Doxygen_MPD_STAR_src_DB Reference Manual

Generated by Doxygen 1.3.7

Wed Sep 6 20:08:50 2006

Contents

1 Doxygen_MPD_STAR_src_DB Class Index **1**

1.1 Doxygen_MPD_STAR_src_DB Class List 1

2 Doxygen_MPD_STAR_src_DB File Index **3**

2.1 Doxygen_MPD_STAR_src_DB File List 3

3 Doxygen_MPD_STAR_src_DB Class Documentation **5**

3.1 time_keeper Struct Reference 5

4 Doxygen_MPD_STAR_src_DB File Documentation **7**

4.1 boo.c File Reference 7

4.2 eventCnts.c File Reference 8

4.3 insTrgCnt.h File Reference 14

4.4 N_insTrgCnt.cc File Reference 22

4.5 resync.c File Reference 31

4.6 resync.h File Reference 36

4.7 syncCol.c File Reference 39

4.8 syncCol.h File Reference 45

4.9 test.c File Reference 49

4.10 time_keeper.c File Reference 51

4.11 time_keeper.h File Reference 53

Chapter 1

Doxygen_MPD_STAR_src_DB Class Index

1.1 Doxygen_MPD_STAR_src_DB Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

time_keeper 5

Chapter 2

Doxygen_MPD_STAR_src_DB File Index

2.1 Doxygen_MPD_STAR_src_DB File List

Here is a list of all files with brief descriptions:

boo.c 7
eventCnts.c 8
insTrgCnt.h 14
N_insTrgCnt.cc 22
resync.c 31
resync.h 36
syncCol.c 39
syncCol.h 45
test.c 49
time_keeper.c 51
time_keeper.h 53

Chapter 3

Doxygen_MPD_STAR_src_DB Class Documentation

3.1 time_keeper Struct Reference

Public Attributes

- clock_t begin_clock
- clock_t save_clock
- time_t begin_time
- time_t save_time

3.1.1 Member Data Documentation

3.1.1.1 clock_t time_keeper::begin_clock

Definition at line 13 of file time_keeper.c.

3.1.1.2 time_t time_keeper::begin_time

Definition at line 14 of file time_keeper.c.

3.1.1.3 clock_t time_keeper::save_clock

Definition at line 13 of file time_keeper.c.

3.1.1.4 time_t time_keeper::save_time

Definition at line 14 of file time_keeper.c.

The documentation for this struct was generated from the following file:

- time_keeper.c

Chapter 4

Doxygen_MPD_STAR_src_DB File Documentation

4.1 boo.c File Reference

```
#include <stdio.h>
#include <string.h>
```

4.2 eventCnts.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <mysql.h>
#include <iostream>
#include "resync.h"
#include "time_keeper.h"
```

Functions

- int main (int argc, char *argv[ ])
- int rowCmp (MYSQL_ROW str1, MYSQL_ROW str2, MYSQL_RES *res)
- void printRow (MYSQL_ROW row, MYSQL_RES *res, int flag)
- void printPrimaryKeys (MYSQL_RES *res)
- void help ()
- void cleanUp (MYSQL *dbH, MYSQL *dbS, int value)

4.2.1 Function Documentation

4.2.1.1 void cleanUp (MYSQL * *dbH*, MYSQL * *dbS*, int *value*)

Definition at line 268 of file eventCnts.c.

```
269 {
270   switch (value)
271     {
272     case 1:
273       printf( "\ntables are synchronized \n\n");
274       prn_time();
275       break;
276     case 2:
277       printf ("\ntables are NOT synchonized \n\n");
278       prn_time();
279       break;
280     default:
281       printf ("SOMETHING IS VERY WRONG! (cleanUp() switch default) \n");
282     }
283
284   mysql_close(dbH);
285   mysql_close(dbS);
286
287   exit(1);
288 }
```

4.2.1.2 void help ()

Definition at line 261 of file eventCnts.c.

```
{
  printf("HELP\n\tresync checks two tables from two different servers (a master and a slave) to \n\t1)
  printf("\nUSAGE:\n\t>>resync databasename host1(master) port user table host2(slave) port\nFor examp

}
```

4.2.1.3 int main (int *argc*, char ∗ *argv*[ ])

Definition at line 17 of file eventCnts.c.

```
{
  if ((argc < 2 || strcmp(argv[1],"--help")==0 ))
    {
      help();
      return(1);
    }
/*   if (argc < 8) */
/*     { */
/*       printf("usage is incorrect - use --help for correct usage\n"); */
/*       return(1); */
/*     } */


/*   char *db = argv[1]; */
/*   char *mhost = argv[2]; */
/*   int mport = atoi(argv[3]); */
/*   char *user = argv[4]; */
/*   char *table = argv[5]; */
/*   char *shost = argv[6]; */
/*   int sport = atoi(argv[7]); */

/*   if ( !strcmp(argv[7],"3336" )) */
/*   { */
/*    pass = "mirr0r"; */

/*   }  */
  char *host = "onldb.starp.bnl.gov";
  char *db1 = "RunLog";
  char *db2 = "RunLog_daq";
  int  port1 = 3501;
  int  port1 = 3503;
  char *user = "deph";
  char * pass = NULL;

  char queryOne[40];
  char queryTwo[255];
  char queryAdd[155];
  char orderBy[155];
  char orderAdd[155];
  int val = 1;

  printf("\nFOR TABLE %s\n",table);
  if (!mysql_real_connect(&dbH,host,user,pass,db1,port1,NULL,0))
    {
      printf("can't connect to  %s %s %s\n",db,mhost,mport);
    }

  if (!mysql_real_connect(&dbS,host,user,pass,db2,port2,NULL,0))
    {
      printf("can't connect to %s %s %s\n",db,shost,sport);
    }


  sprintf(queryOne,"SELECT runNumber, eventCounts from daqSummary");
```

```
if (mysql_real_query (&dbH, queryOne, strlen(queryOne)))
        {
             printf("THIS query failed: %s\n%s\n",queryOne,mysql_error(&dbH));

        }
             res=mysql_store_result(&dbH);


             int max = mysql_num_fields(res);
                  char* fieldList[max];
                  char*   fieldNum[max];

    // initialize to all 0s
    for( int k =0 ; k<max; k++)
    {
       fieldList[k] = 0;
       fieldNum[k] = 0;
    }
    //GET the primary Keys separate them based on type
       int j =0;
                  field = mysql_fetch_fields (res);
               for (int i = 0; i<mysql_num_fields(res) ; i++)
               {
                  field = mysql_fetch_field (res);

                  if(IS_PRI_KEY(field->flags)){
                    if(IS_NUM(field->type)){
                      fieldNum[j] = field->name;
                       //                     printf("FIELD NUM NAME IS %s \n",field->name);
                     }else{
                       //  printf("FIELD NAME IS %s \n",field->name);
                     fieldList[j]= field->name;
                     }
                    j++;
                   }
                }

//GET THE NUMBER OF PRIMARY KEYS
                int count = 0, numbers =0, characters = 0;
                for (int m = 0; m < max; m++)
                  {
                    if (fieldList[m]){count++;}
                    if (fieldNum[m]){count++;}
                  }
//strcat queryTwo here
                sprintf(queryTwo,"SELECT ");
                sprintf(orderBy, " ORDER BY ");

                for (int n = 0; n < count; n++)
                  {
                    if (n < count-1)
                      {
                        if (fieldList[n]){
                          sprintf(queryAdd,"%s, ",fieldList[n]);
                          strcat(queryTwo,queryAdd);
                        }else{
                          sprintf(queryAdd,"%s, ",fieldNum[n]);
                          sprintf(orderAdd,"%s, ",fieldNum[n]);
                          strcat(orderBy,orderAdd);
                          strcat(queryTwo,queryAdd);
                        }
                      }else{
                        if (fieldList[n]){
                          sprintf(queryAdd,"%s FROM %s", fieldList[n],table);
                          strcat(queryTwo,queryAdd);
                        }else{
                          sprintf(queryAdd,"%s FROM %s", fieldNum[n],table);
```

```
                        sprintf(orderAdd,"%s",fieldNum[n]);
                        strcat(orderBy,orderAdd);
                        strcat(queryTwo,queryAdd);
                      }
                    }
                  }
                strcat(queryTwo,orderBy);
                //      printf("QUERY IS %s\n",queryTwo);

                //GET ALL KEYS FROM MASTER TABLE

                start_time();
                if (mysql_real_query (&dbH, queryTwo, strlen(queryTwo)))
                  {
                    printf("THIS query failed: %s\n%s\n",queryTwo,mysql_error(&dbH));

                  }else{
                    resM=mysql_store_result(&dbH);
                  }

                if(resM)
                  {
                    printf("\nNUM OF ROWS FROM %s IS  = %d\n\n",mhost,mysql_num_rows(resM));
                    //GET ALL KEYS FROM SLAVE TABLE
                    prn_time();
                    if (mysql_real_query (&dbS, queryTwo, strlen(queryTwo)))
                      {
                        printf("THIS query failed: %s\n%s\n",queryTwo,mysql_error(&dbS));

                      }else{


                        resS=mysql_store_result(&dbS);

                        if(resS)
                          {
                            printf("\nNUM OF ROWS FROM %s IS  = %d\n\n",shost,mysql_num_rows(resS));
                            prn_time();
                            printPrimaryKeys(resS);

                            int cmp=0;

                                        while(1){
                                          if (cmp<=0)
                                            {
                                              rowM =  mysql_fetch_row(resM);
                                            }
                                          if (cmp >=0)
                                            {
                                              rowS = mysql_fetch_row(resS);
                                            }
                                          if (rowM == 0 && rowS == 0) break;
                                          cmp = rowCmp(rowM,rowS,resS);
                                          if (cmp>0)
                                            {
                                              printRow(rowS,resS,1);
                                              printf("is missing from %s\n",mhost);
                                              val=2;
                                            }
                                          if (cmp<0)
                                            {
                                              printRow(rowM, resM,1);
                                              printf("is missing from %s\n",shost);
                                              val=2;
                                            }
                                        }
                          }else{
```

```
                            printf("NO DATA RETURNED FOR %s\n",table);
                          }
                      }
                  }

                cleanUp(&dbH,&dbS,val);
}
```

4.2.1.4 void printPrimaryKeys (MYSQL_RES ∗ *res*)

Definition at line 248 of file eventCnts.c.

```
{
  printf("primary keys are:\n\n");
  for(int x = 0; x<mysql_num_fields(res); x++)
    {
      field2 = mysql_fetch_field(res);
      if(IS_PRI_KEY(field2->flags))
        {
          printf("***%s***\t\t",field2->name);
        }
    }
  printf("\n");
}
```

4.2.1.5 void printRow (MYSQL_ROW *row*, MYSQL_RES ∗ *res*, int *flag*)

Definition at line 237 of file eventCnts.c.

```
{
  //PRINT OUT A ROW FROM a RESULT SET
  int num = mysql_num_fields(res);

for (int i=0 ; i<mysql_num_fields(res) ; i++)
       {
         // printf("\t%s",field->name );
          printf("\t***%s***\n",row[i] );
       }
}
```

4.2.1.6 int rowCmp (MYSQL_ROW *str1*, MYSQL_ROW *str2*, MYSQL_RES ∗ *res*)

Definition at line 214 of file eventCnts.c.

```
{
  if (str1 == 0) return 1;
  if (str2 == 0) return -1;

  for (int i=0; i<mysql_num_fields(resM);i++)
    {

     if(IS_NUM(field->type)){
       float a = atof(str1[i]);
       float b = atof(str2[i]);
       float ans = a - b;
       if (ans) return ans;
     }else{
```

```

        int ans = strcmp(str1[i],str2[i]);
        if (ans) return ans;
      }
    }
  return 0;

}
```

4.3 insTrgCnt.h File Reference

Functions

- int tCounter (int, MYSQL ∗, MYSQL ∗, MYSQL ∗, int)
- int rundone (int, MYSQL ∗, MYSQL ∗)
- int getX (MYSQL ∗, char ∗)
- int updateFin (MYSQL ∗, int, int)
- int checkTags (MYSQL ∗, int)
- void cleanUp (MYSQL ∗, MYSQL ∗, MYSQL ∗, int, int)

Variables

- MYSQL_RES ∗ res
- MYSQL_RES ∗ res2
- MYSQL_ROW row
- int offlineBit
- int trgVersion
- int threshVersion
- int idx_trigger
- int fileSequence
- int fileStream
- int tCount
- char ∗ name
- int count = 0
- char buff [255]

4.3.1 Function Documentation

4.3.1.1 int checkTags (MYSQL ∗, int)

Definition at line 428 of file N_insTrgCnt.cc.

```
429 {
430   int event = 0;
431   int runTag=0;
432   char query5[255];
433   char query6[255];
434   // MYSQL_RES *res;
435   //MYSQL_ROW row;
436
437   sprintf (query5,"select run from daqEventTag where run = %d",run);
438   sprintf (query6,"select run from daqRunTag where run = %d",run);
439
440   if (mysql_real_query (db, query5, strlen(query5)))
441     {
442        printf("check 1 query failed %s\n",query5);
443       event=0;
444     }else{
445       res = mysql_store_result(db);
446       row = mysql_fetch_row(res);
447       if(row)
448         event=1;
449
450       }
```

```
451    mysql_free_result(res);
452
453
454   if (mysql_real_query (db, query6, strlen(query6)))
455     {
456       int erno = mysql_errno(db);
457       const char* err = mysql_error(db);
458       printf("check 2 query failed %s \t%d\t%s\n",query6,erno,err);
459       runTag=0;
460     }else{
461
462
463     res = mysql_store_result(db);
464     row = mysql_fetch_row(res);
465     if(row)
466       runTag=1;
467     }
468
469    mysql_free_result(res);
470
471    //  printf("runtag = %d and event =%d, run = %d \n",runTag,event,run);
472   if ((runTag && event))
473     return 1;
474   else{ return 0;}
475 }
```

4.3.1.2 void cleanUp (MYSQL *, MYSQL *, MYSQL *, int, int)

Definition at line 477 of file N_insTrgCnt.cc.

```
478 {
479
480   switch (value)
481   {
482     case 1:
483       printf("%d: \t All runs are finished \n", time(NULL));
484       break;
485     case 2:
486       printf("%d: \t end.fin updated to 3, run %d will be rechecked\n", time(NULL),run);
487       break;
488     case 3:
489       printf("%d: \t end.fin updated to 1, with triggerCount for run %d\n",time(NULL),run);
490       break;
491     case 4:
492       printf("%d: \t end.fin updated to 2 for run %d\n",time(NULL),run);
493       break;
494     case 5:
495        printf("%d: \t end.fin update to 1,with NO triggerCount update for run %d\n",time(NULL),run);
496        break;
497     case 6:
498       printf("%d: \t General Query Error ^^^^^message^^^^^ %d\n",time(NULL),run);
499       break;
500     case 7:
501       printf("%d: \t end.fin RESET to 0 %d\n",time(NULL),run);
502       break;
503   case 8:
504     printf("%d: \t end.fin updated to 2, No recorded RUNS, No trigger Updates for run %d\n",time(NULL)
505     break;
506     default:
507       printf("BAD STUFF");
508   }
509
510
511   mysql_close(db_conditions);
```

```
  mysql_close(db_runlog);
  mysql_close(db_trigger);

  exit(1);
}
```

4.3.1.3 int getX (MYSQL *, char *)

Definition at line 349 of file N_insTrgCnt.cc.

```
{
  //  MYSQL_RES *res;
  //MYSQL_ROW row;
  int x;
  if(mysql_real_query(db, query, strlen(query))) {
    printf("Error executing query: %s\n",query);
    return(0);
  }
  res = mysql_store_result(db);
  if(res == NULL) {
    printf("Error storing (%s) for query %s\n",mysql_error(db),query);
    return(0);
  }
  row = mysql_fetch_row(res);
  if(row){
    x= atoi(row[0]);
  }else{ return 0;}
  mysql_free_result(res);

  return x;
}
```

4.3.1.4 int rundone (int, MYSQL *, MYSQL *)

Definition at line 267 of file N_insTrgCnt.cc.

```
{

static char *dbname = "Conditions_rts";
static char *cquery = "select offlineBit, name from triggers where idx_rn=";

  int dbdone=0;

  int nevts;
  int nevts_et;
  int end_testcount;
  int nevbs;
  int dest;
  int ntags;

  char query[255];

  // check config to get list of evbs
  // check end_test run record
  // get count of events from evbs
  // get count of events from eventTag
  sprintf(query, "select count(*) from end where idx_rn=%d",run);
  end_testcount = getX(db_conditions, query);

  if(end_testcount == 0) {
    printf("No end_test run record\n");
```

```
    dbdone = 0;
    goto done;
  }

  sprintf(query, "select count(*) from nodes where task=20 and idx_rn=%d",run);
  nevbs = getX(db_conditions, query);

  if(nevbs == 0)  {
    printf("No evbs in run %d.  Dbs as done as they will get.\n",run);
    dbdone = 1;
    goto done;
  }

  // Get the data destination

  sprintf(query, "select destination from run where idx_rn=%d",run);
  dest = getX(db_conditions, query);

  if((dest != 2) &&
     (dest != 4) &&
     (dest != 3)) {

    printf("No db records will be written for this run, run %d (data dest=%d)\n",run,dest);
    dbdone = 1;
    goto done;
  }

  sprintf(query, "select count(*) from daqRunTag where whichtag=1 and run=%d",run);
  ntags = getX(db_runlog, query);

  if(ntags != nevbs) {
    printf("The number of run tags for run %d does not match the number of evbs %d vs %d\n", run, ntag
    dbdone = 0;
    goto done;
  }

  sprintf(query, "select sum(totalNumberOfEvents) from daqRunTag where run=%d\n",run);
  nevts = getX(db_runlog, query);

  sprintf(query, "select count(*) from daqEventTag where run=%d\n",run);
  nevts_et = getX(db_runlog, query);

  if(nevts != nevts_et) {
    printf("Run tag and event tag disagree on number of events in run %d: %d vs %d\n",
           run, nevts, nevts_et);
    dbdone = 0;
    goto done;
  }

  dbdone = 1;
 done:

  printf(dbdone ? "Run DBs are done\n" : "Run DBs are not done\n");
  return(dbdone);
}
```

4.3.1.5 int tCounter (int, MYSQL *, MYSQL *, MYSQL *, int)

Definition at line 172 of file N_insTrgCnt.cc.

```
{
  int val;
  int updated;
  int updated2;
```

```
char query[255];
char query2[255];
char query3[255];


 sprintf(query,"SELECT name, trgVersion, threshVersion, offlineBit, idx_trigger from triggers where
        if (mysql_real_query (db_conditions, query, strlen(query)))
          {
            printf("THIS query failed: %s\n%s\n",query,mysql_error(db_conditions));
            return(6);
          }else{
            res= mysql_store_result(db_conditions);
            if (res)
              {
                while (( row = mysql_fetch_row (res)) !=NULL)
                  {
                    for (int i = 0; i< mysql_num_fields(res); i++)
                      {
                        name = row[0];
                        trgVersion = atoi(row[1]);
                        threshVersion = atoi(row[2]);
                        offlineBit =  atoi(row[3]);
                        idx_trigger =  atoi(row[4]);
                      }
                    sprintf(query2,"SELECT count(*), fileStream, fileSequence from daqEventTag where
                        (void) mysql_real_escape_string (db_runlog,buff,query2,strlen(query2));

                        if (mysql_real_query (db_runlog, buff, strlen(buff)))
                          {
                            printf("query2 failed: %s\n(%s)\n",query2,mysql_error(db_runlog));
                            return(6);
                          }
                        res2=mysql_store_result(db_runlog);
                        if (res2)
                          {
                            while ((row = mysql_fetch_row (res2)) !=NULL)
                              {
                                for (int j = 0; j < mysql_num_fields (res2);j++)
                                  {
                                    tCount = atoi(row[0]);
                                    fileStream = atoi(row[1]);
                                    fileSequence = atoi(row[2]);
                                   }
                                sprintf(query3,"INSERT into triggerCount set name = '%s', trgVersion
                                if (mysql_real_query (db_trigger, query3, strlen(query3)))
                                  {
                                    printf("insert failed query = %s\n%s\n",query3,mysql_error(db_tr
                                    return(6);

                                  }else{
                                     count++;
                                  }
                              }
                          }else{
                            printf("INNER query returned nothing\n");
                            return(6);
                                  }
                      }


            }else{
              if(!two){
                        updated = updateFin(db_conditions, run,1);
              }else{
                updated = updateFin(db_conditions, run,2);
              }
                        if(updated){
```

```
                            return(5);
                          }else{
                            printf("\nupdate failed for run %d\n", run);
                            return(6);
                          }
                        }
              if(!two){
                 updated2 = updateFin(db_conditions, run,1);
              }else{
                updated2 =  updateFin(db_conditions, run,2);
              }
            if(updated2)
            {
              printf("triggerCount updated with %d inserts \n",count);
              return(3);
            }else{
              printf("update failed for run %d\n%s\n", run,mysql_error(db_conditions));
              return(6);
            }
        }
}
```

4.3.1.6 int updateFin (MYSQL *, int, int)

Definition at line 372 of file N_insTrgCnt.cc.

```
{
 char query4[255];
 char query7[255];
 char query8[255];
 char query9[255];

 switch (val)
   {
   case 1:
    sprintf(query4,"UPDATE end set entryTime = entryTime, fin = 1 where idx_rn = %d",run);
          if (mysql_real_query (db, query4, strlen(query4)))
            {
              printf("updatefailed query: %s",query4);
              return(0);
            }else{
              return(1);
            }
          break;
   case 2:
     sprintf(query7,"UPDATE end set entryTime = entryTime, fin = 2 where idx_rn = %d",run);
          if (mysql_real_query (db, query7, strlen(query7)))
            {
              printf("updatefailed query: %s",query7);
              return(0);
            }else{
              return(1);
            }
          break;
   case 3:
     sprintf(query8,"UPDATE end set entryTime = entryTime, fin = 3 where idx_rn = %d",run);
          if (mysql_real_query (db, query8, strlen(query8)))
            {
              printf("updatefailed query: %s",query8);
              return(0);
            }else{
              return(1);
            }
          break;
```

```
case 4:
    sprintf(query9,"UPDATE end set entryTime = entryTime, fin = 0 where idx_rn = %d",run);
         if (mysql_real_query (db, query9, strlen(query9)))
           {
             printf("updatefailed query: %s",query9);
             return(0);
           }else{
             return(1);
           }
         break;
   default:
     printf("BAD STUFF");
       return(0);
   }

}
```

4.3.2 Variable Documentation

4.3.2.1 char buff[255]

Definition at line 16 of file insTrgCnt.h.

4.3.2.2 int count = 0

Definition at line 15 of file insTrgCnt.h.

4.3.2.3 int fileSequence

Definition at line 13 of file insTrgCnt.h.

4.3.2.4 int fileStream

Definition at line 13 of file insTrgCnt.h.

4.3.2.5 int idx_trigger

Definition at line 13 of file insTrgCnt.h.

4.3.2.6 char* name

Definition at line 14 of file insTrgCnt.h.

4.3.2.7 int offlineBit

Definition at line 13 of file insTrgCnt.h.

4.3.2.8 MYSQL_RES* res

Definition at line 11 of file insTrgCnt.h.

4.3.2.9 MYSQL_RES * res2

Definition at line 11 of file insTrgCnt.h.

4.3.2.10 MYSQL_ROW row

Definition at line 12 of file insTrgCnt.h.

4.3.2.11 int tCount

Definition at line 13 of file insTrgCnt.h.

4.3.2.12 int threshVersion

Definition at line 13 of file insTrgCnt.h.

4.3.2.13 int trgVersion

Definition at line 13 of file insTrgCnt.h.

4.4 N_insTrgCnt.cc File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <mysql.h>
#include <time.h>
#include <iostream>
#include "insTrgCnt.h"
```

Functions

- int main (int argc, char ∗argv[ ])
- int tCounter (int run, MYSQL ∗db_conditions, MYSQL ∗db_runlog, MYSQL ∗db_trigger, int two)
- int rundone (int run, MYSQL ∗db_conditions, MYSQL ∗db_runlog)
- int getX (MYSQL ∗db, char ∗query)
- int updateFin (MYSQL ∗db, int run, int val)
- int checkTags (MYSQL ∗db, int run)
- void cleanUp (MYSQL ∗db_conditions, MYSQL ∗db_runlog, MYSQL ∗db_trigger, int value, int run)

4.4.1 Function Documentation

4.4.1.1 int checkTags (MYSQL ∗ *db*, int *run*)

Definition at line 428 of file N_insTrgCnt.cc.

```
429 {
430   int event = 0;
431   int runTag=0;
432   char query5[255];
433   char query6[255];
434   // MYSQL_RES *res;
435   //MYSQL_ROW row;
436
437   sprintf (query5,"select run from daqEventTag where run = %d",run);
438   sprintf (query6,"select run from daqRunTag where run = %d",run);
439
440   if (mysql_real_query (db, query5, strlen(query5)))
441     {
442        printf("check 1 query failed %s\n",query5);
443       event=0;
444     }else{
445       res = mysql_store_result(db);
446       row = mysql_fetch_row(res);
447       if(row)
448         event=1;
449
450       }
451    mysql_free_result(res);
452
453
454   if (mysql_real_query (db, query6, strlen(query6)))
455     {
```

```
      int erno = mysql_errno(db);
      const char* err = mysql_error(db);
      printf("check 2 query failed %s \t%d\t%s\n",query6,erno,err);
      runTag=0;
    }else{


    res = mysql_store_result(db);
    row = mysql_fetch_row(res);
    if(row)
      runTag=1;
    }

   mysql_free_result(res);

   //  printf("runtag = %d and event =%d, run = %d \n",runTag,event,run);
  if ((runTag && event))
    return 1;
  else{ return 0;}
}
```

4.4.1.2 void cleanUp (MYSQL ∗ *db_conditions*, MYSQL ∗ *db_runlog*, MYSQL ∗ *db_trigger*, int *value*, int *run*)

Definition at line 477 of file N_insTrgCnt.cc.

```
{

  switch (value)
  {
    case 1:
      printf("%d: \t All runs are finished \n", time(NULL));
      break;
    case 2:
      printf("%d: \t end.fin updated to 3, run %d will be rechecked\n", time(NULL),run);
      break;
    case 3:
      printf("%d: \t end.fin updated to 1, with triggerCount for run %d\n",time(NULL),run);
      break;
    case 4:
      printf("%d: \t end.fin updated to 2 for run %d\n",time(NULL),run);
      break;
    case 5:
       printf("%d: \t end.fin update to 1,with NO triggerCount update for run %d\n",time(NULL),run);
       break;
    case 6:
      printf("%d: \t General Query Error ^^^^^message^^^^^ %d\n",time(NULL),run);
      break;
    case 7:
      printf("%d: \t end.fin RESET to 0 %d\n",time(NULL),run);
      break;
  case 8:
    printf("%d: \t end.fin updated to 2, No recorded RUNS, No trigger Updates for run %d\n",time(NULL)
    break;
    default:
      printf("BAD STUFF");
  }


  mysql_close(db_conditions);
  mysql_close(db_runlog);
  mysql_close(db_trigger);

  exit(1);
```

```
}
```

4.4.1.3 int getX (MYSQL ∗ *db*, char ∗ *query*)

Definition at line 349 of file N_insTrgCnt.cc.

```
{
  //  MYSQL_RES *res;
  //MYSQL_ROW row;
  int x;
  if(mysql_real_query(db, query, strlen(query))) {
    printf("Error executing query: %s\n",query);
    return(0);
  }
  res = mysql_store_result(db);
  if(res == NULL) {
    printf("Error storing (%s) for query %s\n",mysql_error(db),query);
    return(0);
  }
  row = mysql_fetch_row(res);
  if(row){
    x= atoi(row[0]);
  }else{ return 0;}
  mysql_free_result(res);

  return x;
}
```

4.4.1.4 int main (int *argc*, char ∗ *argv*[ ])

Definition at line 38 of file N_insTrgCnt.cc.

```
 {

//   if(argc != 2) {
//     printf("%s <run>\n",argv[0]);
//     return(0);
//   }

//   int run = atoi(argv[1]);
//   if(run == 0) {
//     printf("Invalid run %s\n",argv[1]);
//     return(0);
//   }

 static MYSQL db_conditions;
 static MYSQL db_runlog;
 static MYSQL db_trigger;


 char query[255];
 char query2[255];
 char query3[255];
 char requery[255];

 char buff2[255];
 char new_trig[35];
 char queryTime[255];

 int eVal, boo;

```

```

unsigned int field_count;

if(!mysql_real_connect(&db_conditions,
                       "onlsun1",
                       "jml",
                       NULL,
                       "Conditions_rts",
                       3501,
                       NULL,
                       0)) {
  printf("Can't connect to Conditions_rts\n");
  return(0);
}

if(!mysql_real_connect(&db_runlog,
                       "onlsun1",
                       "jml",
                       NULL,
                       "RunLog_daq",
                       3503,
                       NULL,
                       0)) {
  printf("Can't connect to RunLog_daq\n");
  mysql_close(&db_conditions);
  return(0);
}

if(!mysql_real_connect(&db_trigger,
                       "db01",
                       "deph",
                       NULL,
                        "trigger",
                        3316,
                        NULL,
                        0)) {
   printf("Can't Connect to trigger\n");
   mysql_close(&db_conditions);
   mysql_close(&db_runlog);
   return(0);
 }

 sprintf(query,"SELECT idx_rn from end where fin = 0 limit 1");
 int runChecked = getX(&db_conditions, query);

 if(!runChecked)
   {
     sprintf(requery,"SELECT idx_rn from end where fin = 3 limit 1");
     int runReChecked = getX(&db_conditions, requery);
     if(!runReChecked)
       {
         cleanUp(&db_conditions,&db_runlog,&db_trigger,1,runChecked);
       }else{
          int updated2 = updateFin(&db_conditions, runReChecked,4);
          cleanUp(&db_conditions,&db_runlog,&db_trigger,7,runReChecked);
       }
   }else{

  int insRun = rundone(runChecked, &db_conditions, &db_runlog);
  if(!insRun)
     {
       sprintf(queryTime,"SELECT  idxoptime+28800 from end where idx_rn =%d",runChecked);
        if (mysql_real_query (&db_conditions, queryTime, strlen(queryTime)))
        {
             printf("THIS query failed: %s\n%s\n",query,mysql_error(&db_conditions));
             cleanUp(&db_conditions,&db_runlog,&db_trigger,6,runChecked);
        }
```

```
135            res=mysql_store_result(&db_conditions);
136            if (res)
137              {
138                row= mysql_fetch_row (res);
139                if(time(NULL)>atoi(row[0]))
140                {
141                  int t = updateFin(&db_conditions, runChecked, 2);
142                    if(t)
143                      {
144                        boo = tCounter(runChecked, &db_conditions, &db_runlog, &db_trigger,t);
145
146                        cleanUp(&db_conditions,&db_runlog,&db_trigger,4, runChecked);
147                      }
148                }else{
149                  int holdIt = updateFin(&db_conditions, runChecked, 3);
150                    if(holdIt)
151                      {
152                        cleanUp(&db_conditions,&db_runlog,&db_trigger,2, runChecked);
153                      }
154                    }
155              }
156         }else{
157         if(checkTags(&db_runlog, runChecked))
158         {
159           eVal = tCounter(runChecked, &db_conditions,&db_runlog,&db_trigger,0);
160           cleanUp(&db_conditions,&db_runlog,&db_trigger,eVal,runChecked);
161         }else{
162           int skipIt = updateFin(&db_conditions, runChecked, 2);
163           if(skipIt)
164             {
165               cleanUp(&db_conditions,&db_runlog, &db_trigger,8,runChecked);
166             }
167         }
168       }
169     }
170   }
```

4.4.1.5 int rundone (int *run*, MYSQL ∗ *db_conditions*, MYSQL ∗ *db_runlog*)

Definition at line 267 of file N_insTrgCnt.cc.

```
268 {
269
270 static char *dbname = "Conditions_rts";
271 static char *cquery = "select offlineBit, name from triggers where idx_rn=";
272
273   int dbdone=0;
274
275   int nevts;
276   int nevts_et;
277   int end_testcount;
278   int nevbs;
279   int dest;
280   int ntags;
281
282   char query[255];
283
284   // check config to get list of evbs
285   // check end_test run record
286   // get count of events from evbs
287   // get count of events from eventTag
288   sprintf(query, "select count(*) from end where idx_rn=%d",run);
289   end_testcount = getX(db_conditions, query);
290
```

```
291   if(end_testcount == 0) {
292     printf("No end_test run record\n");
293     dbdone = 0;
294     goto done;
295   }
296
297   sprintf(query, "select count(*) from nodes where task=20 and idx_rn=%d",run);
298   nevbs = getX(db_conditions, query);
299
300   if(nevbs == 0)  {
301     printf("No evbs in run %d.  Dbs as done as they will get.\n",run);
302     dbdone = 1;
303     goto done;
304   }
305
306   // Get the data destination
307
308   sprintf(query, "select destination from run where idx_rn=%d",run);
309   dest = getX(db_conditions, query);
310
311   if((dest != 2) &&
312      (dest != 4) &&
313      (dest != 3)) {
314
315     printf("No db records will be written for this run, run %d (data dest=%d)\n",run,dest);
316     dbdone = 1;
317     goto done;
318   }
319
320   sprintf(query, "select count(*) from daqRunTag where whichtag=1 and run=%d",run);
321   ntags = getX(db_runlog, query);
322
323   if(ntags != nevbs) {
324     printf("The number of run tags for run %d does not match the number of evbs %d vs %d\n", run, ntag
325     dbdone = 0;
326     goto done;
327   }
328
329   sprintf(query, "select sum(totalNumberOfEvents) from daqRunTag where run=%d\n",run);
330   nevts = getX(db_runlog, query);
331
332   sprintf(query, "select count(*) from daqEventTag where run=%d\n",run);
333   nevts_et = getX(db_runlog, query);
334
335   if(nevts != nevts_et) {
336     printf("Run tag and event tag disagree on number of events in run %d: %d vs %d\n",
337            run, nevts, nevts_et);
338     dbdone = 0;
339     goto done;
340   }
341
342   dbdone = 1;
343  done:
344
345   printf(dbdone ? "Run DBs are done\n" : "Run DBs are not done\n");
346   return(dbdone);
347 }
```

4.4.1.6 int tCounter (int *run*, MYSQL ∗ *db_conditions*, MYSQL ∗ *db_runlog*, MYSQL ∗ *db_trigger*, int *two*)

Definition at line 172 of file N_insTrgCnt.cc.

```
173 {
```

```
int val;
int updated;
int updated2;
char query[255];
char query2[255];
char query3[255];


 sprintf(query,"SELECT name, trgVersion, threshVersion, offlineBit, idx_trigger from triggers where
        if (mysql_real_query (db_conditions, query, strlen(query)))
          {
            printf("THIS query failed: %s\n%s\n",query,mysql_error(db_conditions));
            return(6);
          }else{
            res= mysql_store_result(db_conditions);
            if (res)
              {
                while (( row = mysql_fetch_row (res)) !=NULL)
                  {
                    for (int i = 0; i< mysql_num_fields(res); i++)
                      {
                        name = row[0];
                        trgVersion = atoi(row[1]);
                        threshVersion = atoi(row[2]);
                        offlineBit =  atoi(row[3]);
                        idx_trigger =  atoi(row[4]);
                      }
                    sprintf(query2,"SELECT count(*), fileStream, fileSequence from daqEventTag where
                        (void) mysql_real_escape_string (db_runlog,buff,query2,strlen(query2));

                        if (mysql_real_query (db_runlog, buff, strlen(buff)))
                          {
                            printf("query2 failed: %s\n(%s)\n",query2,mysql_error(db_runlog));
                            return(6);
                          }
                        res2=mysql_store_result(db_runlog);
                        if (res2)
                          {
                            while ((row = mysql_fetch_row (res2)) !=NULL)
                              {
                                for (int j = 0; j < mysql_num_fields (res2);j++)
                                  {
                                    tCount = atoi(row[0]);
                                    fileStream = atoi(row[1]);
                                    fileSequence = atoi(row[2]);
                                   }
                                sprintf(query3,"INSERT into triggerCount set name = '%s', trgVersion
                                if (mysql_real_query (db_trigger, query3, strlen(query3)))
                                  {
                                    printf("insert failed query = %s\n%s\n",query3,mysql_error(db_tr
                                    return(6);

                                  }else{
                                     count++;
                                  }
                              }
                          }else{
                            printf("INNER query returned nothing\n");
                            return(6);
                                  }
                       }


            }else{
              if(!two){
                        updated = updateFin(db_conditions, run,1);
              }else{
```

```
                  updated = updateFin(db_conditions, run,2);
                }
                          if(updated){
                            return(5);
                          }else{
                            printf("\nupdate failed for run %d\n", run);
                            return(6);
                          }
                        }
              if(!two){
                 updated2 = updateFin(db_conditions, run,1);
              }else{
                updated2 =  updateFin(db_conditions, run,2);
              }
            if(updated2)
            {
              printf("triggerCount updated with %d inserts \n",count);
              return(3);
            }else{
              printf("update failed for run %d\n%s\n", run,mysql_error(db_conditions));
              return(6);
            }
        }
}
```

4.4.1.7 int updateFin (MYSQL ∗ *db*, int *run*, int *val*)

Definition at line 372 of file N_insTrgCnt.cc.

```
{
 char query4[255];
 char query7[255];
 char query8[255];
 char query9[255];

 switch (val)
   {
   case 1:
    sprintf(query4,"UPDATE end set entryTime = entryTime, fin = 1 where idx_rn = %d",run);
          if (mysql_real_query (db, query4, strlen(query4)))
            {
              printf("updatefailed query: %s",query4);
              return(0);
            }else{
              return(1);
            }
          break;
   case 2:
     sprintf(query7,"UPDATE end set entryTime = entryTime, fin = 2 where idx_rn = %d",run);
          if (mysql_real_query (db, query7, strlen(query7)))
            {
              printf("updatefailed query: %s",query7);
              return(0);
            }else{
              return(1);
            }
          break;
   case 3:
     sprintf(query8,"UPDATE end set entryTime = entryTime, fin = 3 where idx_rn = %d",run);
          if (mysql_real_query (db, query8, strlen(query8)))
            {
              printf("updatefailed query: %s",query8);
              return(0);
            }else{
```

```
              return(1);
            }
          break;
 case 4:
     sprintf(query9,"UPDATE end set entryTime = entryTime, fin = 0 where idx_rn = %d",run);
          if (mysql_real_query (db, query9, strlen(query9)))
            {
              printf("updatefailed query: %s",query9);
              return(0);
            }else{
              return(1);
            }
          break;
   default:
     printf("BAD STUFF");
       return(0);
   }

}
```

4.5 resync.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <mysql.h>
#include <iostream>
#include "resync.h"
#include "time_keeper.h"
```

Functions

- int main (int argc, char *argv[ ])
- int rowCmp (MYSQL_ROW str1, MYSQL_ROW str2, MYSQL_RES *res)
- void printRow (MYSQL_ROW row, MYSQL_RES *res, int flag)
- void printPrimaryKeys (MYSQL_RES *res)
- void help ()
- void cleanUp (MYSQL *dbH, MYSQL *dbS, int value)

4.5.1 Function Documentation

4.5.1.1 void cleanUp (MYSQL * *dbH*, MYSQL * *dbS*, int *value*)

Definition at line 259 of file resync.c.

```
260 {
261   switch (value)
262     {
263     case 1:
264       printf( "\ntables are synchronized \n\n");
265       prn_time();
266       break;
267     case 2:
268       printf ("\ntables are NOT synchonized \n\n");
269       prn_time();
270       break;
271     default:
272       printf ("SOMETHING IS VERY WRONG! (cleanUp() switch default) \n");
273     }
274
275   mysql_close(dbH);
276   mysql_close(dbS);
277
278   exit(1);
279 }
```

4.5.1.2 void help ()

Definition at line 252 of file resync.c.

```
{
  printf("HELP\n\tresync checks two tables from two different servers (a master and a slave) to \n\t1)
  printf("\nUSAGE:\n\t>>resync databasename host1(master) port user table host2(slave) port\nFor examp

}
```

4.5.1.3 int main (int *argc*, char * *argv*[ ])

Definition at line 16 of file resync.c.

```
{
  if ((argc < 2 || strcmp(argv[1],"--help")==0 ))
    {
      help();
      return(1);
    }
  if (argc < 8)
    {
      printf("usage is incorrect - use --help for correct usage\n");
      return(1);
    }

  char * pass = NULL;
  char *db = argv[1];
  char *mhost = argv[2];
  int mport = atoi(argv[3]);
  char *user = argv[4];
  char *table = argv[5];
  char *shost = argv[6];
  int sport = atoi(argv[7]);

  if ( !strcmp(argv[7],"3336" ))
  {
   pass = "mirr0r";

  }

  char queryOne[40];
  char queryTwo[255];
  char queryAdd[155];
  char orderBy[155];
  char orderAdd[155];
  int val = 1;

  printf("\nFOR TABLE %s\n",table);
  if (!mysql_real_connect(&dbH,mhost,user,pass,db,mport,NULL,0))
    {
      printf("can't connect to Master %s %s %s\n",db,mhost,mport);
    }

  if (!mysql_real_connect(&dbS,shost,user,pass,db,sport,NULL,0))
    {
      printf("can't connect to Slave %s %s %s\n",db,shost,sport);
    }


  sprintf(queryOne,"SELECT * from %s limit 1",table);
  if (mysql_real_query (&dbH, queryOne, strlen(queryOne)))
         {
              printf("THIS query failed: %s\n%s\n",queryOne,mysql_error(&dbH));

         }
              res=mysql_store_result(&dbH);
              int max = mysql_num_fields(res);
```

```
                    char* fieldList[max];
                    char*   fieldNum[max];

     // initialize to all 0s
     for( int k =0 ; k<max; k++)
     {
        fieldList[k] = 0;
        fieldNum[k] = 0;
     }
     //GET the primary Keys separate them based on type
        int j =0;
                    field = mysql_fetch_fields (res);
                 for (int i = 0; i<mysql_num_fields(res) ; i++)
                 {
                    field = mysql_fetch_field (res);

                    if(IS_PRI_KEY(field->flags)){
                      if(IS_NUM(field->type)){
                        fieldNum[j] = field->name;
                        //                      printf("FIELD NUM NAME IS %s \n",field->name);
                      }else{
                        //  printf("FIELD NAME IS %s \n",field->name);
                      fieldList[j]= field->name;
                      }
                     j++;
                    }
                 }

 //GET THE NUMBER OF PRIMARY KEYS
                 int count = 0, numbers =0, characters = 0;
                 for (int m = 0; m < max; m++)
                   {
                     if (fieldList[m]){count++;}
                     if (fieldNum[m]){count++;}
                   }
 //strcat queryTwo here
                 sprintf(queryTwo,"SELECT ");
                 sprintf(orderBy, " ORDER BY ");

                 for (int n = 0; n < count; n++)
                   {
                     if (n < count-1)
                       {
                         if (fieldList[n]){
                           sprintf(queryAdd,"%s, ",fieldList[n]);
                           strcat(queryTwo,queryAdd);
                         }else{
                           sprintf(queryAdd,"%s, ",fieldNum[n]);
                           sprintf(orderAdd,"%s, ",fieldNum[n]);
                           strcat(orderBy,orderAdd);
                           strcat(queryTwo,queryAdd);
                         }
                       }else{
                         if (fieldList[n]){
                           sprintf(queryAdd,"%s FROM %s", fieldList[n],table);
                           strcat(queryTwo,queryAdd);
                         }else{
                           sprintf(queryAdd,"%s FROM %s", fieldNum[n],table);
                           sprintf(orderAdd,"%s",fieldNum[n]);
                           strcat(orderBy,orderAdd);
                           strcat(queryTwo,queryAdd);
                         }
                       }
                   }
                 strcat(queryTwo,orderBy);
                 //      printf("QUERY IS %s\n",queryTwo);

```

```
                //GET ALL KEYS FROM MASTER TABLE

                start_time();
                if (mysql_real_query (&dbH, queryTwo, strlen(queryTwo)))
                  {
                    printf("THIS query failed: %s\n%s\n",queryTwo,mysql_error(&dbH));

                  }else{
                    resM=mysql_store_result(&dbH);
                  }

                if(resM)
                  {
                    printf("\nNUM OF ROWS FROM %s IS  = %d\n\n",mhost,mysql_num_rows(resM));
                    //GET ALL KEYS FROM SLAVE TABLE
                    prn_time();
                    if (mysql_real_query (&dbS, queryTwo, strlen(queryTwo)))
                      {
                        printf("THIS query failed: %s\n%s\n",queryTwo,mysql_error(&dbS));

                      }else{


                        resS=mysql_store_result(&dbS);

                        if(resS)
                          {
                            printf("\nNUM OF ROWS FROM %s IS  = %d\n\n",shost,mysql_num_rows(resS));
                            prn_time();
                            printPrimaryKeys(resS);

                            int cmp=0;

                                        while(1){
                                          if (cmp<=0)
                                            {
                                              rowM =  mysql_fetch_row(resM);
                                            }
                                          if (cmp >=0)
                                            {
                                              rowS = mysql_fetch_row(resS);
                                            }
                                          if (rowM == 0 && rowS == 0) break;
                                          cmp = rowCmp(rowM,rowS,resS);
                                          if (cmp>0)
                                            {
                                              printRow(rowS,resS,1);
                                              printf("is missing from %s\n",mhost);
                                              val=2;
                                            }
                                          if (cmp<0)
                                            {
                                              printRow(rowM, resM,1);
                                              printf("is missing from %s\n",shost);
                                              val=2;
                                            }
                                        }
                          }else{
                            printf("NO DATA RETURNED FOR %s\n",table);
                          }
                      }
                  }

                cleanUp(&dbH,&dbS,val);
}
```

4.5.1.4 void printPrimaryKeys (MYSQL_RES ∗ *res*)

Definition at line 239 of file resync.c.

```
240 {
241   printf("primary keys are:\n\n");
242   for(int x = 0; x<mysql_num_fields(res); x++)
243     {
244       field2 = mysql_fetch_field(res);
245       if(IS_PRI_KEY(field2->flags))
246         {
247           printf("***%s***\t\t",field2->name);
248         }
249     }
250   printf("\n");
251 }
```

4.5.1.5 void printRow (MYSQL_ROW *row*, MYSQL_RES ∗ *res*, int *flag*)

Definition at line 228 of file resync.c.

```
229 {
230   //PRINT OUT A ROW FROM a RESULT SET
231   int num = mysql_num_fields(res);
232
233 for (int i=0 ; i<mysql_num_fields(res) ; i++)
234        {
235          // printf("\t%s",field->name );
236           printf("\t***%s***\n",row[i] );
237        }
238 }
```

4.5.1.6 int rowCmp (MYSQL_ROW *str1*, MYSQL_ROW *str2*, MYSQL_RES ∗ *res*)

Definition at line 205 of file resync.c.

```
206 {
207   if (str1 == 0) return 1;
208   if (str2 == 0) return -1;
209
210   for (int i=0; i<mysql_num_fields(resM);i++)
211     {
212
213      if(IS_NUM(field->type)){
214        float a = atof(str1[i]);
215        float b = atof(str2[i]);
216        float ans = a - b;
217        if (ans) return ans;
218      }else{
219
220         int ans = strcmp(str1[i],str2[i]);
221         if (ans) return ans;
222      }
223     }
224   return 0;
225
226 }
```

4.6 resync.h File Reference

Functions

- int rowCmp (MYSQL_ROW, MYSQL_ROW, MYSQL_RES ∗)
- void printRow (MYSQL_ROW, MYSQL_RES ∗, int)
- void printPrimaryKeys (MYSQL_RES ∗)
- void cleanUp (MYSQL ∗, MYSQL ∗, int)
- void help ()

Variables

- MYSQL dbH
- MYSQL dbS
- MYSQL_FIELD ∗ field
- MYSQL_FIELD ∗ field2
- MYSQL_RES ∗ res
- MYSQL_RES ∗ resM
- MYSQL_RES ∗ resS
- MYSQL_ROW rowM = 0 rowS = 0

4.6.1 Function Documentation

4.6.1.1 void cleanUp (MYSQL ∗, MYSQL ∗, int)

Definition at line 268 of file eventCnts.c.

```
269 {
270   switch (value)
271     {
272     case 1:
273       printf( "\ntables are synchronized \n\n");
274       prn_time();
275       break;
276     case 2:
277       printf ("\ntables are NOT synchonized \n\n");
278       prn_time();
279       break;
280     default:
281       printf ("SOMETHING IS VERY WRONG! (cleanUp() switch default) \n");
282     }
283
284   mysql_close(dbH);
285   mysql_close(dbS);
286
287   exit(1);
288 }
```

4.6.1.2 void help ()

Definition at line 261 of file eventCnts.c.

```
{
  printf("HELP\n\tresync checks two tables from two different servers (a master and a slave) to \n\t1)
  printf("\nUSAGE:\n\t>>resync databasename host1(master) port user table host2(slave) port\nFor examp

}
```

4.6.1.3 void printPrimaryKeys (MYSQL_RES *)

Definition at line 248 of file eventCnts.c.

```
{
  printf("primary keys are:\n\n");
  for(int x = 0; x<mysql_num_fields(res); x++)
    {
      field2 = mysql_fetch_field(res);
      if(IS_PRI_KEY(field2->flags))
        {
          printf("***%s***\t\t",field2->name);
        }
    }
  printf("\n");
}
```

4.6.1.4 void printRow (MYSQL_ROW, MYSQL_RES *, int)

Definition at line 237 of file eventCnts.c.

```
{
  //PRINT OUT A ROW FROM a RESULT SET
  int num = mysql_num_fields(res);

for (int i=0 ; i<mysql_num_fields(res) ; i++)
       {
         // printf("\t%s",field->name );
          printf("\t***%s***\n",row[i] );
       }
}
```

4.6.1.5 int rowCmp (MYSQL_ROW, MYSQL_ROW, MYSQL_RES *)

Definition at line 214 of file eventCnts.c.

```
{
  if (str1 == 0) return 1;
  if (str2 == 0) return -1;

  for (int i=0; i<mysql_num_fields(resM);i++)
    {

     if(IS_NUM(field->type)){
       float a = atof(str1[i]);
       float b = atof(str2[i]);
       float ans = a - b;
       if (ans) return ans;
     }else{

        int ans = strcmp(str1[i],str2[i]);
```

```
230         if (ans) return ans;
231      }
232     }
233   return 0;
234
235 }
```

4.6.2 Variable Documentation

4.6.2.1 MYSQL dbH [static]

Definition at line 3 of file resync.h.

4.6.2.2 MYSQL dbS [static]

Definition at line 3 of file resync.h.

4.6.2.3 MYSQL_FIELD* field

Definition at line 5 of file resync.h.

4.6.2.4 MYSQL_FIELD * field2

Definition at line 5 of file resync.h.

4.6.2.5 MYSQL_RES* res

Definition at line 6 of file resync.h.

4.6.2.6 MYSQL_RES * resM

Definition at line 6 of file resync.h.

4.6.2.7 MYSQL_RES * resS

Definition at line 6 of file resync.h.

4.6.2.8 MYSQL_ROW rowM = 0 rowS = 0

Definition at line 7 of file resync.h.

4.7 syncCol.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <mysql.h>
#include <iostream>
#include "syncCol.h"
#include "time_keeper.h"
```

Defines

- #define FILENAME "/star/u/deph/src/out.put"

Functions

- int main (int argc, char *argv[ ])
- int rowCmp (MYSQL_ROW str1, MYSQL_ROW str2, MYSQL_RES *res)
- void printRow (MYSQL_ROW row, MYSQL_RES *res, int flag, FILE *fptr)
- void printPrimaryKeys (MYSQL_RES *res)
- void help ()
- void cleanUp (MYSQL *dbH, MYSQL *dbS, int value)

4.7.1 Define Documentation

4.7.1.1 #define FILENAME "/star/u/deph/src/out.put"

Definition at line 18 of file syncCol.c.

4.7.2 Function Documentation

4.7.2.1 void cleanUp (MYSQL * *dbH*, MYSQL * *dbS*, int *value*)

Definition at line 319 of file syncCol.c.

```
320 {
321   switch (value)
322     {
323     case 1:
324       printf( "\ntables are synchronized \n\n");
325       prn_time();
326       break;
327     case 2:
328       printf ("\ntables are NOT synchonized \n\n");
329       prn_time();
330       break;
331     default:
332       printf ("SOMETHING IS VERY WRONG! (cleanUp() switch default) \n");
333     }
334
```

```
  mysql_close(dbH);
  mysql_close(dbS);


  exit(1);
}
```

4.7.2.2 void help ()

Definition at line 312 of file syncCol.c.

```
{
  printf("HELP\n\tcolsyn checks two columns to \n\t1) see if they are syncronized \n\t2) identify the
  printf("\nUSAGE:\n\t>>resync databasename host1(master) port user table host2(slave) port\nFor examp

}
```

4.7.2.3 int main (int *argc*, char ∗ *argv*[ ])

Definition at line 20 of file syncCol.c.

```
{
  if ((argc < 2 || strcmp(argv[1],"--help")==0 ))
    {
      help();
      return(1);
    }
  if (argc < 8)
    {
      printf("usage is incorrect - use --help for correct usage\n");
      return(1);
    }

  char * pass = NULL;
  char *db = argv[1];
  char *table1 = argv[2];
  char *mhost = argv[3];
  int mport = atoi(argv[4]);
  char *user = argv[5];
  char *table2 = argv[6];
  char *shost = argv[7];
  int sport = atoi(argv[8]);

  char * table;

  if ( !strcmp(argv[8],"3336" ))
  {
   pass = "mirr0r";

  }

  char queryOne[40];
  char queryTable1[40];
  char queryTable2[40];
  char queryTwo[255];
  char queryThree[255];
  char queryFour[255];
  char queryAdd[155];
  char orderBy[155];
  char orderAdd[155];
  int val = 1;
```

```

FILE *file_ptr;
file_ptr = fopen(FILENAME, "w");


printf("\nFOR TABLES %s and %s \n",table1,table2);
printf("\n\tWhich columns do you want to compare\n\n");



if (!mysql_real_connect(&dbH,mhost,user,pass,db,mport,NULL,0))
  {
    printf("can't connect to Master %s %s %s\n",db,mhost,mport);
  }

if (!mysql_real_connect(&dbS,shost,user,pass,db,sport,NULL,0))
  {
    printf("can't connect to Slave %s %s %s\n",db,shost,sport);
  }


sprintf(queryOne,"SELECT * from %s limit 1",table1);
sprintf(queryTwo,"SELECT * from %s limit 1",table2);
if (mysql_real_query (&dbH, queryOne, strlen(queryOne)))
        {
             printf("THIS query failed: %s\n%s\n",queryOne,mysql_error(&dbH));

        }
if (mysql_real_query (&dbS, queryTwo, strlen(queryTwo)))
        {
             printf("THIS query failed: %s\n%s\n",queryOne,mysql_error(&dbH));
        }

             res=mysql_store_result(&dbH);
             res2=mysql_store_result(&dbS);

             printf("%s columns are\n",table1);

             field1 = mysql_fetch_fields (res);
              for (int i = 0; i<mysql_num_fields(res) ; i++)
                {
                  field1 = mysql_fetch_field (res);
                  printf("FIELD NAME IS %s \n",field1->name);
                }

              printf("\n%s columns are\n",table2);

              field2 = mysql_fetch_fields (res2);
              for (int i = 0; i<mysql_num_fields(res2) ; i++)
                {
                  field2 = mysql_fetch_field (res2);
                  printf("FIELD NAME IS %s \n",field2->name);
                }

              int max = mysql_num_fields(res);
                   char* fieldList[max];
                   char * fieldNum[max];

     // initialize to all 0s
     for( int k =0 ; k<max; k++)
     {
        fieldList[k] = 0;fieldNum[k] = 0;
     }

     printf("MAX IS %d\n",max);
     printf("input each column <<return>> (one column per line) then type \"done\" <<return>> \n");
     char * x;
```

```
      char s[100];

    int cnt = 0;
    char *d ="done";

      printf("++++++\n");

      while(1)
        {

          fgets(s,100,stdin);
          s[strlen(s)-1]= '\0';
          // printf("%s\n",s);
          // printf("\n%s,%s\n",d,s);

          if(strcmp(d,s))
            {
              x = (char*) malloc(strlen(s)+1);
              strcpy(x,s);
              fieldList[cnt] = x;

              cnt++;
            }else{
              break;
            }
        }
      x=0;
      free(x);
      printf("you inputed: ");
      for (int i = 0; i < cnt ; i++)
        {
          printf("%s\n",fieldList[i]);
        }

//strcat queryTwo here
      sprintf(queryThree,"SELECT distinct ");
      sprintf(queryFour,"SELECT distinct ");
      sprintf(orderBy, " ORDER BY ");

      for (int n = 0; n < cnt; n++)
        {
          if (n < cnt-1)
            {
              if (fieldList[n]){
                sprintf(queryAdd,"%s, ",fieldList[n]);
                strcat(queryThree,queryAdd);
                strcat(queryFour,queryAdd);
                sprintf(orderAdd," %s, ",fieldList[n]);
                strcat(orderBy,orderAdd);
              }
            }else{
              if (fieldList[n]){
                sprintf(queryAdd,"%s FROM ", fieldList[n]);
                strcat(queryThree,queryAdd);
                strcat(queryFour,queryAdd);

                sprintf(orderAdd,"%s ",fieldList[n]);
                strcat(orderBy,orderAdd);
              }

              // strcat(queryThree,queryAdd);
            }
        }
      sprintf(queryTable1,"%s where fileStream > 99 ",table1);
      sprintf(queryTable2,"%s ",table2);
      strcat(queryThree,queryTable1);
      strcat(queryFour, queryTable2);
```

```
        strcat(queryThree,orderBy);
        strcat(queryFour, orderBy);
        printf("QUERY ONE IS %s\n",queryThree);
        printf("QUERY TWO IS %s\n",queryFour);

        //GET ALL KEYS FROM MASTER TABLE

        start_time();
        if (mysql_real_query (&dbH, queryThree, strlen(queryThree)))
          {
            printf("THIS query failed: %s\n%s\n",queryThree,mysql_error(&dbH));
          }else{
            resM=mysql_store_result(&dbH);
          }

        if(resM)
          {
            printf("\nNUM OF ROWS FROM %s IS  = %d\n\n",table1,mysql_num_rows(resM));
            //GET ALL KEYS FROM SLAVE TABLE
            prn_time();
            if (mysql_real_query (&dbS, queryFour, strlen(queryFour)))
              {
                printf("THIS query failed: %s\n%s\n",queryFour,mysql_error(&dbS));
              }else{

                resS=mysql_store_result(&dbS);

                if(resS)
                  {
                    printf("\nNUM OF ROWS FROM %s IS  = %d\n\n",table2,mysql_num_rows(resS));
                    prn_time();
                    //   printPrimaryKeys(resS);

                    int cmp=0;

                    while(1){
                      if (cmp<=0)
                        {
                          rowM =  mysql_fetch_row(resM);
                        }
                      if (cmp >=0)
                        {
                          rowS = mysql_fetch_row(resS);
                        }
                      if (rowM == 0 && rowS == 0) break;
                      cmp = rowCmp(rowM,rowS,resS);
                      if (cmp>0)
                        {
                          printRow(rowS,resS,1,file_ptr);
                          fprintf(file_ptr,"is missing from %s\n",table1);
                          val=2;
                        }
                      if (cmp<0)
                        {
                          printRow(rowM, resM,1,file_ptr);
                          fprintf(file_ptr,"is missing from %s\n",table2);
                          val=2;
                        }
                    }
                  }else{
                    printf("NO DATA RETURNED FOR %s\n",table);
                  }
              }
          }
        fclose(file_ptr);
        cleanUp(&dbH,&dbS,val);
}
```

4.7.2.4 void printPrimaryKeys (MYSQL_RES ∗ *res*)

Definition at line 299 of file syncCol.c.

```
{
  printf("primary keys are:\n\n");
  for(int x = 0; x<mysql_num_fields(res); x++)
    {
      field2 = mysql_fetch_field(res);
      if(IS_PRI_KEY(field2->flags))
        {
          printf("***%s***\t\t",field2->name);
        }
    }
  printf("\n");
}
```

4.7.2.5 void printRow (MYSQL_ROW *row*, MYSQL_RES ∗ *res*, int *flag*, FILE ∗ *fptr*)

Definition at line 288 of file syncCol.c.

```
{
  //PRINT OUT A ROW FROM a RESULT SET
  int num = mysql_num_fields(res);

for (int i=0 ; i<mysql_num_fields(res) ; i++)
       {
         // printf("\t%s",field->name );
          fprintf(fptr,"\t***%s***\n",row[i] );
       }
}
```

4.7.2.6 int rowCmp (MYSQL_ROW *str1*, MYSQL_ROW *str2*, MYSQL_RES ∗ *res*)

Definition at line 264 of file syncCol.c.

```
{
  if (str1 == 0) return 1;
  if (str2 == 0) return -1;

  for (int i=0; i<mysql_num_fields(res);i++)
    {

     if(IS_NUM(field1->type)){

       float a = atof(str1[i]);
       float b = atof(str2[i]);
       float ans = a - b;
       if (ans) return ans;
     }else{

        int ans = strcmp(str1[i],str2[i]);
        if (ans) return ans;
     }
    }
  return 0;

}
```

4.8 syncCol.h File Reference

Functions

- int rowCmp (MYSQL_ROW, MYSQL_ROW, MYSQL_RES ∗)
- void printRow (MYSQL_ROW, MYSQL_RES ∗, int, FILE ∗)
- void printPrimaryKeys (MYSQL_RES ∗)
- void cleanUp (MYSQL ∗, MYSQL ∗, int)
- void help ()

Variables

- MYSQL dbH
- MYSQL dbS
- MYSQL_FIELD ∗ field
- MYSQL_FIELD ∗ field1
- MYSQL_FIELD ∗ field2
- MYSQL_RES ∗ res
- MYSQL_RES ∗ res2
- MYSQL_RES ∗ resS
- MYSQL_RES ∗ resM
- MYSQL_ROW rowM = 0 rowS = 0

4.8.1 Function Documentation

4.8.1.1 void cleanUp (MYSQL ∗, MYSQL ∗, int)

Definition at line 268 of file eventCnts.c.

```
{
  switch (value)
    {
    case 1:
      printf( "\ntables are synchronized \n\n");
      prn_time();
      break;
    case 2:
      printf ("\ntables are NOT synchonized \n\n");
      prn_time();
      break;
    default:
      printf ("SOMETHING IS VERY WRONG! (cleanUp() switch default) \n");
    }

  mysql_close(dbH);
  mysql_close(dbS);

  exit(1);
}
```

4.8.1.2 void help ()

Definition at line 261 of file eventCnts.c.

```
{
  printf("HELP\n\tresync checks two tables from two different servers (a master and a slave) to \n\t1)
  printf("\nUSAGE:\n\t>>resync databasename host1(master) port user table host2(slave) port\nFor examp

}
```

4.8.1.3 void printPrimaryKeys (MYSQL_RES *)

Definition at line 248 of file eventCnts.c.

```
{
  printf("primary keys are:\n\n");
  for(int x = 0; x<mysql_num_fields(res); x++)
    {
      field2 = mysql_fetch_field(res);
      if(IS_PRI_KEY(field2->flags))
        {
          printf("***%s***\t\t",field2->name);
        }
    }
  printf("\n");
}
```

4.8.1.4 void printRow (MYSQL_ROW, MYSQL_RES *, int, FILE *)

Definition at line 288 of file syncCol.c.

```
{
  //PRINT OUT A ROW FROM a RESULT SET
  int num = mysql_num_fields(res);

for (int i=0 ; i<mysql_num_fields(res) ; i++)
       {
         // printf("\t%s",field->name );
          fprintf(fptr,"\t***%s***\n",row[i] );
       }
}
```

4.8.1.5 int rowCmp (MYSQL_ROW, MYSQL_ROW, MYSQL_RES *)

Definition at line 214 of file eventCnts.c.

```
{
  if (str1 == 0) return 1;
  if (str2 == 0) return -1;

  for (int i=0; i<mysql_num_fields(resM);i++)
    {

     if(IS_NUM(field->type)){
       float a = atof(str1[i]);
       float b = atof(str2[i]);
```

```
225         float ans = a - b;
226         if (ans) return ans;
227       }else{
228
229          int ans = strcmp(str1[i],str2[i]);
230          if (ans) return ans;
231       }
232      }
233    return 0;
234
235 }
```

4.8.2 Variable Documentation

4.8.2.1 MYSQL dbH [static]

Definition at line 3 of file syncCol.h.

4.8.2.2 MYSQL dbS [static]

Definition at line 3 of file syncCol.h.

4.8.2.3 MYSQL_FIELD* field

Definition at line 5 of file syncCol.h.

4.8.2.4 MYSQL_FIELD * field1

Definition at line 5 of file syncCol.h.

4.8.2.5 MYSQL_FIELD * field2

Definition at line 5 of file syncCol.h.

4.8.2.6 MYSQL_RES* res

Definition at line 6 of file syncCol.h.

4.8.2.7 MYSQL_RES * res2

Definition at line 6 of file syncCol.h.

4.8.2.8 MYSQL_RES * resM

Definition at line 6 of file syncCol.h.

4.8.2.9 MYSQL_RES * resS

Definition at line 6 of file syncCol.h.

4.8.2.10 MYSQL_ROW rowM = 0 rowS = 0

Definition at line 7 of file syncCol.h.

4.9 test.c File Reference

```
#include <mysql.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

Functions

- int main (int argc, char ∗argv[ ])

4.9.1 Function Documentation

4.9.1.1 int main (int *argc*, char ∗ *argv*[ ])

Definition at line 6 of file test.c.

```
7 {
8         static MYSQL db;
9         MYSQL_RES *res;
10         MYSQL_ROW row;
11         MYSQL_FIELD *field;
12         unsignef long col_len;
13
14         if(!mysql_real_connect(&db,
15                 "onlsun1",
16                 "stardb",
17                 NULL,
18                 "RunLog",
19                 3501,
20                 NULL,
21                 0)) {
22                 printf("Error: %s\n",mysql_error(&db));
23         }
24
25         char *q = "select a.runNumber, a.name, a.offlineTriggerID as offlineTrgId,format(a.prescale,1)
26         mysql_real_query(&db, q, strlen(q));
27
28         res = mysql_store_result(&db);
29         if(!res) printf("Error? %s (%s)\n", q, mysql_error(&db));
30
31         mysql_field_seek (res, 0);
32
33         for (unsigned int i; i < mysql_num_fields(res); i++)
34         {
35            field = mysql_fetch_field (res)
36            col_len = strlen (field->name);
37            if (col_len < field->max_length)
38                col_len = field->max_length;
39            if (col_len < 4 && !IS_NOT_NULL (field->flags))
40                col_len = 4;  /* 4 = length of the word "NULL" */
41              field->max_length = col_len; /* reset column info */
42          }
43
44 row = mysql_fetch_row(res);
45
46
47
48         if(row)
```

```
49                // printf("last run is %d\n",atoi(row[0]));
50                 printf("last run is %d\n",atoi(row[0]));
51
52         else
53                 printf("Error? %s\n",mysql_error(&db));
54         mysql_close(&db);
55 }
```

4.10 time_keeper.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
```

Classes

- struct time_keeper

Defines

- #define MAXSTRING 100

Functions

- void start_time (void)
- double prn_time (void)

Variables

- time_keeper tk

4.10.1 Define Documentation

4.10.1.1 #define MAXSTRING 100

Definition at line 10 of file time_keeper.c.

4.10.2 Function Documentation

4.10.2.1 double prn_time (void)

Definition at line 25 of file time_keeper.c.

```
26 {
27   char s1[MAXSTRING], s2[MAXSTRING];
28   int  field_width, n1,n2;
29   double clocks_per_second = (double) CLOCKS_PER_SEC, user_time, real_time;
30
31   user_time = (clock() - tk.save_clock) / clocks_per_second;
32   real_time = difftime(time(NULL), tk.save_time);
33
34   tk.save_clock = clock();
35   tk.save_time = time(NULL);
36
37   n1 = sprintf(s1, "%.1f", user_time);
38   n2 = sprintf(s2, "%.1f", real_time);
39
40   field_width=(n1>n2) ? n1 : n2 ;
```

```

  printf("%s%*.1f%s\n%s%*.1f%s\n\n",
         "User time: ", field_width, user_time, " seconds",
         "Real time: ", field_width, real_time, " seconds");
  return user_time;
}
```

4.10.2.2 void start_time (void)

Definition at line 19 of file time_keeper.c.

```
{
  tk.begin_clock = tk.save_clock = clock();
  tk.begin_time = tk.save_time = time(NULL);
}
```

4.10.3 Variable Documentation

4.10.3.1 time_keeper tk [static]

Definition at line 17 of file time_keeper.c.

4.11 time_keeper.h File Reference

Functions

- void start_time (void)
- void prn_time (void)

4.11.1 Function Documentation

4.11.1.1 void prn_time (void)

Definition at line 25 of file time_keeper.c.

```
26 {
27   char s1[MAXSTRING], s2[MAXSTRING];
28   int  field_width, n1,n2;
29   double clocks_per_second = (double) CLOCKS_PER_SEC, user_time, real_time;
30
31   user_time = (clock() - tk.save_clock) / clocks_per_second;
32   real_time = difftime(time(NULL), tk.save_time);
33
34   tk.save_clock = clock();
35   tk.save_time = time(NULL);
36
37   n1 = sprintf(s1, "%.1f", user_time);
38   n2 = sprintf(s2, "%.1f", real_time);
39
40   field_width=(n1>n2) ? n1 : n2 ;
41
42   printf("%s%*.1f%s\n%s%*.1f%s\n\n",
43          "User time: ", field_width, user_time, " seconds",
44          "Real time: ", field_width, real_time, " seconds");
45   return user_time;
46 }
```

4.11.1.2 void start_time (void)

Definition at line 19 of file time_keeper.c.

```
20 {
21   tk.begin_clock = tk.save_clock = clock();
22   tk.begin_time = tk.save_time = time(NULL);
23 }
```

Index

begin_clock
  time_keeper, 5
begin_time
  time_keeper, 5
boo.c, 7
buff
  insTrgCnt.h, 20

checkTags
  insTrgCnt.h, 14
  N_insTrgCnt.cc, 22
cleanUp
  eventCnts.c, 8
  insTrgCnt.h, 15
  N_insTrgCnt.cc, 23
  resync.c, 31
  resync.h, 36
  syncCol.c, 39
  syncCol.h, 45
count
  insTrgCnt.h, 20

dbH
  resync.h, 38
  syncCol.h, 47
dbS
  resync.h, 38
  syncCol.h, 47

eventCnts.c, 8
eventCnts.c
  cleanUp, 8
  help, 8
  main, 9
  printPrimaryKeys, 12
  printRow, 12
  rowCmp, 12

field
  resync.h, 38
  syncCol.h, 47
field1
  syncCol.h, 47
field2
  resync.h, 38
  syncCol.h, 47

FILENAME
  syncCol.c, 39
fileSequence
  insTrgCnt.h, 20
fileStream
  insTrgCnt.h, 20

getX
  insTrgCnt.h, 16
  N_insTrgCnt.cc, 24

help
  eventCnts.c, 8
  resync.c, 31
  resync.h, 36
  syncCol.c, 40
  syncCol.h, 45

idx_trigger
  insTrgCnt.h, 20
insTrgCnt.h, 14
insTrgCnt.h
  buff, 20
  checkTags, 14
  cleanUp, 15
  count, 20
  fileSequence, 20
  fileStream, 20
  getX, 16
  idx_trigger, 20
  name, 20
  offlineBit, 20
  res, 20
  res2, 20
  row, 21
  rundone, 16
  tCount, 21
  tCounter, 17
  threshVersion, 21
  trgVersion, 21
  updateFin, 19

main
  eventCnts.c, 9
  N_insTrgCnt.cc, 24
  resync.c, 32

syncCol.c, 40
test.c, 49
MAXSTRING
time_keeper.c, 51

N_insTrgCnt.cc, 22
N_insTrgCnt.cc
checkTags, 22
cleanUp, 23
getX, 24
main, 24
rundone, 26
tCounter, 27
updateFin, 29
name
insTrgCnt.h, 20

offlineBit
insTrgCnt.h, 20

printPrimaryKeys
eventCnts.c, 12
resync.c, 34
resync.h, 37
syncCol.c, 43
syncCol.h, 46
printRow
eventCnts.c, 12
resync.c, 35
resync.h, 37
syncCol.c, 44
syncCol.h, 46
prn_time
time_keeper.c, 51
time_keeper.h, 53

res
insTrgCnt.h, 20
resync.h, 38
syncCol.h, 47
res2
insTrgCnt.h, 20
syncCol.h, 47
resM
resync.h, 38
syncCol.h, 47
resS
resync.h, 38
syncCol.h, 47
resync.c, 31
cleanUp, 31
help, 31
main, 32
printPrimaryKeys, 34
printRow, 35
rowCmp, 35
resync.h, 36
cleanUp, 36
dbH, 38
dbS, 38
field, 38
field2, 38
help, 36
printPrimaryKeys, 37
printRow, 37
res, 38
resM, 38
resS, 38
rowCmp, 37
rowM, 38
row
insTrgCnt.h, 21
rowCmp
eventCnts.c, 12
resync.c, 35
resync.h, 37
syncCol.c, 44
syncCol.h, 46
rowM
resync.h, 38
syncCol.h, 47
rundone
insTrgCnt.h, 16
N_insTrgCnt.cc, 26

save_clock
time_keeper, 5
save_time
time_keeper, 5
start_time
time_keeper.c, 52
time_keeper.h, 53
syncCol.c, 39
syncCol.c
cleanUp, 39
FILENAME, 39
help, 40
main, 40
printPrimaryKeys, 43
printRow, 44
rowCmp, 44
syncCol.h, 45
syncCol.h
cleanUp, 45
dbH, 47
dbS, 47
field, 47
field1, 47

field2, 47
help, 45
printPrimaryKeys, 46
printRow, 46
res, 47
res2, 47
resM, 47
resS, 47
rowCmp, 46
rowM, 47

tCount
insTrgCnt.h, 21
tCounter
insTrgCnt.h, 17
N_insTrgCnt.cc, 27
test.c, 49
main, 49
threshVersion
insTrgCnt.h, 21
time_keeper, 5
begin_clock, 5
begin_time, 5
save_clock, 5
save_time, 5
time_keeper.c, 51
MAXSTRING, 51
prn_time, 51
start_time, 52
tk, 52
time_keeper.h, 53
prn_time, 53
start_time, 53
tk
time_keeper.c, 52
trgVersion
insTrgCnt.h, 21

updateFin
insTrgCnt.h, 19
N_insTrgCnt.cc, 29