

## The Meqi Source Code

The Meqi source code was developed using Microsoft Visual C++ Version 6 and compiles without issuing any warning messages. Because it uses only standard ASCII C++ functions, it should compile on most C++ compilers without major adjustments to the code, but no attempt has been made to do so. The code is simply being made available as is. This brief introduction to the organization of the code is designed to aid one in enhancing the functionality of Meqi to better suit one's needs and in possibly using parts of the code in other software.

A general outline of the main() program is presented followed by a brief introduction to each of the four header files. The third section discusses various means that have been set up to navigate the code. Enhancements to the code have basically fallen along four different lines which are discussed in the fourth section. The last section briefly discusses parts of the code that might be useful and easily transported to other software.

### 1. The main() function

Meqi's main function entails is functionally divided into four parts. The first part reads the command file and stores all of the instructions read from the command file meqi.csv. The second part processes the input files and concludes with the formation of the basic output table of chemotypic indices. The third part operates on the output table based on instructions in taken from the command file. The fourth and final part enables the user to operate on the output table via options specified in the command window from which Meqi is run.

The code is organized by sections. These sections and their subsections are listed starting at line 21 of the meqi.cpp file. Section 0, which constitutes the part 1, opens, reads and closes the command file **meqi.csv** using the function ReadMeqiCmds( ). Generally speaking, each row of the command file is stored as an instance of the LISTS class contained in an array of LISTS pointers called **pParmtrs**. The information in the non-header rows of the command tables is stored in the two-dimensional array **cmds** where each row constitutes a single command that completes the designation of a corresponding chemotypic index. All user-specified instructions obtained from the command file are read from pParmtrs and cmds arrays.

The LISTS class and all other classes are defined in the meqiClasses.h header. An instance of a LISTS class is a list of lists. Each row of any table or file that Meqi reads is viewed as a list of lists. In most CSV tables, the lists are separated by commas and the members or words within each list are separated by spaces. Meqi allows any character (the list separator) to be used to distinguish the lists comprising a line and any other character (the word separator) to be used to distinguish the members or words within a list. Instances of the LISTS class are used throughout Meqi.

The second part is the core of Meqi in which the chemotypic indices are computed. Each structure is processed within the main while loop. A structure is read in section 1 into an instance of a GRAPH class identified by the pointer **origptr** using the function GetSdfGraph( ). This pointer is defined at the start of section 1.6 in a while loop over the records in the input files. This loop extends from start of section 1 to the end of section 5. The information in origptr constitutes all that Meqi retains from the mol file. This information is called the *original structure*.

The structure section in the programming template has a header followed by a variable number of operational subsections. Each operational subsection has a header followed by a variable number of command tables. This hierarchy of user specifications defines a corresponding hierarchy of graph representations of the original structure that give rise to the different chemotypic indices that are computed. Section 2 of the code creates the instances of the GRAPH class that defines

this hierarchy. The top of the hierarchy reflects those options specified in the structure section header of the programming template. The corresponding operations are performed on the original structure and result in the formation of the *parent structure* common to all subsequent graph representations. The information constituting the parent structure is stored in an instance of the GRAPH class addressed via the pointer **ggptr**. This pointer is redefined at the start of section 2.1.

Once the parent structure is formed, Meqi sequentially processes the user options specified in the structure subsection headers. The second through fifth lines of the header starting with the Prefix option in the programming template results in the creation of an *operational structure* for that subsection defined by the GRAPH pointer **secPtr**. This pointer is reinitialized in a for loop over the number of structure subsections extending from the start of section 2.3 in the code to the end of section 5.1.

The processing of the commands in `cmds` begins at the start of section 2.4. Each “command” is a six character string of digits representing a single choice for each of the six option categories defined by the columns of a command table. The *i*'th digit corresponds to a selection from the *i*'th column of the relevant command table. Before a command is processed, the values of its six options are stored in the following variables:

**catOpt** – Substructure category

**trtOpt** – Treatment

**attOpt** – Attachment

**embOpt** – Embedding

**topOpt** – Topology

**varOpt** – Variable.

These variables are updated in a `cmds` for loop over those consecutive rows of the `Ccmds` array that correspond to all of the commands generated from all of the command tables within the relevant structure subsection. This loop extends from the start of section 2.4 to the end of section 5.1.

The *screening structure* is formed from the operational structure based on the choices in the sixth line of the structure subsection header starting with the lower-size bound option. The screening structure is defined by the GRAPH pointer **scrnPtr** formed in section 2.5 of the code. The relevant `catOpt` option is incorporated in the screening structure at this point.

Each of the four `trtOpt` options requires its own subsection of code. These subsections are section 3.1 (the group treatment), section 3.2 (the list treatment), section 3.4 (the first treatment) and section 3.4 (the relational treatment). Processing is completed on any command entailing the list treatment at the end of section 3.2 with the appropriate values written to the output file identified by the file pointer **out\_csv** at which point processing returns to the next command at the start of the `cmds` for loop.

For the other three treatment options, the information in the first four command options at the end of section 3 resides in a GRAPH instance identified by the pointer **subPtr**. Section 4 completes the processing of the remaining `topOpt` and `varOpt` options of the command. The information for these options is taken into account in a GRAPH instance identified by the pointer **topPtr**. The relevant information is written to `out_csv` in section 5.1 at which point processing returns to the next command at the start of the `cmds` for loop.

The basic output file defined by the file pointer `out_csv` is completed by the end of section 5. Section 6 computes summary information that is written to the screen and to the `summary.txt` file identified by the pointer **summary**. Section 7 sets the stage for working with the files that exist

at the end of section 5 and files that can be constructed using the command line options specified in the table operations menu. The means of identifying and storing the relevant file types and sizes is created in section 7.3.

Meqi allows two ways of analyzing and elaborating the information in the basic output file. One means, part three, is by adding sections to the programming template so that these manipulations can be automated into a Meqi run. Only one such section, the index joining section, has been added so far, but well illustrates this capability. Section 8 contains the relevant code. Adding another analysis section entails a change to the programming template, a corresponding change to the function `ReadMeqiCmds()` and a corresponding addition to section 8.

Another means, part four, is provided via command-line table operations. These are covered in section T. Adding another analysis capability is relatively straight forward and numerous examples of such additions appear in that section.

## 2. The header files

Meqi has four header files: `meqiClasses.h`, `meqiConstants.h`, `meqiFunctions.h` and `meqiUtilities.h`. The `meqiClasses.h` header file contains the classes that have been defined. The `GRAPH` class contains the arrays of the `VERTEX`, `EDGE` and `NEIGHBORS` classes. This class is only used in the `main()` function and in the `GetSdfGraph()` function in the `meqiFunctions.h` header file. The `LISTS` class was discussed earlier and is used throughout the `main()` function, in many of the `GRAPH` functions and in many of the functions in the `meqiFunctions.h` header file. The `BTREE` class is a binary-tree type class used in counting string occurrences. The `HIST` class is designed for work with histograms and has received only occasional use.

The `meqiConstants.h` header file contains a variety of constants used throughout the code.

The `meqiFunctions.h` header file contains the major functions defined globally that essentially operate over complete files or are particularly unique to the Meqi code. With the exceptions of `ReadMeqiCmds()`, `GetSdfGraph()`, `HeaderRows()`, which sets up the headers for the output table, and `SetParameters()`, which defines operative parameters read from the command file, these functions are used in working with the tables following the creation of the basic output file.

The `meqiUtilities` header file contains miscellaneous functions easily applicable outside of the Meqi code.

## 3. Navigating the code

In addition to a brief description of their purpose and structure that is given at the start of the definition and code for each function, the `main()` and `ReadMeqiCmds()` functions contain extensive outlines of their structure. These outlines enable one to quickly search and find the relevant subcode.

A **debug** variable in `main()` can be set that enable one to see when critical operations are executed and selected results of those operations when relevant. Similarly a **defile** variable can be set so that one can see when files are opened and closed. All of the major functions have a similar debug variable that can be set to indicate when the function was entered, when selected steps in the function are executed along with relevant results and when the function was exited.

## 4. Some possible enhancements to the code

Although Meqi was written with possible commercial applications in mind, it was primarily designed as a software environment for conducting research into chemotypic methodologies. In that light it was designed to present the user with great flexibility in trying out ideas and methods of analysis related to chemotypic indices as well as to be so structured that it could be easily be

enhanced to incorporate new chemotypic analysis capabilities. As a consequence, a number of issues related to general application have not been addressed even though clearly needed.

Possibly the most obvious is Meqi's current inability to read structural information not in the form of an SD file. The GetSdfGraph( ) function could be augmented to read other formats such as SMILES strings.

Another significant enhancement that lies outside the developer's expertise would be the replacement of the ChemGrfs( ) in the GRAPH class with another function that better takes into account issues related to aromaticity and tautomers.

One can automate the table operations within the Meqi structure in a reasonably straight forward manner by adding the relevant section to the Meqi programming template along the lines that was done for the index joining section. Alternatively, the code could be greatly reduced and simplified by excising all code not required in sections 0 through 5.

Should one wish to simply investigate another chemotypic table analysis methodology, it is fairly easy to add the relevant option to Section T.

### **5. More generally usable parts of the code**

Although most of the Meqi code probably has little application outside of its use within the Meqi software, parts of the code may be of more general interest. Something akin to the GRAPH class is likely to be common to most software handling 2D structural information, but some of the functions in this class may be of particular interest to those interested in computing a broad assortment of chemotypic indices.

The assignment of chemotypic identifiers to the computed chemotypes is applicable to any structural representation that can be represented as a labeled pseudograph. This is done with the MeqnumAlg( ) function of the GRAPH class and with the MorgnumAlg( ) that it calls.

It has been this programmer's experience that the LISTS class is a very powerful programming tool when working with information in any string or record that is naturally viewed as a list of lists.

Finally, the BTREE class has proved adequate for the counting needs encountered so far in the table operations.