KANDİLLİ EARTHQUAKE SEISMIC DATA SERVER AND ANALYSIS

by

Emin Şenay & Atilla Soner Balkır

Submitted to the Department of Computer Engineering

in partial fulfillment of the requirements

for the degree of

Bachelor of Science

in

Computer Engineering

Boğaziçi University

June 2007

TABLE OF CONTENTS

ABSTRACT

Project Name : Kandilli Earthquake Seismic Data Server and Analysis

Project Team : Emin Şenay, Atilla Soner Balkır

Term : Spring 2007

Keywords : Grid Computing, Kandilli Earthquake Analysis, TR-GRID, Earthquake, Kandilli Seismic Data Server, Google Map, SEE-GRID

Summary : This project is formed of two parts. First one is to develop an application running on TR-GRID which analyzes the seismic data and identifies the earthquakes by using the seismic data server located in TR-GRID. Seismic data server takes the seismic files produced by the stations of Kandilli Observatory and Earthquake Research Institute using mirroring software. The second part is about developing a web application which displays the obtained results in a user friendly way.

In this document, all relevant information will be provided in terms of progress and programming.

1. Introduction

This document is written for Cmpe 492 Senior Project course given by Computer Engineering Department at Boğaziçi University. This project is mainly aimed to develop grid-enabled applications with the collaboration of ULAKBIM and Kandilli Observatory and Earthquake Research Institute.

2. Previous Work

The project is based on a project done by Didem Unat in Cmpe 492 Course given in spring 2006 Semester. The project is composed of three parts. In the first part, a mirroring application was developed to copy the seismic data files (SAC file format) from the server of Kandilli Observatory and Earthquake Research Institute. The files are copied and registered into TR-GRID storage elements.

In our project, these registered files are going to be used by our application running on GRID infrastructure. The application will assume that all needed files are in the GRID infrastructure. The correctness of this assumption heavily depends on the robustness of the application developed by Didem Unat.

During the correctness tests of this application, we found out that some directories and seismic data files were not available for a few hours. On the other hand, lerek.ulakbim.gov.tr and TR-GRID infrastructure was shut down for a period of time between November and December 2006 because of a maintenance work. When we checked the data afterwards, we

have seen that all SAC files are properly registered to TR-GRID when the application started after the maintenance. This shows that, even though the application is designed to work robustly, it might have some defects leading to data incompleteness.

2.1. Structure of TR-GRID

The TR-GRID infrastructure needs a number of steps to be connected and run an application on it. There are two ways for connecting to TR-GRID. First one is using a preinstalled user interface by requesting an account and the second one is installing the user interface yourself. The user interface is actually a process running on a Linux computer; however Scientific Linux 3.0.5 is recommended.

We choose to use the first choice and took an account from the user interface server, levrek.ulakbim.gov.tr.

Connecting to levrek is done by ssh commands. For example:

ssh esenay@levrek.ulakbim.gov.tr

After connecting to the server, some environment variables are needed to be set. These are set with the following commands:

export LCG_CATALOG_TYPE=lfc

export LFC_HOST=lfc.ulakbim.gov.tr

The last step before connecting to the GRID is creating a proxy. The user interface computer and GRID are needed to be connected in a secure way, so using our digital certificates; we create a proxy with the given command:

voms-proxy-init --voms trgridd

trgridd: The virtual organization that we are connecting to.

After these commands, we can use the GRID with a number of tools containing LCG-Utils. Also, to run an application on grid architecture, job submission procedure should be followed.

2.1.1. Lcg Utilities

The LCG Utilities package is the main end user command line tool for data management provided by LCG. It includes UNIX like commands in order to perform regular operations on the grid architecture. The commands are basically variations of the UNIX commands prefixed with the ***lfc*** tag.

For example, in UNIX, to list the files in a directory, ***ls*** command is used. On grid architecture, the equivalent of this command is **lfc-ls**. A sample is as follows:

lfc-ls -l /grid/trgridd/

Similarly, in UNIX, to make a directory, ***mkdir*** command is used. On a grid architecture, the equivalent of this command is ***lfc-mkdir***. A sample is as follows:

lfc-mkdir /grid/trgridd/soner-emin

To copy a local file to the GRID and register it, the following command is used:

lcg-cr -v -d se.ulakbim.gov.tr -l lfn:/grid/trgridd/soner-emin/helloworld.jdl --vo trgridd file:///home_levrek/bouncmpe/esenay/20061223/491testjdl.jdl

To copy a file from TR GRID to the local system, the following command is used:

lcg-cp --vo trgridd lfn:/grid/trgridd/soner-emin/helloworld.jdl file:/home_levrek/bouncmpe/esenay/helloworld1.jdl

2.1.2. Job Submission

Running a program on the grid architecture is different from running a simple application on a normal PC and requires a few steps before the execution starts. The whole process is called ***job submission.*** Once a job is submitted to the GRID successfully, it is scheduled automatically by the GRID and then run. To submit a job to the GRID, a formal language called *JDL (Job Description Language)* is defined.

Before submitting a job, a JDL file is written that contains information about the name of the executable file, the standard input file, standard output file as well as the standard error file. Once a job finishes its execution, all the standard output is written to the standard output file instead of the screen. If during the execution, any error occurs, it is written to the standard error file. The executable program takes its input from the input file, which is also specified in the JDL file.

Apart from these, a simple JDL file contains the following lines:

Type: Type of the process.

JobType: Type of the job.

Executable: Name of the executable file.

StdOutput: Name of the standard output file.

StdError: Name of the standard error file.

InputSandbox: List of input files.

OutputSandbox: List of output files.

RetryCount: Number of retries.

A sample JDL file is as follows:

```
Type="Job";
JobType="Normal";
Executable = "fileop.o";
StdOutput = "std.out";
StdError = "std.err";
InputSandbox = {"fileop.o"};
OutputSandbox = {"output.txt"};
RetryCount = 3;
```

The following command is used for submitting a job to the GRID:

```
edg-job-submit 491testjdl.jdl
```

which yields an output as the following:

The job has been successfully submitted to the Network Server.

Use edg-job-status command to check job current status. Your job identifier (edg_jobId) is: https://rb.ulakbim.gov.tr:9000/E_XWzpmO2i4fk0cJpGecFw

Once a job is submitted, it is not run instantaneously. Since there may be many applications waiting to be run in the queue, the job is initially scheduled to be run. In order to see the status of an application on the GRID, the following command is used:

edg-job-status https://rb.ulakbim.gov.tr:9000/E_XWzpmO2i4fk0cJpGecFw

which yields an output as the following:

Status info for the Job: https://rb.ulakbim.gov.tr:9000/E_XWzpmO2i4fk0cJpGecFw
Current Status: Running
Status Reason: unavailable
Destination: ce.ulakbim.gov.tr:2119/jobmanager-lcgpbs-trgridd
reached on: Sat Dec 23 15:53:52 2006

2.2. About SEE-GRID

South Eastern European GRid-enabled eInfrastructure Development, a project to support and develop grid computing in Southeastern Europe, allowing countries there to participate in Pan-European and worldwide grid computing initiatives. SEE-GRID can also be used for running our application.

2.3. The Web Interface

The program running on the GRID only calculates and determines whether an earthquake has occurred in a specific area or not. However, its output is not reachable from the Internet so ordinary people are not able to see it.

We have designed a web interface which is updated regularly by our application running on the GRID. The application is going to update the information on the web on a regular basis, so people are going to be able to use it and check whether an earthquake has occurred or not.

Furthermore, we have also added another page that displays the earthquake analysis stations, which gather information on the geological movements in the ground and prepare the SAC files required as input by our application.

The web interface requires the following software tools to compile and run:

1- MySQL Database
2- Apache Tomcat Application Server
3- JSP, HTML
4- Google Maps API

We have designed two database tables for the web interface. The first one contains the stations information such as the name of the station the latitude and longitude values of the station, type of the station and the depth of the station. The second table contains the earthquake information such as the latitude and longitude values of the earthquake, the

magnitude of the earthquake and the date ant time values of the earthquake. There is also a third table in the database, but it is not concerned with the web interface. Detailed explanation about this database table can be found in the architecture section.

We preferred to use the Google Maps API since it is easy to code, well designed and contains detailed information about the places to be displayed. By using the Google Maps API, a user can click on the web interface to see the details of an earthquake or a station. Moreover, three kinds of displays have been included:

Map view

Satellite view

Hybrid view

The earthquakes are displayed according to their magnitudes so that earthquakes with larger magnitudes are displayed to cover a larger area on the map and earthquakes with a smaller magnitude cover a smaller area.

A search tool is designed to list the earthquake analysis stations according to their types and search the earthquakes that occurred on a given time interval, latitude and longitude values or magnitude. The interface initially displays everything on a detailed table which is able to sort the data according to the fields provided. The user can also click on the ***show the map*** link and see everything on the map visually.

The operation of the whole system can be summarized as follows. The application running on the GRID analyzes the SAC files and records all the earthquake information to the database

hourly. The web interface connects to the database and gathers the earthquake information which updated dynamically.

3. Goals

As it is written previously, the goal of this project is developing an application which runs on TR-GRID, analyzes the seismic data and identifies the earthquakes by using the data grid of Kandilli Observatory and Earthquake Research Institute. Also, the results obtained by the application should be displayed in a user friendly manner.

The algorithm of the analysis part is based on a previous algorithm running on a single PC in Kandilli Observatory and Earthquake Research Institute. This algorithm is modified to be run on a parallel architecture efficiently.

4. Advantages of the GRID

The most important reason for using the GRID is to accomplish a distributed seismic data server with a high bandwidth. If the data were only stored in the Kandilli Earthquake Observatory and Research Institute's web server, people trying to download and use the data might suffer from the low network speeds since the bandwidth is limited and shared amongst clients requesting data. However, in the GRID, the data is stored in several *storage nodes* instead of a single server. This provides a better resource management and a higher bandwidth as different parts of the data are stored in different storage nodes each with their own bandwidths. Another benefit comes from the organized structure of the data stored in the GRID. The seismic data files are catalogued and stored using a directory structure based on their retrieval dates from the seismic stations. This ensures that the users of the data handle it easier and without much effort. Moreover, the amount of seismic data keeps increasing as new files are retrieved from the earthquake stations every hour. Keeping the data on a single

PC would eventually be infeasible and hard to handle since the space requirements increase continuously. One final gain is for the future work that might include parallel applications using the seismic data on the GRID. Once an application that requires massive amount of mathematical calculations is developed on the GRID or ported to work on the GRID, the seismic data will be readily available.

5. Implementation Details

In this part of the report, we are going to give the implementation details of the project such as the structure of a seismic data file, the functions and executable programs used previously at the Kandilli Observatory and Earthquake Research Institute in order to analyze a seismic data file, locate an earthquake and calculate the magnitude of an earthquake as well.

5.1. The Architecture

This section explains the internals of the earthquake epicenter application in consecutive steps.

- A cron job is run hourly on levrek.ulakbim.gov.tr in order to submit a job to the GRID. This job basically submits the earthquake epicenter analysis application.
- A database is installed under the domain zumrut.levrek.grid.boun.edu.tr. The database has got two main functions. The first is to keep the information such as the coordinates as well as the times of the earthquakes as it provides the necessary data for the web interface. The second is to provide information for the jobs running on the GRID. When the earthquake epicenter application starts to run on the GRID, it has to find out which data to process because there is a huge amount of data involving hourly seismic information starting from April 14th, 2006. (The date that the mirroring software was launched) The database contains an entry for every hour starting from April 14th 2006 00:00 AM to December 31st, 2020. Each entry also contains a Boolean

successful/unsuccessful flag initially set to unsuccessful. Once the job is launched on the GRID, it retrieves the current date and time from the system, and then queries the database for an unsuccessful entry closest to the current date time value. After the closest unsuccessful entry is received from the database, the application accesses to the logical file directory of the GRID, in other words the storage nodes, in order to retrieve the corresponding seismic data (SAC) files from the storage node to the worker node. The copying of files from the storage nodes to the worker nodes takes about 8 minutes for an hourly data of almost 100 stations.

- The seismic files stored in the storage nodes are compressed to ZIP format. Thus, the application running on the worker node unzips all the files after the copying is finished.
- Once all the files are copied and extracted, the application starts to search for a pick value in the seismic data (SAC) files for each station's hourly seismic data file. A pick means a change in the usual geographic state of the ground by means of a seismic wave and a geological movement designating an earthquake. A pick travels from one side of Turkey to the other side (ie, from east to west or vice versa) in no more than 30 seconds. However, a SAC file contains all the pick values for a complete hour. Thus, we have to determine whether the picks determined at different stations belong to the same geographical event or not. By doing this, we aim to filter independent picks as well as the noise in the data in order to obtain better and more accurate results. Normally, this is done manually in the Kandilli Earthquake Observatory and Research Institute by the people running the earthquake epicenter analysis application (HYPO 71) written for a single PC. We wanted to automate this process by filtering the noise and independent data. To achieve this, we designed a windowing mechanism where the hourly data is divided into 2 minutes of windows. Instead of running the picker

function on an hourly data, we run it for each station's corresponding windows. Since a pick travels from one side of Turkey to the other side in no more than 30 seconds, we create sub-windows of 30 seconds from each 2-minute window. When a pick is determined at the same sub-window of 4 or more stations, we consider it a *real* pick and run HYPO71 to determine the coordinates of the earthquake.

- If a real pick is found and the coordinates of the earthquake are calculated, the result is inserted to the database.
- After the application is done with the SAC files successfully, it accesses the database again and changes the corresponding date time entry's Boolean flag as *successful*. By doing so, we guarantee that the corresponding date's seismic data files are analyzed successfully and if any earthquakes are found, the results are inserted into the database. This ensures that only one copy of the application is run on a specific hourly data and only once. If during the execution, the seismic data files are not present due to a fail in the storage nodes or the mirroring software or the job is aborted or some other run time error occurs, the Boolean flag stays as *unsuccessful*, meaning that the same data will be retrieved and analyzed once again by another copy of the earthquake epicenter application.
- Once the application has nothing to do with the seismic data files, it simply deletes them from the worker nodes preventing unnecessary space occupancy.
- When a user wants to see the earthquakes of a specific date time or coordinates, she enters the search keys to the web interface and requests information. The web interface queries the database and displays the results.

5.2. Class Documentation

5.2.1. bashScript Class Reference

```
#include <bashScript.h>
```

5.2.1.1. Public Member Functions

- string **getDataFiles** (string destDir)
- string **getDataFiles** (string destDir, string *dateList)
- void **getTime** (string *dateList)

5.2.1.2. Detailed Description

bashScript class. It is used for copying the data from grid to the local computer. It creates a bash command to copy the data. For example a command which is produced from this class can be as follows: for i in `lfc-ls /grid/trgridd/kandilli/barbar.koeri.boun.edu.tr/wData/2007/05/20/12/` ; do lcg-cp --vo trgridd lfn:/grid/trgridd/kandilli/barbar.koeri.boun.edu.tr/wData/2007/05/20/12/$i file:`pwd`/$i ; done

---

5.2.1.3. Member Function Documentation

string bashScript::getDataFiles (string *destDir*)

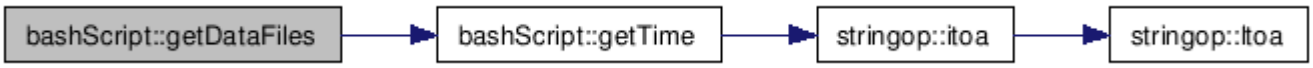
Returns the bash command which can be used to copy the seismic data from the grid to a given directory name.

The seismic data to be copied is the hourly data which belongs to the hour before the algorithm is run.

References getTime().

Referenced by main().

Here is the call graph for this function:



Here is the caller graph for this function:

string bashScript::getDataFiles (string *destDir*, string * *dateList*)

Returns the bash command which can be used to copy the seismic data from the grid to a given directory name.

The seismic data to be copied is the hourly data which is given by dateList array.

The elements of the dateList array must be as follows:

dateList[0]: Year

dateList[1]: Month

dateList[2]: Day

dateList[3]: Hour

void bashScript::getTime (string * *dateList*)

Returns the current year, month, day and the hour before the current hour.

It sets the given array's appropriate elements to return these information.

dateList : Array to be set

dateList[0]: Year

dateList[1]: Month

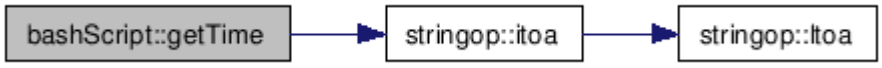
dateList[2]: Day

dateList[3]: Hour

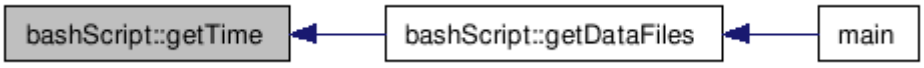
References stringop::itoa().

Referenced by getDataFiles().

Here is the call graph for this function:



Here is the caller graph for this function:



5.2.2. DB Class Reference

5.2.2.1. Public Member Functions

- **DB** ()
- Connection **getConnection** ()
- boolean **openConnection** ()
- boolean **closeConnection** ()

5.2.2.2. Detailed Description

DB class. It is used for connecting the earthquake database.

5.2.2.3. Constructor & Destructor Documentation

DB::DB ()

Default constructor

5.2.2.4. Member Function Documentation

Connection DB::getConnection ()

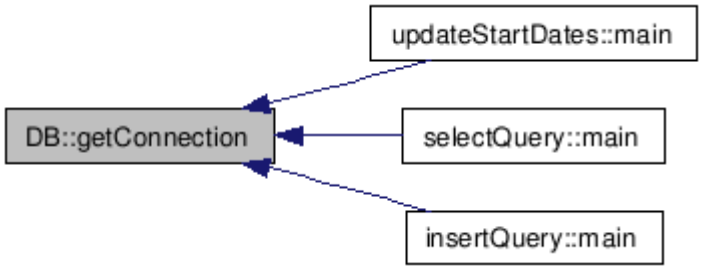
Returns the current connection if there is any; otherwise a new connection is established and returned.

References openConnection().

Referenced by updateStartDates::main(), selectQuery::main(), and insertQuery::main().

Here is the call graph for this function:

Here is the caller graph for this function:

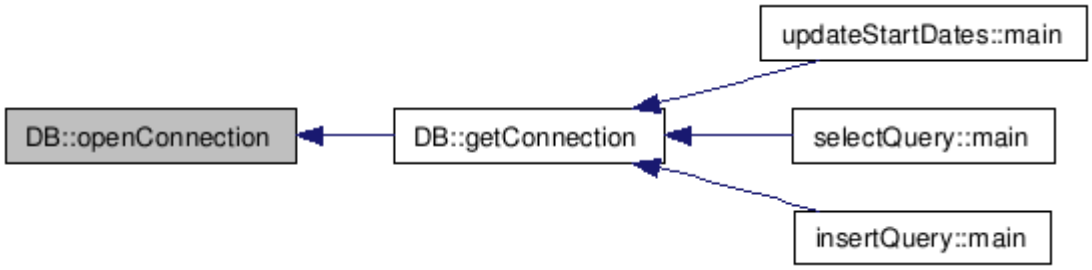


boolean DB::openConnection ()

Establishes a new connection.

Referenced by getConnection().

Here is the caller graph for this function:



boolean DB::closeConnection ()

Closes the current connection.

5.2.3. eqDb Class Reference

```
#include <eqDb.h>
```

5.2.3.1. Public Member Functions

- void **insertLocation** ()

5.2.3.2. Detailed Description

eqDb class. Functions located in this class constitute a part of the database module of the program. The other parts are called from **bashScript** files before the program is initialized.

5.2.3.3. Member Function Documentation

void eqDb::insertLocation ()

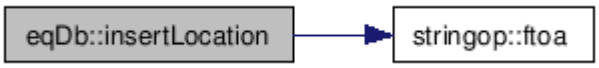
Inserts the earthquakes found by earthquake location algorithm to a database.

When the earthquake location algorithm finds an earthquake, it generates a file called "2" and writes the necessary information except the earthquake time to this file. The time information can be found in another file called "1", the input file of the earthquake location algorithm, which is created by the program. Once all information is taken from those files, a Java executable, which inserts the data to the database, is called with the insert command.

References stringop::ftoa().

Referenced by main().

Here is the call graph for this function:



Here is the caller graph for this function:

void eqDb::updateHour (string *year*, string *month*, string *day*, string *hour*)

Updates the database table named “startdates”. Sets the successful value to 1 for a given hour.

Referenced by main().

Here is the caller graph for this function:

5.2.4. eqFileOp Class Reference

```
#include <eqFileOp.h>
```

5.2.4.1. Public Member Functions

- void **extract** (const char *sourceDir, const char *destinationDir, const char *type1, const char *type2)
- string * **listFiles** (const char *dirName, const char *type1, const char *type2, int &numberOfElements)

5.2.4.2. Detailed Description

eqFileOp class. This class is about the seismic file operations. It can be used for extracting compressed seismic files and returning the names of the seismic files.

5.2.4.3. Member Function Documentation

void eqFileOp::extract (const char * *sourceDir*, const char * *destinationDir*, const char * *type1*, const char * *type2*)

Extracts the compressed seismic files located in a given source directory. The destination directory is also given.

The file name format of the compressed files is as follows:

date.station_name.type1.type2.zip

date: The hour which the data belongs to. Given as yyyymmddhh format.

station_name: The name of the station which sends the data.

type1: Specific information of the type of the station. It can be one of these values: BHE,BNH,BHZ,SHE,SHN,SHZ

type2: Specific information of the data. It can be one of these values: IU, KO, GE, TK

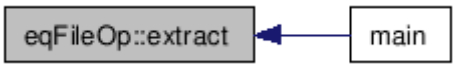
By specifying the type1 and type2 parameters, one can extract only the necessary data for usage.

If a parameter is specified as NULL, all types of files are extracted.

For example, if type1 is equal to NULL and type2 is equal to KO, the files which suits date.station_name.*.KO.zip are extracted.

Referenced by main().

Here is the caller graph for this function:



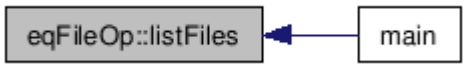
string * eqFileOp::listFiles (const char * *dirName*, const char * *type1*, const char * *type2*, int & *numberOfElements*)

Returns an array of file names which are located in the given source directory. The number of elements in the array is also returned.

Instead of returning a list of all files, one can use the type1 and type2 parameters to return a specific set of elements. More information about these parameters is given in extract method.

Referenced by main().

Here is the caller graph for this function:


eqFileOp::listFiles
main

5.2.5. insertQuery Class Reference

5.2.5.1. Static Public Member Functions

- static void **main** (String[] Args)

5.2.5.2. Detailed Description

Class for inserting new items to the database.

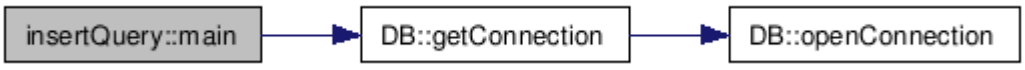
5.2.5.3. Member Function Documentation

static void insertQuery::main (String[] *Args*) `[static]`

Inserts the given query. Each space character is considered as a new argument to the program. There are three spaces in the program which is calling this program. (Insert into xxx(a,b,c...) values....) All 4 words are reconnected before inserting a new record.

References DB, and DB::getConnection().

Here is the call graph for this function:



5.2.6. pickFilter Class Reference

```
#include <pickFilter.h>
```

5.2.6.1. Public Member Functions

- void **filterData** (**window** windowList[], int size)
- void **filterWindow** (**window** windowList[], int size, int sec)

5.2.6.2. Detailed Description

PickFilter class. It introduces some functions about filtering picks in windows.

5.2.6.3. Member Function Documentation

void pickFilter::filterData (window *windowList*[], int *size*)

Deletes false picks in windows due to the noise in the data.

The starting index of the pick is stored in a variable named ndxpk. The time passed between two samples is stored in a variable named delta. Usually, delta is a value between 0.02 and 0.05.

Data is hourly taken, so there are 3600/delta possible values of ndxpk. The probability of the starting time of the pick from two stations having same value is very low. Therefore, the picks having same ndxpk value are considered as false picks and they are deleted from the pick window.

Size: Length of the window array.

References ndxpk.

Referenced by main().

Here is the caller graph for this function:

void pickFilter::filterWindow (window *windowList*[], int *size*, int *sec*)

Deletes picks from given window list based on given sub-window size.

Normally, if one pick is received from one station at time t, all other picks of the same earthquake must be received at most at time t+x. This function takes the x value as sec parameter. Based on this parameter, for each member of the given window list, it finds the sub-window of x seconds which has maximum number of picks. After that, it deletes the picks which are not in this sub-window.

size: The length of the window array.

References ndxpk.

Referenced by main().

Here is the caller graph for this function:

5.2.7. sac Class Reference

```
#include <sac.h>
```

5.2.7.1. Public Member Functions

- void **calculatePickValue** (string fileName, **window** windowList[], int windowSec, int windowSlideSec)
- void **calculatePickTime** (long **nzyear**, long **nzjday**, long **nzhour**, long **nzmin**, long **nzsec**, long **nzmsec**, long **ndxpk**, float **delta**, int windowNumber, int windowSize, **window** windowList[])
- void **setWindow** (int windowSec, int windowSlideSec)
- int **getWindowListSize** (int windowSlideSec)
- void **createHypoHeader** (std::ofstream &out)
- void **createHypoFooter** (std::ofstream &out)
- void **readFile** (string fileName)
- void **callHYP02d** ()

5.2.7.2. Detailed Description

sac class. Used in operations related to SAC data.

5.2.7.3. Member Function Documentation

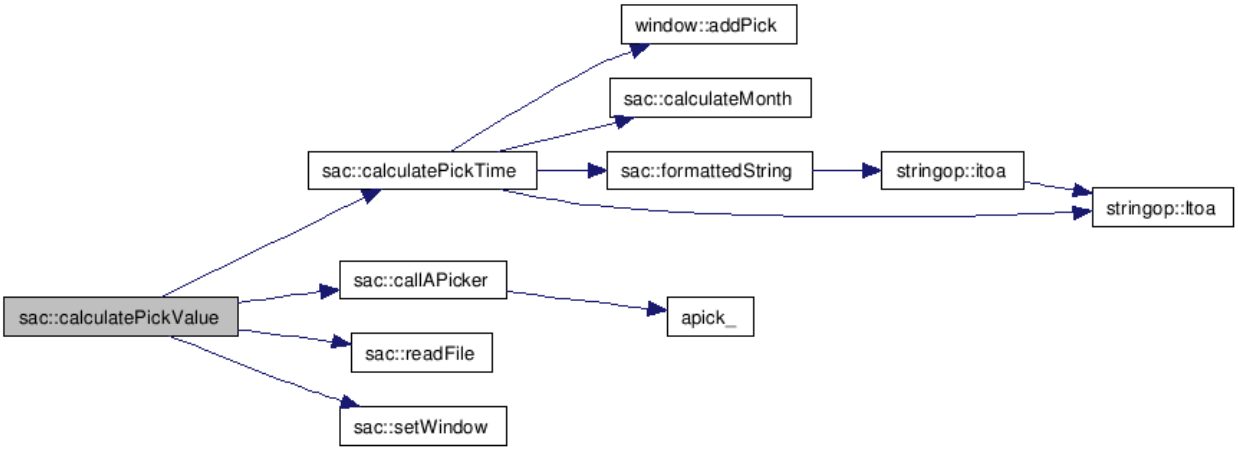
void sac::calculatePickValue (string *fileName*, window *windowList*[], int *windowSec*, int *windowSlideSec*)

Reads the input parameters from the given input file and calculates the pick time

References calculatePickTime(), data, delta, ndxpk, nlen, nzhour, nzjday, nzmin, nzmsec, nzsec, nzyear, readFile(), setWindow(), windowNum, and windowSlideNum.

Referenced by main().

Here is the call graph for this function:



Here is the caller graph for this function:

void sac::calculatePickTime (long *nzyear*, long *nzjday*, long *nzhour*, long *nzmin*, long *nzsec*, long *nzmsec*, long *ndxpk*, float *delta*, int *windowNumber*, int *windowSize*, window *windowList*[])

This function calculates the time (year, day of year, hour, minutes, seconds and milliseconds) of the pick.

It takes the time variables which are read from the SAC file and sets the start time to these variables. After that, it adds the number of seconds which is passed from the beginning of the data to the pick start to the start time. The number of seconds since the pick start can be found by multiplying ndxpk and delta values of the class. Lastly, the function converts the calculated time

to year, month, day of month, hour, minutes, seconds and milliseconds format and adds this pick to the current window.

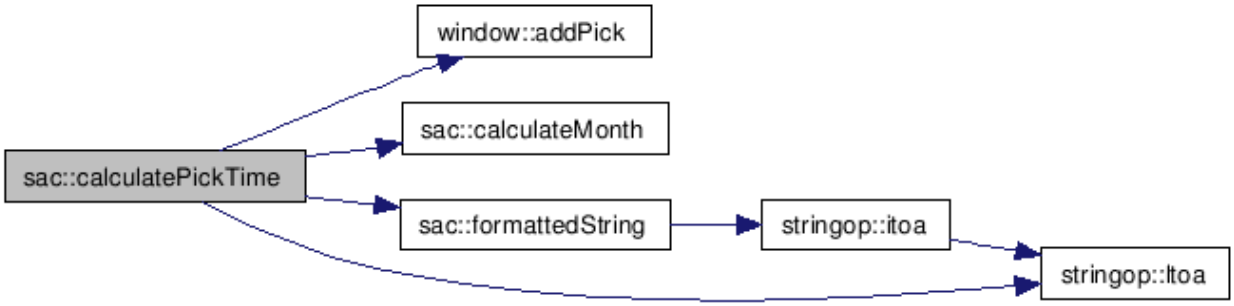
All time values are two digit integers except the second, which is a three digit integer.

For example 0705231341042.18 stand for 23 May 2007 13:41:42:18.

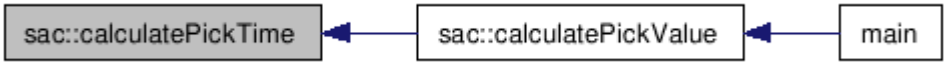
References window::addPick(), iqual, kstnm, and stringop::ltoa().

Referenced by calculatePickValue().

Here is the call graph for this function:



Here is the caller graph for this function:



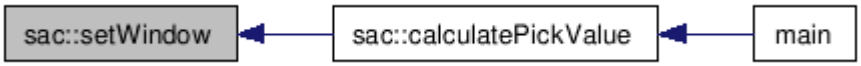
void sac::setWindow (int *windowSec*, int *windowSlideSec*)

Given the window size (in seconds) and the amount of time between two successive windows (in seconds), this function sets the appropriate index values of both.

References delta, windowNum, and windowSlideNum.

Referenced by calculatePickValue().

Here is the caller graph for this function:



int sac::getWindowListSize (int *windowSlideSec*)

Calculates and returns the length of the window array.

windowSlideSec: the amount of time between two successive windows (in seconds).

In an hourly data, normally there will be 3600/windowSlideSec different elements of the array. However, sometimes the data from the last minutes of the last hour and the first minutes of the next hour are received. Thus, the required array length is doubled.

This function is required to be called before calculatePickValue function.

Referenced by main().

Here is the caller graph for this function:

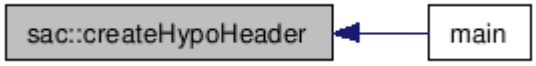
void sac::createHypoHeader (std::ofstream & *out*)

Creates the header for input file of the earthquake location algorithm.

Reads the template.txt file and copies it to a given output stream.

Referenced by main().

Here is the caller graph for this function:



void sac::createHypoFooter (std::ofstream & *out*)

Creates the footer for input file of the earthquake location algorithm.

Referenced by main().

Here is the caller graph for this function:

void sac::readFile (string *fileName*)

Reads the SAC file and initializes necessary parameters which will be used in other functions.

Parameters that are being initialized:

float* data: The SAC data array

float nlen: Length of the SAC data array

float delta: Time interval in seconds between two SAC data elements.

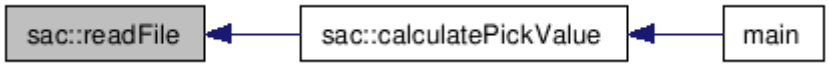
float ndxst: Used for defining the beginning of the pick search. Used internally by aPicker function.

fileName: Name of the SAC file

References cAry, data, delta, kstnm, ndxst, nlen, nzhour, nzjday, nzmin, nzmsec, nzsec, and nzyear.

Referenced by calculatePickValue().

Here is the caller graph for this function:



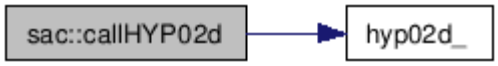
void sac::callHYP02d ()

Calls HYP02d fortran function. This function is used for locating the earthquake.

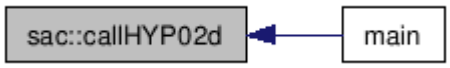
References hyp02d_().

Referenced by main().

Here is the call graph for this function:



Here is the caller graph for this function:



5.2.8. sac_header Struct Reference

```
#include <sac.h>
```

5.2.8.1. Detailed Description

sac header struct. Used while reading a SAC file.

5.2.9. selectQuery Class Reference

5.2.9.1. Static Public Member Functions

- static void **main** (String[] Args)

5.2.9.2. Detailed Description

selectQuery class is used for determining the date and hour on which earthquake location program runs.

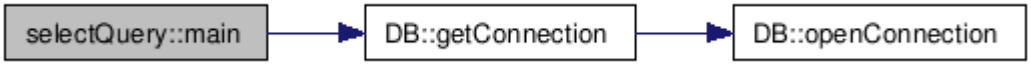
5.2.9.3. Member Function Documentation

static void selectQuery::main (String[] *Args*) `[static]`

It takes the next "unsuccessful" hour value from the database and executes the earthquake location program on this time. For the hour to be "unsuccessful", either the program never worked with that time values or the execution on that time is unsuccessful because of a specific reason such as data inavailability.

References DB, and DB::getConnection().

Here is the call graph for this function:



5.2.10. stringop Class Reference

```
#include <stringop.h>
```

5.2.10.1. Public Member Functions

- string **itoa** (int value, int base)
- string **ltoa** (long value, int base)
- char * **ftoa** (double f, double sigfigs)

5.2.10.2. Detailed Description

This class is used for converting integers, long integers and floats to their string representations.

5.2.10.3. Member Function Documentation

string stringop::itoa (int *value*, int *base*)

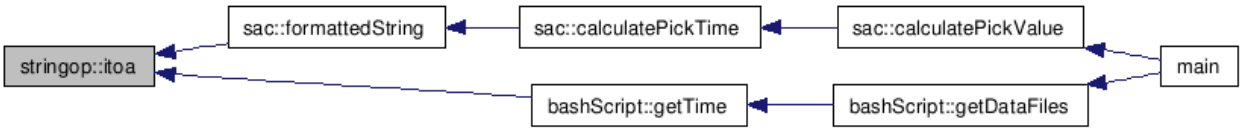
Converts the given integer of given base to a string.

References ltoa().

Referenced by bashScript::getTime().

Here is the call graph for this function:

Here is the caller graph for this function:

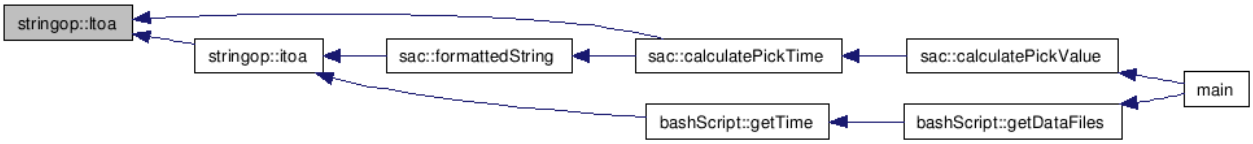


string stringop::ltoa (long *value*, int *base*)

Converts the given long of given base to a string.

Referenced by sac::calculatePickTime(), and itoa().

Here is the caller graph for this function:

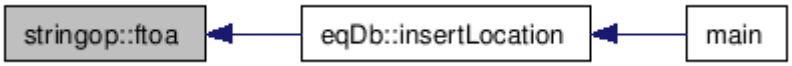


char * stringop::ftoa (double *f*, double *sigfigs*)

Converts the given float to a string. Number of significant figures of the string value must be given.

Referenced by eqDb::insertLocation().

Here is the caller graph for this function:



5.2.11. updateStartDates Class Reference

5.2.11.1. Detailed Description

updateStartDates class is used for updating the database table named “startdates”. This table used for determining hourly data files which will be copied from the storage element. The earthquake epicenter location function then runs in these hourly data. After that, it sets a flag in this table so that another instance of the program will not rerun on the same data again.

5.2.11.2. Public Member Functions

- static void **main** (String[] args)

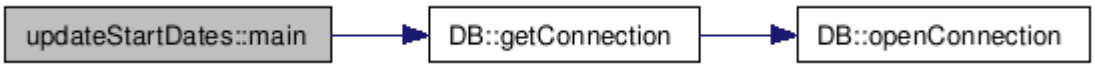
5.2.11.3. Member Function Documentation

static void updateStartDates::main (String[] *args*) `[static]`

Executes the update query which sets the “successful” flag. The use of this flag is explained above.

References DB, and DB::getConnection().

Here is the call graph for this function:



5.2.12. window Class Reference

```
#include <window.h>
```

5.2.12.1. Public Member Functions

- **window** ()
- void **addPick** (string newPick, double newPickTime, long newNdxpk, float newDelta)

5.2.12.2. Public Attributes

- list< string > **pickList**
- list< double > **timeList**
- list< long > **ndxpk**

5.2.12.3. Detailed Description

This class is used for storing picks in a time interval.

5.2.12.4. Constructor & Destructor Documentation

window::window ()

Default constructor

5.2.12.5. Member Function Documentation

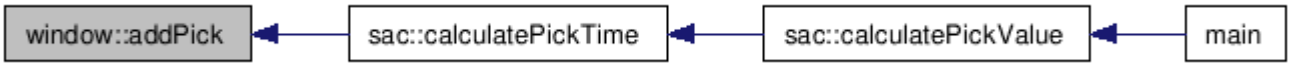
void window::addPick (string *newPick*, double *newPickTime*, long *newNdxpk*, float *newDelta*)

Adds a new pick to the window.

References delta, ndxpk, pickList, and timeList.

Referenced by sac::calculatePickTime().

Here is the caller graph for this function:



5.2.12.6. Member Data Documentation

list<string> window::pickList

List of picks as they are seen on the output file (1).

Referenced by addPick(), and main().list<double> window::timeList

List of picks in a comparable format.

Referenced by addPick().list<long> window::ndxpk

List of pick start indices

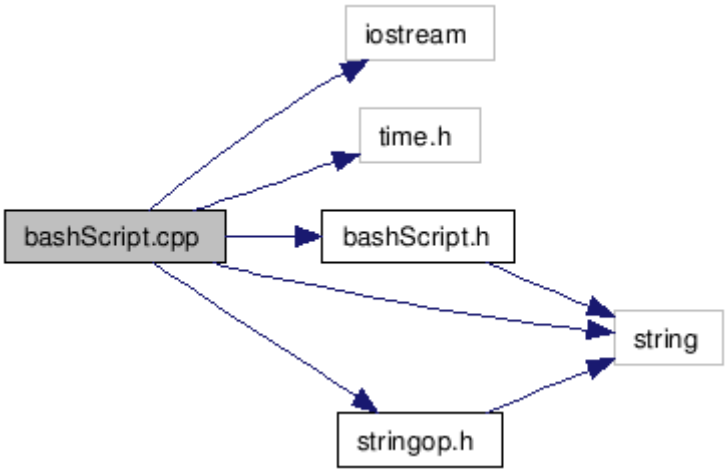
Referenced by addPick().

5.3. File Documentation

5.3.1. bashScript.cpp File Reference

```
#include <iostream>

#include <time.h>

#include "bashScript.h"

#include "stringop.h"

#include <string>
```

Include dependency graph for bashScript.cpp:



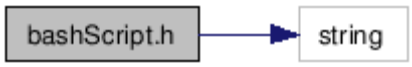
5.3.1.1. Namespaces

- namespace **std**

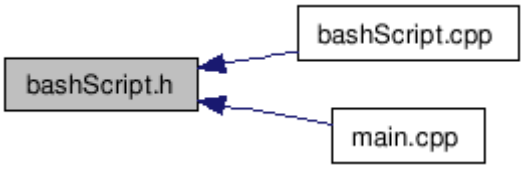
5.3.2. bashScript.h File Reference

```
#include <string>
```

Include dependency graph for bashScript.h:



This graph shows which files directly or indirectly include this file:



5.3.2.1. Classes

- class **bashScript**

5.3.3. DB.java File Reference

5.3.3.1. Namespaces

- namespace **java::sql**

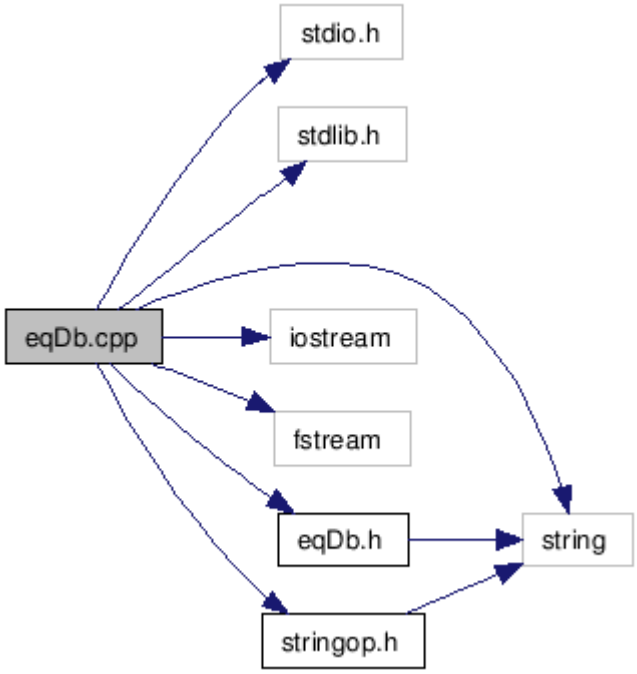
5.3.3.2. Classes

- class **DB**

5.3.4. eqDb.cpp File Reference

```
#include <stdio.h>

#include <stdlib.h>

#include <string>

#include <iostream>

#include <fstream>

#include "eqDb.h"

#include "stringop.h"
```

Include dependency graph for eqDb.cpp:

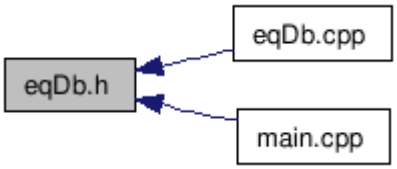


5.3.5. eqDb.h File Reference

```
#include <string>
```

Include dependency graph for eqDb.h:

This graph shows which files directly or indirectly include this file:



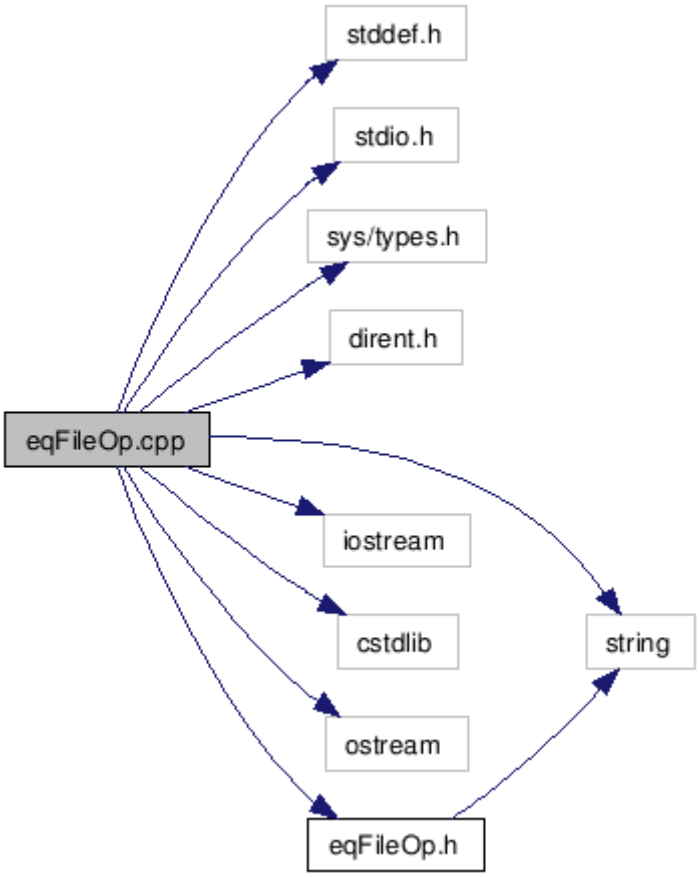
5.3.5.1. Classes

- class **eqDb**

5.3.6. eqFileOp.cpp File Reference

```
#include <stddef.h>

#include <stdio.h>

#include <sys/types.h>

#include <dirent.h>

#include <string>

#include <iostream>

#include <cstdlib>

#include <ostream>

#include "eqFileOp.h"
```

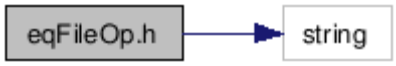
Include dependency graph for eqFileOp.cpp:



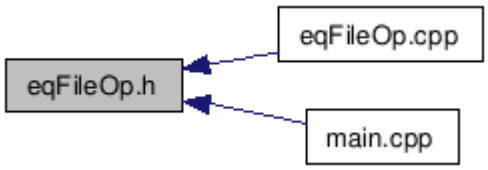
5.3.7. eqFileOp.h File Reference

```
#include <string>
```

Include dependency graph for eqFileOp.h:



This graph shows which files directly or indirectly include this file:



5.3.7.1. Classes

- class **eqFileOp**

5.3.8. insertQuery.java File Reference

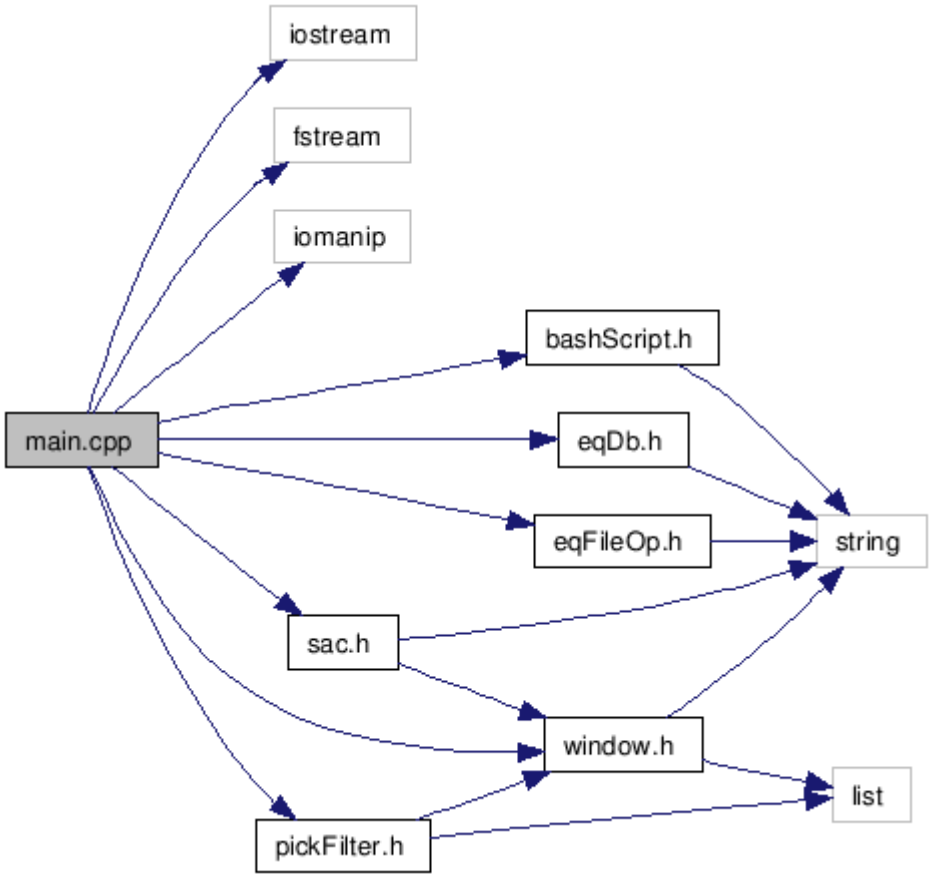
5.3.8.1. Classes

- class **insertQuery**

5.3.9. main.cpp File Reference

```
#include <iostream>

#include <fstream>

#include <iomanip>

#include "sac.h"

#include "eqFileOp.h"

#include "bashScript.h"

#include "eqDb.h"

#include "pickFilter.h"

#include "window.h"
```

Include dependency graph for main.cpp:



5.3.9.1. Functions

- int **main** (int argc, char **argv)

5.3.9.2. Function Documentation

int main (int *argc*, char ** *argv*)

Main method of the program. It takes the date and the hour from the arguments (default 2007 05 19 20), takes the seismic data of that hour from the grid, extracts the necessary data files to be used for finding the epicenter of the earthquake, calculates the picks in the data files, filters noisy data, executes the earthquake epicenter location subroutines and inserts the earthquakes that are found by the previous subroutines. By default, the data is first copied from the grid and after the

data is read, they are deleted from the system. However, this behavior can be changed for testing purposes. After specifiying the date and hour information in the terminal, one can specify a number (running mode). The running modes are as follows:

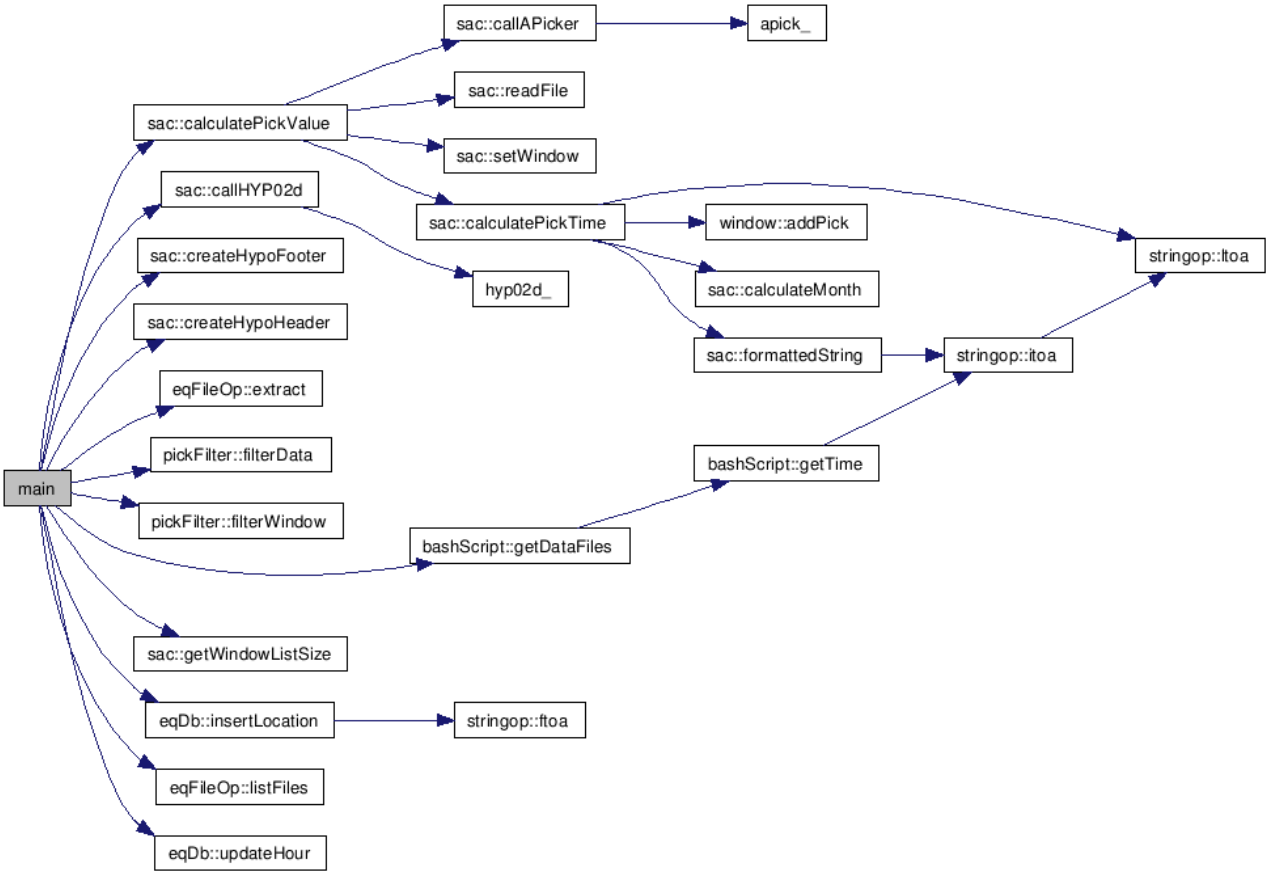
0- First copy the data and delete (default) 1- Copy the data but not delete at the end of the program. 2- Do not copy the data, but delete them at the end of the program. 3- Neither copy nor delete the data.

The program uses four important variables in execution. These are WINDOWSLIDESEC, WINDOWSEC\ SUBWINDOWSEC

WINDOWSLIDESEC: The time between two windows used by the picker function (in seconds). WINDOWSEC: The length of the window used by the picker (in seconds). SUBWINDOWSEC: The maximum allowed delay in the start time of the pick for the same earthquake. If a pick is found at time t by one station, pick time of the other stations must be at most t+SUBWINDOWSEC for the pick to be considered as the same pick. PICKTHRESHOLD: Minimum number of picks which must exist in the same subwindow for those picks to be considered as an earthquake.

References sac::calculatePickValue(), sac::callHYP02d(), sac::createHypoFooter(), sac::createHypoHeader(), eqFileOp::extract(), pickFilter::filterData(), pickFilter::filterWindow(), bashScript::getDataFiles(), sac::getWindowListSize(), eqDb::insertLocation(), eqFileOp::listFiles(), PICKTHRESHOLD, window::pickList, SUBWINDOWSEC, WINDOWSEC, and WINDOWSLIDESEC.

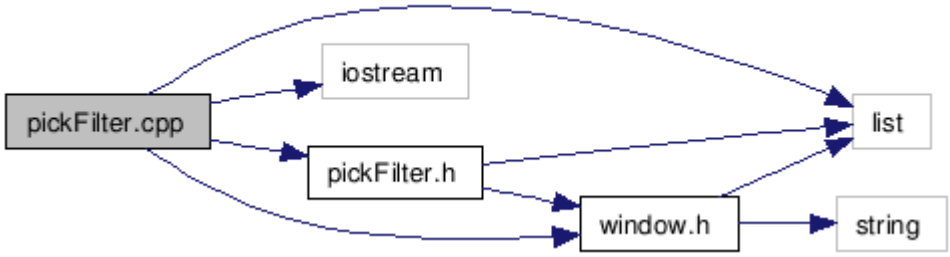
Here is the call graph for this function:


sac::callAPicker
apick_
sac::readFile
sac::calculatePickValue
sac::setWindow
sac::callHYP02d
sac::calculatePickTime
window::addPick
stringop::ltoa
sac::createHypoFooter
hyp02d_
sac::calculateMonth
sac::createHypoHeader
sac::formattedString
stringop::itoa
eqFileOp::extract
pickFilter::filterData
bashScript::getTime
main
pickFilter::filterWindow
bashScript::getDataFiles
sac::getWindowListSize
eqDb::insertLocation
stringop::ftoa
eqFileOp::listFiles
eqDb::updateHour

5.3.10. pickFilter.cpp File Reference

```
#include <list>

#include <iostream>

#include "window.h"

#include "pickFilter.h"
```

Include dependency graph for pickFilter.cpp:

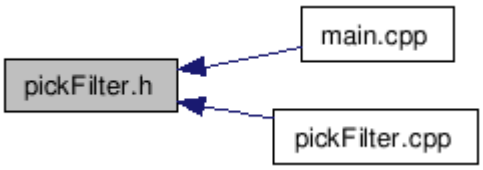


5.3.11. pickFilter.h File Reference

```
#include <list>

#include "window.h"
```

Include dependency graph for pickFilter.h:

This graph shows which files directly or indirectly include this file:



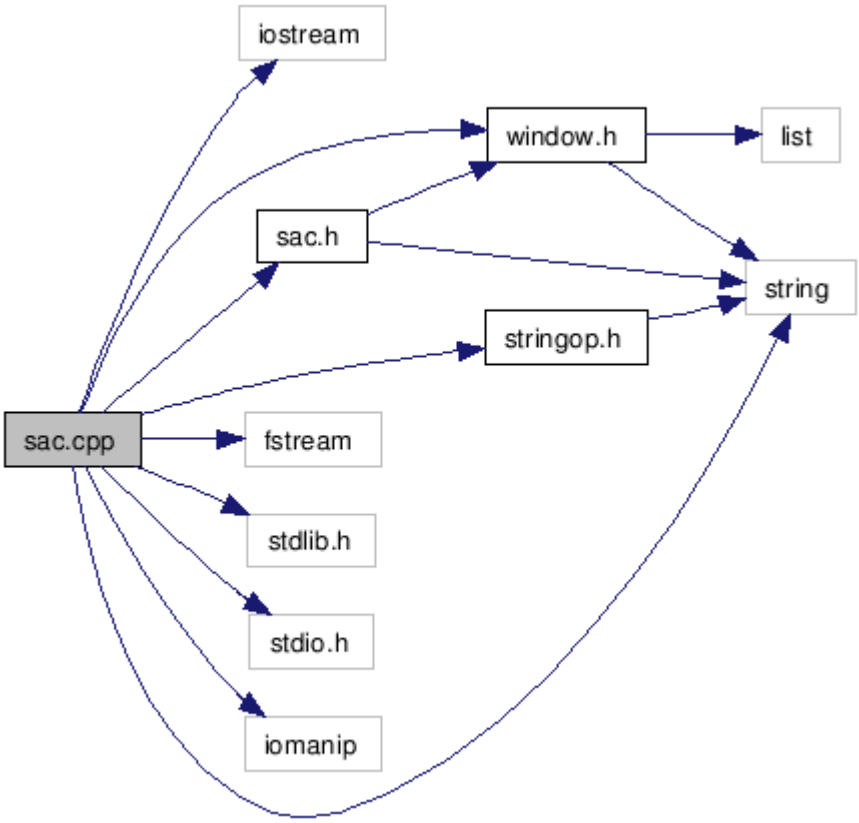
5.3.11.1. Classes

- class **pickFilter**

5.3.12. sac.cpp File Reference

```
#include <iostream>

#include "sac.h"

#include <fstream>

#include <string>

#include <stdlib.h>

#include <stdio.h>

#include <iomanip>

#include "stringop.h"

#include "window.h"
```

Include dependency graph for sac.cpp:



5.3.12.1. Functions

- void **hyp02d_** ()
- void **apick_** (const float *arry, float &**nlen**, float &**delta**, float &**ndxst**, const float ***cAry**, long int &**ndxpk**, float &**nlncda**, long int &**itype**, long int &**iqual**, float &**updwn**)

5.3.12.2. Variables

- float * **data**
- float **nlen**
- float **delta**
- float **ndxst**
- float **cAry** [] = {0.95,1.0,0.15,0.005,11.0,0.0039,1000000.0,-0.1,2.0,3.0,1.0,3,40,3,0,0,0}
- long int **ndxpk**
- long int **itype**
- long int **iqual**
- long int **nzyear**
- long int **nzhour**
- long int **nzmin**
- long int **nzsec**
- long int **nzmsec**
- long int **nzjday**
- float **nlncda**
- float **updwn**
- char **kstnm** [8]
- int **windowNum**
- int **windowSlideNum**

5.3.12.3. Function Documentation

void apick_ (const float * *arry*, float & *nlen*, float & *delta*, float & *ndxst*, const float * *cAry*, long int & *ndxpk*, float & *nlncda*, long int & *itype*, long int & *iqual*, float & *updwn*)

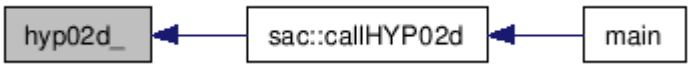
Referenced by sac::callAPicker()

Here is the caller graph for this function:

void hyp02d_ ()

Referenced by sac::callHYP02d().

Here is the caller graph for this function:

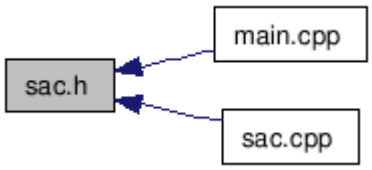


5.3.13. sac.h File Reference

```
#include <string>

#include "window.h"
```

Include dependency graph for sac.h:

This graph shows which files directly or indirectly include this file:



5.3.13.1. Classes

- struct **sac_header**
- class **sac**

5.3.14. selectQuery.java File Reference

5.3.14.1. Namespaces

- namespace **java::util**
- namespace **java::io**

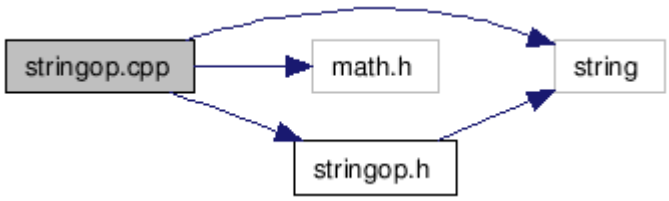
5.3.14.2. Classes

- class **selectQuery**

5.3.15. stringop.cpp File Reference

```
#include <string>

#include <math.h>

#include "stringop.h"
```

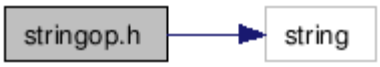
Include dependency graph for stringop.cpp:



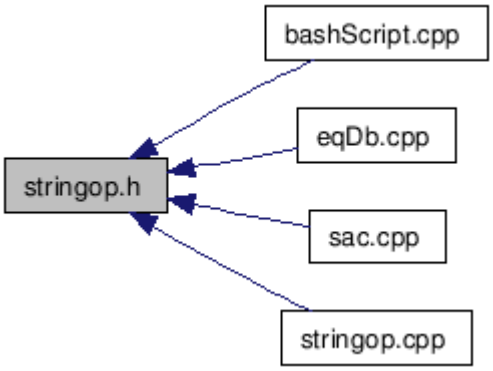
5.3.16. stringop.h File Reference

```
#include <string>
```

Include dependency graph for stringop.h:



This graph shows which files directly or indirectly include this file:



5.3.16.1. Classes

- class **stringop**

5.3.17. updateStartDates.java File Reference

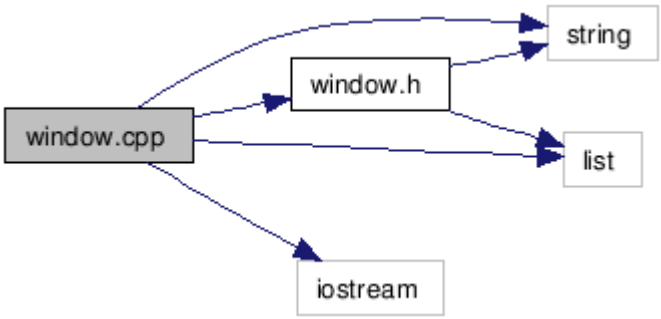
5.3.17.1. Classes

- class **updateStartDates**

5.3.18. window.cpp File Reference

```
#include "window.h"

#include <string>

#include <iostream>

#include <list>
```

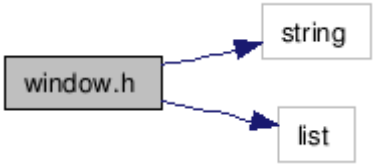
Include dependency graph for window.cpp:



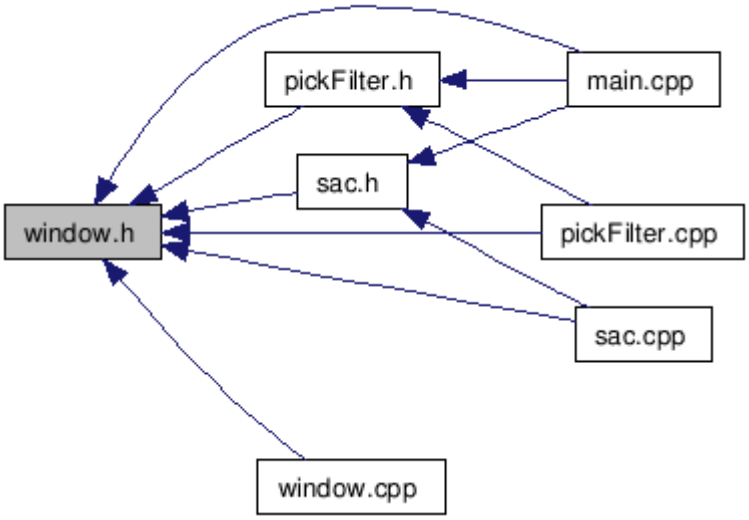
5.3.19. window.h File Reference

```
#include <string>

#include <list>
```

Include dependency graph for window.h:



This graph shows which files directly or indirectly include this file:



5.3.19.1. Classes

- class **window**

5.3.20. BUILD File Reference

BUILD bash script file is the main file for building the project. It compiles Java files with Sun Java Compiler (javac). Fortran files are compiled with Intel Fortran Compiler (ifort) and C++ files are compiled with Intel C++ Compiler (icpc). Files that are compiled with Intel compliers are then linked and a.out executable is obtained.

5.3.21. BUILDFAST File Reference

BUILDFAST bash script file is the lite version of the BUILD file. It does not compile Java and Fortran files. It only compiles C++ files and it uses the previous compiled versions of the Fortran files while linking. The resulting a.out file is obtained faster. The purpose of this file is to save the time of the developers while they are working on C++ files.

5.3.22. RUNME File Reference

RUNME bash script file is the main file to run the project. It sets the necessary environment variables and executes the selectQuery class, which iteratively executes the main program for different time and hour values.

5.3.23. RUNSINGLEHOUR File Reference

RUNSINGLEHOUR bash script file can also be used for executing the main earthquake epicenter location program. This time, after setting necessary environment variables, instead of executing the selectQuery class which automatically sets the date and time for the program, the script runs the main program with a user specified date and time value. For example *./RUNSINGLEHOUR 2007 06 11 03* command will cause the earthquake epicenter program run on the data of 11 June 2007 - 03:00.

5.4. Previously Used Functions

Generally at the Kandilli Observatory and Earthquake Research Institute, the SAC files are read using Visual BASIC routines and then the corresponding data is passed to the appropriate Fortran 90 methods. There are two basic Fortran 90 programs used by the institute in order to locate and calculate the magnitude of an earthquake. HYPO71 was used for many years in the 1970's and early 1980's for routine earthquake processing at the US Geological Survey in Menlo Park, CA, and also at many seismic networks around the World including Kandilli. The program does not get its input directly from a SAC file. Instead, another widely used routine called APicker.F90 is used to process a SAC file and generate the corresponding input data necessary to invoke Hypo71. Today, these programs are called dynamically using Visual BASIC from a single PC.

5.5. Migrating to TR-GRID

Since we are going to use the TR-GRID infrastructure, we needed to migrate the previous programs written in Visual BASIC to C++, which is a compatible programming language with TR-GRID. Because the previously used programs are written in Fortran, we needed to call these routines dynamically from C++. We preferred to use Intel C and Intel Fortran compilers for the implementation.

When calling a subroutine written in a different programming language, C++ requires the addition of an *extern "C"* directive, which includes the prototype of the corresponding subroutine.

An example used for calling *apick* is as follows:

```
extern "C"
{
        extern void apick_(const float* arry,float& nlen,float& delta,float& ndxst,const float*
cAry,float& ndxpk,float& nlncda,float& itype,float& iqual,float& updwn);
}
```

In order to call this method from C++, first the Fortran 90 file that includes the subroutine must be compiled as:

```
ifort -c  aPicker.f90
```

Then the C++ files that are used to call these methods are compiled and linked dynamically as:

```
icpc –o output.o c_file.cpp aPicker.o
```

The final executable is then called as:

```
./output.o
```

and produces the data needed for invoking Hypo71.

The SAC files located on TR-GRID are normally compressed as ZIP files. Names of these zip files are given after the different types of earthquake stations. Apicker needs only a specific

group of these types. We implemented a routine that takes as input the types of the ZIP files and then extracts them. In the future, if we need to process other types of ZIP files, it will be much easier to scale the program.

5.6. Building and Running the Program

To compile all the files and link the C++ and Fortran files used in the project, one should use the *BUILD* file. If only C++ files are needed to be compiled and then linked to the Fortran object files, which are created before, *BUILDFAST* file should be used.

There are two options for running the program. These are automatic running and manual running. The RUNME file stands for the automatic execution and RUNSINGLEHOUR file stands for the manual execution.

Details about the building and running can be found in the file references of these four files.

6. Conclusion and Future Work

We have implemented an earthquake epicenter analysis application which detects the coordinates and time of an earthquake and enters this information to the database. As pointed out early, the picker software currently used in the Kandilli Earthquake Observatory and Research Institute is operated manually and the pick values are collected by the users who run the program. On the other hand, we tried to automate this by designing filtering methods that filter the noise as well as irrelevant pick values using data windowing techniques discussed previously. The filtering methods however, do not always successfully detect all the picks correctly or may consider some noisy data as pick values leading to false positive earthquake

detections. Thus, the filtering method should be improved to work better. Moreover, the magnitude and the depth of the earthquakes can be calculated as well.

The mirroring software written previously by Didem Unat, that copies the seismic data files to the storage nodes of the GRID sometimes fails to work correctly, as discussed before. As a result, some seismic data files might not be available to be processed. Nevertheless, our earthquake epicenter analysis application works collectively with the database, marking the previously processed data entries as *successful* and leaving the incomplete or not available data entries as *unsuccessful.* Hence, the failures in the mirroring software do not effect the execution of our program. This is because each time a copy of our program is executed; it queries the database for the data entries in the table marked as *unsuccessful* again.

Finally, the earthquake epicenter analysis application is only a prototype to illustrate that the seismic data stored in the GRID can be used successfully by an application working on the GRID. New seismic software applications that use this data for any other purpose might be developed to run on the GRID, taking advantage of the availability of the data and the high computation power of the GRID.

REFERENCES

1- Developing Grid Enabled Applications on TR-GRID, Didem Unat, Boğaziçi University Cmpe 492 Project in 2006 Spring term.

2- http://www.grid.org.tr/

3- http://wiki.grid.org.tr/index.php/Ana_Sayfa

4- http://www.gridpp.ac.uk/wiki/LCG_Utils

5- http://glite.web.cern.ch/glite/documentation/default.asp

6- https://twiki.cern.ch/twiki/bin/view/LCG/LfcAdminGuide