[2]);
442:
443:   while(fgets(bufr[15],sizeof(bufr[15]),file[2])){
444:     if(sscanf(bufr[15],"%s%f%f%f%d%d%d",wellconc[21][n[10]].wellid[22],&wellconc[21][n[10]].easting[23],&wellconc[21][n[10]].northing[24],&wellconc[21][n[10]]
 .elevation[25],&wellconc[21][n[10]].col[26],&wellconc[21][n[10]].row[27],&wellconc[21][n[10]].lay[28])==7){
445: /*      printf("%s %10.2f %10.2f %10.2f %5d %5d %5d\n",wellconc[n].wellid,wellconc[n].easting,wellconc[n].northing,wellconc[
 n].elevation,wellconc[n].col,wellconc[n].row,wellconc[n].lay);   */
446:
447:           /* count nodes per well and find min/max elevation */
448:           if(n[10]>0){
449:                   if(strcmp(wellconc[21][n[10]-1].wellid[22],wellconc[21][n[10]].wellid[22])==0){ /* working on same well */
450:                           npw[11]++;
451:                           maxelev[13]=(maxelev[13]>wellconc[21][n[10]-1].elevation[25]) ? maxelev[13] : wellconc[21][n[10]-1].elevation[25];
452:                           minelev[12]=(minelev[12]<wellconc[21][n[10]-1].elevation[25]) ? minelev[12] : wellconc[21][n[10]-1].elevation[25];
453:                   }
```

Footnotes:
1: *masscap.cpp:124*
2: *masscap.cpp:393*
3: *masscap.cpp:124*
4: *masscap.cpp:162*
5: *masscap.cpp:123*
6: *masscap.cpp:124*
7: *masscap.cpp:76*
8: *masscap.cpp:74*
9: *masscap.cpp:73*
10: *masscap.cpp:395*
11: *masscap.cpp:395*
12: *masscap.cpp:396*
13: *masscap.cpp:396*
14: *masscap.cpp:125*
15: *masscap.cpp:394*
16: *masscap.cpp:394*
17: *masscap.cpp:126*
18: *masscap.cpp:205*
19: *masscap.cpp:395*
20: *masscap.cpp:77*
21: *masscap.cpp:98*
22: *masscap.cpp:84*
23: *masscap.cpp:85*
24: *masscap.cpp:86*
25: *masscap.cpp:87*
26: *masscap.cpp:95*
27: *masscap.cpp:94*
28: *masscap.cpp:96*

```
         else /* new well encountered */
         {
            maxelev[1]=(maxelev[1]>wellconc[2][n[3]-1].elevation[4]) ? maxelev[1] : wellconc[2][n[3]-1].elevation[4];
          minelev[5]=(minelev[5]<wellconc[2][n[3]-1].elevation[4]) ? minelev[5] : wellconc[2][n[3]-1].elevation[4];
            wellsum[6][unique_wells[7]].nodes_per_well[8]=npw[9];
            fprintf(summary_file[10],"Well %19s contained %3d nodes between %10.2f ft and %10.2f ft\n",wellconc[2][n[3]
-1].wellid[11],wellsum[6][unique_wells[7]].nodes_per_well[8],maxelev[1],minelev[5]);
      minelev[5]=100000;
      maxelev[1]=-100000;
            npw[9]=1;
            unique_wells[7]++;
            if(unique_wells[7]>max_well[12])
                Abort_well[13](unique_wells[7]);
         }
     }
     n[3]++;
     if(n[3]>max_nodes[14])
            Abort_nodes[15](n[3]);

   k[16]=0;
   }
  else /* n must be decremented back one! */
   if(sscanf(bufr[17],"%f%f",&wellconc[2][n[3]-1].time[18][k[16]],&wellconc[2][n[3]-1].conc[19][k[16]])==2){
        /* convert time to years */
    wellconc[2][n[3]-1].time[18][k[16]]/=365.25;
/*  printf("%10.2f %10.2f\n",wellconc[n-1].time[k],wellconc[n-1].conc[k]); */

   k[16]++;
     if(k[16]>max_time_steps[20])
            Abort_times[21](k[16]);
         }

   }

  /* process last record */
  maxelev[1]=(maxelev[1]>wellconc[2][n[3]-1].elevation[4]) ? maxelev[1] : wellconc[2][n[3]-1].elevation[4];
  minelev[5]=(minelev[5]<wellconc[2][n[3]-1].elevation[4]) ? minelev[5] : wellconc[2][n[3]-1].elevation[4];
    wellsum[6][unique_wells[7]].nodes_per_well[8]=npw[9];
  fprintf(summary_file[10],"Well %19s contained %3d nodes between %10.2f ft and %10.2f ft\n",wellconc[2][n[3]-1].wellid[11],wellsum[6]
[unique_wells[7]].nodes_per_well[8],maxelev[1],minelev[5]);

    total_nodes[22]=n[3];
    total_time_steps[23]=k[16];
    unique_wells[7]++;

    /* write summary data */

  fprintf(summary_file[10],"\n\nTotal well nodes = %d\n",total_nodes[22]);
  fprintf(summary_file[10],"Total wells (unique well ids)  = %d\n",unique_wells[7]);
  fprintf(summary_file[10],"Total concentration time steps = %d\n\n",total_time_steps[23]);

  fclose(file[24]);

```

Footnotes:
1: *masscap.cpp:396*
2: *masscap.cpp:98*
3: *masscap.cpp:395*
4: *masscap.cpp:87*
5: *masscap.cpp:396*
6: *masscap.cpp:119*
7: *masscap.cpp:125*
8: *masscap.cpp:111*
9: *masscap.cpp:395*
10: *masscap.cpp:123*
11: *masscap.cpp:84*
12: *masscap.cpp:76*
13: *masscap.cpp:172*
14: *masscap.cpp:74*
15: *masscap.cpp:183*
16: *masscap.cpp:395*
17: *masscap.cpp:394*
18: *masscap.cpp:88*
19: *masscap.cpp:89*
20: *masscap.cpp:73*
21: *masscap.cpp:194*
22: *masscap.cpp:125*
23: *masscap.cpp:125*
24: *masscap.cpp:393*

```
505:  }
506:
507: /*************************************** read_flow_data ****************************************************/
508:
509: void read_flow_data(void){
510:
511:  FILE *file2;
512:  char bufr[120];
513:  int n,timecount;
514:  float time_yr;
515:
516:  /* open nodal based well file */
517:
518:  if((file2[1]=fopen(input_file2[2],"rt"))==NULL){
519:   printf("Can't open the modified modflow well/time file");
520:   Abort[3](input_file2[2]);}
521:  else
522:      printf("\nModified flow rate file %s is open!\n",input_file2[2]);
523:
524:  fprintf(summary_file[4],"\n******************************************************************************\n");
525:  fprintf(summary_file[4],"Table 2. Summary of flow rates on a nodal basis obtained from file: %s \n\n",input_file2[2]);
526:
527:
528:  timecount[5]=0;
529:
530:  while(fgets(bufr[6],sizeof(bufr[6]),file2[1])){
531:      if(sscanf(bufr[6],"%d%f",&nodeflow[7][timecount[5]].numwell[8],&nodeflow[7][timecount[5]].time[9])==2){ /* go read layer,row,col,fl
     ow */
532:          time_yr[10]=nodeflow[7][timecount[5]].time[9]/(float)365.25;
533:          fprintf(summary_file[4],"Well stress period #%3d  = %10.2f years has %3d nodes \n",timecount[5]+1,time_yr[10],nod
     eflow[7][timecount[5]].numwell[8]);
534:         /* handle situation where -1 can be used in the well file to indicate the use of previous time step  */
535:         if(nodeflow[7][timecount[5]].numwell[8]==-1){ /* use previous values */
536:             for(n[11]=0;n[11]<nodeflow[7][timecount[5]-1].numwell[8];n[11]++){
537:              nodeflow[7][timecount[5]].numwell[8]=nodeflow[7][timecount[5]-1].numwell[8];
538:              nodeflow[7][timecount[5]].lay[12][n[11]]=nodeflow[7][timecount[5]-1].lay[12][n[11]];
539:              nodeflow[7][timecount[5]].row[13][n[11]]=nodeflow[7][timecount[5]-1].row[13][n[11]];
540:              nodeflow[7][timecount[5]].col[14][n[11]]=nodeflow[7][timecount[5]-1].col[14][n[11]];
541:              nodeflow[7][timecount[5]].flow_rate[15][n[11]]=nodeflow[7][timecount[5]-1].flow_rate[15][n[11]];}
542:         }else
543:          for(n[11]=0;n[11]<nodeflow[7][timecount[5]].numwell[8];n[11]++){
544:             fgets(bufr[6],sizeof(bufr[6]),file2[1]);
545:             sscanf(bufr[6],"%d%d%d%f",&nodeflow[7][timecount[5]].lay[12][n[11]],&nodeflow[7][timecount[5]].row[13][n[11]],&nodeflow[7][tim
     ecount[5]].col[14][n[11]],&nodeflow[7][timecount[5]].flow_rate[15][n[11]]);
546:       /* convert negative flows to positve values */
547:             if(nodeflow[7][timecount[5]].flow_rate[15][n[11]]<(float)0.0)
548:                 nodeflow[7][timecount[5]].flow_rate[15][n[11]]*=(float)-1.0;
549: /* debug statement      fprintf(summary_file,"stress period #%d %d %d %d %f \n",timecount+1,nodeflow[timecount].lay[n],nodef
     low[timecount].row[n],nodeflow[timecount].col[n],nodeflow[timecount].flow_rate[n]); */
550:          }
551:          timecount[5]++;
552:      }
553:      num_stress_periods[16]=timecount[5];
```

Footnotes:
1: *masscap.cpp:511*
2: *masscap.cpp:124*
3: *masscap.cpp:162*
4: *masscap.cpp:123*
5: *masscap.cpp:513*
6: *masscap.cpp:512*
7: *masscap.cpp:108*
8: *masscap.cpp:104*
9: *masscap.cpp:105*
10: *masscap.cpp:514*
11: *masscap.cpp:513*
12: *masscap.cpp:103*
13: *masscap.cpp:101*
14: *masscap.cpp:102*
15: *masscap.cpp:106*
16: *masscap.cpp:125*

```
        }

        /* set very large time for next array space */
        nodeflow[1][timecount[2]].time[3]=(float)3652500.;
        fprintf(summary_file[4],"Well stress period #%3d  = %10.2f years has been set for interpolation usage\n",timecount[2]+1,nodef
low[1][timecount[2]].time[3]/365.25);
        fprintf(summary_file[4],"Total number of actual stress periods = %d \n\n\n",num_stress_periods[5]);

        fclose(file2[6]);

}
```

Footnotes:
1: *masscap.cpp:108*
2: *masscap.cpp:513*
3: *masscap.cpp:105*
4: *masscap.cpp:123*
5: *masscap.cpp:125*
6: *masscap.cpp:511*