Grader PHP Documentation

8.0

Generated by Doxygen 1.13.2

1 Hierarchical Index

1.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

2 Class Index

2.1 Class List

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

3 File Index

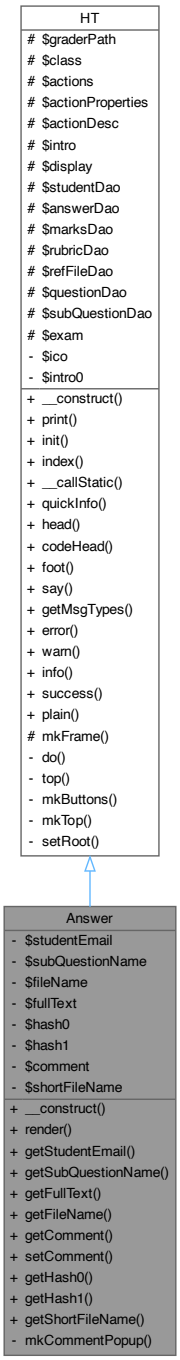
3.1 File List

Here is a list of all files with brief descriptions:

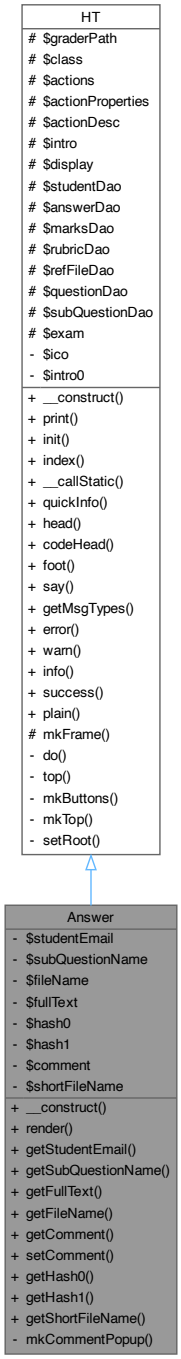
4 Class Documentation

4.1 Answer Class Reference

Inheritance diagram for Answer:



Collaboration diagram for Answer:



Public Member Functions

- __construct ($studentEmail, $subQuestionName, $fileName, $fullText='', $hash0='', $hash1='', $comment='')
- render ($firstAnswer="")
- getStudentEmail ()
- getSubQuestionName ()
- getFullText ()

- getFileName ()
- getComment ()
- setComment ($comment)
- getHash0 ()
- getHash1 ()
- getShortFileName ()

Public Member Functions inherited from HT

- __construct ()
- print ($toStr=true)

Static Private Member Functions

- static mkCommentPopup ()

Private Attributes

- $studentEmail
- $subQuestionName
- $fileName
- $fullText
- $hash0
- $hash1
- $comment
- $shortFileName

Additional Inherited Members

Static Public Member Functions inherited from HT

- static init ()
- static index ()
- static __callStatic ($method, $args)
- static quickInfo ($side)
- static head ($toStr=false, $title="Grader by Manoj", $refresh='')
- static codeHead ($title="Code Listing by Manoj")
- static foot ($toStr=false)
- static say ($what, $class='plain', $toStr=false, $showIcon=false)
- static getMsgTypes ()
- static error ($what, $toStr=false)
- static warn ($what, $toStr=false)
- static info ($what, $toStr=false)
- static success ($what, $toStr=false)
- static plain ($what, $toStr=false)

Static Protected Member Functions inherited from HT

- static mkFrame ($side='left')

Static Protected Attributes inherited from HT

- static $graderPath
- static $class = __CLASS__
- static $actions
- static $actionProperties = [ ]
- static $actionDesc
- static $intro
- static $display = ["class" => "success", "title" => ""]
- static $studentDao
- static $answerDao
- static $marksDao
- static $rubricDao
- static $refFileDao
- static $questionDao
- static $subQuestionDao
- static $exam

4.1.1 Detailed Description

Class Answer

Represents a student's answer submission for a sub-question, including file content, hash information for file integrity, and associated comments for grading.

Extends the HT class, assumes interaction with student and exam DAOs for rendering grading interfaces and handling auto-grading configurations.

Inherits from the HT class for rendering HTML and messaging.

Definition at line 14 of file Answer.php.

4.1.2 Member Function Documentation

__construct()

```
Answer::__construct (
              $studentEmail,
              $subQuestionName,
              $fileName,
              $fullText = '',
              $hash0 = '',
              $hash1 = '',
              $comment = '')
```

Answer constructor.

Parameters

string	*$studentEmail*	The student's email address.
string	*$subQuestionName*	The sub-question name.
string	*$fileName*	The full file path to the student's answer.
string	*$fullText*	(Optional) Preloaded file content; if empty, file is read.

string	*$hash0*	(Optional) Primary file hash; calculated if empty.
string	*$hash1*	(Optional) Secondary file hash; calculated if empty.
string	*$comment*	(Optional) Comments related to the answer.

Definition at line 67 of file Answer.php.

```
00075       {
00076         $this->studentEmail = $studentEmail;
00077         $this->subQuestionName = $subQuestionName;
00078         $this->fileName = $fileName;
00079         if ($fullText) {
00080             $this->fullText = $fullText;
00081         } else {
00082             $this->fullText = file_get_contents($this->fileName);
00083         }
00084         if ($hash0) {
00085             $this->hash0 = $hash0;
00086         } else {
00087             $this->hash0 = Report::hash($this->fileName);
00088         }
00089         if ($hash1) {
00090             $this->hash1 = $hash1;
00091         } else {
00092             $this->hash1 = Report::hash($this->fileName, false);
00093         }
00094         $this->comment = $comment;
00095         $student = self::$studentDao->getByEmail($studentEmail)[0];
00096         $studentFolder = $student->getAnswerFolder();
00097         $this->shortFileName = trim(
00098             str_replace($studentFolder, "", $fileName),
00099             '/'
00100         );
00101     }
```

References $comment, $fileName, $fullText, $hash0, $hash1, $studentEmail, and $subQuestionName.

getComment()

```
Answer::getComment ()
```

Gets the comment associated with this answer.

Returns

string The comment text.

Definition at line 268 of file Answer.php.

```
00269     {
00270         return $this->comment;
00271     }
```

References $comment.

getFileName()

```
Answer::getFileName ()
```

Gets the full file name (path) of the answer.

Returns

string The full file path.

Definition at line 258 of file Answer.php.

```
00259     {
00260         return $this->fileName;
00261     }
```

References $fileName.

getFullText()

```
Answer::getFullText ()
```

Gets the full content of the answer file.

Returns

string The full file content.

Definition at line 248 of file Answer.php.

```
00249     {
00250         return $this->fullText;
00251     }
```

References $fullText.

getHash0()

```
Answer::getHash0 ()
```

Gets the primary hash value of the answer file.

Returns

string The primary hash.

Definition at line 290 of file Answer.php.

```
00291     {
00292         return $this->hash0;
00293     }
```

References $hash0.

getHash1()

```
Answer::getHash1 ()
```

Gets the secondary hash value of the answer file.

Returns

string The secondary hash.

Definition at line 299 of file Answer.php.

```
00300     {
00301         return $this->hash1;
00302     }
```

References $hash1.

getShortFileName()

```
Answer::getShortFileName ()
```

Gets the file name relative to the student's answer folder.

Returns

string The short file name.

Definition at line 309 of file Answer.php.

```
00310     {
00311         return $this->shortFileName;
00312     }
```

References $shortFileName.

getStudentEmail()

```
Answer::getStudentEmail ()
```

Gets the student's email associated with this answer.

Returns

string The student's email.

Definition at line 228 of file Answer.php.

```
00229     {
00230         return $this->studentEmail;
00231     }
```

References $studentEmail.

getSubQuestionName()

```
Answer::getSubQuestionName ()
```

Gets the sub-question name this answer belongs to.

Returns

string The sub-question name.

Definition at line 238 of file Answer.php.

```
00239     {
00240         return $this->subQuestionName;
00241     }
```

References $subQuestionName.

mkCommentPopup()

```
static Answer::mkCommentPopup ()  [static], [private]
```

Generates JavaScript code for opening a popup for editing comments.

Returns

string JavaScript code for the comment editing popup.

Definition at line 108 of file Answer.php.

```
00109     {
00110         $file = "editComment";
00111         return "onclick=
00112         \"var h = screen.height * 0.3;
00113         var t = screen.height * 0.05;
00114         var w = screen.width * 0.25;
00115         var l = screen.width * 0.65;
00116         window.open('', '$file', 'width='+ w +',height=' + h + ',left='+ l +',top='+ t
      +',resizable=yes,scrollbars=yes,toolbar=yes,menubar=no,location=no,directories=no,status=yes');
00117           document.getElementById('$file').submit();
00118           return false;\"";
00119     }
```

render()

```
Answer::render (
              $firstAnswer = "")
```

Renders the HTML representation of this answer for grading or viewing, including buttons for file viewing and auto-grading options if applicable.

Parameters

string	*$firstAnswer*	(Optional) Additional attribute to set for the first rendered answer.

Returns

string The HTML content representing the answer.

Definition at line 128 of file Answer.php.

```
00129     {
00130         // var_dump($this); return;
00131         $subQuestionName = $this->subQuestionName;
00132         $exam = self::$exam;
00133         $gradable = false;
00134         // Grade the question only if it is in CFG::$gradeQuestions
00135         foreach ($exam->getQuestions() as $question) {
00136             if (in_array($question->getNum(), CFG::$gradeQuestions)) {
00137                 foreach ($question->getSubQuestions() as $subQuestion) {
00138                     if ($subQuestionName == $subQuestion->getName()) {
00139                         $gradable = true;
00140                         break 2;
00141                     }
00142                 }
00143             }
00144         }
00145         if (!$gradable) {
00146             return "";
00147         }
00148         $autoGraders = [];
00149         $autoFiles = [];
00150         static $autoDone = [];
00151         $student = self::$studentDao->getByObject($this)[0];
00152         // $submissionFolder = $student->getAnswerFolder();
```

```
00153         // foreach (CFG::$autoGrade as $grader => $files) {
00154         //  $autoGrader = $submissionFolder . $grader;
00155         //  $autoGraders[] = $autoGrader;
00156         //  $autoFiles = array_merge($autoFiles, $files);
00157         // }
00158
00159         $answerFolder = $student->getAnswerFolder();
00160         foreach (CFG::$autoGradables as $answerFile => $autoGrade) {
00161             $autoGrader = $autoGrade['grader'];
00162             $autoGraders[] = $answerFolder . $autoGrader;
00163             $autoFiles[] = $answerFile;
00164         }
00165
00166         $fileName = $this->fileName;
00167         $shortFileName = $this->shortFileName;
00168         $basename = basename($fileName);
00169         if (!self::$refFileDao->isGradable($shortFileName)) {
00170             return "";
00171         }
00172         $fullText = htmlentities($this->fullText, ENT_QUOTES);
00173         $popup = htmlentities(self::mkCommentPopup(), ENT_QUOTES);
00174         $td = "";
00175         $tdFileDB = "<td align='center'>
00176         <form method='post' action='?do' target='right2'>
00177         <input type='submit' $firstAnswer name='viewfile'
00178         title='$fileName' value='$basename: DB' >
00179         <input type='hidden' name='do' value='viewFile'>
00180         <input type='hidden' name='student_email' value='$this->studentEmail'>
00181         <input type='hidden' name='subquestion_name'
00182         value='$subQuestionName'>
00183         <input type='hidden' name='filename' value='$fileName'>
00184         <input type='hidden' name='showDB' value='Yes'>
00185         <input type='hidden' name='fulltext' value='$fullText'>
00186         </form></td>
00187         ";
00188         $tdFileHDD = "<td align='center'>
00189         <form method='post' action='?do' target='right2'>
00190         <input type='submit' $firstAnswer name='viewfile'
00191         title='$fileName' value='$basename' >
00192         <input type='hidden' name='do' value='viewFile'>
00193         <input type='hidden' name='commentable' value='$fileName'>
00194         <input type='hidden' name='popup' value='$popup'>
00195         <input type='hidden' name='student_email' value='$this->studentEmail'>
00196         <input type='hidden' name='subquestion_name'
00197         value='$subQuestionName'>
00198         <input type='hidden' name='filename' value='$fileName'>
00199         <input type='hidden' name='fulltext' value='$fullText'>
00200         </form></td>
00201         ";
00202         if (in_array($basename, $autoFiles)) {
00203             foreach ($autoGraders as $autoGrader) {
00204                 if (!in_array($autoGrader, $autoDone)) {
00205                     $td .= $tdFileHDD;
00206                     $autoDone[] = $autoGrader;
00207                     $graderURL = str_replace(realpath($_SERVER['DOCUMENT_ROOT']), "", $autoGrader);
00208                     $td .= "<td align='center'>
00209                     <a href='$graderURL' target='right2'><input type='button'
00210                     $firstAnswer value='$basename: Auto Grade'></a></td>";
00211                     break;
00212                 }
00213             }
00214             if (CFG::isQuiz()) {
00215                 $td .= $tdFileDB . $tdFileHDD;
00216             }
00217         } else {
00218             $td .= $tdFileHDD;
00219         }
00220         return $td;
00221     }
```

References CFG::$autoGradables, HT::$exam, $fileName, $fullText, CFG::$gradeQuestions, $shortFileName, $subQuestionName, and CFG::isQuiz().

Here is the call graph for this function:

setComment()

```
Answer::setComment (
              $comment)
```

Sets or updates the comment associated with this answer.

Parameters

string	*$comment*	The comment text.

Returns

self The current Answer instance (for method chaining).

Definition at line 279 of file Answer.php.

```
00279                                       : self
00280     {
00281         $this->comment = $comment;
00282         return $this;
00283     }
```

References $comment.

4.1.3 Member Data Documentation

$comment

```
Answer::$comment  [private]
```

Definition at line 49 of file Answer.php.

Referenced by __construct(), getComment(), and setComment().

$fileName

```
Answer::$fileName  [private]
```

Definition at line 29 of file Answer.php.

Referenced by __construct(), getFileName(), and render().

$fullText

```
Answer::$fullText  [private]
```

Definition at line 34 of file Answer.php.

Referenced by __construct(), getFullText(), and render().

$hash0

```
Answer::$hash0  [private]
```

Definition at line 39 of file Answer.php.

Referenced by __construct(), and getHash0().

$hash1

```
Answer::$hash1  [private]
```

Definition at line 44 of file Answer.php.

Referenced by __construct(), and getHash1().

$shortFileName

```
Answer::$shortFileName  [private]
```

Definition at line 54 of file Answer.php.

Referenced by getShortFileName(), and render().

$studentEmail

```
Answer::$studentEmail  [private]
```

Definition at line 19 of file Answer.php.

Referenced by __construct(), and getStudentEmail().

$subQuestionName

```
Answer::$subQuestionName  [private]
```

Definition at line 24 of file Answer.php.

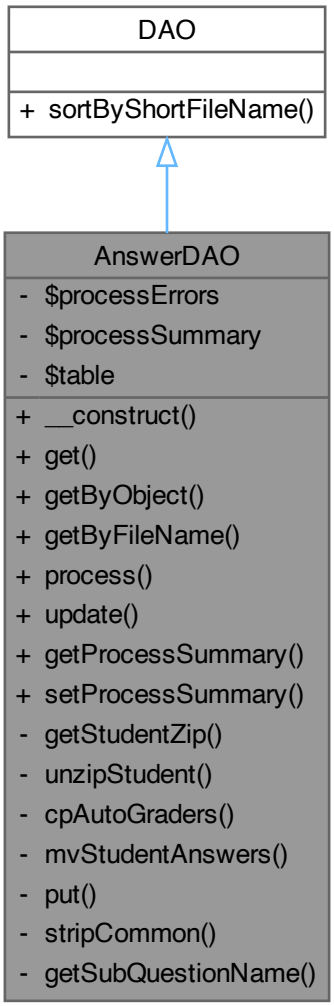
Referenced by __construct(), getSubQuestionName(), and render().

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

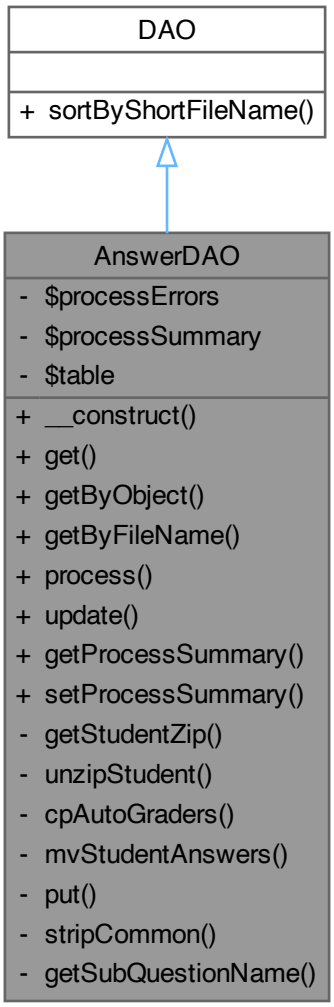
- Answer.php

4.2 AnswerDAO Class Reference

Inheritance diagram for AnswerDAO:



Collaboration diagram for AnswerDAO:



Public Member Functions

- __construct ()
- get ($where=[ ], $verbose=false)
- getByObject ($student, $subQuestion=null, $answer=null)
- getByFileName ($fileName)
- process ($student)
- update ($answer)
- getProcessSummary ()
- setProcessSummary ($processSummary)

Private Member Functions

- getStudentZip ($student)

- unzipStudent ($student)
- cpAutoGraders ($student)
- mvStudentAnswers ($student)
- put ($student, $answers)

Static Private Member Functions

- static stripCommon ($strings)
- static getSubQuestionName ($shortFileName)

Private Attributes

- $processErrors = ""
- $processSummary = ""

Static Private Attributes

- static $table

Additional Inherited Members

Static Public Member Functions inherited from DAO

- static sortByShortFileName ($objects)

4.2.1 Detailed Description

Class AnswerDAO

Handles the CRUD operations for student answers, including processing and storing answers submitted via ZIP files, and managing associated grading workflows.

Definition at line 9 of file AnswerDAO.php.

4.2.2 Member Function Documentation

__construct()

```
AnswerDAO::__construct ()
```

Initializes the AnswerDAO and sets the table name.

Definition at line 23 of file AnswerDAO.php.

```
00024     {
00025         $table = CFG::mkTableName("answers");
00026         self::$table = $table;
00027     }
```

References $table, and CFG::mkTableName().

Here is the call graph for this function:

cpAutoGraders()

```
AnswerDAO::cpAutoGraders (
              $student)  [private]
```

Copies auto-grader files into the student's answer folder for automatic grading.

Parameters

Student	*$student*	The student object.

Returns

bool True on successful copy.

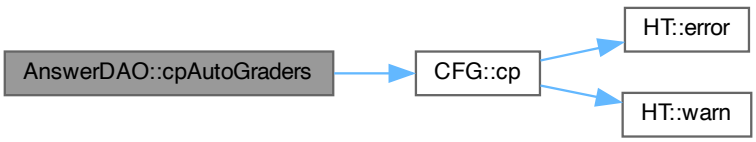
Definition at line 192 of file AnswerDAO.php.

```
00193     { // No error handling for now!
00194         $answerFolder = $student->getAnswerFolder();
00195         foreach (CFG::$autoGradables as $file => $autoGrade) {
00196             $grader = $autoGrade['grader'];
00197             $from = CFG::$solutionFolder . $grader;
00198             $to = $answerFolder . "/" . $grader;
00199             CFG::cp($from, $to, $mkDir = true);
00200             $helpers = $autoGrade['helpers'] ?? [];
00201             foreach ($helpers as $helper) {
00202                 $from = CFG::$solutionFolder . $helper;
00203                 $to = $answerFolder . "/" . $helper;
00204                 CFG::cp($from, $to, $mkDir = true);
00205             }
00206         }
00207         return true;
00208     }
```

References CFG::$autoGradables, CFG::$solutionFolder, and CFG::cp().

Referenced by process().

Here is the call graph for this function:



Here is the caller graph for this function:

get()

```
AnswerDAO::get (
                $where = [],
                $verbose = false)
```

Retrieves Answer records based on filtering criteria.

Parameters

array	*$where*	Key-value pairs for WHERE clause filtering.
bool	*$verbose*	If true, display error when no answers found.

Returns

Answer[] Array of Answer objects.

Definition at line 36 of file AnswerDAO.php.

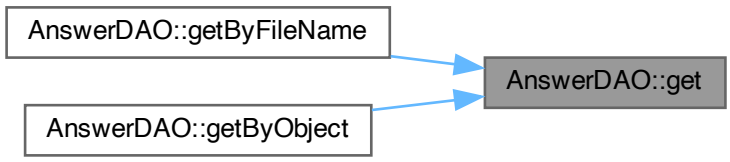
```
00037     {
00038         $table = self::$table;
00039         $answers = [];
00040         $rows = DB::$db->get(
00041             table: $table,
00042             where: $where
00043         );
00044         $got = count($rows);
00045         if ($verbose && $got <= 0) { // Not found in the DB
00046             HT::error("<strong>AnswerDAO</strong>:
00047                 No answers in the DB!<br>
00048                 Run AnswerDAO->process(student) for each student first.");
00049         } else {
00050             foreach ($rows as $row) {
00051                 $answers[] = new Answer(
00052                     $row['student_email'],
00053                     $row['subquestion_name'],
00054                     $row['filename'],
00055                     $row['fulltext'],
00056                     $row['hash0'],
00057                     $row['hash1'],
00058                     $row['comment']
00059                 );
00060             }
00061         }
00062         return $answers;
00063     }
```

References DB::$db, $table, and HT::error().

Referenced by getByFileName(), and getByObject().

Here is the call graph for this function:

Here is the caller graph for this function:



getByFileName()

```
AnswerDAO::getByFileName (
              $fileName)
```

Retrieves Answer objects associated with a specific file name.

Parameters

string	*$fileName*	Name of the file to search for.

Returns

Answer[] Array of Answer objects matching file name.

Definition at line 101 of file AnswerDAO.php.

```
00102     {
00103         $where = ['filename' => $fileName];
00104         return $this->get($where);
00105     }
```

References get().

Here is the call graph for this function:

getByObject()

```
AnswerDAO::getByObject (
              $student,
              $subQuestion = null,
              $answer = null)
```

Retrieves Answer objects based on given Student, SubQuestion, and optionally, specific Answer.

Parameters

Student	*$student*	The student object.
SubQuestion | null	*$subQuestion*	Optional subquestion filter.
Answer | null	*$answer*	Optional specific answer filter.

Returns

Answer[] Filtered array of Answer objects.

Definition at line 73 of file AnswerDAO.php.

```
00074     {
00075         $where = ['student_email' => $student->getEmail()];
00076         if ($subQuestion) {
00077             $where['subquestion_name'] = $subQuestion->getName();
00078         }
00079         $got = $this->get($where);
00080         if ($answer) { // Use the short file name to filter, not
00081             $answers = [];
00082             foreach ($got as $a) {
00083                 if ($answer->getShortFileName() == $a->getShortFileName()) {
00084                     $answers[] = $a;
00085                 }
00086             }
00087         } else {
00088             $answers = $got;
00089         }
00090         // Sort the answers by short file name, for aesthetics
00091         $answers = self::sortByShortFileName($answers);
00092         return $answers;
00093     }
```

References get().

Here is the call graph for this function:

getProcessSummary()

```
AnswerDAO::getProcessSummary ()
```

Retrieves the accumulated process summary for error reporting.

Returns

string The accumulated process summary.

Definition at line 459 of file AnswerDAO.php.

```
00460     {
00461         return $this->processSummary;
00462     }
```

References $processSummary.

getStudentZip()

```
AnswerDAO::getStudentZip (
              $student)  [private]
```

Locates and returns the ZIP file submitted by a student. (Or the one that was created by Grader, if finals.)

Parameters

Student	*$student*	The student object.

Returns

string Path to the located ZIP file.

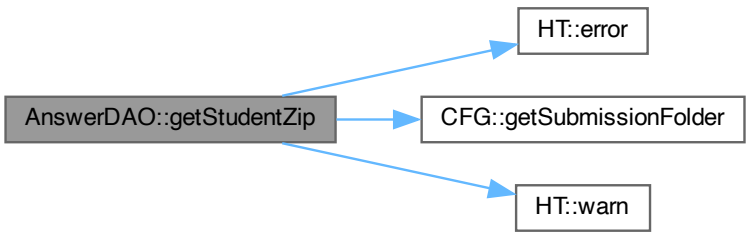
Definition at line 114 of file AnswerDAO.php.

```
00115     {
00116         $zipFile = "";
00117         // Try the emailed zips first
00118         $email = $student->getEmail();
00119         $section = $student->getSection();
00120         $name = $student->getName();
00121         $leadIn = $student->mkLeadIn();
00122         $gPat = CFG::$emailFolder . "*$email*.zip";
00123         $fNames = glob($gPat);
00124         $nFiles = count($fNames);
00125         if ($nFiles == 0) { // Try the section folder
00126             $gPat = CFG::getSubmissionFolder($section) . "*$email*.zip";
00127             $fNames = glob($gPat);
00128             $nFiles = count($fNames);
00129         }
00130         if ($nFiles == 0) { // Try the section folder with student name
00131             $gPat = CFG::getSubmissionFolder($section) . "*$name*.zip";
00132             $fNames = glob($gPat);
00133             $nFiles = count($fNames);
00134         }
00135         if ($nFiles > 1) {
00136             $this->processErrors .= HT::warn(
00137                 "$leadIn Found $nFiles Zip files. Will use the latest one.",
00138                 true
00139             );
00140             // Ref: https://stackoverflow.com/a/35925596
00141             usort($fNames, function ($a, $b) {
00142                 return filemtime($b) - filemtime($a);
00143             });
00144             $nFiles = 1;
00145         }
00146         if ($nFiles != 1) {
00147             $this->processErrors .= HT::error(
00148                 "$leadIn Found $nFiles Zip files. Please investigate!",
00149                 true
00150             );
00151             $student->setStatus("Absent")->update();
00152         } else {
00153             $zipFile = $fNames[0];
00154             $student->setZipFile($zipFile);
00155         }
00156         return $zipFile;
00157     }
```

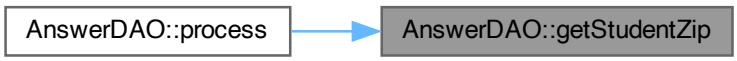
References CFG::$emailFolder, HT::error(), CFG::getSubmissionFolder(), and HT::warn().

Referenced by process().

Here is the call graph for this function:



Here is the caller graph for this function:



getSubQuestionName()

```
static AnswerDAO::getSubQuestionName (
              $shortFileName)  [static], [private]
```

Retrieves the subquestion name associated with a given short file name.

Parameters

string	*$shortFileName*	The short file name.

Returns

string Subquestion name, if found.

Definition at line 292 of file AnswerDAO.php.

```
00293     {
00294         $rubricDao = new RubricDAO();
00295         $rubrics = $rubricDao->get([
00296             'short_filename' => $shortFileName
00297         ]);
00298         $subQuestionName = "";
00299         if (count($rubrics) > 0) {
00300             $subQuestionName = $rubrics[0]->getSubQuestionName();
00301         }
00302         return $subQuestionName;
00303     }
```

mvStudentAnswers()

```
AnswerDAO::mvStudentAnswers (
              $student)  [private]
```

Moves student answer files from the working directory to the target answer folder.

Parameters

Student	*$student*	The student object.

Returns

string[] Array mapping short file names to their new paths.

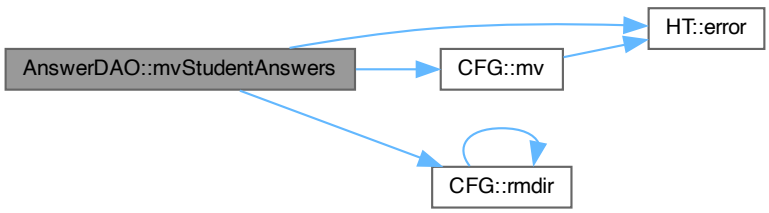
Definition at line 245 of file AnswerDAO.php.

```
00246     {
00247         // Iterate over the files in unzipped (in foo)
00248         $refFileDAO = new RefFileDAO();
00249         $answers = glob(
00250             "foo/{*,*/*,*/*/*,*/*/*/*,*/*/*/*/*}/*.{php,html}",
00251             GLOB_BRACE
00252         );
00253         if (empty($answers)) {
00254             return [];
00255         }
00256         $shortNames = self::stripCommon($answers);
00257         $answerFolder = $student->getAnswerFolder($student);
00258         if (realpath($answerFolder)) {
00259             $student->delete();
00260         }
00261         @mkdir($answerFolder, 0777, true);
00262         $moved = [];
00263         foreach ($answers as $key => $a) {
00264             $shortFileName = $shortNames[$key];
00265             $refFile = $refFileDAO->getRefFile($shortFileName, 'resources');
00266             if (empty($refFile)) {
00267                 $leadIn = $student->mkLeadIn();
00268                 $this->processErrors .= HT::error(
00269                     "$leadIn Created an unknown file <code>$shortFileName</code>",
00270                     true
00271                 );
00272                 continue;
00273             }
00274             $refShortFileName = $refFile->getShortFileName();
00275             $target = $answerFolder . $refShortFileName;
00276             CFG::mv($a, $target);
00277             $moved[$refShortFileName] = $target;
00278         }
00279         CFG::rmdir("foo");
00280         return $moved;
00281     }
```

References HT::error(), CFG::mv(), and CFG::rmdir().

Referenced by process().

Here is the call graph for this function:



Here is the caller graph for this function:

process()

```
AnswerDAO::process (
              $student)
```

Main processing pipeline: extracts answers from ZIP, maps them to subquestions, stores them.

Parameters

Student	*$student*	The student to process.

Returns

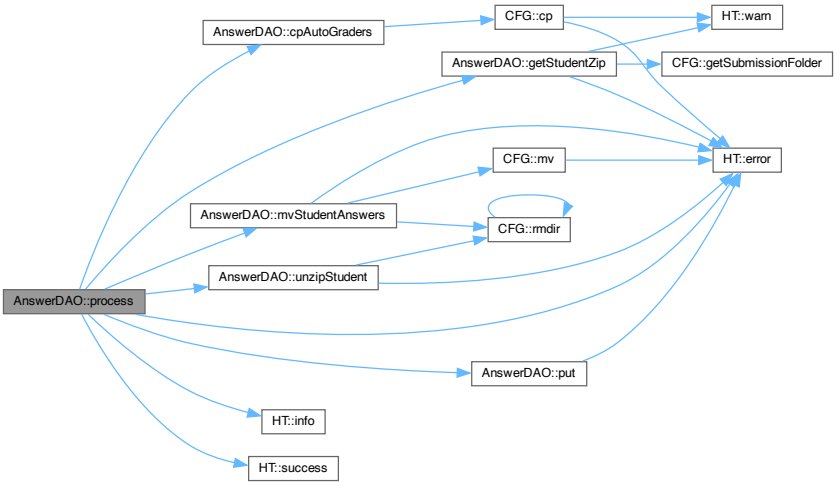
void

Definition at line 358 of file AnswerDAO.php.

```
00359     {
00360         $name = $student->getName();
00361         $studentEmail = $student->getEmail();
00362         $section = $student->getSection();
00363         $msg = $student->mkLeadIn();
00364         if ($zipFile = $this->getStudentZip($student)) {
00365             $dirName = realpath(dirname($zipFile));
00366             $baseName = basename($zipFile);
00367             $msg .= HT::success("Located the zip file: <br>
00368              Folder: <code>$dirName</code><br>
00369              File: <code>$baseName</code>", true);
00370             if ($this->unzipStudent($student)) {
00371                 $answerFiles = $this->mvStudentAnswers($student);
00372                 $count = count($answerFiles);
00373                 if ($count) {
00374                     $msg .= HT::success("$count files successfully unzipped.", true);
00375                 } else {
00376                     $msg .= HT::error("No files unzipped.", true);
00377                 }
00378                 $answers = [];
00379                 foreach ($answerFiles as $shortFileName => $fileName) {
00380                     $subQuestionName = self::getSubQuestionName($shortFileName);
00381                     if ($subQuestionName) {
00382                         $answers[] = new Answer(
00383                             $studentEmail,
00384                             $subQuestionName,
00385                             $fileName
00386                         );
00387                     }
00388                 }
00389                 $count = count($answers);
00390                 if ($count) {
00391                     $msg .= HT::success("$count Answer objects successfully created.", true);
00392                 } else {
00393                     $msg .= HT::error("No Answer objects created.", true);
00394                 }
00395                 $this->put($student, $answers);
00396                 $count = DB::$db->affectedRows();
00397                 if ($count) {
00398                     $msg .= HT::success("$count records went into the database.", true);
00399                 } else {
00400                     $msg .= HT::error("No database rows affected.", true);
00401                 }
00402                 $count = $this->cpAutoGraders($student);
00403                 if ($count) {
00404                     $msg .= HT::success("Attempted autograded question. Copied.", true);
00405                 } else {
00406                     $msg .= HT::error("No autograders copied.", true);
00407                 }
00408             }
00409         }
00410         $msg .= $this->processErrors;
00411         $this->processSummary .= $this->processErrors;
00412         $this->processErrors = "";
00413         HT::info($msg);
00414     }
```

References DB::$db, $processErrors, cpAutoGraders(), HT::error(), getStudentZip(), HT::info(), mvStudentAnswers(), put(), HT::success(), and unzipStudent().

Here is the call graph for this function:



put()

```
AnswerDAO::put (
              $student,
              $answers)  [private]
```

Saves an array of Answer objects to the database for a given student.

Parameters

Student	*$student*	The student object.
Answer[ ]	*$answers*	Array of Answer objects to save.

Returns

void

Definition at line 312 of file AnswerDAO.php.

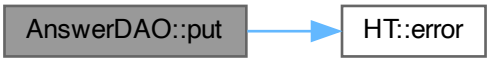
```
00313     {
00314         $leadIn = $student->mkLeadIn();
00315         if (count($answers) <= 0) { // No answers found. An error!
00316             $this->processErrors .= HT::error(
00317                 "$leadIn No answers found.
00318             Probably absent or alias-zipping.",
00319                 true
00320             );
00321         } else {
00322             $table = self::$table;
00323             $columns = [
00324                 'student_email',
00325                 'subquestion_name',
00326                 'filename',
00327                 'fulltext',
00328                 'hash0',
00329                 'hash1',
00330             ];
00331             $rows = [];
00332             foreach ($answers as $answer) {
```

```
00333                 $studentEmail = $student->getEmail();
00334                 $subQuestionName = $answer->getSubQuestionName();
00335                 $fileName = $answer->getFileName();
00336                 $fullText = DB::$db->getFileContent($fileName);
00337                 $hash0 = Report::hash($fileName);
00338                 $hash1 = Report::hash($fileName, false);
00339                 $rows[] = [
00340                     $studentEmail,
00341                     $subQuestionName,
00342                     $fileName,
00343                     $fullText,
00344                     $hash0,
00345                     $hash1,
00346                 ];
00347             }
00348             DB::$db->update($table, $columns, $rows);
00349         }
00350     }
```

References DB::$db, $table, and HT::error().

Referenced by process().

Here is the call graph for this function:



Here is the caller graph for this function:

setProcessSummary()

```
AnswerDAO::setProcessSummary (
            $processSummary)
```

Sets the accumulated process summary for error reporting.

Parameters

string	*$processSummary*	The summary text to set.

Returns

self

Definition at line 470 of file AnswerDAO.php.

```
00470                                                   : self
00471     {
00472         $this->processSummary = $processSummary;
00473         return $this;
00474     }
```

References $processSummary.

stripCommon()

```
static AnswerDAO::stripCommon (
              $strings)  [static], [private]
```

Strips common prefixes or paths from an array of file paths for standardization.

Parameters

array | string	*$strings*	Array of file paths or a single string.

Returns

array|string Array of stripped paths or stripped string.

Definition at line 216 of file AnswerDAO.php.

```
00217     {
00218         if (CFG::isQuiz()) {
00219             $stripped = str_replace("foo/", "", $strings);
00220         } else {
00221             // need to remove silly paths students have used
00222             $difPosition = 10000; // just a large number
00223             $count = count($strings);
00224             for ($i = 1; $i < $count; $i++) {
00225                 $position = strspn($strings[$i - 1] ^ $strings[$i], "\0");
00226                 $difPosition = min([$difPosition, $position]);
00227             }
00228             if ($strings[0][$difPosition - 1] == 'q') {
00229                 $difPosition--;
00230             }
00231             $stripped = [];
00232             foreach ($strings as $string) {
00233                 $stripped[] = substr($string, $difPosition);
00234             }
00235         }
00236         return $stripped;
00237     }
```

References CFG::isQuiz().

Here is the call graph for this function:

unzipStudent()

```
AnswerDAO::unzipStudent (
              $student)  [private]
```

Unzips the student's uploaded ZIP file into a working directory.

Parameters

Student	*$student*	The student object.

Returns

bool True on success, false on failure.

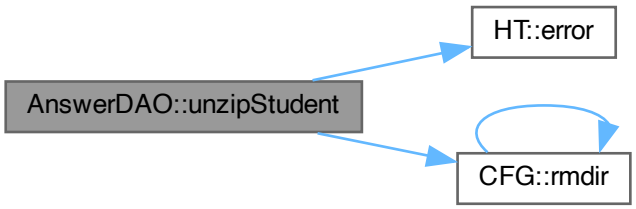
Definition at line 165 of file AnswerDAO.php.

```
00166     {
00167         $zipFile = $student->getZipFile();
00168         $zip = new ZipArchive;
00169         $res = $zip->open($zipFile);
00170         if ($res === TRUE) {
00171             CFG::rmdir("foo");
00172             $zip->extractTo("foo");
00173         } else {
00174             $leadIn = $student->mkLeadIn();
00175             $this->processErrors .= HT::error(
00176                 "$leadIn Error unzipping to <var>foo</var>!<br>
00177              [<var>$zipFile</var>]",
00178                 true
00179             );
00180             return false;
00181         }
00182         $zip->close();
00183         return true;
00184     }
```

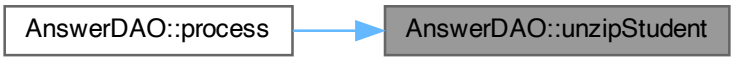
References HT::error(), and CFG::rmdir().

Referenced by process().

Here is the call graph for this function:



Here is the caller graph for this function:



update()

```
AnswerDAO::update (
              $answer)
```

Updates an existing Answer record in the database.

Parameters

Answer	*$answer*	The Answer object containing updated data.

Returns

void

Definition at line 422 of file AnswerDAO.php.

```
00423     {
00424         $table = self::$table;
00425         $columns = [
00426             'student_email',
00427             'subquestion_name',
00428             'filename',
00429             'fulltext',
00430             'hash0',
00431             'hash1',
00432             'comment'
00433         ];
00434         $studentEmail = $answer->getStudentEmail();
00435         $subQuestionName = $answer->getSubQuestionName();
00436         $fileName = $answer->getFileName();
00437         $fulltext = DB::$db->sanitize($answer->getFullText());
00438         $hash0 = $answer->getHash0();
00439         $hash1 = $answer->getHash1();
00440         $comment = $answer->getComment();
00441         $rows = [
00442             $studentEmail,
00443             $subQuestionName,
00444             $fileName,
00445             $fulltext,
00446             $hash0,
00447             $hash1,
00448             $comment
00449         ];
00450         // DB::$db->put($table, $columns, $rows);
00451         DB::$db->update($table, $columns, $rows);
00452     }
```

References DB::$db, and $table.

4.2.3 Member Data Documentation

$processErrors

```
AnswerDAO::$processErrors = ""  [private]
```

Definition at line 15 of file AnswerDAO.php.

Referenced by process().

$processSummary

```
AnswerDAO::$processSummary = ""  [private]
```

Definition at line 18 of file AnswerDAO.php.

Referenced by getProcessSummary(), and setProcessSummary().

$table

```
AnswerDAO::$table  [static], [private]
```

Definition at line 12 of file AnswerDAO.php.

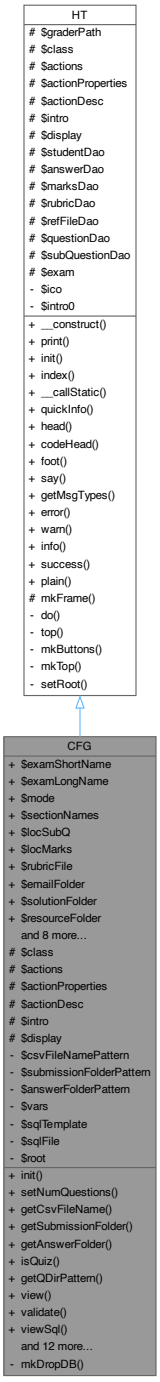
Referenced by __construct(), get(), put(), and update().

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

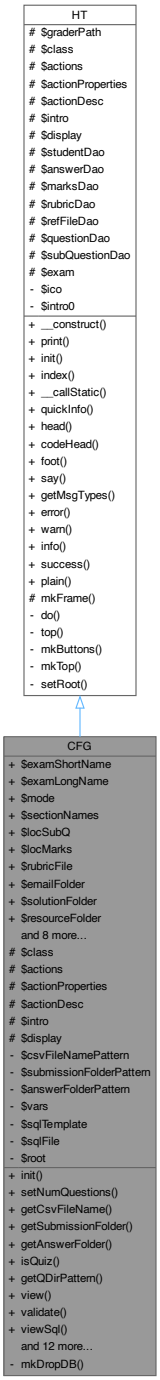
- AnswerDAO.php

4.3 CFG Class Reference

Inheritance diagram for CFG:



Collaboration diagram for CFG:



Static Public Member Functions

- static init ()
- static setNumQuestions ($numQuestions)
- static getCsvFileName ($sectionName, $studentName="")
- static getSubmissionFolder ($sectionName, $studentName="")
- static getAnswerFolder ($sectionName, $studentName="")

- static isQuiz ()
- static getQDirPattern ()
- static view ()
- static validate ()
- static viewSql ()
- static rmdir ($dir)
- static mv ($from, $to)
- static cpr ($from, $to)
- static cp ($from, $to, $mkDir=false)
- static mkFileName ($fileName)
- static getDbPrefix ()
- static mkSql ($toString=false)
- static runSql ()
- static dropDB ()
- static mkTableName ($table)
- static toSnake ($input)
- static toCamel ($input)

Static Public Member Functions inherited from HT

- static init ()
- static index ()
- static __callStatic ($method, $args)
- static quickInfo ($side)
- static head ($toStr=false, $title="Grader by Manoj", $refresh='')
- static codeHead ($title="Code Listing by Manoj")
- static foot ($toStr=false)
- static say ($what, $class='plain', $toStr=false, $showIcon=false)
- static getMsgTypes ()
- static error ($what, $toStr=false)
- static warn ($what, $toStr=false)
- static info ($what, $toStr=false)
- static success ($what, $toStr=false)
- static plain ($what, $toStr=false)

Static Public Attributes

- static $examShortName
- static $examLongName
- static $mode = "LT"
- static $sectionNames
- static $locSubQ
- static $locMarks
- static $rubricFile = "./rubric.csv"
- static $emailFolder
- static $solutionFolder
- static $resourceFolder
- static $configFile = "../config.php"
- static $gradeQuestions
- static $autoGradables
- static $dbUser
- static $dbPass
- static $dbName
- static $dbPrefix
- static $numQuestions

Static Protected Attributes

- static $class = __CLASS__
- static $actions
- static $actionProperties
- static $actionDesc
- static $intro
- static $display

Static Protected Attributes inherited from HT

- static $graderPath
- static $class = __CLASS__
- static $actions
- static $actionProperties = [ ]
- static $actionDesc
- static $intro
- static $display = ["class" => "success", "title" => ""]
- static $studentDao
- static $answerDao
- static $marksDao
- static $rubricDao
- static $refFileDao
- static $questionDao
- static $subQuestionDao
- static $exam

Static Private Member Functions

- static mkDropDB ()

Static Private Attributes

- static $csvFileNamePattern
- static $submissionFolderPattern
- static $answerFolderPattern
- static $vars
- static $sqlTemplate = "../model/setup.template.sql"
- static $sqlFile = 'setup.sql'
- static $root

Additional Inherited Members

Public Member Functions inherited from HT

- __construct ()
- print ($toStr=true)

Static Protected Member Functions inherited from HT

- static mkFrame ($side='left')

4.3.1 Detailed Description

Class CFG

Manages configuration and setup for an exam grading system, including exam details, database connection, grading configuration, and utility methods.

This class reads settings typically defined in an external `config.php` file, validates the required folders and files, and manages SQL database setup for grading purposes.

Inherits from the HT class for rendering HTML and messaging.

Definition at line 14 of file CFG.php.

4.3.2 Member Function Documentation

cp()

```
static CFG::cp (
              $from,
              $to,
              $mkDir = false)  [static]
```

Copies a file or directory.

Parameters

string	*$from*	
string	*$to*	
bool	*$mkDir*	

Returns

bool

Definition at line 709 of file CFG.php.

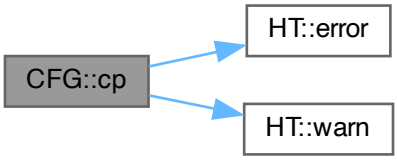
```
00710   {
00711     $error = "Error copying <code>$from</code> to <code>$to</code>:";
00712     $error = str_replace('///', '/', $error);
00713     if (is_dir($from)) {
00714       // HT::error("$error Source is a folder.");
00715       // try cpr after creating the $to folder: Dicey!
00716       if (!is_dir($to)) {
00717         mkdir($to, 0777, true);
00718       }
00719       self::cpr($from, $to);
00720       return false;
00721     }
00722     if (!is_readable($from)) {
00723       HT::error("$error Source not found/readable.");
00724       return false;
00725     }
00726     if (is_dir($to)) {
00727       if (!file_exists($to)) {
00728         HT::error("$error Target not found/writable.");
00729         return false;
00730       }
00731       $to .= "/" . basename($from);
00732     }
00733     $toDir = dirname($to);
00734     if (!file_exists($toDir)) {
00735       if ($mkDir && mkdir($toDir)) {
```

```
00736         HT::warn("$error Target directory was not found. Created it.");
00737       } else {
00738         HT::error("$error Target directory not found/writable.");
00739         return false;
00740       }
00741     }
00742     copy($from, $to);
00743     return true;
00744   }
```

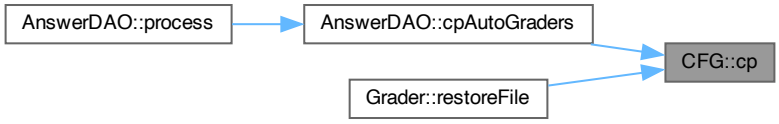
References HT::error(), and HT::warn().

Referenced by AnswerDAO::cpAutoGraders(), and Grader::restoreFile().

Here is the call graph for this function:



Here is the caller graph for this function:



cpr()

```
static CFG::cpr (
              $from,
              $to)  [static]
```

Recursively copies a directory.

Parameters

string	*$from*	
string	*$to*	

Returns

void

Definition at line 679 of file CFG.php.

```
00680   {
00681     if (is_dir($from)) {
00682       $dir_handle = opendir($from);
00683       while ($file = readdir($dir_handle)) {
00684         if ($file != "." && $file != "..") {
00685           if (is_dir($from . "/" . $file)) {
00686             if (!is_dir($to . "/" . $file)) {
00687               mkdir($to . "/" . $file);
00688             }
00689             self::cpr($from . "/" . $file, $to . "/" . $file);
00690           } else {
00691             copy($from . "/" . $file, $to . "/" . $file);
00692           }
00693         }
00694       }
00695       closedir($dir_handle);
00696     } else {
00697       copy($from, $to);
00698     }
00699   }
```

dropDB()

```
static CFG::dropDB ()  [static]
```

Drops all database tables related to the exam.

Returns

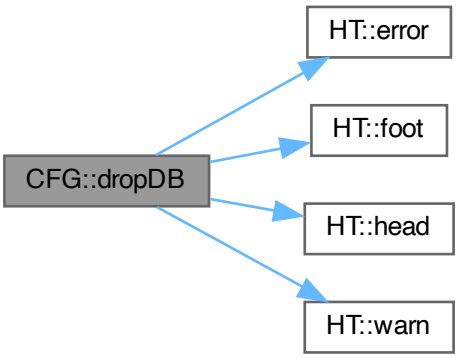
void

Definition at line 869 of file CFG.php.

```
00870   {
00871     HT::head();
00872     $sql = self::mkDropDB();
00873     if (empty($sql)) {
00874       HT::error("Error creating drop DB script!");
00875       return false;
00876     }
00877     $dbName = self::$dbName;
00878     if (!isset($_POST['confirmDropDB'])) {
00879       $str = "Dropping DB ($dbName)!<br><pre><code>$sql</code></pre><br>
00880           Really drop it? <table align='right'><tr>
00881           <td><form method='post' action='?do'>
00882           <input type='hidden' name='do' value='dropDB'>
00883           <input type='hidden' name='confirmDropDB' value='dummy'>
00884           <button type='submit' class='error'>
00885           <strong>Proceed to drop it</strong>
00886           </button></form></td>
00887           </tr></table>";
00888       HT::warn($str);
00889     } else {
00890       DB::$db->multiQuery($sql);
00891       $error = DB::$db->getError();
00892       if ($error) {
00893         $error = "<br>$error";
00894       }
00895       HT::error("Dropping DB ($dbName)!<br><pre><code>$sql</code></pre>$error");
00896       HT::warn("Please run the DB setup script again.
00897       <a href='?do=runSql'>Run it!</a>");
00898       // self::runSql();
00899     }
00900     HT::foot();
00901   }
```

References DB::$db, $dbName, $sql, HT::error(), HT::foot(), HT::head(), and HT::warn().

Here is the call graph for this function:



getAnswerFolder()

```
static CFG::getAnswerFolder (
              $sectionName,
              $studentName = "")  [static]
```

Returns answer folder path.

Parameters

string	*$sectionName*	
string	*$studentName*	

Returns

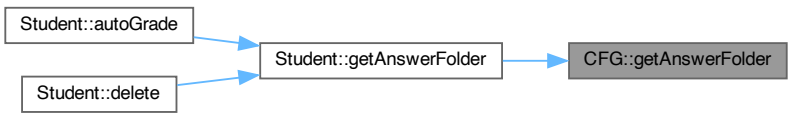
string

Definition at line 233 of file CFG.php.

```
00234   {
00235     return str_replace(
00236       ["{sectionName}", "{studentName}"],
00237       [trim($sectionName), trim($studentName)],
00238       self::$answerFolderPattern
00239     );
00240   }
```

Referenced by Student::getAnswerFolder().

Here is the caller graph for this function:



getCsvFileName()

```
static CFG::getCsvFileName (
              $sectionName,
              $studentName = "")  [static]
```

Generates CSV file name based on section and student name.

Parameters

string	*$sectionName*	
string	*$studentName*	

Returns

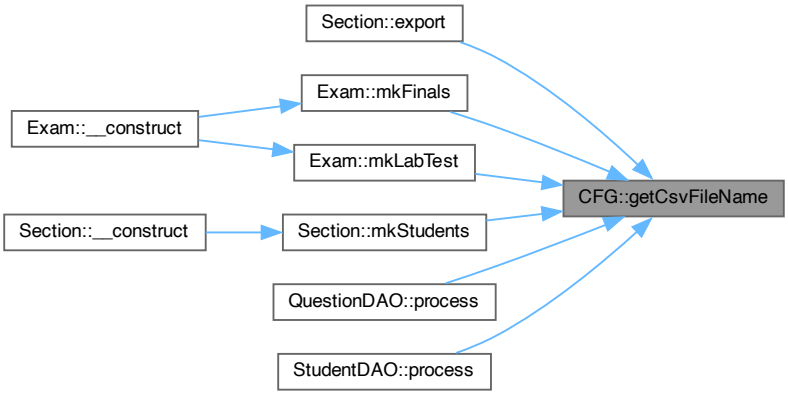
string

Definition at line 201 of file CFG.php.

```
00202   {
00203     return str_replace(
00204       ["{sectionName}", "{studentName}"],
00205       [trim($sectionName), trim($studentName)],
00206       self::$csvFileNamePattern
00207     );
00208   }
```

Referenced by Section::export(), Exam::mkFinals(), Exam::mkLabTest(), Section::mkStudents(), QuestionDAO::process(), and StudentDAO::process().

Here is the caller graph for this function:



getDbPrefix()

```
static CFG::getDbPrefix ()  [static]
```

Generates database table prefix based on exam short name.

Returns

string

Definition at line 767 of file CFG.php.

```
00768   {
00769     $dbPrefix = preg_replace("/\s+/", "", self::$examShortName);
00770     $dbPrefix = strtolower($dbPrefix) . "_";
00771     return $dbPrefix;
00772   }
```

References $dbPrefix.

getQDirPattern()

```
static CFG::getQDirPattern ()  [static]
```

Creates a regex pattern for question directories.

Returns

string

Definition at line 257 of file CFG.php.

```
00258   {
00259     // Create a regular expression like "@/q[123]/..@
00260     $pattern = "@/[qQ][";
00261     for ($i = 1; $i <= self::$numQuestions; $i++) {
00262       $pattern .= $i;
00263     }
00264     $pattern .= "]/..@";
00265     return $pattern;
00266   }
```

getSubmissionFolder()

```
static CFG::getSubmissionFolder (
              $sectionName,
              $studentName = "")  [static]
```

Returns submission folder path.

Parameters

string	*$sectionName*	
string	*$studentName*	

Returns

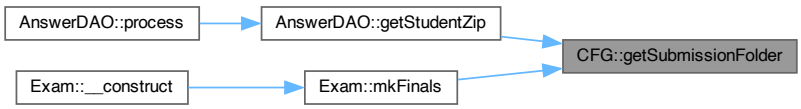
string

Definition at line 217 of file CFG.php.

```
00218   {
00219     return str_replace(
00220       ["{sectionName}", "{studentName}"],
00221       [trim($sectionName), trim($studentName)],
00222       self::$submissionFolderPattern
00223     );
00224   }
```

Referenced by AnswerDAO::getStudentZip(), and Exam::mkFinals().

Here is the caller graph for this function:



init()

```
static CFG::init ()  [static]
```

Initializes configuration variables from global $GLOBALS. Also prepares DB connection and paths.

Returns

void

Definition at line 167 of file CFG.php.

```
00168   {
00169     $vars = self::$vars;
00170     foreach ($vars as $v) {
00171       if (isset($GLOBALS[$v])) { // Necessary for $mode = "ft"
00172         self::$$v = $GLOBALS[$v];
00173       }
00174     }
00175     self::$dbName = strtolower(self::$dbName);
00176     $dbHost = "localhost";
00177     $db = new DB($dbHost, self::$dbUser, self::$dbPass, self::$dbName);
00178     self::$dbPrefix = self::getDbPrefix();
00179     self::$configFile = realpath(self::$configFile);
00180     self::$root = $GLOBALS['root'];
00181   }
```

References $vars.

isQuiz()

```
static CFG::isQuiz ()  [static]
```

Checks if the mode is set to 'FT' (Final Test).

Returns

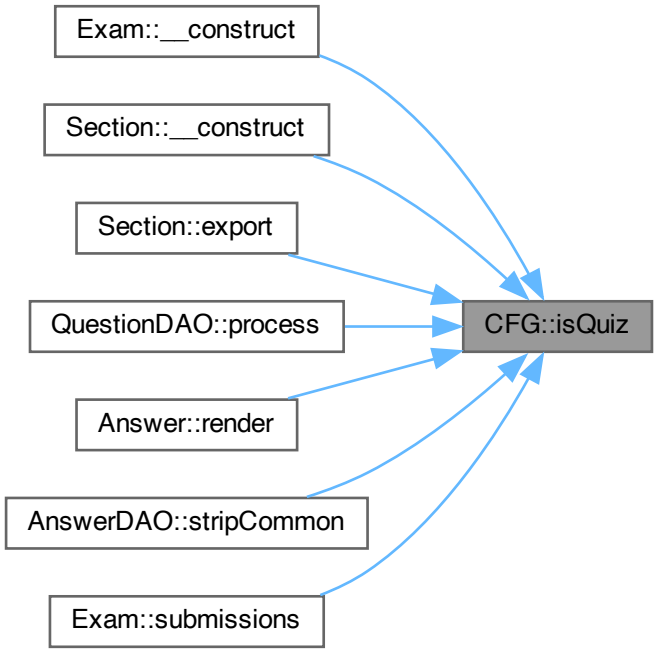
bool

Definition at line 247 of file CFG.php.

```
00248   {
00249     return trim(strtolower(self::$mode)) == "ft";
00250   }
```

Referenced by Exam::__construct(), Section::__construct(), Section::export(), QuestionDAO::process(), Answer::render(), AnswerDAO::stripCommon(), and Exam::submissions().

Here is the caller graph for this function:



mkDropDB()

```
static CFG::mkDropDB ()  [static], [private]
```

Generates SQL to drop all exam-related tables.

Returns

string

Definition at line 806 of file CFG.php.

```
00807   {
00808     $dbPrefix = self::getDbPrefix();
00809     $dbName = strtolower(self::$dbName);
00810     $search = ["{dbName}", "{dbPrefix}"];
00811     $replace = [$dbName, $dbPrefix];
00812     $sql = 'DROP DATABASE IF EXISTS `{dbName}`;';
00813     $sql = str_replace($search, $replace, $sql);
00814
00815
00816     // Pattern for matching table names (SQL LIKE syntax)
00817     $pattern = "$dbPrefix%";
00818
00819     // SQL to get the list of tables that match the wildcard pattern
00820     $sql0 = "SELECT TABLE_NAME FROM INFORMATION_SCHEMA.TABLES WHERE TABLE_SCHEMA = '$dbName' AND
      TABLE_NAME LIKE '$pattern'";
00821
00822     $sql = "";
00823     if (DB::$db->query($sql0)) {
00824       $tables = DB::$db->fetchAll();
00825       foreach ($tables as $row) {
00826         $table = $row["TABLE_NAME"];
00827         $sql .= "DROP TABLE `$table`;\n";
00828       }
00829     } else {
00830       echo "No tables found";
00831     }
00832     // var_dump($sql); die;
00833     return $sql;
00834   }
```

References DB::$db, $dbName, $dbPrefix, $sql, and $sql0.

mkFileName()

```
static CFG::mkFileName (
              $fileName)  [static]
```

Converts absolute path to relative path under document root.

Parameters

string	*$fileName*	

Returns

string

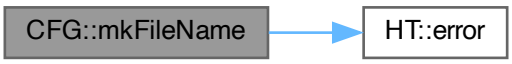
Definition at line 752 of file CFG.php.

```
00753   {
00754     $root = $_SERVER['DOCUMENT_ROOT'];
00755     $fileName0 = realpath($fileName);
00756     if (stripos($fileName, $root) !== 0) { // Does not start with DocRoot
00757       HT::error("$fileName: not found under $root.");
00758     }
00759     return substr($fileName0, strlen($root));
00760   }
```

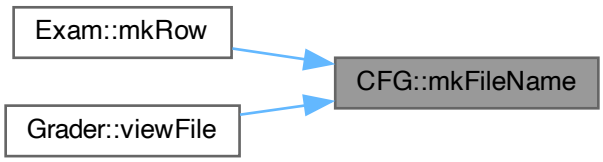
References $root, and HT::error().

Referenced by Exam::mkRow(), and Grader::viewFile().

Here is the call graph for this function:



Here is the caller graph for this function:



mkSql()

```
static CFG::mkSql (
              $toString = false)  [static]
```

Creates and saves SQL file from template. Danger: System calls. To be customized for Windows.

Parameters

bool	*$toString*	

Returns

string

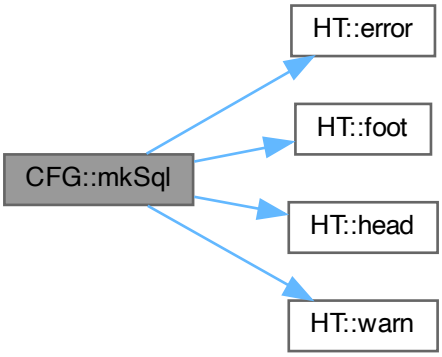
Definition at line 780 of file CFG.php.

```
00781   {
00782     $template = self::$sqlTemplate;
00783     $dbPrefix = self::getDbPrefix();
00784     $dbName = strtolower(self::$dbName);
00785     $search = ["{dbName}", "{dbPrefix}"];
00786     $replace = [$dbName, $dbPrefix];
00787     $sql = file_get_contents($template);
00788     $sql = str_replace($search, $replace, $sql);
00789     $sqlFile = self::$root . DIRECTORY_SEPARATOR . self::$sqlFile;
00790     HT::head($toString);
00791     if (!file_put_contents($sqlFile, $sql)) {
00792       HT::error("Error creating $sqlFile while setting up the database.<br>
00793       Cannot safely continue without fixing this issue!");
```

```
00794       return "";
00795     }
00796     HT::warn("Created $sqlFile.");
00797     HT::foot($toString);
00798     return $sql;
00799   }
```

References $dbName, $dbPrefix, $sql, $sqlFile, HT::error(), HT::foot(), HT::head(), and HT::warn().

Here is the call graph for this function:



mkTableName()

```
static CFG::mkTableName (
              $table)  [static]
```

Prepends database prefix to table name if not already present.

Parameters

string	*$table*	

Returns

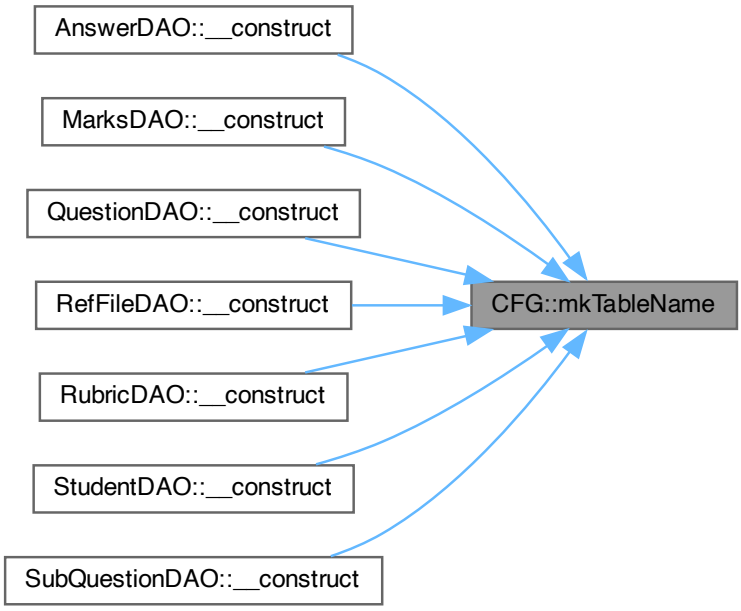
string

Definition at line 909 of file CFG.php.

```
00910   {
00911     $pf = self::getDbPrefix();
00912     if (strpos($table, $pf) !== 0) {
00913       $table = $pf . $table;
00914     }
00915     return $table;
00916   }
```

Referenced by AnswerDAO::__construct(), MarksDAO::__construct(), QuestionDAO::__construct(), RefFileDAO::__construct(), RubricDAO::__construct(), StudentDAO::__construct(), and SubQuestionDAO::__construct().

Here is the caller graph for this function:



mv()

```
static CFG::mv (
              $from,
              $to)  [static]
```

Moves a file or folder (with copy as fallback). Danger: System calls.

Parameters

string	*$from*	
string	*$to*	

Returns

bool

Definition at line 639 of file CFG.php.

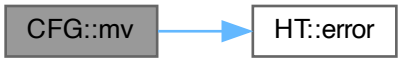
```
00640   {
00641     $toFolder = dirname($to);
00642     if (!is_dir($toFolder)) {
00643       mkdir($toFolder, 0777, true);
00644     }
00645     if (is_dir($to)) {
00646       $to .= "/" . $from;
00647     }
00648     // Make a subfolder if needed
```

```
00649     $error = "Error moving <code>$from</code> to <code>$to</code>:";
00650     if (is_dir($from)) {
00651       HT::error("$error Source is a folder.");
00652       return false;
00653     }
00654     if (!is_readable($from)) {
00655       HT::error("$error Source not found.");
00656       return false;
00657     }
00658     if (!rename($from, $to)) {
00659       if (copy($from, $to)) {
00660         if (!unlink($from)) {
00661           HT::error("Error moving $from to $to: Could not remove source.");
00662           return false;
00663         }
00664       } else {
00665         HT::error("Error moving $from to $to: Could not copy source.");
00666         return false;
00667       }
00668     }
00669     return true;
00670   }
```

References HT::error().

Referenced by AnswerDAO::mvStudentAnswers().

Here is the call graph for this function:



Here is the caller graph for this function:

rmdir()

```
static CFG::rmdir (
              $dir)  [static]
```

Recursively deletes a directory. Danger: System call.

Parameters

string	*$dir*	

Returns

bool

Definition at line 621 of file CFG.php.

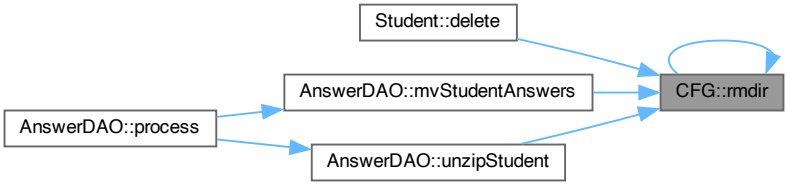
```
00622   {
00623     if (is_dir($dir)) {
00624       $files = array_diff(scandir($dir), array('.', '..'));
00625       foreach ($files as $file) {
00626         (is_dir("$dir/$file")) ? self::rmdir("$dir/$file") : unlink("$dir/$file");
00627       }
00628       return rmdir($dir);
00629     }
00630   }
```

References rmdir().

Referenced by Student::delete(), AnswerDAO::mvStudentAnswers(), rmdir(), and AnswerDAO::unzipStudent().

Here is the call graph for this function:

Here is the caller graph for this function:



runSql()

```
static CFG::runSql ()  [static]
```

Runs SQL script to create and set up database.

Returns

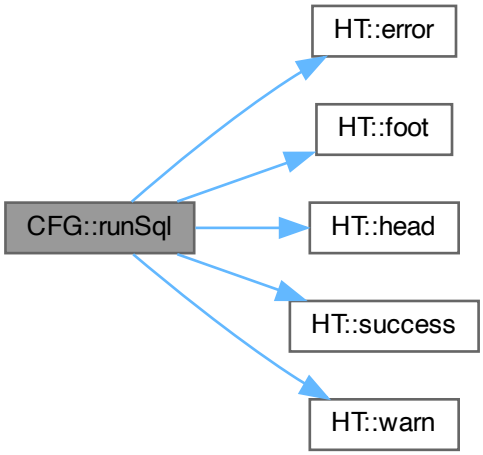
bool

Definition at line 841 of file CFG.php.

```
00842   {
00843     $template = self::$sqlTemplate;
00844     $sql = self::mkSql();
00845     if (empty($sql)) {
00846       HT::error("Error running $template!");
00847       return false;
00848     }
00849     $sqlFile = self::$root . DIRECTORY_SEPARATOR . self::$sqlFile;
00850     $db = new DB(dbPass: self::$dbPass);
00851     $status = $db->importSQL($sqlFile);
00852     if ($status) {
00853       $msg = HT::success("Executed {$status['success']} successfully.", true);
00854       $msg .= HT::error("{$status['errors']} statements failed.", true);
00855     } else {
00856       $msg = HT::error("Error setting up the database [CFG::runSql fails].", true);
00857     }
00858     HT::head();
00859     HT::warn("<h4>Result from <code>CFG::runSQL():</code></h4>" . $msg);
00860     HT::foot();
00861   }
```

References $sql, $sqlFile, HT::error(), HT::foot(), HT::head(), HT::success(), and HT::warn().

Here is the call graph for this function:



setNumQuestions()

```
static CFG::setNumQuestions (
              $numQuestions)  [static]
```

Sets the number of questions for the exam.

Parameters

int	*$numQuestions*	

Returns

void

Definition at line 189 of file CFG.php.

```
00190   {
00191     self::$numQuestions = $numQuestions;
00192   }
```

References $numQuestions.

toCamel()

```
static CFG::toCamel (
              $input)  [static]
```

Converts snake_case to camelCase. Ref: https://gist.github.com/carousel/1aacbea013d230768b3dec1a1

Parameters

string	*$input*	

Returns

string

Definition at line 937 of file CFG.php.

```
00938   {
00939     return lcfirst(str_replace(' ', '', ucwords(str_replace('_', ' ', $input))));
00940   }
```

toSnake()

```
static CFG::toSnake (
              $input)  [static]
```

Converts camelCase to snake_case. Ref: https://gist.github.com/carousel/1aacbea013d230768b3dec1a1

Parameters

string	*$input*	

Returns

string

Definition at line 925 of file CFG.php.

```
00926   {
00927     return strtolower(preg_replace('/(?<!^)[A-Z]/', '_$0', $input));
00928   }
```

validate()

```
static CFG::validate ()  [static]
```

Validates config settings, directories, and required files.

Returns

void

Definition at line 319 of file CFG.php.

```
00320   {
00321     HT::head();
00322     $error = false;
00323     $configFile = self::$configFile;
00324     $str = HT::info(
00325       "Validating <strong>" . self::$examShortName . ": " .
00326         self::$examLongName . "</strong><br>
00327       Config file: <a href='?do=view'><code>$configFile</code></a>",
00328       true
00329     );
00330     $root = self::$root;
00331     $realpath = realpath($root);
00332     if (!is_dir($root)) {
00333       $str .= HT::error("Exam folder (<code>$realpath</code>) not found!<br>
00334       Edit <code>$configFile</code> to set <code>\$root.</code>", true);
00335       $error = true;
00336       goto end;
00337     } else {
00338       $str .= HT::success("Exam folder (<code>$realpath</code>) located.", true);
00339     }
00340     if (self::isQuiz()) {
00341       $str .= HT::success(
00342         "Grading a finals (an eLearn quiz, as opposed to a Lab Test)!",
00343         true
00344       );
00345       self::$sectionNames = ['All'];
00346     }
00347     if (empty(self::$sectionNames)) {
00348       $str .= HT::error(
00349         "Section names (<code>\$sectionNames</code>) not specified!<br>
00350         Edit <code>$configFile</code> to set it as an array.",
00351         true
00352       );
00353       $error = true;
00354       goto end;
00355     } else {
00356       $str1 = implode(", ", self::$sectionNames);
00357       $str .= HT::success(
00358         "Found these section[s]: <strong>$str1</strong>.",
00359         true
00360       );
00361       $str0 = "";
00362       foreach (self::$sectionNames as $sectionName) {
00363         $csvFileName = self::getCsvFileName($sectionName);
00364         if (is_file($csvFileName)) {
00365           $csvFileName = realpath($csvFileName);
00366           $str0 .= HT::success(
00367             " &rarr; eLearn Grade Export file located:<br>
00368              <code>$csvFileName</code>",
00369             true
00370           );
00371         } else {
00372           $str0 .= HT::error(
00373             " &rarr; eLearn Grade Export file  not found.<br>
00374              <code>$csvFileName</code><br>
00375             Please create it: Export the grade for the section $sectionName as CSV.",
00376             true
00377           );
00378           $error = true;
00379         }
00380       }
00381       if ($error) {
00382         $str .= HT::warn($str0, true);
00383       } else {
00384         $str .= HT::info($str0, true);
00385       }
00386       $str0 = "";
00387       foreach (self::$sectionNames as $sectionName) {
00388         $submissionFolder = self::getSubmissionFolder($sectionName);
```

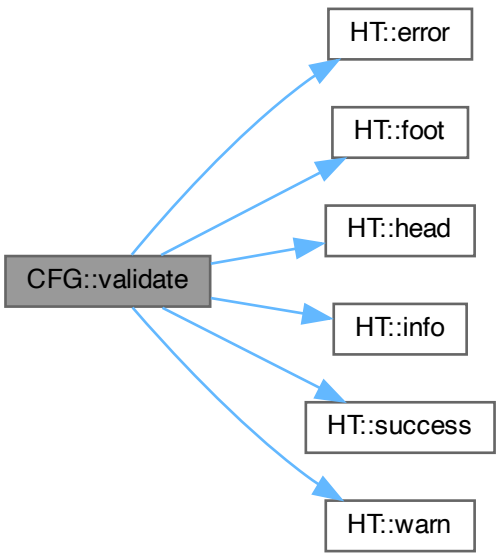
```
00389         if (is_dir($submissionFolder)) {
00390           $submissionFolder = realpath($submissionFolder);
00391           $nZips = count(glob($submissionFolder . "/*.zip"));
00392           if ($nZips > 0) {
00393             $str0 .= HT::success(
00394               " &rarr; Found $nZips student submissions in<br>
00395                <code>$submissionFolder</code>",
00396               true
00397             );
00398           } else {
00399             $str0 .= HT::warn(
00400               " &rarr; No zip files in the student submission folder
00401                <code>$submissionFolder</code>",
00402               true
00403             );
00404             $error = true;
00405           }
00406         } else {
00407           $str0 .= HT::error(
00408             " &rarr; Student Submission folder not found:<br>
00409              <code>$submissionFolder</code> <br>
00410             Please download the student submissions from $sectionName
00411             and copy them there.",
00412             true
00413           );
00414           $error = true;
00415         }
00416       }
00417       foreach (self::$sectionNames as $sectionName) {
00418         $answerFolder = self::getanswerFolder($sectionName);
00419         if (is_dir($answerFolder)) {
00420           $answerFolder = realpath($answerFolder);
00421         } else {
00422           $str0 .= HT::error(
00423             " &rarr; Student answer folder not found:<br>
00424              <code>$answerFolder</code> <br>
00425             Please create it so that their zip files can be unzipped.",
00426             true
00427           );
00428           $error = true;
00429         }
00430       }
00431       if ($error) {
00432         $str .= HT::warn($str0, true);
00433       } else {
00434         $str .= HT::info($str0, true);
00435       }
00436       $rubricFile = self::$rubricFile;
00437       if (!is_file($rubricFile)) {
00438         $str .= HT::error(
00439           "Rubric file not found: <br>
00440            <code>$rubricFile</code><br>
00441           Create it before proceeding.",
00442           true
00443         );
00444         $error = true;
00445       } else {
00446         $rubricFile = realpath($rubricFile);
00447         $str .= HT::success(
00448           "Rubric File located:<br>
00449            <code>$rubricFile</code><br>",
00450           true
00451         );
00452       }
00453       $emailFolder = self::$emailFolder;
00454       if (empty($emailFolder) || !is_dir($emailFolder)) {
00455         $str .= HT::warn(
00456           "Email Submission folder not found. <br>
00457            <code>$emailFolder</code><br>
00458           Create it and copy the emailed zip files there.",
00459           true
00460         );
00461       } else {
00462         $emailFolder = realpath(self::$emailFolder);
00463         $nZips = count(glob($emailFolder . "/*.zip"));
00464         $str .= HT::success(
00465           "Email Submission folder located with $nZips files:<br>
00466            <code>$emailFolder</code>",
00467           true
00468         );
00469       }
00470     }
00471     $solutionFolder = self::$solutionFolder;
00472     if (!is_dir($solutionFolder)) {
00473       $str .= HT::error(
00474         "Solution folder not found: <br>
00475          <code>$solutionFolder</code><br>
```

```
00476         Create it and copy the reference solutions there.",
00477         true
00478       );
00479       $error = true;
00480     } else {
00481       $solutionFolder = realpath($solutionFolder);
00482       $gPattern = "$solutionFolder/{*,*/*,*/*/*}.*";
00483       $nFiles = count(glob($gPattern, GLOB_BRACE));
00484       if ($nFiles <= 0) {
00485         $str .= HT::warn(
00486           "Solution folder is empty: <br>
00487          <code>$solutionFolder</code><br>
00488         Copy the reference solutions there.",
00489           true
00490         );
00491       } else {
00492         $str .= HT::success(
00493           "Solution folder located with $nFiles files:<br>
00494          <code>$solutionFolder</code>",
00495           true
00496         );
00497       }
00498     }
00499     $resourceFolder = self::$resourceFolder;
00500     if (!is_dir($resourceFolder)) {
00501       $str .= HT::error(
00502         "Resource folder not found: <br>
00503          <code>$resourceFolder</code><br>
00504         Create it and copy the resource files there.",
00505         true
00506       );
00507       $error = true;
00508     } else {
00509       $resourceFolder = realpath($resourceFolder);
00510       $gPattern = "$resourceFolder/{*,*/*,*/*/*}.*";
00511       $nFiles = count(glob($gPattern, GLOB_BRACE));
00512       if ($nFiles <= 0) {
00513         $str .= HT::warn(
00514           "Resource folder is empty: <br>
00515          <code>$resourceFolder</code><br>
00516         Copy the resource files there.",
00517           true
00518         );
00519       } else {
00520         $str .= HT::success(
00521           "Resource folder located with $nFiles files:<br>
00522          <code>$resourceFolder</code>",
00523           true
00524         );
00525       }
00526     }
00527     $gradeQuestions = self::$gradeQuestions;
00528     if (empty($gradeQuestions)) {
00529       $str .= HT::error(
00530         "No questions are marked gradable (<code>\$gradeQuestions</code>)! <br>
00531         Edit <code>$configFile</code> to set it as an array.",
00532         true
00533       );
00534       $error = true;
00535     } else {
00536       $str .= HT::success(
00537         "Will assist in grading questions <strong>"
00538           . implode(", ", $gradeQuestions) . "</strong>",
00539         true
00540       );
00541     }
00542     $autoGradables = self::$autoGradables;
00543     if (empty($autoGradables)) {
00544       $str .= HT::warn(
00545         "No questions are marked for autograding (<code>\$autoGradables</code>)!<br>
00546         Edit <code>$configFile</code> to set it if needed.",
00547         true
00548       );
00549     } else {
00550       foreach ($autoGradables as $file => $autoGraded) {
00551         $autoGraderBase = $autoGraded['grader'];
00552         $autoGrader = "$solutionFolder/$autoGraderBase";
00553         $subQuestionName = $autoGraded['subQuestionName'];
00554         if (file_exists($autoGrader)) {
00555           $str .= HT::success(
00556             "Autograder <code>$autoGraderBase</code> will grade <code>$subQuestionName
    [$file]</code>",
00557             true
00558           );
00559         } else {
00560           $str .= HT::error(
00561             "Autograder <code>$autoGraderBase</code> for <code>$subQuestionName [$file]</code> not
```

```
      found.<br>
00562             Copy it to <code>$solutionFolder</code>",
00563             true
00564           );
00565           $error = true;
00566         }
00567         $autoHelpers = $autoGraded['helpers'] ?? [];
00568         foreach ($autoHelpers as $helper) {
00569           $helperFull = "$solutionFolder/$helper";
00570           if (file_exists($helperFull)) {
00571             $str .= HT::success(
00572               "Helper file <code>$helper</code> for <code>$subQuestionName [$file]</code> located",
00573               true
00574             );
00575           } else {
00576             $str .= HT::error(
00577               "Helper file <code>$helper</code> for <code>$subQuestionName [$file]</code> not
      found.<br>
00578               Copy it to <code>$solutionFolder</code>",
00579               true
00580             );
00581             $error = true;
00582           }
00583         }
00584       }
00585     }
00586     end:
00587     if ($error) {
00588       $configFile = self::$configFile;
00589       $str .= HT::error(
00590         "Fix the errors (red boxes) and validate again.<br>
00591          The comments in <a href='?do=view'><code>$configFile</code></a> may help.",
00592         true
00593       );
00594     }
00595     HT::warn($str);
00596     HT::foot();
00597   }
```

References $autoGradables, $configFile, $emailFolder, $gradeQuestions, $resourceFolder, $root, $rubricFile, $solutionFolder, HT::error(), HT::foot(), HT::head(), HT::info(), HT::success(), and HT::warn().

Here is the call graph for this function:



view()

```
static CFG::view ()  [static]
```

Displays configuration file and included sub-config if available.

Returns

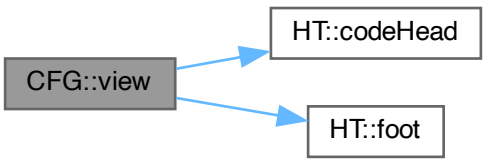
void

Definition at line 277 of file CFG.php.

```
00278   {
00279     $configFile = self::$configFile;
00280     $ext = pathinfo($configFile, PATHINFO_EXTENSION);
00281     $configFileContents = file_get_contents($configFile);
00282     try {
00283       $includedConfigFile = explode("require_once ", $configFileContents)[1];
00284       $includedConfigFile = explode('"', $includedConfigFile)[1];
00285       $includedConfigFile = str_replace('$root', self::$root, $includedConfigFile);
00286       $includedConfigFileContent = file_get_contents($includedConfigFile);
00287     } catch (Exception $e) {
00288       echo $e->getMessage();
00289     }
00290     $fullText0 = htmlentities($configFileContents);
00291     $fullText1 = htmlentities($includedConfigFileContent);
00292     $realPath0 = "<div
00293       style='color: #373;
00294       background: #ecffd6;
00295       border: 1px solid #617c42;
00296       padding:4px;
00297       font-weight:bold;
00298       text-align:left;'><code>" .
00299       realpath($configFile) . "</code></div>";
00300     $realPath1 = "<div
00301       style='color: #373;
00302       background: #ecffd6;
00303       border: 1px solid #617c42;
00304       padding:4px;
00305       font-weight:bold;
00306       text-align:left;'><code>" .
00307       realpath($includedConfigFile) . "</code></div>";
00308     HT::codeHead($configFile);
00309     echo "$realPath0<pre><code class='language-$ext'>$fullText0</code></pre>";
00310     echo "$realPath1<pre><code class='language-$ext'>$fullText1</code></pre>";
00311     HT::foot();
00312   }
```

References $configFile, HT::codeHead(), and HT::foot().

Here is the call graph for this function:



viewSql()

```
static CFG::viewSql ()  [static]
```

Displays the generated SQL file.

Returns

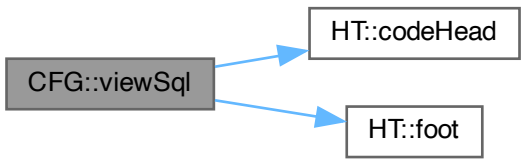
void

Definition at line 604 of file CFG.php.

```
00605   {
00606     $fileName = self::$root . DIRECTORY_SEPARATOR . self::$sqlFile;
00607     self::mkSql();
00608     $ext = pathinfo($fileName, PATHINFO_EXTENSION);
00609     $fullText = htmlentities(file_get_contents($fileName), ENT_QUOTES);
00610     HT::codeHead($fileName);
00611     echo "<pre><code class='language-$ext'>$fullText</code></pre>";
00612     HT::foot();
00613   }
```

References HT::codeHead(), and HT::foot().

Here is the call graph for this function:



4.3.3 Member Data Documentation

$actionDesc

```
CFG::$actionDesc  [static], [protected]
```

Initial value:

```
= [
    'view' => 'View (not edit) the current settings in your <code>config.php</code> file.',
    'validate' => 'Validate your configuration in <code>config.php</code> file and check for the existence
      an contents of the folders specified.',
    'viewSql' => 'Examine the SQL file that will be run to set up your database. Do not edit this file. It
      is generated based on your <code>config.php</code> and a template',
    'runSql' => 'Run the generated SQL file to set up the database. If you see a red box about errors in the
      database, try this button. Clicking the button again is harmless: It does not reset the database.',
    'dropDB' => 'Drop all database tables and restart: This button will really reset the database. If you
      have graded some students, all the entered grades will really disappear.'
  ]
```

Definition at line 143 of file CFG.php.

$actionProperties

```
CFG::$actionProperties  [static], [protected]
```

Initial value:

```
= [
    'view' => [
      'target' => 'left',
      'clear' => 'yes'
    ],
    'validate' => [
      'target' => 'right'
    ],
    'viewSql' => [
      'target' => 'left',
      'clear' => 'yes'
    ],
    'runSql' => [
      'target' => 'right'
    ],
    'dropDB' => [
      'class' => 'error',
      'target' => 'right'
    ]
  ]
```

Definition at line 123 of file CFG.php.

$actions

```
CFG::$actions  [static], [protected]
```

Initial value:

```
= [
    'view' => 'View Configuration',
    'validate' => 'Validate Config and Folders',
    'viewSql' => 'View SQL File',
    'runSql' => 'Setup Database',
    'dropDB' => 'Drop Database',
  ]
```

Definition at line 116 of file CFG.php.

$answerFolderPattern

```
CFG::$answerFolderPattern  [static], [private]
```

Definition at line 49 of file CFG.php.

$autoGradables

```
CFG::$autoGradables  [static]
```

Definition at line 96 of file CFG.php.

Referenced by Student::autoGrade(), AnswerDAO::cpAutoGraders(), Section::export(), Answer::render(), and validate().

$class

```
CFG::$class = __CLASS__  [static], [protected]
```

Definition at line 113 of file CFG.php.

$configFile

```
CFG::$configFile = "../config.php"  [static]
```

Definition at line 81 of file CFG.php.

Referenced by validate(), and view().

$csvFileNamePattern

```
CFG::$csvFileNamePattern  [static], [private]
```

Definition at line 31 of file CFG.php.

$dbName

```
CFG::$dbName  [static]
```

Definition at line 105 of file CFG.php.

Referenced by dropDB(), HT::index(), mkDropDB(), and mkSql().

$dbPass

```
CFG::$dbPass  [static]
```

Definition at line 102 of file CFG.php.

Referenced by HT::index().

$dbPrefix

```
CFG::$dbPrefix  [static]
```

Definition at line 108 of file CFG.php.

Referenced by getDbPrefix(), HT::index(), Stat::mistakes(), mkDropDB(), mkSql(), and Stat::mkSql().

$dbUser

```
CFG::$dbUser  [static]
```

Definition at line 99 of file CFG.php.

Referenced by HT::index().

$display

```
CFG::$display  [static], [protected]
```

Initial value:

```
= [
    "class" => "plain",
    "title" => "Setup and Configuration"
  ]
```

Definition at line 156 of file CFG.php.

$emailFolder

```
CFG::$emailFolder  [static]
```

Definition at line 46 of file CFG.php.

Referenced by AnswerDAO::getStudentZip(), and validate().

$examLongName

```
CFG::$examLongName  [static]
```

Definition at line 22 of file CFG.php.

Referenced by Grader::quickSummary(), and HT::top().

$examShortName

```
CFG::$examShortName  [static]
```

Definition at line 19 of file CFG.php.

Referenced by Exam::__construct().

$gradeQuestions

```
CFG::$gradeQuestions  [static]
```

Definition at line 93 of file CFG.php.

Referenced by QuestionDAO::get(), Exam::mkFinals(), Exam::mkLabTest(), Answer::render(), and validate().

$intro

```
CFG::$intro  [static], [protected]
```

Initial value:

```
= "<h4>Configuration Menu</h4>
  This subPage allows you to view and validate your configuration file.
  Validation means checking for the existence of the necessary folders and files.
  This is also the place where you will set up the database tables. If necessary,
  you can reset your DB also here. See the descriptions of what the buttons do
  on the right."
```

Definition at line 150 of file CFG.php.

$locMarks

```
CFG::$locMarks  [static]
```

Definition at line 37 of file CFG.php.

Referenced by Exam::mkLabTest(), and QuestionDAO::process().

$locSubQ

```
CFG::$locSubQ  [static]
```

Definition at line 34 of file CFG.php.

Referenced by Exam::mkLabTest(), and QuestionDAO::process().

$mode

```
CFG::$mode = "LT"  [static]
```

Definition at line 25 of file CFG.php.

$numQuestions

```
CFG::$numQuestions  [static]
```

Definition at line 111 of file CFG.php.

Referenced by setNumQuestions().

$resourceFolder

```
CFG::$resourceFolder  [static]
```

Definition at line 55 of file CFG.php.

Referenced by RefFile::__construct(), RefFileDAO::process(), and validate().

$root

```
CFG::$root  [static], [private]
```

Definition at line 90 of file CFG.php.

Referenced by mkFileName(), and validate().

$rubricFile

```
CFG::$rubricFile = "./rubric.csv"  [static]
```

Definition at line 40 of file CFG.php.

Referenced by RubricDAO::process(), and validate().

$sectionNames

```
CFG::$sectionNames  [static]
```

Definition at line 28 of file CFG.php.

Referenced by Exam::mkLabTest(), and QuestionDAO::process().

$solutionFolder

```
CFG::$solutionFolder  [static]
```

Definition at line 52 of file CFG.php.

Referenced by RefFile::__construct(), AnswerDAO::cpAutoGraders(), RefFileDAO::process(), and validate().

$sqlFile

```
CFG::$sqlFile = 'setup.sql'  [static], [private]
```

Definition at line 87 of file CFG.php.

Referenced by mkSql(), and runSql().

$sqlTemplate

```
CFG::$sqlTemplate = "../model/setup.template.sql"  [static], [private]
```

Definition at line 84 of file CFG.php.

$submissionFolderPattern

```
CFG::$submissionFolderPattern  [static], [private]
```

Definition at line 43 of file CFG.php.

$vars

```
CFG::$vars  [static], [private]
```

Initial value:

```
= [
    "examShortName",
    "examLongName",
    "mode",
    "sectionNames",
    "csvFileNamePattern",
    "locSubQ",
    "locMarks",
    "rubricFile",
    "submissionFolderPattern",
    "answerFolderPattern",
    "emailFolder",
    "solutionFolder",
    "resourceFolder",
    "gradeQuestions",
    "autoGradables",
    "dbUser",
    "dbPass",
    "dbName"
  ]
```

Definition at line 59 of file CFG.php.

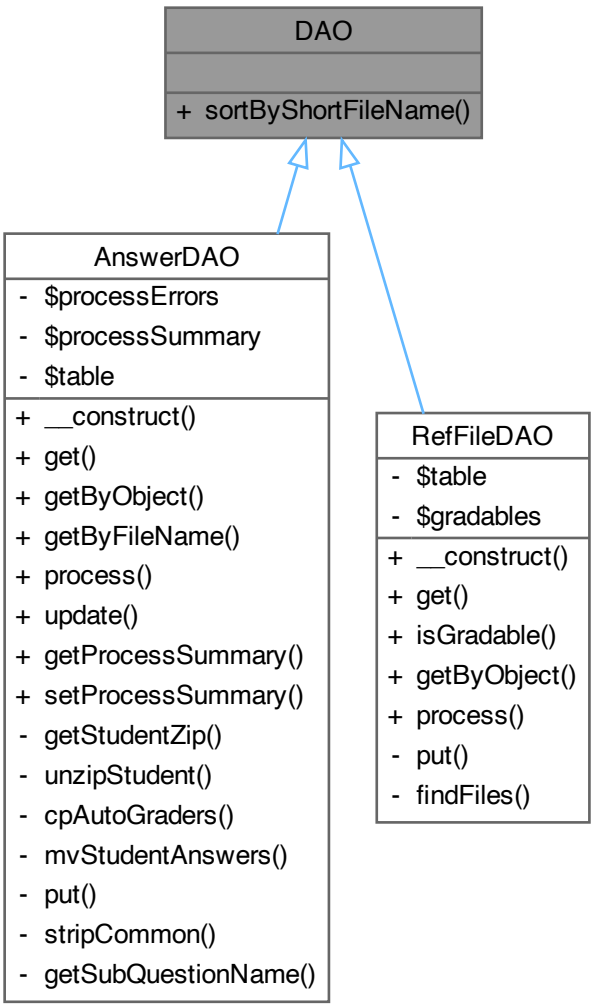
Referenced by init().

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

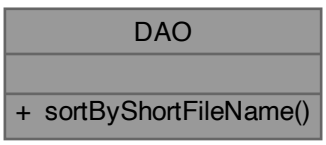
- CFG.php

4.4 DAO Class Reference

Inheritance diagram for DAO:



Collaboration diagram for DAO:



Static Public Member Functions

- static sortByShortFileName ($objects)

4.4.1 Detailed Description

Class DAO

Base class for future refactoring of all Data Access Objects (DAOs) by inheritance. Provides common utility methods shared across DAOs.

Definition at line 9 of file DAO.php.

4.4.2 Member Function Documentation

sortByShortFileName()

```
static DAO::sortByShortFileName (
              $objects)  [static]
```

Sorts an array of objects by their short file name in ascending order.

Parameters

object[ ]	*$objects*	Array of objects that implement getShortFileName().

Returns

object[] Sorted array of objects indexed by their short file name.

Definition at line 17 of file DAO.php.

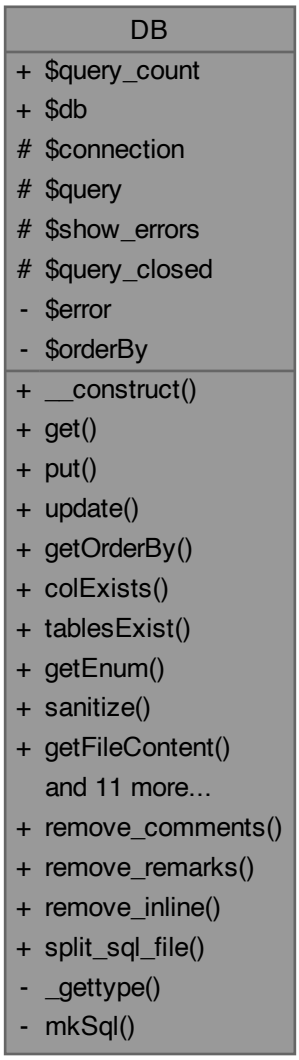
```
00018     {
00019         $toSort = [];
00020         foreach ($objects as $o) {
00021             $toSort[$o->getShortFileName()] = $o;
00022         }
00023         ksort($toSort);
00024         return $toSort;
00025     }
```

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

- DAO.php

4.5 DB Class Reference

Collaboration diagram for DB:



Public Member Functions

- __construct ($dbHost='localhost', $dbUser='root', $dbPass='', $dbName='', $charset='utf8')
- get ($table, $what='', $where=[ ])
- put ($table, $columns, $rows)
- update ($table, $columns, $rows)
- getOrderBy ($table, $columns)
- colExists ($table, $column)
- tablesExist ($dbPrefix)
- getEnum ($table, $field)

- sanitize ($str)
- getFileContent ($file)
- query ($query)
- fetchAll ($callback=null)
- fetchArray ()
- close ()
- numRows ()
- affectedRows ()
- lastInsertID ()
- error ($error)
- getError ()
- multiQuery ($mSql)
- importSQL ($sqlFile, $gzip=false)

Static Public Member Functions

- static remove_comments (&$output)
- static remove_remarks ($sql)
- static remove_inline ($sql)
- static split_sql_file ($sql, $delimiter=";")

Public Attributes

- $query_count = 0

Static Public Attributes

- static $db

Protected Attributes

- $connection
- $query
- $show_errors = true
- $query_closed = true

Private Member Functions

- _gettype ($var)

Static Private Member Functions

- static mkSql ($table, $columns, $rows, $insert="REPLACE")

Private Attributes

- $error = ""

Static Private Attributes

- static $orderBy = ["section", "student_email"]

4.5.1 Detailed Description

Class DB

Database abstraction and utility class for handling MySQL connections and queries.

Features:

- Safe and persistent database connection.
- Query preparation and execution.
- CRUD (Create, Read, Update, Delete) operations.
- Bulk SQL imports and execution.
- Utility methods for column existence, enum retrieval, and SQL comment removal.

This class is adapted from a public domain database class with added functionality for grading systems.

Ref https://codeshack.io/super-fast-php-mysql-database-class/

Modified:

1. Automate DB setup in the constructor using try-catch
2. Add some utility methods

Definition at line 23 of file DB.php.

4.5.2 Member Function Documentation

__construct()

```
DB::__construct (
              $dbHost = 'localhost',
              $dbUser = 'root',
              $dbPass = '',
              $dbName = '',
              $charset = 'utf8')
```

DB constructor. Initializes and connects to a MySQL database.

Parameters

string	*$dbHost*	Database host.
string	*$dbUser*	Database username.
string	*$dbPass*	Database password.
string	*$dbName*	Database name.

string	*$charset*	Character set for connection.

Definition at line 61 of file DB.php.

```
00067     {
00068     try {
00069       mysqli_report(MYSQLI_REPORT_ERROR | MYSQLI_REPORT_STRICT);
00070       $configFile = realpath("config.php");
00071       $setupMsg = "Edit the config file <code>$configFile</code>
00072       and go to <strong>Setup and Configuration</strong> and set it up.";
00073       $this->connection = new mysqli($dbHost, $dbUser, $dbPass, $dbName);
00074       if ($this->connection->connect_error) { // For WAMP
00075         $error = $this->connection->connect_error;
00076         $this->error = "WAMP: Database connection error: $error.
00077         $setupMsg";
00078       } else {
00079         $this->connection->set_charset($charset);
00080         self::$db = $this;
00081       }
00082     } catch (Exception $e) { // for MAMP
00083       $error = $e->getMessage();
00084       $configFile = realpath("config.php");
00085       $this->error = "MAMP: Database connection error: $error.
00086       $setupMsg";
00087     }
00088   }
```

References $error, and error().

Here is the call graph for this function:

_gettype()

```
DB::_gettype (
              $var)  [private]
```

Returns the type of variable for binding parameters.

Parameters

mixed	*$var*	

Returns

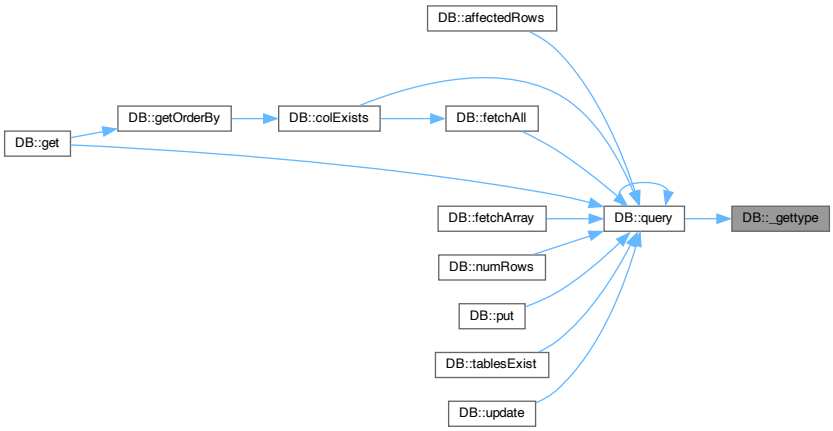
string

Definition at line 671 of file DB.php.

```
00672   {
00673     if (is_string($var))
00674       return 's';
00675     if (is_float($var))
00676       return 'd';
00677     if (is_int($var))
00678       return 'i';
00679     return 'b';
00680   }
```

Referenced by query().

Here is the caller graph for this function:



affectedRows()

```
DB::affectedRows ()
```

Gets number of affected rows from last operation.

Returns

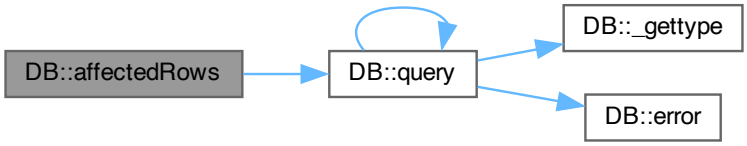
int

Definition at line 397 of file DB.php.

```
00398   {
00399     return $this->query->affected_rows;
00400   }
```

References query().

Here is the call graph for this function:



close()

```
DB::close ()
```

Closes the database connection.

Returns

bool

Definition at line 376 of file DB.php.

```
00377   {
00378     return $this->connection->close();
00379   }
```

colExists()

```
DB::colExists (
              $table,
              $column)
```

Checks if a column exists in a table.

Parameters

string	*$table*	
string	*$column*	

Returns

bool

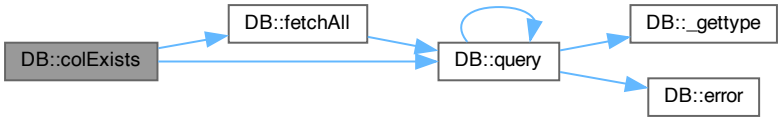
Definition at line 204 of file DB.php.

```
00205   {
00206     $sql = "SELECT *
00207       FROM `information_schema`.COLUMNS
00208       WHERE TABLE_NAME = '$table'
00209       AND COLUMN_NAME = '$column'";
00210     return !empty($this->query($sql)->fetchAll());
00211   }
```

References $sql, fetchAll(), and query().

Referenced by getOrderBy().

Here is the call graph for this function:



Here is the caller graph for this function:

error()

```
DB::error (
              $error)
```

Outputs or stores an error message.

Parameters

string	*$error*	

Returns

void

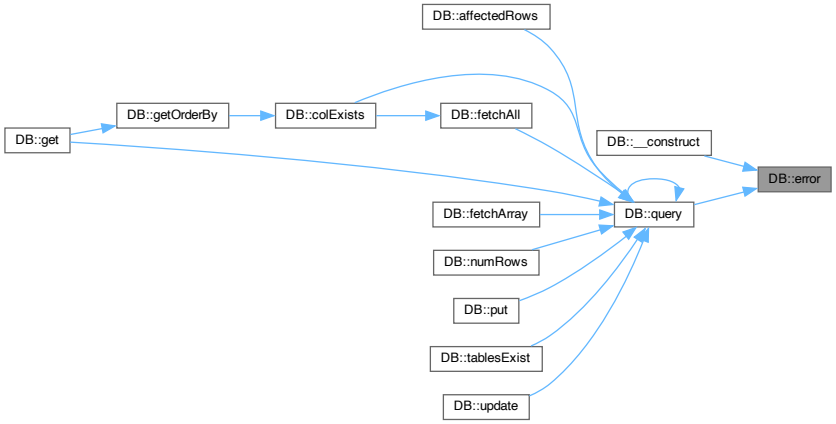
Definition at line 418 of file DB.php.

```
00419   {
00420     if ($this->show_errors) {
00421       exit($error);
00422     }
00423   }
```

References $error.

Referenced by __construct(), and query().

Here is the caller graph for this function:



fetchAll()

```
DB::fetchAll (
              $callback = null)
```

Fetches all rows from the last query result.

Parameters

callable | null	*$callback*	

Returns

array

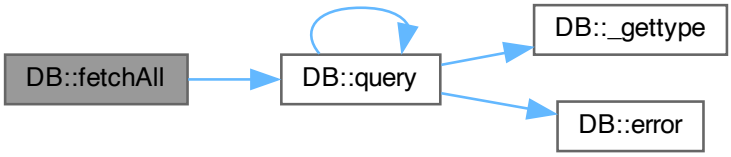
Definition at line 318 of file DB.php.

```
00319   {
00320     $params = array();
00321     $row = array();
00322     $meta = $this->query->result_metadata();
00323     while ($field = $meta->fetch_field()) {
00324       $params[] = &$row[$field->name];
00325     }
00326     call_user_func_array(array($this->query, 'bind_result'), $params);
00327     $result = array();
00328     while ($this->query->fetch()) {
00329       $r = array();
00330       foreach ($row as $key => $val) {
00331         $r[$key] = $val;
00332       }
00333       if ($callback != null && is_callable($callback)) {
00334         $value = call_user_func($callback, $r);
00335         if ($value == 'break')
00336           break;
00337       } else {
00338         $result[] = $r;
00339       }
00340     }
00341     $this->query->close();
00342     $this->query_closed = TRUE;
00343     return $result;
00344   }
```

References query().

Referenced by colExists().

Here is the call graph for this function:



Here is the caller graph for this function:

fetchArray()

```
DB::fetchArray ()
```

Fetches a single row as an associative array.

Returns

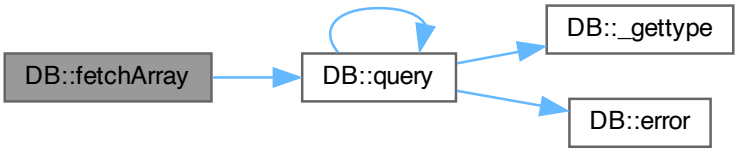
array

Definition at line 351 of file DB.php.

```
00352   {
00353     $params = array();
00354     $row = array();
00355     $meta = $this->query->result_metadata();
00356     while ($field = $meta->fetch_field()) {
00357       $params[] = &$row[$field->name];
00358     }
00359     call_user_func_array(array($this->query, 'bind_result'), $params);
00360     $result = array();
00361     while ($this->query->fetch()) {
00362       foreach ($row as $key => $val) {
00363         $result[$key] = $val;
00364       }
00365     }
00366     $this->query->close();
00367     $this->query_closed = TRUE;
00368     return $result;
00369   }
```

References query().

Here is the call graph for this function:



get()

```
DB::get (
              $table,
              $what = '',
              $where = [])
```

Retrieves records from a table based on conditions.

Parameters

string	*$table*	Full table name (with prefix).
string | array	*$what*	Array of columns to get. Empty => *.
array	*$where*	Associative array: column => value. Empty => no WHERE clause.

Returns

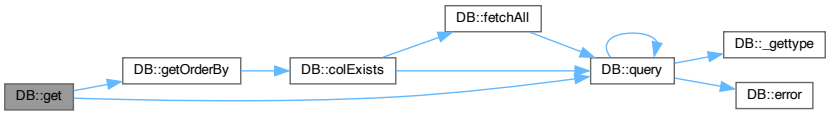
array Collated output

Definition at line 100 of file DB.php.

```
00101   {
00102     if (is_array($what)) {
00103       if (in_array("*", $what)) {
00104         $what = [];
00105       }
00106       $columns = $what;
00107     } else {
00108       if (empty($what) || $what == "*") {
00109         $columns = [];
00110       } else {
00111         $columns = [$what];
00112       }
00113     }
00114     if (empty($columns)) {
00115       $what = "*";
00116     } else {
00117       $what = "`" . implode("`, `", $columns) . "`";
00118     }
00119     $keys = $where;
00120     if (empty($keys)) {
00121       $where = "";
00122     } else {
00123       $where = [];
00124       foreach ($keys as $key => $val) {
00125         $where[] = "`$key` = '$val'";
00126       }
00127       $where = "WHERE " . implode(" AND ", $where);
00128     }
00129     $sql = "SELECT $what FROM $table $where";
00130     $orderBy = $this->getOrderBy($table, self::$orderBy);
00131     $sql .= "$orderBy";
00132     $rows = $this->query($sql)->fetchAll();
00133     return $rows;
00134   }
```

References $orderBy, $sql, getOrderBy(), and query().

Here is the call graph for this function:



getEnum()

```
DB::getEnum (
              $table,
              $field)
```

Gets ENUM field values from a table.

Parameters

string	*$table*	
string	*$field*	

Returns

array

Definition at line 234 of file DB.php.

```
00235   {
00236     $sql = "SHOW COLUMNS FROM {$table} WHERE Field = '{$field}'";
00237     $result = self::$db->query($sql);
00238     $row = $result->fetchAll();
00239     $type = $row[0]['Type'];
00240     preg_match('/enum\((.*)\)$/', $type, $matches);
00241     $vals = explode(',', $matches[1]);
00242     return $vals;
00243   }
```

References $sql.

getError()

```
DB::getError ()
```

Returns last error message.

Returns

string

Definition at line 430 of file DB.php.

```
00431   {
00432     return $this->error;
00433   }
```

References $error.

getFileContent()

```
DB::getFileContent (
                $file)
```

Retrieves and sanitizes file content.

Parameters

string	*$file*	

Returns

string

Definition at line 262 of file DB.php.

```
00263   {
00264     return $this->sanitize(file_get_contents($file));
00265   }
```

References sanitize().

Here is the call graph for this function:

getOrderBy()

```
DB::getOrderBy (
              $table,
              $columns)
```

Returns ORDER BY clause if columns exist.

Parameters

string	*$table*	
array	*$columns*	

Returns

string

Definition at line 177 of file DB.php.

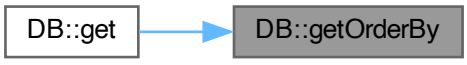
```
00178   {
00179     if (!is_array($columns)) {
00180       $columns = [$columns];
00181     }
00182     foreach ($columns as $key => $column) {
00183       if (!$this->colExists($table, $column)) {
00184         unset($columns[$key]);
00185       }
00186     }
00187     if (count($columns) > 0) {
00188       $orderBy = "ORDER BY `" . implode("`, `", $columns) . "`";
00189     } else {
00190       $orderBy = "";
00191     }
00192     return $orderBy;
00193   }
```

References $orderBy, and colExists().

Referenced by get().

Here is the call graph for this function:

Here is the caller graph for this function:



importSQL()

```
DB::importSQL (
              $sqlFile,
              $gzip = false)
```

Imports SQL from file, optionally gzipped.

Parameters

string	*$sqlFile*	
bool	*$gzip*	

Returns

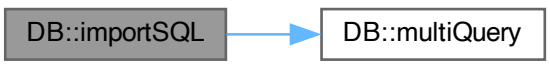
array|false

Definition at line 622 of file DB.php.

```
00623   {
00624     if (!is_readable($sqlFile)) {
00625       return false;
00626     }
00627     $sql = file_get_contents($sqlFile);
00628     if ($gzip) {
00629       $sql = gzdecode($sql);
00630     }
00631     return $this->multiQuery($sql);
00632   }
```

References $sql, and multiQuery().

Here is the call graph for this function:



lastInsertID()

```
DB::lastInsertID ()
```

Gets last insert ID from auto-increment.

Returns

int

Definition at line 407 of file DB.php.

```
00408   {
00409     return $this->connection->insert_id;
00410   }
```

mkSql()

```
static DB::mkSql (
              $table,
              $columns,
              $rows,
              $insert = "REPLACE")  [static], [private]
```

Internal utility to create SQL statement.

Parameters

string	*$table*	
array	*$columns*	
array	*$rows*	
string	*$insert*	

Returns

string

Definition at line 646 of file DB.php.

```
00647   {
00648     $values = [];
00649     foreach ($rows as $key => $row) {
00650       if (is_array($row)) { // Multiple rows
00651         $values[$key] = "('" . implode("', '", $row) . "')";
00652       } else { // Single row
00653         $values = "('" . implode("', '", $rows) . "')";
00654       }
00655     }
00656     if (is_array($values)) { // Multiple rows
00657       $values = implode(",\n ", $values);
00658     }
00659     $sql = "$insert INTO $table
00660         ( `" . implode("`, `", $columns) . "`)
00661         VALUES $values";
00662     return $sql;
00663   }
```

References $sql.

multiQuery()

```
DB::multiQuery (
              $mSql)
```

Executes multiple SQL queries from a string.

Parameters

string	*$mSql*	

Returns

array ['success' => int, 'errors' => int, 'message' => string]

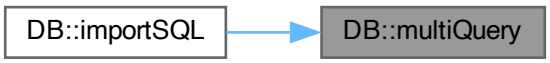
Definition at line 595 of file DB.php.

```
00596   {
00597     $mSql = self::remove_comments($mSql);
00598     $mSql = self::remove_remarks($mSql);
00599     $lines = self::split_sql_file($mSql);
00600     $errors = $success = 0;
00601     $message = "";
00602     foreach ($lines as $sql) {
00603       try {
00604         $this->connection->query($sql, MYSQLI_STORE_RESULT);
00605         ++$success;
00606       } catch (Exception $e) {
00607         ++$errors;
00608         $message .= "\n$errors: " . $e->getMessage();
00609       }
00610     }
00611     $message = trim($message);
00612     return compact('success', 'errors', 'message');
00613   }
```

References $sql.

Referenced by importSQL().

Here is the caller graph for this function:



numRows()

`DB::numRows ()`

Gets number of rows from last query.

Returns

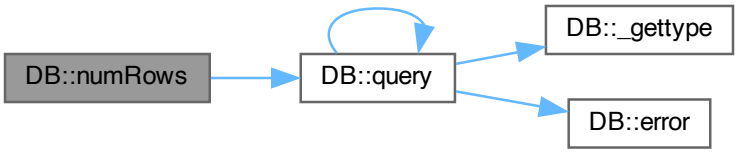
int

Definition at line 386 of file DB.php.

```
00387   {
00388     $this->query->store_result();
00389     return $this->query->num_rows;
00390   }
```

References query().

Here is the call graph for this function:



put()

```
DB::put (
                $table,
                $columns,
                $rows)
```

Inserts or replaces rows in a table.

Parameters

string	*$table*	
array	*$columns*	
array	*$rows*	

Returns

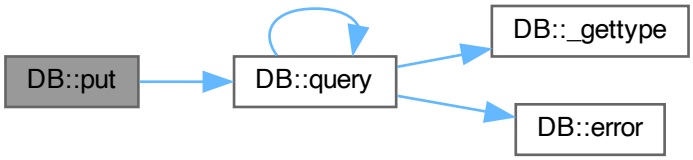
void

Definition at line 144 of file DB.php.

```
00145   {
00146     $sql = self::mkSql($table, $columns, $rows, "REPLACE");
00147     $this->query($sql);
00148   }
```

References $sql, and query().

Here is the call graph for this function:



query()

```
DB::query (
              $query)
```

Executes a raw SQL query.

Parameters

string	*$query*	

Returns

$this

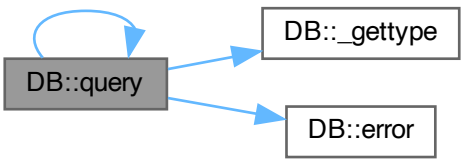
Definition at line 275 of file DB.php.

```
00276   {
00277     if (!$this->query_closed) {
00278       $this->query->close();
00279     }
00280     if ($this->query = $this->connection->prepare($query)) {
00281       if (func_num_args() > 1) {
00282         $x = func_get_args();
00283         $args = array_slice($x, 1);
00284         $types = '';
00285         $args_ref = array();
00286         foreach ($args as $k => &$arg) {
00287           if (is_array($args[$k])) {
00288             foreach ($args[$k] as $j => &$a) {
00289               $types .= $this->_gettype($args[$k][$j]);
00290               $args_ref[] = &$a;
00291             }
00292           } else {
00293             $types .= $this->_gettype($args[$k]);
00294             $args_ref[] = &$arg;
00295           }
00296         }
00297         array_unshift($args_ref, $types);
00298         call_user_func_array(array($this->query, 'bind_param'), $args_ref);
00299       }
00300       $this->query->execute();
00301       if ($this->query->errno) {
00302         $this->error('Unable to process MySQL query (check your params) - ' . $this->query->error);
00303       }
00304       $this->query_closed = FALSE;
00305       $this->query_count++;
00306     } else {
00307       $this->error('Unable to prepare MySQL statement (check your syntax) - ' .
      $this->connection->error);
00308     }
00309     return $this;
00310   }
```

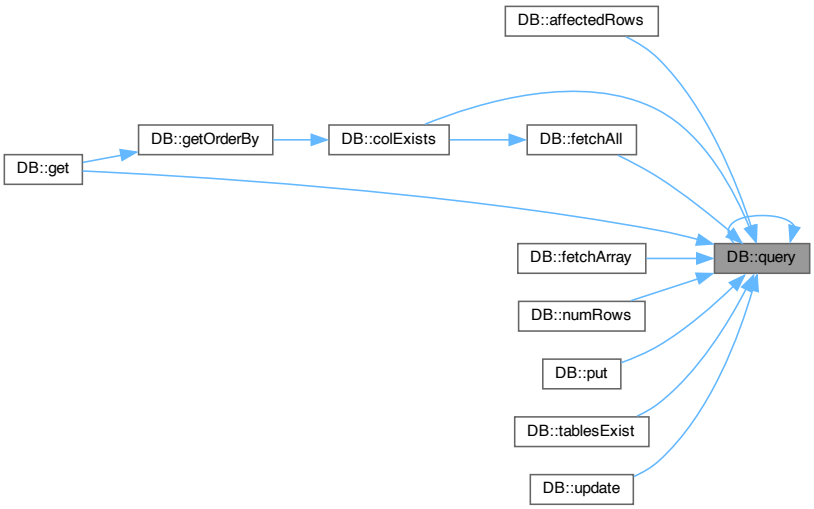
References $query, _gettype(), error(), and query().

Referenced by affectedRows(), colExists(), fetchAll(), fetchArray(), get(), numRows(), put(), query(), tablesExist(), and update().

Here is the call graph for this function:



Here is the caller graph for this function:



remove_comments()

```
static DB::remove_comments (
            & $output)  [static]
```

Removes comments from SQL file. To import an sql: From my own DbHelper.php, which took it from phpBB

Parameters

string	*$sql*	

Returns

string

Definition at line 444 of file DB.php.

```
00445   {
00446     $lines = explode("\n", $output);
00447     $output = "";
00448
00449     $linecount = count($lines);
00450
00451     $in_comment = false;
00452     for ($i = 0; $i < $linecount; $i++) {
00453       if (preg_match("/^\/\*/", preg_quote($lines[$i]))) {
00454         $in_comment = true;
00455       }
00456
00457       if (!$in_comment) {
00458         $output .= $lines[$i] . "\n";
00459       }
00460
00461       if (preg_match("/\*\/$/", preg_quote($lines[$i]))) {
00462         $in_comment = false;
00463       }
00464     }
00465
00466     unset($lines);
00467     return $output;
00468   }
```

remove_inline()

```
static DB::remove_inline (
              $sql)  [static]
```

Removes inline SQL comments (/ ∗ ∗ /).

Parameters

string	*$sql*	

Returns

string

Definition at line 503 of file DB.php.

```
00504   {
00505     $regex = array('/\/\*.*\*\//U');
00506     $sql = preg_replace($regex, "", $sql);
00507     $sql = trim($sql);
00508     return $sql;
00509   }
```

References $sql.

remove_remarks()

```
static DB::remove_remarks (
              $sql)  [static]
```

Removes SQL remark lines (starting with #). Strips the sql comment lines out of an uploaded sql file. From phpBB. Modified.

Parameters

string	*$sql*	

Returns

string

Definition at line 478 of file DB.php.

```
00479   {
00480     $lines = explode("\n", $sql);
00481     $linecount = count($lines);
00482     $output = "";
00483
00484     for ($i = 0; $i < $linecount; $i++) {
00485       if (($i != ($linecount - 1)) || (strlen($lines[$i]) > 0)) {
00486         if (isset($lines[$i][0]) && $lines[$i][0] != "#") {
00487           $output .= $lines[$i] . "\n";
00488         } else {
00489           $output .= "\n";
00490         }
00491       }
00492     }
00493
00494     return $output;
00495   }
```

References $sql.

sanitize()

```
DB::sanitize (
              $str)
```

Sanitizes a string for safe SQL usage.

Parameters

string	*$str*	

Returns

string

Definition at line 251 of file DB.php.

```
00252   {
00253     return $this->connection->real_escape_string($str);
00254   }
```

Referenced by getFileContent().

Here is the caller graph for this function:

split_sql_file()

```
static DB::split_sql_file (
              $sql,
              $delimiter = ";")  [static]
```

Splits SQL file into individual queries.

Parameters

string	*$sql*	
string	*$delimiter*	

Returns

array

Definition at line 518 of file DB.php.

```
00519   {
00520     $sql = trim($sql);
00521     $tokens = explode($delimiter, $sql);
00522     $output = array();
00523     $matches = array();
00524
00525     // this is faster than calling count($oktens) every time thru the loop.
00526     $token_count = count($tokens);
00527     for ($i = 0; $i < $token_count; $i++) {
00528       $tokens[$i] = self::remove_inline($tokens[$i]);
```

```
00529       // Don't wanna add an empty string as the last thing in the array.
00530       if (($i != ($token_count - 1)) || (strlen($tokens[$i] > 0))) {
00531         // This is the total number of single quotes in the token.
00532         $total_quotes = preg_match_all("/'/", $tokens[$i], $matches);
00533         // Counts single quotes that are preceded by an odd number of backslashes,
00534         // which means they're escaped quotes.
00535         $escaped_quotes = preg_match_all("/(?<!\\\\)(\\\\\\\\)*\\\\'/", $tokens[$i], $matches);
00536
00537         $unescaped_quotes = $total_quotes - $escaped_quotes;
00538
00539         // If the number of unescaped quotes is even, then the delimiter did NOT occur inside a string
      literal.
00540         if (($unescaped_quotes % 2) == 0) {
00541           // It's a complete sql statement.
00542           $output[] = $tokens[$i];
00543         } else {
00544           // incomplete sql statement. keep adding tokens until we have a complete one.
00545           // $temp will hold what we have so far.
00546           $temp = $tokens[$i] . $delimiter;
00547
00548           // Do we have a complete statement yet?
00549           $complete_stmt = false;
00550
00551           for ($j = $i + 1; (!$complete_stmt && ($j < $token_count)); $j++) {
00552             // This is the total number of single quotes in the token.
00553             $total_quotes = preg_match_all("/'/", $tokens[$j], $matches);
00554             // Counts single quotes that are preceded by an odd number of backslashes,
00555             // which means they're escaped quotes.
00556             $escaped_quotes = preg_match_all("/(?<!\\\\)(\\\\\\\\)*\\\\'/", $tokens[$j], $matches);
00557
00558             $unescaped_quotes = $total_quotes - $escaped_quotes;
00559
00560             if (($unescaped_quotes % 2) == 1) {
00561               // odd number of unescaped quotes. In combination with the previous incomplete
00562               // statement(s), we now have a complete statement. (2 odds always make an even)
00563               $output[] = $temp . $tokens[$j];
00564
00565               // exit the loop.
00566               $complete_stmt = true;
00567               // make sure the outer loop continues at the right point.
00568               $i = $j;
00569             } else {
00570               // even number of unescaped quotes. We still don't have a complete statement.
00571               // (1 odd and 1 even always make an odd)
00572               $temp .= $tokens[$j] . $delimiter;
00573             }
00574           } // for..
00575         } // else
00576       }
00577     }
00578     foreach ($output as $k => $o) {
00579       $o = trim($o);
00580       if (empty($o)) {
00581         unset($output[$k]);
00582       } else {
00583         $output[$k] = $o;
00584       }
00585     }
00586     return $output;
00587   }
```

References $sql.

tablesExist()

```
DB::tablesExist (
                $dbPrefix)
```

Checks if any tables exist with a given prefix.

Parameters

string	*$dbPrefix*	

Returns

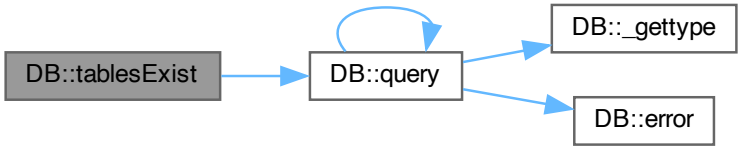
bool

Definition at line 219 of file DB.php.

```
00220   {
00221     $sql = "SHOW TABLES LIKE '$dbPrefix%'";
00222     $results = $this->query($sql);
00223     $results = $results->fetchAll();
00224     return !empty($results);
00225   }
```

References $sql, and query().

Here is the call graph for this function:



update()

```
DB::update (
                $table,
                $columns,
                $rows)
```

Inserts or updates rows based on duplicate key.

Parameters

string	*$table*	
array	*$columns*	
array	*$rows*	

Returns

void

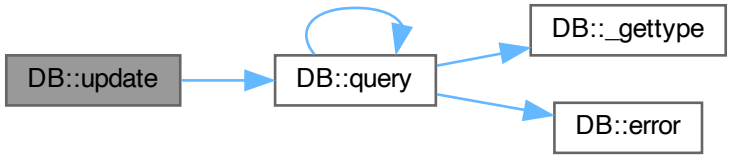
Definition at line 158 of file DB.php.

```
00159   {
00160     $sql = self::mkSql($table, $columns, $rows, "INSERT");
00161     $update = "";
00162     foreach ($columns as $column) {
00163       $update .= "`$column`=VALUES(`$column`), ";
00164     }
00165     $update = trim($update, ", ");
00166     $sql .= " ON DUPLICATE KEY UPDATE $update";
00167     $this->query($sql);
```

```
00168   }
```

References $sql, and query().

Here is the call graph for this function:



4.5.3 Member Data Documentation

$connection

```
DB::$connection  [protected]
```

Definition at line 26 of file DB.php.

$db

```
DB::$db  [static]
```

Definition at line 41 of file DB.php.

Referenced by RefFile::__construct(), CFG::dropDB(), AnswerDAO::get(), MarksDAO::get(), QuestionDAO::get(), RefFileDAO::get(), RubricDAO::get(), StudentDAO::get(), SubQuestionDAO::get(), Stat::getRows(), Stat::mistakes(), CFG::mkDropDB(), Grader::postComment(), AnswerDAO::process(), AnswerDAO::put(), QuestionDAO::put(), RefFileDAO::put(), RubricDAO::put(), StudentDAO::put(), SubQuestionDAO::put(), AnswerDAO::update(), MarksDAO::update(), and StudentDAO::update().

$error

```
DB::$error = ""  [private]
```

Definition at line 44 of file DB.php.

Referenced by __construct(), error(), and getError().

$orderBy

```
DB::$orderBy = ["section", "student_email"]  [static], [private]
```

Definition at line 47 of file DB.php.

Referenced by get(), and getOrderBy().

$query

```
DB::$query  [protected]
```

Definition at line 29 of file DB.php.

Referenced by query().

$query_closed

```
DB::$query_closed = true  [protected]
```

Definition at line 35 of file DB.php.

$query_count

```
DB::$query_count = 0
```

Definition at line 38 of file DB.php.

$show_errors

```
DB::$show_errors = true  [protected]
```

Definition at line 32 of file DB.php.

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

- DB.php

4.6 Exam Class Reference

Inheritance diagram for Exam:

Collaboration diagram for Exam:

Public Member Functions

- __construct ()
- render ($student)
- grade ()
- csv ()
- export ()

- stats ()
- __call ($method, $args)
- getQuestions ()
- getSections ()

Public Member Functions inherited from HT

- __construct ()
- print ($toStr=true)

Static Public Member Functions

- static view ()
- static process ()
- static submissions ()
- static rubrics ()
- static refFiles ()

Static Public Member Functions inherited from HT

- static init ()
- static index ()
- static __callStatic ($method, $args)
- static quickInfo ($side)
- static head ($toStr=false, $title="Grader by Manoj", $refresh='')
- static codeHead ($title="Code Listing by Manoj")
- static foot ($toStr=false)
- static say ($what, $class='plain', $toStr=false, $showIcon=false)
- static getMsgTypes ()
- static error ($what, $toStr=false)
- static warn ($what, $toStr=false)
- static info ($what, $toStr=false)
- static success ($what, $toStr=false)
- static plain ($what, $toStr=false)

Static Protected Member Functions

- static delete ()

Static Protected Member Functions inherited from HT

- static mkFrame ($side='left')

Static Protected Attributes

- static $class = __CLASS__
- static $actions
- static $actionProperties
- static $intro
- static $actionDesc
- static $display

Static Protected Attributes inherited from HT

- static $graderPath
- static $class = __CLASS__
- static $actions
- static $actionProperties = [ ]
- static $actionDesc
- static $intro
- static $display = ["class" => "success", "title" => ""]
- static $studentDao
- static $answerDao
- static $marksDao
- static $rubricDao
- static $refFileDao
- static $questionDao
- static $subQuestionDao
- static $exam

Private Member Functions

- mkLabTest ()
- mkFinals ()

Static Private Member Functions

- static mkRow ($subQuestion, $refFile)

Private Attributes

- $name
- $questions = [ ]
- $sections = [ ]

4.6.1 Detailed Description

Class Exam

Represents an exam object with its associated questions and student sections.

Handles the processing of exam-related data such as:

- Importing rubrics, reference files, solutions, and student submissions.
- Generating questions and subquestions from CSV files.
- Handling grading, CSV exports, and statistical reports.

Inherits from HT for UI rendering and interaction.

Definition at line 15 of file Exam.php.

4.6.2 Member Function Documentation

__call()

```
Exam::__call (
              $method,
              $args)
```

Magic method to catch undefined method calls and show error.

Parameters

string	*$method*	
array	*$args*	

Returns

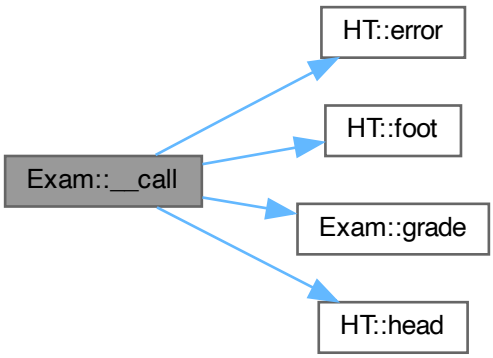
void

Definition at line 533 of file Exam.php.

```
00534     {
00535         HT::head();
00536         HT::error("Got an unknown dispatch request: " . __CLASS__ .
00537             "->$method!");
00538         $this->grade();
00539         HT::foot();
00540     }
```

References HT::error(), HT::foot(), grade(), and HT::head().

Here is the call graph for this function:



__construct()

```
Exam::__construct ()
```

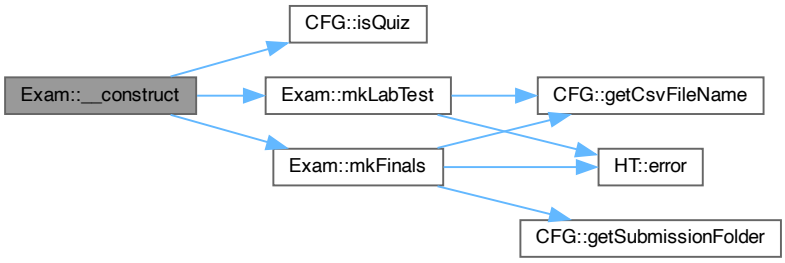
Exam constructor. Initializes the exam name and generates questions/sections based on mode (Quiz or Lab Test).

Definition at line 108 of file Exam.php.

```
00109     {
00110         $this->name = CFG::$examShortName;
00111         if (CFG::isQuiz()) {
00112             $this->mkFinals();
00113         } else {
00114             $this->mkLabTest();
00115         }
00116     }
```

References CFG::$examShortName, CFG::isQuiz(), mkFinals(), and mkLabTest().

Here is the call graph for this function:



csv()

```
Exam::csv ()
```

Exports grades as CSV.

Returns

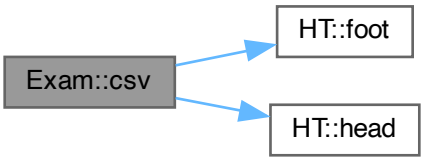
void

Definition at line 286 of file Exam.php.

```
00287     {
00288         HT::head();
00289         foreach ($this->sections as $section) {
00290             $section->csv();
00291         }
00292         HT::foot();
00293     }
```

References HT::foot(), and HT::head().

Here is the call graph for this function:



delete()

```
static Exam::delete ()  [static], [protected]
```

Deletes all processed student submissions.

Returns

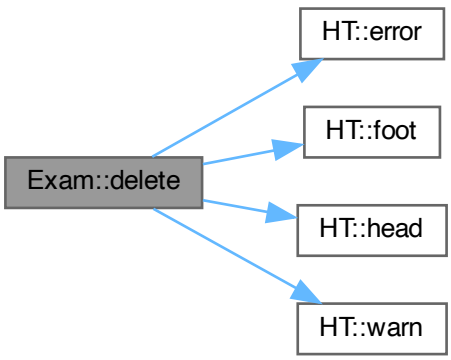
void

Definition at line 477 of file Exam.php.

```
00478     {
00479         HT::head();
00480         if (!isset($_POST['confirmDelete'])) {
00481             $str = "Deleting all unzipped student files!<br>
00482               Really delete them? <table align='right'><tr>
00483               <td><form method='post' action='?do'>
00484               <input type='hidden' name='do' value='delete'>
00485               <input type='hidden' name='confirmDelete' value='dummy'>
00486               <button type='submit' class='error'>
00487               <strong>Proceed to delete</strong>
00488               </button></form></td>
00489               </tr></table>";
00490             HT::warn($str);
00491         } else {
00492             HT::error("Deleting all student submissions...");
00493             self::init();
00494             $exam = self::$exam;
00495             foreach ($exam->getSections() as $section) {
00496                 $section->delete();
00497             }
00498         }
00499         HT::foot();
00500     }
```

References HT::$exam, HT::error(), HT::foot(), HT::head(), and HT::warn().

Here is the call graph for this function:



export()

```
Exam::export ()
```

Exports all exam data.

Returns

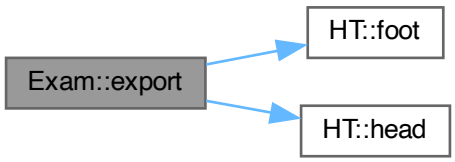
void

Definition at line 300 of file Exam.php.

```
00301     {
00302         HT::head();
00303         foreach ($this->sections as $section) {
00304             $section->export();
00305         }
00306         HT::foot();
00307     }
```

References HT::foot(), and HT::head().

Here is the call graph for this function:



getQuestions()

```
Exam::getQuestions ()
```

Gets all questions in the exam.

Returns

Question[]

Definition at line 549 of file Exam.php.

```
00550     {
00551         return $this->questions;
00552     }
```

References $questions.

getSections()

```
Exam::getSections ()
```

Gets all sections in the exam.

Returns

Section[]

Definition at line 559 of file Exam.php.

```
00560     {
00561         return $this->sections;
00562     }
```

References $sections.

grade()

```
Exam::grade ()
```

Grades all student sections.

Returns

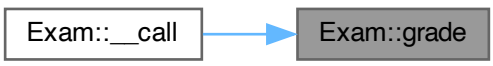
void

Definition at line 274 of file Exam.php.

```
00275     {
00276         foreach ($this->sections as $section) {
00277             $section->grade();
00278         }
00279     }
```

Referenced by __call().

Here is the caller graph for this function:



mkFinals()

```
Exam::mkFinals ()  [private]
```

Processes the Finals/Quiz mode: Generates data from either database or CSV, depending on 'force' POST flag.

Returns

void

Definition at line 180 of file Exam.php.

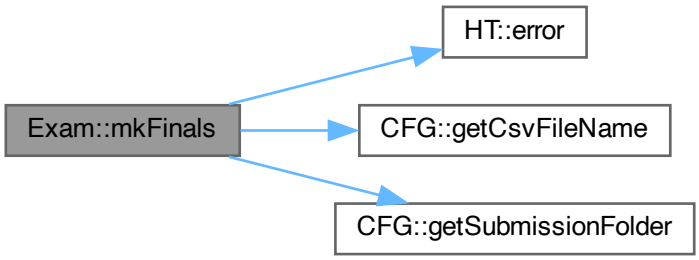
```
00181     {
00182         $sectionName = "All";
00183         if (empty($_POST['force'])) { // Use DB
00184             $students = self::$studentDao->getStudents();
00185             $section = new Section($sectionName);
00186             $this->sections[] = $section->setStudents($students);
00187             $questions = self::$questionDao->get();
00188             foreach ($questions as $question) {
00189                 $qNum = $question->getNum();
00190                 $subQuestions = self::$subQuestionDao->get(['question_num' => $qNum]);
00191                 $question->setSubQuestions($subQuestions);
00192                 $this->questions[$qNum] = $question;
00193             }
00194         } else {
00195             $csv = CFG::getCsvFileName("");
00196             if (!is_file($csv)) {
00197                 HT::error("Error loading eLearn report $csv<br>
00198             Please run an <strong>Attempt Details</strong> report.");
00199                 return;
00200             }
00201             // $lockFile = CFG::getSubmissionFolder($sectionName) . "mkFinal.lock";
00202             // if (file_exists($lockFile)) {
00203             //     HT::error("Students have been processed already!<br>
00204             //     If you need to reprocess, first remove the lock file:<br>
00205             //     <code>$lockFile</code>.");
00206             //     return;
00207             // }
00208             // touch($lockFile);
00209             $file = fopen($csv, 'r');
00210             $headers = fgetcsv($file);
00211             $answerIdx = array_search('Answer', $headers);
00212             $questionIdx = array_search('Q #', $headers);
00213             $nameIdx = array_search('FirstName', $headers);
00214             $emailIdx = array_search('Username', $headers);
00215             $marksIdx = array_search('Out Of', $headers);
00216             $qNum = CFG::$gradeQuestions[0];
00217             $qName = "q$qNum";
00218             $zip = new ZipArchive;
00219             $students = [];
00220             while ($line = fgetcsv($file)) {
00221                 if ($line[$questionIdx] == $qNum) {
00222                     $email = $line[$emailIdx];
00223                     $name = $line[$nameIdx];
00224                     $student = new Student($email, $name, $sectionName);
00225                     $zipFile = CFG::getSubmissionFolder($sectionName) . $email . ".zip";
00226                     $student->setZipFile($zipFile);
00227                     $res = $zip->open($zipFile, ZipArchive::CREATE);
00228                     if ($res === TRUE) {
00229                         // Trying to restore line-breaks gobbled by eLearn CSV export
00230                         // TODO: Need to find the right logic rather than hardcode it
00231                         $answer = str_replace("   ", "\n   ", $line[$answerIdx]);
00232                         // https://stackoverflow.com/a/710967
00233                         $answer = preg_replace('/^\h*\v+/m', '', $answer);
00234                         $zip->addFromString("$qName/Olympics.php", $answer);
00235                         $zip->close();
00236                     }
00237                     $marks[$qName] = $line[$marksIdx];
00238                     $students[] = $student;
00239                 }
00240             }
00241             fclose($file);
00242             $section = new Section($sectionName);
00243             $this->sections[] = $section->setStudents($students);
00244             self::$studentDao->put($section, $students);
00245             $this->questions[$qNum] = new Question($qNum, $marks);
00246             self::$questionDao->put($this->questions);
00247             $subQuestion = new SubQuestion($qName, $marks[$qName], $qNum);
00248             self::$subQuestionDao->put([$subQuestion]);
00249         }
```

```
00250     }
```

References CFG::$gradeQuestions, $name, $questions, HT::error(), CFG::getCsvFileName(), and CFG::getSubmissionFolder().

Referenced by __construct().

Here is the call graph for this function:



Here is the caller graph for this function:

mkLabTest()

```
Exam::mkLabTest ()  [private]
```

Processes the Lab Test mode: Generates questions and sections from CSV file.

Returns

void

Definition at line 125 of file Exam.php.

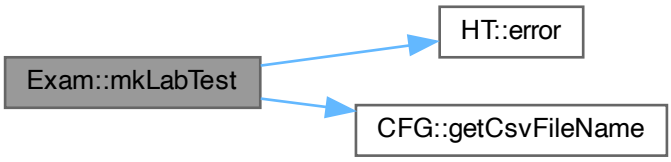
```
00126     {
00127         // Section Names
00128         $sectionNames = CFG::$sectionNames;
00129         // Locations of subQuestion number and marks in the CSV file header
00130         $locSubQ = CFG::$locSubQ;
00131         $locMarks = CFG::$locMarks;
00132         // From the header row of the given CSV file (eLearn grade export),
00133         // construct the question and its subQuestions
00134         // $csv = "./CSV/IS113-LT1-{$sectionNames[0]}.csv";
```

```
00135         $csv = CFG::getCsvFileName($sectionNames[0]);
00136         if (!is_file($csv)) {
00137             HT::error("Error loading $csv");
00138             return;
00139         }
00140         $file = fopen($csv, 'r');
00141         $line = fgetcsv($file);
00142         fclose($file);
00143         // Filter only the questions column headers
00144         foreach ($line as $key => $value) {
00145             if (empty($value) || (preg_match('@[qQ].[ A-Za-z]@', $value) == 0)) {
00146                 unset($line[$key]);
00147             }
00148         }
00149         // Get the marks for each subQuestion and question numbers
00150         $marks = $questionNumbers = [];
00151         foreach ($line as $key => $value) {
00152             $tokens = explode(" ", $value);
00153             $subQuestionName = strtolower(trim($tokens[$locSubQ]));
00154             $marks[$subQuestionName] = explode(":", $tokens[$locMarks])[1];
00155             // Use preg_match() function to check match
00156             preg_match('!\d+!', $subQuestionName, $match);
00157             if (!in_array($match[0], $questionNumbers)) {
00158                 $questionNumbers[] = $match[0];
00159             }
00160         }
00161         // We now have question numbers, subQuestion names and marks
00162         // Add questions (which will add subQuestions too)
00163         // Add only gradeable questions
00164         foreach ($questionNumbers as $qNum) {
00165             if (in_array($qNum, CFG::$gradeQuestions)) {
00166                 $this->questions[$qNum] = new Question($qNum, $marks);
00167             }
00168         }
00169         // Add sections (which will add students too)
00170         foreach ($sectionNames as $sectionName) {
00171             $this->sections[] = new Section($sectionName);
00172         }
00173     }
```

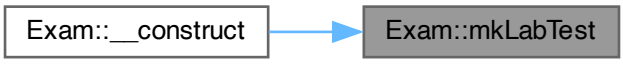
References CFG::$gradeQuestions, CFG::$locMarks, CFG::$locSubQ, CFG::$sectionNames, HT::error(), and CFG::getCsvFileName().

Referenced by __construct().

Here is the call graph for this function:



Here is the caller graph for this function:



mkRow()

```
static Exam::mkRow (
              $subQuestion,
              $refFile)  [static], [private]
```

Creates a table row for a subquestion and reference file.

Parameters

SubQuestion	*$subQuestion*	
RefFile	*$refFile*	

Returns

string HTML table row.

Definition at line 511 of file Exam.php.

```
00512     {
00513         if ($refFile->isGradable()) {
00514             $shown = "<td class='success'>Yes</td>";
00515         } else {
00516             $shown = "<td class='error'>No</td>";
00517         }
00518         $refFileName = CFG::mkFileName($refFile->getFileName());
00519         $row = "<tr><td>" . $subQuestion->getName() .
00520             $shown .
00521             "</td><td><code>" . $refFileName .
00522             "</code></td></tr>";
00523         return $row;
00524     }
```

References CFG::mkFileName().

Here is the call graph for this function:

process()

```
static Exam::process ()  [static]
```

Processes rubrics, reference files, and student submissions.

Returns

void

Definition at line 352 of file Exam.php.

```
00353     {
00354         self::init();
00355         self::rubrics();
00356         self::refFiles();
00357         self::submissions();
00358     }
```

refFiles()

```
static Exam::refFiles ()  [static]
```

Displays and processes reference and solution files.

Returns

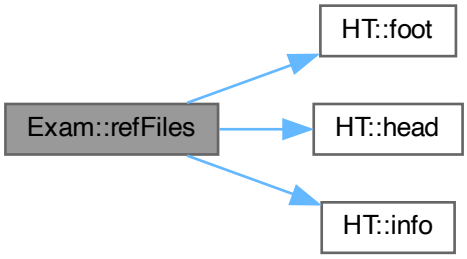
void

Definition at line 440 of file Exam.php.

```
00441     {
00442         self::init();
00443         HT::head();
00444         $exam = self::$exam;
00445         $resSrc = "<table width='98%' align='center' class='grader warn'>
00446                 <tr><th colspan='100%'>Resources</th></tr>
00447                 <tr><th width='10%'>SubQ</th><th width='10%'>Show?</th>
00448                 <th width='80%'>File Name</th></tr>";
00449         $solSrc = "<table width='98%' align='center' class='grader warn'>
00450                 <tr><th colspan='100%'>Solutions</th></tr>
00451                 <tr><th width='10%'>SubQ</th><th width='10%'>Show?</th>
00452                 <th width='80%'>File Name</th></tr>";
00453         foreach ($exam->questions as $question) {
00454             foreach ($question->getSubQuestions() as $subQuestion) {
00455                 self::$refFileDao->process($subQuestion);
00456                 $resources = self::$refFileDao->getByObject($subQuestion, 'resources');
00457                 $solutions = self::$refFileDao->getByObject($subQuestion, 'solutions');
00458                 foreach ($resources as $resource) {
00459                     $resSrc .= self::mkRow($subQuestion, $resource);
00460                 }
00461                 foreach ($solutions as $solution) {
00462                     $solSrc .= self::mkRow($subQuestion, $solution);
00463                 }
00464             }
00465         }
00466         $resSrc .= "</table><br>";
00467         $solSrc .= "</table><br>";
00468         HT::info($resSrc . $solSrc);
00469         HT::foot();
00470     }
```

References HT::$exam, HT::foot(), HT::head(), and HT::info().

Here is the call graph for this function:



render()

```
Exam::render (
              $student)
```

Renders exam content for a given student.

Parameters

Student	*$student*	

Returns

string Rendered HTML.

Definition at line 260 of file Exam.php.

```
00261     {
00262         $srcDoc = "";
00263         foreach ($this->questions as $question) {
00264             $srcDoc .= $question->render($student);
00265         }
00266         return $srcDoc;
00267     }
```

rubrics()

```
static Exam::rubrics ()  [static]
```

Processes and imports rubrics into the system.

Returns

void

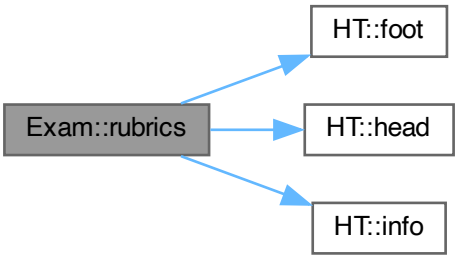
Definition at line 390 of file Exam.php.

```
00391     {
00392         self::init();
00393         HT::head();
00394         $exam = self::$exam;
00395         $qtnSrc = "<table width='60%' align='center' class='warn grader'>
00396                 <tr><th width='30%'>Question Number</th><th width='10%'></th>
00397                 <th width='60%'>SubQuestion</th></tr>";
00398         $rubSrc = "";
00399         foreach ($exam->questions as $question) {
00400             self::$questionDao->process();
00401             $qtnSrc .= "<tr><td>" . $question->getNum() . " (Marks:" .
00402                 $question->getMarks() .
00403                 ")</td><td></td><td></td></tr>";
00404             foreach ($question->getSubQuestions() as $subQuestion) {
00405                 self::$subQuestionDao->process($question);
00406                 self::$rubricDao->process($subQuestion);
00407                 $subQuestionName = $subQuestion->getName();
00408                 $qtnSrc .= "<tr><td></td><td>&rarr;</td>
00409                 <td>$subQuestionName  (Marks:" . $subQuestion->getMarks() .
00410                     ")</td></tr>";
00411                 $class = "success";
00412                 $rubSrc .= "<table width='95%' align='center' class='grader $class'>
00413                 <tr><th>Marks</th><th>Rubric (for $subQuestionName)</th>
00414                 <th>File Name</th></tr>";
00415                 $rubrics = self::$rubricDao->getByObject($subQuestion);
00416                 foreach ($rubrics as $k => $rubric) {
00417                     $marks = $rubric->getMarks();
00418                     $rubricName = $rubric->getRubricName();
00419                     $rubricText = $rubric->getRubricText();
00420                     $rubricText = htmlentities($rubricText);
00421                     $shortFileName = $rubric->getShortFileName();
00422                     $rubSrc .= "<tr><td> $marks </td>
00423                     <td style='text-align:left;'> $rubricName:
00424                     <code>$rubricText</code></td>
00425                     <td><code>$shortFileName</code></td></tr>";
00426                 }
00427                 $rubSrc .= "</table><br>";
00428             }
00429         }
00430         $qtnSrc .= "</table><br>";
00431         HT::info($qtnSrc . $rubSrc);
00432         HT::foot();
```

```
00433     }
```

References $class, HT::$exam, HT::foot(), HT::head(), and HT::info().

Here is the call graph for this function:



stats()

```
Exam::stats ()
```

Displays exam statistics.

Returns

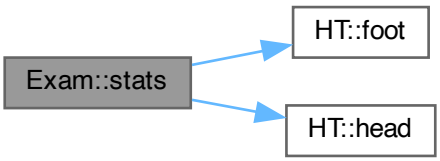
void

Definition at line 314 of file Exam.php.

```
00315     {
00316         HT::head();
00317         foreach ($this->sections as $section) {
00318             $section->stats();
00319         }
00320         HT::foot();
00321     }
```

References HT::foot(), and HT::head().

Here is the call graph for this function:



submissions()

```
static Exam::submissions ()  [static]
```

Imports and processes student submissions.

Returns

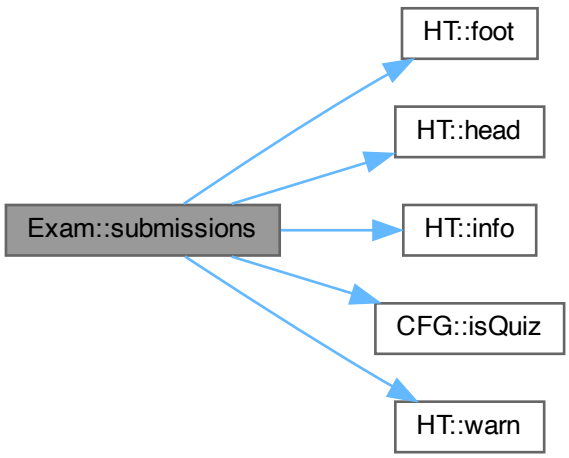
void

Definition at line 365 of file Exam.php.

```
00366     {
00367         self::init();
00368         HT::head();
00369         $exam = self::$exam;
00370         foreach ($exam->sections as $section) {
00371             HT::warn($section->getName());
00372             if (CFG::isQuiz()) {
00373             } else {
00374                 self::$studentDao->process($section);
00375             }
00376             foreach ($section->getStudents() as $student) {
00377                 self::$answerDao->process($student);
00378             }
00379         }
00380         HT::info("<h2>Error Message Summary of Submissions</h2>" .
00381             self::$answerDao->getProcessSummary());
00382         HT::foot();
00383     }
```

References HT::$exam, HT::foot(), HT::head(), HT::info(), CFG::isQuiz(), and HT::warn().

Here is the call graph for this function:



view()

```
static Exam::view ()  [static]
```

Displays exam overview and total marks.

Returns

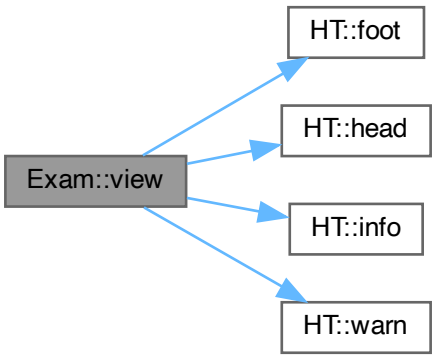
void

Definition at line 331 of file Exam.php.

```
00332     {
00333         self::init();
00334         $str = "";
00335         $exam = self::$exam;
00336         $marks = 0;
00337         foreach ($exam->questions as $question) {
00338             $marks += $question->getMarks();
00339             $str .= HT::info($question->print(), true);
00340         }
00341         $str .= HT::info("<h3>Total Marks = $marks</h3>", true);
00342         HT::head();
00343         HT::warn($str);
00344         HT::foot();
00345     }
```

References HT::$exam, HT::foot(), HT::head(), HT::info(), and HT::warn().

Here is the call graph for this function:



4.6.3 Member Data Documentation

$actionDesc

```
Exam::$actionDesc  [static], [protected]
```

Initial value:

```
= [
        'view' => 'Examine the questions, subQuestions and their marks as
        processed from the eLearn grade book exported (CSV) files.
        <span style="color:red;">In Finals Mode (as opposed to Lab-Test Mode),
```

```
        this button reprocesses the eLearn output file, and repopulates
        the DB. Use with care!</span>',
        'process' => 'Does what the next three buttons do:
        Import <code>rubrics.csv</code> to create rubrics, import
        the grade book downloads from eLearn to create questions, subQuestions
        and students, and import student submissions to create answers and
        prepare for grading. You can also use Grader to verify that students
        have submitted valid zip files: Grader will produce color-coded status
        messages per student.',
        'rubrics' => 'Import <code>rubrics.csv</code> to create questions,
        subQuestions and their rubrics in the database.',
        'refFiles' => 'Process the resource and solution files into the database',
        'submissions' => 'Import student submissions to create answers and prepare
        for grading. This button can be used to validate the submitted zip files.',
        'delete' => 'Delete the student submissions: The uncompressed files, not
        the zip files downloaded from eLearn.'
    ]
```

Definition at line 74 of file Exam.php.

$actionProperties

```
Exam::$actionProperties  [static], [protected]
```

Initial value:

```
= [
        'view' => [
            'force' => 'force',
            'class' => 'error'
        ],
        'rubrics' => [
            'class' => 'warn'
        ],
        'refFiles' => [
            'class' => 'warn',
            'target' => 'right'
        ],
        'submissions' => [
            'class' => 'warn'
        ],
        'delete' => [
            'class' => 'error',
            'target' => 'right'
        ]
    ]
```

Definition at line 42 of file Exam.php.

$actions

```
Exam::$actions  [static], [protected]
```

Initial value:

```
= [
        'view' => 'Exam Details',
        'process' => 'Process Everything!',
        'rubrics' => 'Questions and Rubrics',
        'refFiles' => 'Resources and Solutions',
        'submissions' => 'Submissions',
        'delete' => 'Delete Submissions'
    ]
```

Definition at line 32 of file Exam.php.

$class

```
Exam::$class = __CLASS__  [static], [protected]
```

Definition at line 27 of file Exam.php.

Referenced by rubrics().

$display

```
Exam::$display  [static], [protected]
```

Initial value:

```
= [
        "class" => "info",
        "title" => "Preprocess Files"
    ]
```

Definition at line 97 of file Exam.php.

$intro

```
Exam::$intro  [static], [protected]
```

Initial value:

```
= "<h4>Process Menu</h4>
    Here, you can process the files to set up Grader:
    <ul>
    <li>Process grade book downloads and <code>rubrics.csv</code> to create
    questions, subQuestions and rubrics.</li>
    <li>Process the reference resource files and solutions.</li>
    <li>Process student submissions, as downloaded from eLearn.</li>
    </ul>"
```

Definition at line 64 of file Exam.php.

$name

```
Exam::$name  [private]
```

Definition at line 18 of file Exam.php.

Referenced by mkFinals().

$questions

```
Exam::$questions = []  [private]
```

Definition at line 21 of file Exam.php.

Referenced by getQuestions(), and mkFinals().

$sections

```
Exam::$sections = []  [private]
```

Definition at line 24 of file Exam.php.

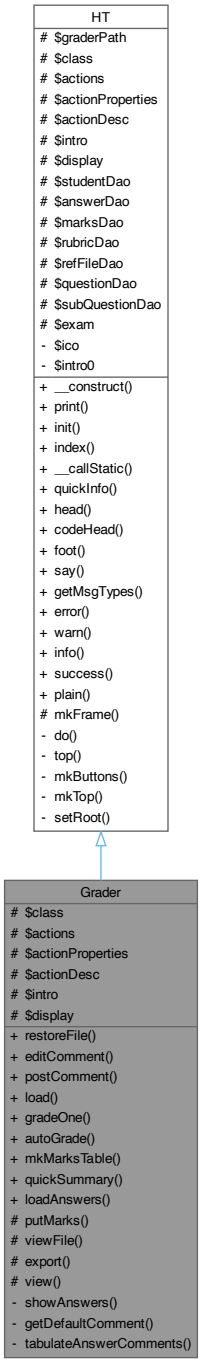
Referenced by getSections().

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

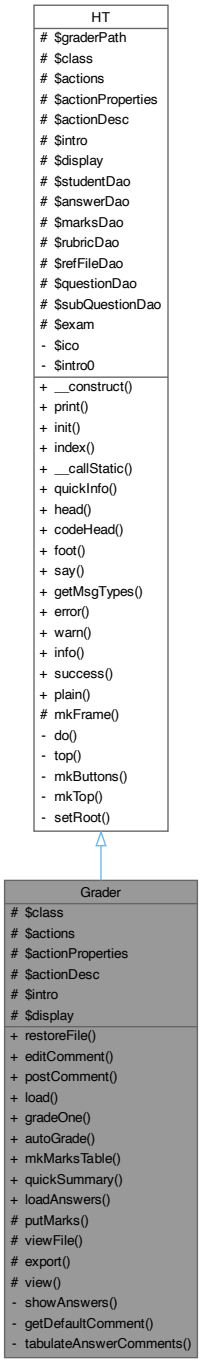
- Exam.php

4.7 Grader Class Reference

Inheritance diagram for Grader:



Collaboration diagram for Grader:



Static Public Member Functions

- static restoreFile ()
- static editComment ()
- static postComment ()
- static load ()
- static gradeOne ()

- static autoGrade ()
- static mkMarksTable ($student)
- static quickSummary ()
- static loadAnswers ()

Static Public Member Functions inherited from HT

- static init ()
- static index ()
- static __callStatic ($method, $args)
- static quickInfo ($side)
- static head ($toStr=false, $title="Grader by Manoj", $refresh='')
- static codeHead ($title="Code Listing by Manoj")
- static foot ($toStr=false)
- static say ($what, $class='plain', $toStr=false, $showIcon=false)
- static getMsgTypes ()
- static error ($what, $toStr=false)
- static warn ($what, $toStr=false)
- static info ($what, $toStr=false)
- static success ($what, $toStr=false)
- static plain ($what, $toStr=false)

Static Protected Member Functions

- static putMarks ()
- static viewFile ()
- static export ()
- static view ()

Static Protected Member Functions inherited from HT

- static mkFrame ($side='left')

Static Protected Attributes

- static $class = __CLASS__
- static $actions
- static $actionProperties
- static $actionDesc
- static $intro
- static $display

Static Protected Attributes inherited from HT

- static $graderPath
- static $class = __CLASS__
- static $actions
- static $actionProperties = [ ]
- static $actionDesc
- static $intro
- static $display = ["class" => "success", "title" => ""]
- static $studentDao
- static $answerDao
- static $marksDao
- static $rubricDao
- static $refFileDao
- static $questionDao
- static $subQuestionDao
- static $exam

Static Private Member Functions

- static showAnswers ($student)
- static getDefaultComment ($answer)
- static tabulateAnswerComments ($answer)

Additional Inherited Members

Public Member Functions inherited from HT

- __construct ()
- print ($toStr=true)

4.7.1 Detailed Description

Class Grader

Manages the full grading workflow for student exams, including:

- Loading students and their submissions.
- Manual and automatic grading via rubrics.
- Handling grader interfaces and exporting grades.
- Editing and managing grading comments.

Inherits from HT for UI rendering and control.

Definition at line 14 of file Grader.php.

4.7.2 Member Function Documentation

autoGrade()

```
static Grader::autoGrade ()  [static]
```

Automatically grades all student submissions where applicable.

Returns

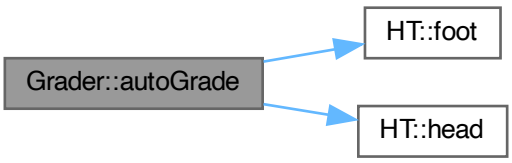
void

Definition at line 504 of file Grader.php.

```
00505   {
00506     self::init();
00507     HT::head();
00508     $exam = self::$exam;
00509     foreach ($exam->getSections() as $section) {
00510       foreach ($section->getStudents() as $student) {
00511         $student->autoGrade();
00512       }
00513     }
00514     HT::foot();
00515   }
```

References HT::$exam, HT::foot(), and HT::head().

Here is the call graph for this function:



editComment()

```
static Grader::editComment ()  [static]
```

Provides a form interface for editing grading comments.

Returns

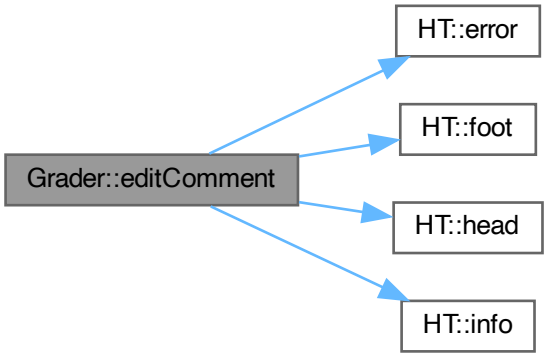
void

Definition at line 284 of file Grader.php.

```
00285   {
00286     self::init();
00287     $fileName = $_POST['filename'];
00288     $answer = self::$answerDao->getByFileName($fileName);
00289     $nAnswer = count($answer);
00290     if ($nAnswer == 0) {
00291       HT::error("Sorry, could not find the answer in the DB!<br>
00292               File: $fileName. This should never happen.");
00293       return;
00294     } elseif ($nAnswer > 1) {
00295       HT::error("Too many answers for $fileName!<br>
00296               This should never happen.");
00297       return;
00298     }
00299     $answer = $answer[0];
00300     $studentEmail = $answer->getStudentEmail();
00301     $student = self::$studentDao->getByEmail($studentEmail)[0];
00302     $studentName = $student->getName();
00303     $comment = $answer->getComment();
00304     if (empty($comment)) {
00305       $comment = self::getDefaultComment($answer);
00306     }
00307     $msg = $student->getMessage();
00308     $str = HT::info("<form method='post'>
00309       <table width='90%' align='center'>
00310       <tr><td width='20%'>Name: </td>
00311       <td><input type='text' value='$studentName $msg'
00312       style='width:90%' disabled></td></tr>
00313       <tr><td>File: </td>
00314       <td><input type='text' value='$fileName' style='width:90%' disabled></td>
00315       </tr><tr>
00316       <td valign='top'>Comment:</td>
00317       <td>
00318       <textarea style='width:90%;height:70vh;' name='comment'>$comment</textarea>
00319       </td>
00320       </tr>
00321       <tr>
00322       <th colspan='100%'><input type='submit' value='Update Comment' name='postComment'
00323       align='center'></th>
00324       </tr>
00325       </table>
00326       <input type='hidden' name='do' value='postComment'>
00327       <input type='hidden' name='student_email' value='$studentEmail'>
00328       <input type='hidden' name='filename' value='$fileName'>
00329       </form>", true);
00330     HT::head();
00331     echo $str;
00332     HT::foot();
00333     return;
00334   }
```

References HT::error(), HT::foot(), HT::head(), and HT::info().

Here is the call graph for this function:



export()

```
static Grader::export ()  [static], [protected]
```

Exports final grades as CSV files for upload to LMS.

Returns

void

Definition at line 443 of file Grader.php.

```
00444   {
00445     self::init();
00446     $exam = self::$exam;
00447     $exam->export();
00448   }
```

References HT::$exam.

getDefaultComment()

```
static Grader::getDefaultComment (
              $answer)  [static], [private]
```

Generates default comment based on existing rubrics.

Parameters

Answer	*$answer*	

Returns

string

Definition at line 216 of file Grader.php.

```
00217   {
00218     self::init();
00219     $subQuestionName = $answer->getSubQuestionName();
00220     $shortFileName = $answer->getShortFileName();
00221     $rubrics = self::$rubricDao->get([
00222       'subquestion_name' => $subQuestionName,
00223       'short_filename' => $shortFileName
00224     ]);
00225     $comment = "";
00226     foreach ($rubrics as $rubric) {
00227       $comment .= $rubric->getRubricName() . " [" .
00228         $rubric->getRubricText() . "]: \n\n";
00229     }
00230     return $comment;
00231   }
```

gradeOne()

```
static Grader::gradeOne ()  [static]
```

Loads grading interface for a single student.

Returns

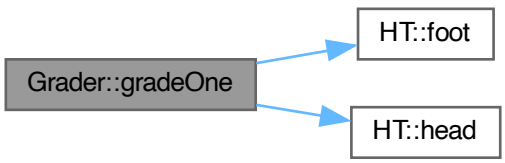
void

Definition at line 480 of file Grader.php.

```
00481   {
00482     $email = $_POST['email'];
00483     self::init();
00484     $student = self::$studentDao->getByEmail($email)[0];
00485     $left1 = htmlentities($student->render(), ENT_QUOTES);
00486     $questions = self::$exam->getQuestions();
00487     $left2 = "";
00488     foreach ($questions as $question) {
00489       $left2 .= htmlentities($question->render($student), ENT_QUOTES);
00490     }
00491     HT::head();
00492     echo "<iframe name='left1' srcdoc='$left1'
00493     style='border:none;width:98vw;height:100px;min-height:100px;' ></iframe>";
00494     echo "<iframe name='left2' srcdoc='$left2'
00495     style='border:none;width:98vw;height:90vh;' ></iframe>";
00496     HT::foot();
00497   }
```

References HT::foot(), and HT::head().

Here is the call graph for this function:



load()

```
static Grader::load ()  [static]
```

Loads student list and prepares grading interface.

Returns

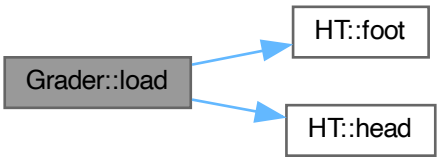
void

Definition at line 467 of file Grader.php.

```
00468   {
00469     self::init();
00470     HT::head();
00471     self::$exam->grade();
00472     HT::foot();
00473   }
```

References HT::foot(), and HT::head().

Here is the call graph for this function:



loadAnswers()

```
static Grader::loadAnswers ()  [static]
```

Loads and displays student answers for manual review.

Returns

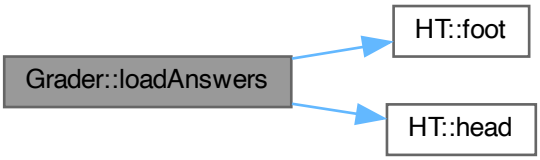
void

Definition at line 707 of file Grader.php.

```
00708   {
00709     $email = $_POST['email'];
00710     self::init();
00711     $student = self::$studentDao->getByEmail($email)[0];
00712     // $right1 = htmlentities(self::showSummary($student));
00713     $right1 = htmlentities(self::showAnswers($student), ENT_QUOTES);
00714     HT::head();
00715     echo "<iframe name='right1' srcdoc='$right1'
00716     style='border:none;width:98vw;height:150px;min-height:150px;' ></iframe>";
00717     echo "<iframe name='right2' srcdoc="
00718     style='border:none;width:98vw;height:85vh;' ></iframe>";
00719     HT::foot();
00720   }
```

References HT::foot(), and HT::head().

Here is the call graph for this function:



mkMarksTable()

```
static Grader::mkMarksTable (
              $student)  [static]
```

Generates a summary marks table for a student.

Parameters

Student	*$student*	

Returns

string HTML table of marks.

Definition at line 609 of file Grader.php.

```
00610   {
00611     // Make a table of marks
00612     $marksTable = "<table class='grader success' align='center'>
00613     <tr><th>Question</th><th>Marks</th><th>Out Of</th></tr>";
00614     $subQuestions = self::$subQuestionDao->get();
00615     $total = $totalOutOF = 0;
00616     foreach ($subQuestions as $subQuestion) {
00617       $marks = self::$marksDao->getByObject($student, $subQuestion);
00618       $subQuestionName = $subQuestion->getName();
00619       $subTotal = 0;
00620       foreach ($marks as $mark) {
00621         $subTotal += $mark->getMarks();
00622       }
00623       $total += $subTotal;
00624       $outOf = $subQuestion->getMarks();
00625       $totalOutOF += $outOf;
00626       $subTotal = number_format($subTotal, 3);
00627       $outOf = number_format($outOf, 0);
00628       $marksTable .= "<tr>
00629       <td>$subQuestionName</td>
00630       <td>$subTotal</td>
00631       <td>$outOf</td>
00632       </tr>";
00633     }
00634     $total = number_format($total, 3);
00635     $totalOutOF = number_format($totalOutOF, 0);
00636     $marksTable .= "<tr><th>Total</th><th>$total</th><th>$totalOutOF</th></tr></table>";
00637     return $marksTable;
00638   }
```

postComment()

```
static Grader::postComment ()  [static]
```

Handles form submission to update grading comments in the database.

Returns

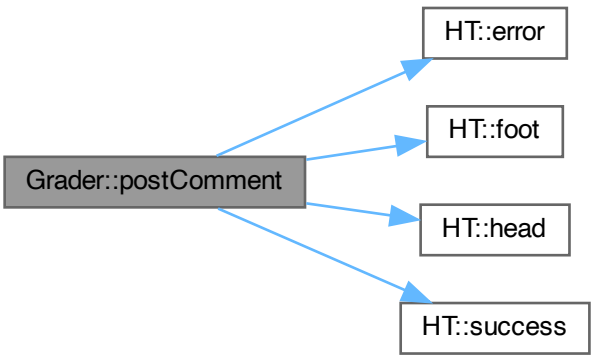
void

Definition at line 341 of file Grader.php.

```
00342   {
00343     self::init();
00344     $fileName = $_POST['filename'];
00345     $answer = self::$answerDao->getByFileName($fileName);
00346     $nAnswer = count($answer);
00347     if ($nAnswer == 0) {
00348       HT::error("Sorry, could not find the answer in the DB!<br>
00349             File: $fileName. This should never happen.");
00350       return;
00351     } elseif ($nAnswer > 1) {
00352       HT::error("Too many answers for $fileName!<br>
00353             This should never happen.");
00354       return;
00355     }
00356     $answer = $answer[0];
00357     $studentEmail = $answer->getStudentEmail();
00358     $student = self::$studentDao->getByEmail($studentEmail)[0];
00359     $studentName = $student->getName();
00360     $comment = $_POST['comment'];
00361     $str = HT::success("
00362       <table width='90%' align='center'>
00363       <tr><td width='20%'>Name: </td><td><input type='text' value='$studentName'
00364       style='width:90%' disabled></td></tr>
00365       <tr><td>File: </td><td><input type='text' value='$fileName'
00366       style='width:90%' disabled></td></tr>
00367       <tr><td valign='top'>Comment:</td><td>
00368       <textarea style='width:90%;height:70vh;' disabled>$comment</textarea>
00369       </td></tr>
00370       <tr><th colspan='100%'>Comment Posted to the DB</th></tr>
00371       </table>");
00372     $comment = DB::$db->sanitize($_POST['comment']);
00373     $answer->setComment($comment);
00374     self::$answerDao->update($answer);
00375     HT::head();
00376     echo $str;
00377     HT::foot();
00378     return;
00379   }
```

References DB::$db, HT::error(), HT::foot(), HT::head(), and HT::success().

Here is the call graph for this function:



putMarks()

```
static Grader::putMarks ()  [static], [protected]
```

Inserts or updates marks in the database for a graded subquestion.

Returns

void

Definition at line 148 of file Grader.php.

```
00149   {
00150     $full = isset($_POST['putFullMarks']);
00151     self::init();
00152     $row = $_POST;
00153     $subQuestionName = $row['subquestion_name'];
00154     $subQuestion = self::$subQuestionDao->get(
00155       ['subquestion_name' => $subQuestionName]
00156     )[0];
00157     $rubrics = self::$rubricDao->getByObject($subQuestion);
00158     $studentEmail = $row['student_email'];
00159     $student = self::$studentDao->getByEmail($studentEmail)[0];
00160     $rubSrc = "<table class='grader' width='80%' align='center'>
00161       <tr><th>Rubric</th><th>Award</th><th>Ref</th></tr>";
00162     $totAwarded = $totQuestion = 0.0;
00163     foreach ($rubrics as $rubric) {
00164       $rubricName = $rubric->getRubricName();
00165       if (isset($row[$rubricName])) {
00166         $marks = self::$marksDao->getByObject($student, $subQuestion, $rubric);
00167         if (empty($marks)) {
00168           $marks = new Marks($studentEmail, $subQuestionName, $rubricName, 0);
00169         } else {
00170           $marks = $marks[0];
00171         }
00172         $rubricName = $rubric->getRubricName();
00173         if ($full) {
00174           $awarded = (float) $row["full-$rubricName"];
00175         } else {
00176           $awarded = (float) $row[$rubricName];
00177         }
00178         $marks->setMarks($awarded);
00179         self::$marksDao->update($marks);
00180         $refMarks = $rubric->getMarks();
00181         $rubricText = $rubric->getRubricText();
00182         $max = max([$refMarks, 0]);
00183         if ($awarded == $max) {
00184           $class = "success";
00185         } else if ($awarded > 0) {
00186           $class = "warn";
00187         } else {
00188           $class = "error";
00189         }
00190         $totQuestion += $max;
00191         $totAwarded += $awarded;
00192         $rubricText = htmlentities($rubricText, ENT_QUOTES);
00193         $rubSrc .= "<tr class='$class'>
00194         <td style='text-align:left;'> $rubricName: <var>$rubricText</var></td>
00195         <td>$awarded</td><td> $awarded </td> </tr>";
00196       }
00197     }
00198     $rubSrc .= "<tr><th>Total</th><th>$totAwarded</th>
00199     <th>$totQuestion</th></tr></table>";
00200     $reload = "<script type='text/javascript'>var done='no';
00201       window.onload=function()
00202       {window.parent.parent.left.left1.document.getElementById('reload').submit();
00203       /* alert('Marks Updated: [$subQuestionName = $totAwarded/$totQuestion]');*/}
00204       </script>";
00205     echo $reload;
00206     // HT::success($rubSrc);
00207     // HT::foot();
00208   }
```

References $class.

quickSummary()

```
static Grader::quickSummary ()  [static]
```

Generates a quick grading summary and email preview for a student.

Returns

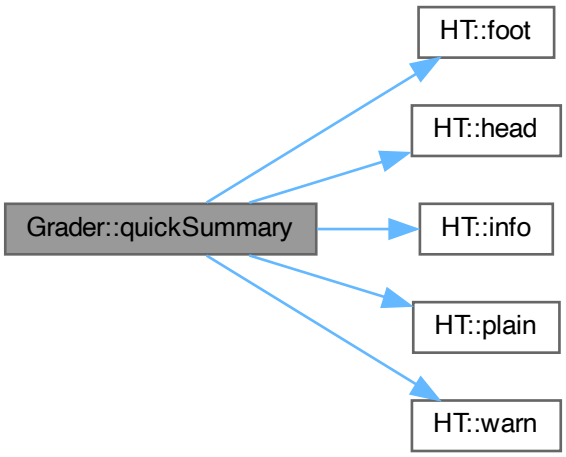
void

Definition at line 645 of file Grader.php.

```
00646   {
00647     $email = $_POST['email'];
00648     self::init();
00649     $student = self::$studentDao->getByEmail($email)[0];
00650     $answers = self::$answerDao->getByObject($student);
00651
00652     $comment = "";
00653     foreach ($answers as $answer) {
00654       $comment .= self::tabulateAnswerComments($answer);
00655     }
00656     if (!empty($comment)) {
00657       $comment = "<h2>Manual Grading (Overrides or Augments Auto-Grading)</h2>$comment";
00658     } else {
00659       $comment = "<h2>No Manual Grading Needed!</h2>";
00660     }
00661
00662     $subject = CFG::$examLongName . " Feedback";
00663
00664     $studentName = $student->getName();
00665     $marksTable = self::mkMarksTable($student);
00666     $intro = "<p>Dear " . ucwords(strtolower($studentName)) . ",</p>" .
00667       "<p>Here is the $subject for questions 1 and 3.
00668     The grading is done first through automated code, in two passes.
00669     Then, the auto-graded results are carefully reviewed and adjusted
00670     as necessary.</p>" .
00671       "<p>You will find the rubrics and results of the auto-grading part in the
00672     first yellow box, followed by the manual adjustments (if any)
00673     in the second grey box.</p>" .
00674       "<p>If you feel that the rubrics are not correctly applied to your
00675     answers, do get in touch with me. Note that we will not discuss
00676     the merits of the rubrics (because they are applied to all students),
00677     but only their correct application.</p>
00678     <p>Here's the top-line summary:</p>
00679     $marksTable
00680     ";
00681
00682     $emailSmu = "$email@smu.edu.sg";
00683     $msg =
00684       HT::head(true) .
00685       HT::info("<h2>$studentName</h2>" .
00686         "<span onclick='emailThis(\"$emailSmu\", \"$subject\")'
00687         title='Click to email this'>
00688         <h3>Email $subject</h3></span>
00689         <p>{$student->getMessage()}</p>", true) .
00690       HT::info($intro, true) .
00691       HT::warn(html_entity_decode($student->getComment()), true) .
00692       HT::plain($comment, true) .
00693       HT::foot(true);
00694     // echo "<iframe name='left2' srcdoc='$msg'
00695     // style='border:none;width:98vw;height:90vh;'  ></iframe>";
00696     echo $msg;
00697     return;
00698   }
```

References CFG::$examLongName, $intro, HT::foot(), HT::head(), HT::info(), HT::plain(), and HT::warn().

Here is the call graph for this function:



restoreFile()

```
static Grader::restoreFile ()  [static]
```

Restores a student's file from the database version.

Returns

void

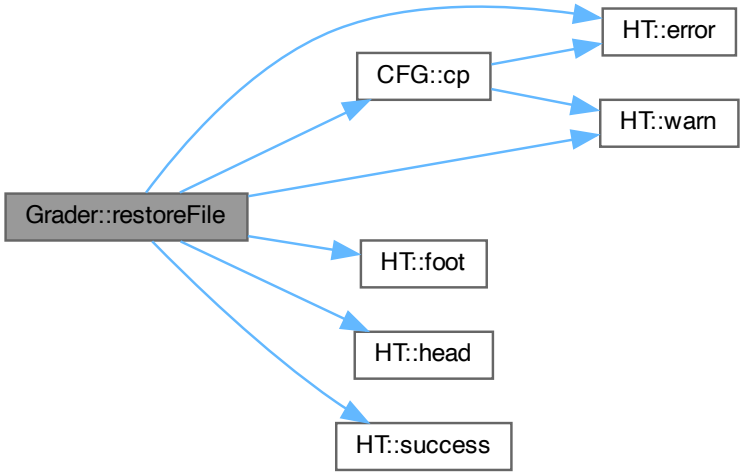
Definition at line 238 of file Grader.php.

```
00239   {
00240     self::init();
00241     $fileName = $_POST['filename'];
00242     $answer = self::$answerDao->getByFileName($fileName);
00243     $nAnswer = count($answer);
00244     if ($nAnswer == 0) {
00245       HT::error("Sorry, could not find the answer in the DB!<br>
00246               File: $fileName. This should never happen.");
00247       return;
00248     } elseif ($nAnswer > 1) {
00249       HT::error("Too many answers for $fileName!<br>
00250               This should never happen.");
00251       return;
00252     }
00253     $answer = $answer[0];
00254     $fileText = file_get_contents($fileName);
00255     $fileDB = $answer->getFullText();
00256     $str = "";
00257     if ($fileText == $fileDB) {
00258       $str .= HT::warn("Text in the file is identical to the one in the DB. Nothing to restore!",
      true);
00259     } else {
00260       $pathInfo = pathinfo($fileName);
00261       $to = "{$pathInfo['dirname']}/{$pathInfo['filename']}-edited.{$pathInfo['extension']}";
00262       if (file_exists($to)) {
00263         $str .= HT::error("Target file $to exists. Please keep it safe before trying to overwrite
      it!", true);
00264       } else {
00265         if (CFG::cp($fileName, $to)) {
```

```
00266           $str .= HT::success("Copied $fileName to $to successfully!", true);
00267         }
00268       }
00269       if (file_put_contents($fileName, $fileDB)) {
00270         $str .= HT::warn("Restored student update from DB to $fileName. (Overwrote it!)", true);
00271       }
00272     }
00273     HT::head();
00274     echo $str;
00275     HT::foot();
00276     return;
00277   }
```

References CFG::cp(), HT::error(), HT::foot(), HT::head(), HT::success(), and HT::warn().

Here is the call graph for this function:



showAnswers()

```
static Grader::showAnswers (
              $student)  [static], [private]
```

Displays all answers for a student, with click-to-grade functionality.

Parameters

Student	*$student*	

Returns

string Rendered HTML.

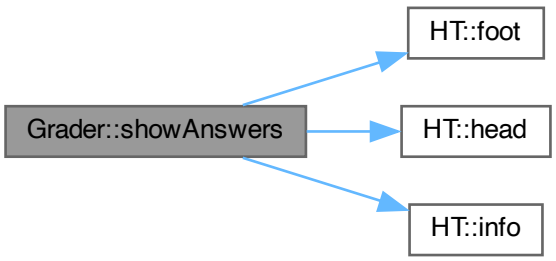
Definition at line 94 of file Grader.php.

```
00095   {
00096     $wrap = 5;
00097     $srcDoc = HT::head(true);
00098     $answers = self::$answerDao->getByObject($student);
00099     $studentName = $student->getName();
00100     $studentEmail = $student->getEmail();
00101     $tabAns = "<table align='center' title='Grader::showAnswers()'><tr>
00102       <td colspan='100%' style='text-align:center;font-size:120%;'>
00103       Answers: <strong title='Grader::showAnswers()'>$studentName</strong>
00104       </td></tr><tr>";
00105     $loadAnswers = "";
00106     $iAns = 0;
00107     if (count($answers) > 0) { // Student attempted
00108       $firstAnswer = "id='firstAnswer'";
00109       foreach ($answers as $answer) {
00110         $td = $answer->render($firstAnswer);
00111         if ($td) {
00112           $tabAns .= $td;
00113           $firstAnswer = "";
00114           $iAns++;
00115           if ($iAns % $wrap == 0) {
00116             $tabAns .= "</tr><tr>";
00117           }
00118         }
00119       }
00120       $loadAnswers = "<script type='text/javascript'>
00121           window.onload=function()
00122           {document.getElementById('firstAnswer').click();
00123
      window.parent.parent.frames['left'].frames['left1'].document.getElementById('$studentEmail').submit();}

00124           </script>";
00125     } else { // Absent?
00126       $tabAns .= "<td>Not Attempted</td>";
00127     }
00128     $colSpan = $wrap - $iAns % $wrap;
00129     if ($colSpan) {
00130       $colSpan = "colspan='$colSpan'";
00131     } else {
00132       $colSpan = "";
00133     }
00134     $closeTable = "<th $colSpan><button onclick='history.back()' class='success'>
00135     <strong>Go Back</strong></button></th></tr></table>";
00136     $tabAns .= $closeTable;
00137     $srcDoc .= HT::info($tabAns, true);
00138     $srcDoc .= $loadAnswers;
00139     $srcDoc .= HT::foot(true);
00140     return $srcDoc;
00141   }
```

References HT::foot(), HT::head(), and HT::info().

Here is the call graph for this function:



tabulateAnswerComments()

```
static Grader::tabulateAnswerComments (
              $answer)  [static], [private]
```

Tabulates rubric comments for a student's answer.

Parameters

Answer	*$answer*	

Returns

string HTML table of comments and marks.

Definition at line 525 of file Grader.php.

```
00526   {
00527     $data = $answer->getComment();
00528     $comment0 = "<h3>Question {$answer->getSubQuestionName()}:
00529       <code>{$answer->getShortFileName()}</code> </h3>";
00530     $rows = [];
00531     if (!empty($data)) {
00532       // This is ugly. For the third question, we take the whole entry as comment
00533       if (strpos($data, 'q3-1') !== false) {
00534         $comment1 =
00535           "<table align='center' width='80%' class='grader warn'>
00536               <tr><th>Rerunning the Auto-Grader</th> </tr>
00537               <tr><td style='text-align: left;'><pre>$data</pre></td></tr>
00538             </table>";
00539       } else {
00540         // Convert data into paragraphs (split by double newlines)
00541         $data = str_replace(["\r\n", "\r"], "\n", $data);
00542         $entries = preg_split("/\r?\n\s*\r?\n/", trim($data));
00543         foreach ($entries as $entry) {
00544           // Match rubric number and rubric text
00545           if (preg_match('/^(q\d+[a-zA-Z]*-\d+) \[(.*?)\]:\s*(.*)$/s', trim($entry), $matches)) {
00546             $rubricNumber = $matches[1]; // e.g., q1a-6
00547
00548             $rubricText = htmlspecialchars(trim($matches[2])); // e.g., Missing or wrong <label>
    </label>
00549             $comment = htmlspecialchars(trim($matches[3])); // The rest is the comment
00550
00551             // Extract marks if present
00552             $marks = '';
00553             if (preg_match('/= (-?[\d.]+)/', $comment, $markMatches)) {
00554               $marks = $markMatches[1];
00555               $comment = trim(str_replace($markMatches[0], '', $comment));
00556             }
00557
00558             if (!empty($comment)) {
00559               // Add only if there's a valid comment
00560               $comment = $trimmed = preg_replace('/\s*marks\s*$/i', '', $comment);
00561               $commentTD = "<td style='text-align: left;'>" . nl2br($comment) . "</td>";
00562               // $commentTD = "<td style='text-align: left;'>$comment</td>";
00563               $class = $marks > 0 ? "success" : "error";
00564               $rubric = self::$rubricDao->get(["rubric_name" => $rubricNumber]);
00565               if (is_array($rubric)) {
00566                 $rubric = $rubric[0];
00567                 $outOf = $rubric->getMarks();
00568                 if ($marks >= 0) {
00569                   $outOf = abs($outOf);
00570                 }
00571               }
00572               $rows[] = "<tr class='$class'>
00573                       <td>{$rubricNumber}</td>
00574                       <td style='text-align: left;'>{$rubricText}</td>
00575                       $commentTD
00576                       <td>{$marks}</td>
00577                       <td>{$outOf}</td>
00578                   </tr>";
00579             }
00580           }
00581         }
00582
00583         // Generate the HTML table
00584         $comment1 =  "<table align='center' width='80%' class='grader warn'>
```

```
00585           <tr>
00586               <th>Rubric Number</th>
00587               <th>Rubric Text</th>
00588               <th>Comment</th>
00589               <th>Marks</th>
00590               <th>Out Of</th>
00591           </tr>
00592           " . implode("\n", $rows) . "
00593           </table>";
00594       }
00595       $comment0 .= $comment1;
00596       // Debug: Append the raw grading data
00597       // $comment0 .= "<div class='error'><pre><code>" . htmlspecialchars($data) .
      "</code></pre></div>";
00598       // var_dump($comment);
00599       return $comment0;
00600     }
00601   }
```

References $class.

view()

```
static Grader::view ()  [static], [protected]
```

Displays all marks for review before export.

Returns

void

Definition at line 455 of file Grader.php.

```
00456   {
00457     self::init();
00458     $exam = self::$exam;
00459     $exam->csv();
00460   }
```

References HT::$exam.

viewFile()

```
static Grader::viewFile ()  [static], [protected]
```

Displays a file's content for review, with optional restore and comment options.

Returns

void

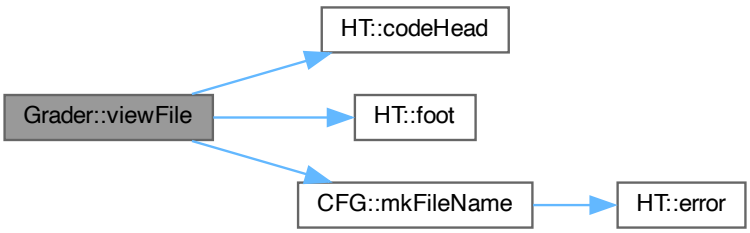
Definition at line 386 of file Grader.php.

```
00387   {
00388     $fileName = $_POST['filename'];
00389     if (isset($_POST['showDB'])) {
00390       $fulltext = $_POST['fulltext'];
00391       $output = "<form method='post' id='restoreFile'>
00392       <input type='hidden' name='do' value='restoreFile'>
00393       <input type='hidden' name='filename' value='$fileName'>
00394       <input type='submit' name='comment' value='Restore File'>
00395       </form>";
00396     } else {
00397       $fulltext = file_get_contents($fileName);
00398       $fileName0 = CFG::mkFileName($fileName);
00399       $output = "<a href='$fileName0'>
00400     <input type='submit' name='run' value='View Output'></a>";
```

```
00401     }
00402     $fulltext = htmlentities($fulltext);
00403     $ext = pathinfo($fileName, PATHINFO_EXTENSION);
00404     $commentable = isset($_POST['commentable']);
00405     HT::codeHead($fileName);
00406     $comment = "";
00407     if ($commentable) {
00408       $popup = $_POST['popup'];
00409       $student_email = $_POST['student_email'];
00410       $subquestion_name = $_POST['subquestion_name'];
00411       $comment = "<form method='post' id='editComment' target='editComment'>
00412       <input type='hidden' name='do' value='editComment'>
00413       <input type='hidden' name='filename' value='$fileName'>
00414       <input type='hidden' name='popup' value='$popup'>
00415       <input type='hidden' name='student_email' value='$student_email'>
00416       <input type='hidden' name='subquestion_name' value='$subquestion_name'>
00417       <input type='submit' name='comment' value='Edit Comment' $popup>
00418       </form>";
00419     }
00420     $realPath = "<div
00421     title='Grader::viewFile()'
00422     style='color: #373;
00423     background: #ecffd6;
00424     border: 1px solid #617c42;
00425     padding:4px;
00426     font-weight:bold;
00427     text-align:left;'><code>" .
00428       CFG::mkFileName($fileName) . "</code></div>";
00429     echo "<table padding='20' style='position:absolute;top:60px;right:15px;'>
00430     <tr><td>$comment</td><td>$output</td></tr>
00431     </table>";
00432     echo "$realPath<pre><code class='language-$ext'>$fulltext</code></pre>";
00433     HT::foot();
00434   }
```

References HT::codeHead(), HT::foot(), and CFG::mkFileName().

Here is the call graph for this function:



4.7.3 Member Data Documentation

$actionDesc

Grader::$actionDesc [static], [protected]

Initial value:

```
= [
    'load' => 'Load Student List so that you can grade each one. The list is
    color-coded to indicate the students who were absent for the test.
    Once loaded, a student can be graded by clicking on their
    <button>Grade</button> or <button>Continue</button> button.
    <span style="color:red">Note that the answer files viewed and the output shown
    are from the current/edited versions (from the disk).</span>',
    'export' => 'After the grading is complete, click on this button to
    generate CSV files that can be imported to eLearn. The file names will
```

```
    be <code>Export-{gradebook}.csv</code> where <code>{gradebook}</code>
    is the name of the grade book exported from eLearn. The files will be
    in the same  folder as the grade book exports.',
    'view' => 'View the marks that will be exported when the next button
    is clicked.',
    '$autoGradables' => 'Run the autograder (typically for Q3, but as specified in
    <code>config.php</code>) for all students. It will update the database
    with the marks computed. The autograder file should output the total marks
    in some recognizable form, as specified in  <code>config.php &rarr;
    $autoGrade[subQuestionName]</code>, which specifies the subquestion
    name (usually q3) and the rubric name (usually q3-1), so that the marks
    can be inserted into the database. For those students whose file crashes,
    Grader will put zero in the database, but will give you buttons to view
    and manually grade q3.'
  ]
```

Definition at line 47 of file Grader.php.

$actionProperties

```
Grader::$actionProperties  [static], [protected]
```

Initial value:

```
= [
    'load' => [
      'target' => 'left',
      'clear' => 'yes'
    ],
    'view' => [
      'target' => 'right'
    ],
    'export' => [
      'target' => 'right'
    ],
    'autoGrade' => [
      'target' => 'left',
      'class' => 'error',
      'clear' => 'yes'
    ]
  ]
```

Definition at line 28 of file Grader.php.

$actions

```
Grader::$actions  [static], [protected]
```

Initial value:

```
= [
    'load' => 'Load Student List',
    'view' => 'View Marks',
    'export' => 'Export Grades (CSV)',
    'autoGrade' => 'Auto-Grade',
  ]
```

Definition at line 20 of file Grader.php.

$class

```
Grader::$class = __CLASS__  [static], [protected]
```

Definition at line 17 of file Grader.php.

Referenced by putMarks(), and tabulateAnswerComments().

$display

```
Grader::$display  [static], [protected]
```

Initial value:

```
= [
    "class" => "warn",
    "title" => "Grade Students"
  ]
```

Definition at line 81 of file Grader.php.

$intro

```
Grader::$intro  [static], [protected]
```

Initial value:

```
= "<h4>Grade Students</h4>
    <p>The most time-consuming process, grading the students, is triggered here.
    Click on the load button to get a list of students, and click on the
    buttons in the list to grade/regrade etc. Grader can export the marks that
    can be directly imported to eLearn.</p>
  "
```

Definition at line 73 of file Grader.php.

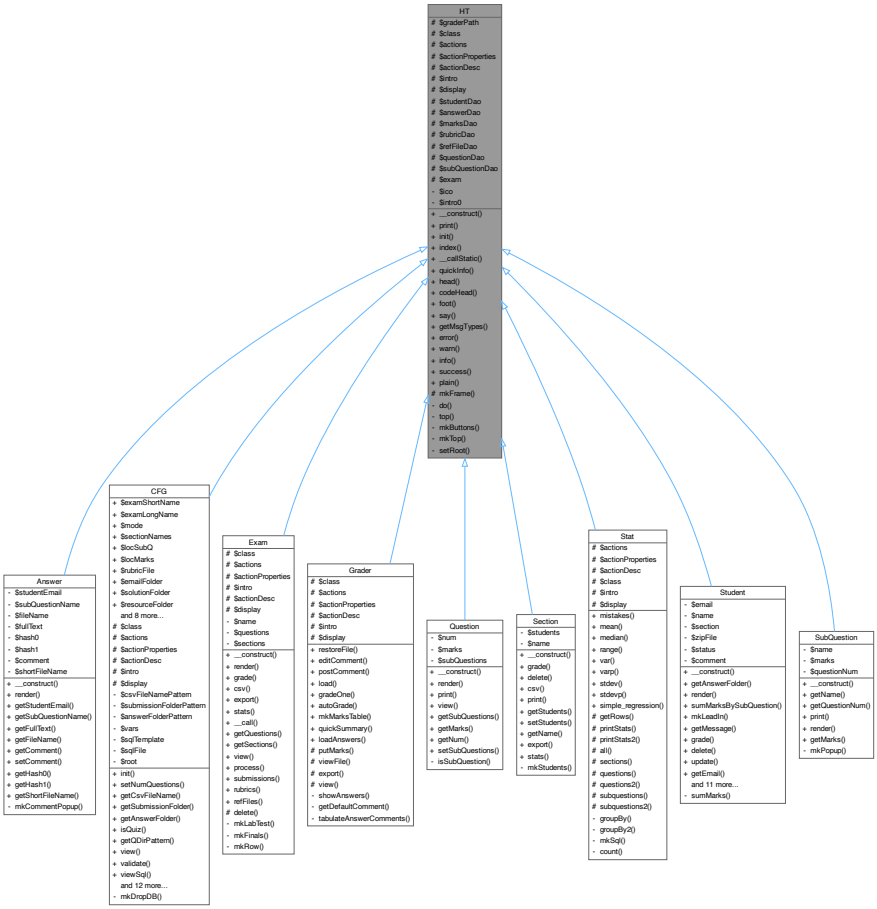
Referenced by quickSummary().

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

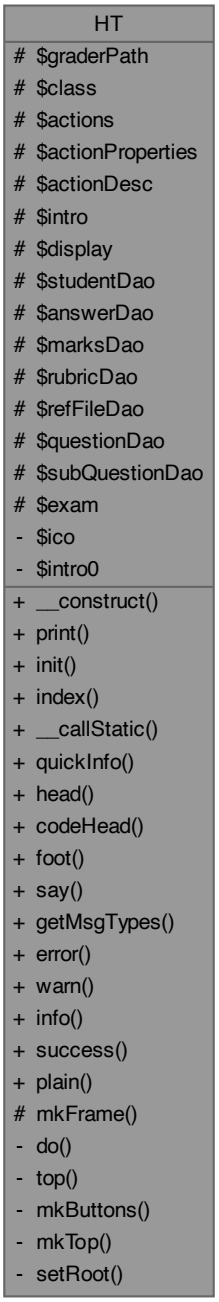
- Grader.php

4.8 HT Class Reference

Inheritance diagram for HT:



Collaboration diagram for HT:



Public Member Functions

- __construct ()
- print ($toStr=true)

Static Public Member Functions

- static init ()

- static index ()
- static __callStatic ($method, $args)
- static quickInfo ($side)
- static head ($toStr=false, $title="Grader by Manoj", $refresh='')
- static codeHead ($title="Code Listing by Manoj")
- static foot ($toStr=false)
- static say ($what, $class='plain', $toStr=false, $showIcon=false)
- static getMsgTypes ()
- static error ($what, $toStr=false)
- static warn ($what, $toStr=false)
- static info ($what, $toStr=false)
- static success ($what, $toStr=false)
- static plain ($what, $toStr=false)

Static Protected Member Functions

- static mkFrame ($side='left')

Static Protected Attributes

- static $graderPath
- static $class = __CLASS__
- static $actions
- static $actionProperties = [ ]
- static $actionDesc
- static $intro
- static $display = ["class" => "success", "title" => ""]
- static $studentDao
- static $answerDao
- static $marksDao
- static $rubricDao
- static $refFileDao
- static $questionDao
- static $subQuestionDao
- static $exam

Static Private Member Functions

- static do ($method='')
- static top ()
- static mkButtons ()
- static mkTop ($actions)
- static setRoot ()

Static Private Attributes

- static $ico
- static $intro0

4.8.1 Detailed Description

Class HT

Base class providing core HTML rendering, user interface (UI), message display utilities, and dispatch handling for Grader system.

This class is extended by CFG, Exam, Grader, and Report.

Major functionalities:

- Generates UI frames and menus for grading workflows.
- Displays status/info/error messages in styled banners.
- Provides common buttons and navigation.
- Handles dispatch calls dynamically via __callStatic.
- Generates DAO instances for interaction with database objects.
- HTML rendering for user interface elements like buttons and action menus.

Definition at line 19 of file HT.php.

4.8.2 Member Function Documentation

__callStatic()

```
static HT::__callStatic (
              $method,
              $args)  [static]
```

Handles static dispatch calls dynamically based on $_POST or $_GET['do'].

Parameters

string	*$method*	
array	*$args*	

Returns

void

Definition at line 254 of file HT.php.

```
00255   { // Handle static dispatch request: $class::$method()
00256     static::do($method);
00257     exit();
00258   }
```

__construct()

```
HT::__construct ()
```

Definition at line 169 of file HT.php.

```
00170   {
00171     self::head();
00172     self::error("Attempt to instantiate static class: " . static::$class);
00173     self::foot();
00174     die;
00175   }
```

codeHead()

```
static HT::codeHead (
              $title = "Code Listing by Manoj")  [static]
```

Render specialized code view header with syntax highlighting.

Parameters

string	*$title*	

Returns

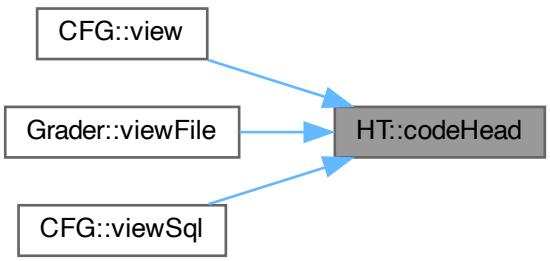
void

Definition at line 544 of file HT.php.

```
00545   {
00546     $path = ".";
00547     $str = "<!DOCTYPE html>
00548     <html>
00549       <head>
00550         <title>$title</title>
00551         <link rel='stylesheet' href='$path/highlight/styles/agate.min.css'>
00552         <script src='$path/highlight/highlight.min.js'></script>
00553         <script>hljs.highlightAll();</script>
00554       </head>
00555       <body>
00556       ";
00557     echo $str;
00558   }
```

Referenced by CFG::view(), Grader::viewFile(), and CFG::viewSql().

Here is the caller graph for this function:



do()

```
static HT::do (
              $method = '')  [static], [private]
```

Handles unknown dispatch requests, redirecting to UI subpages or showing errors.

Parameters

string	*$method*	

Returns

void

Definition at line 265 of file HT.php.

```
00266   {
00267     $class = static::$class;
00268     if (array_key_exists('do', $_POST)) {
00269       $action = $_POST['do'];
00270     } else if (array_key_exists('do', $_GET)) {
00271       $action = $_GET['do'];
00272     } else {
00273       self::head();
00274       self::error("Got an unknown static dispatch request: $class::$method()!<br>
00275         Dispatched to HT::do(), but failed.");
00276       var_dump("POST:", $_POST, "GET:", $_GET);
00277       xdebug_print_function_stack(
00278         "Should not be here!",
00279         XDEBUG_STACK_NO_DESC
00280       );
00281       self::foot();
00282       exit();
00283     }
00284     if (method_exists($class, $action)) {
00285       $class::$action();
00286       exit();
00287     } else { // In the HT class, it's just a redirection to subPages
00288       header("Location: ui/$action.php");
00289       exit();
00290     }
00291   }
```

References $class.

error()

```
static HT::error (
              $what,
              $toStr = false)  [static]
```

Display error message.

Parameters

string	*$what*	
bool	*$toStr*	

Returns

string|void

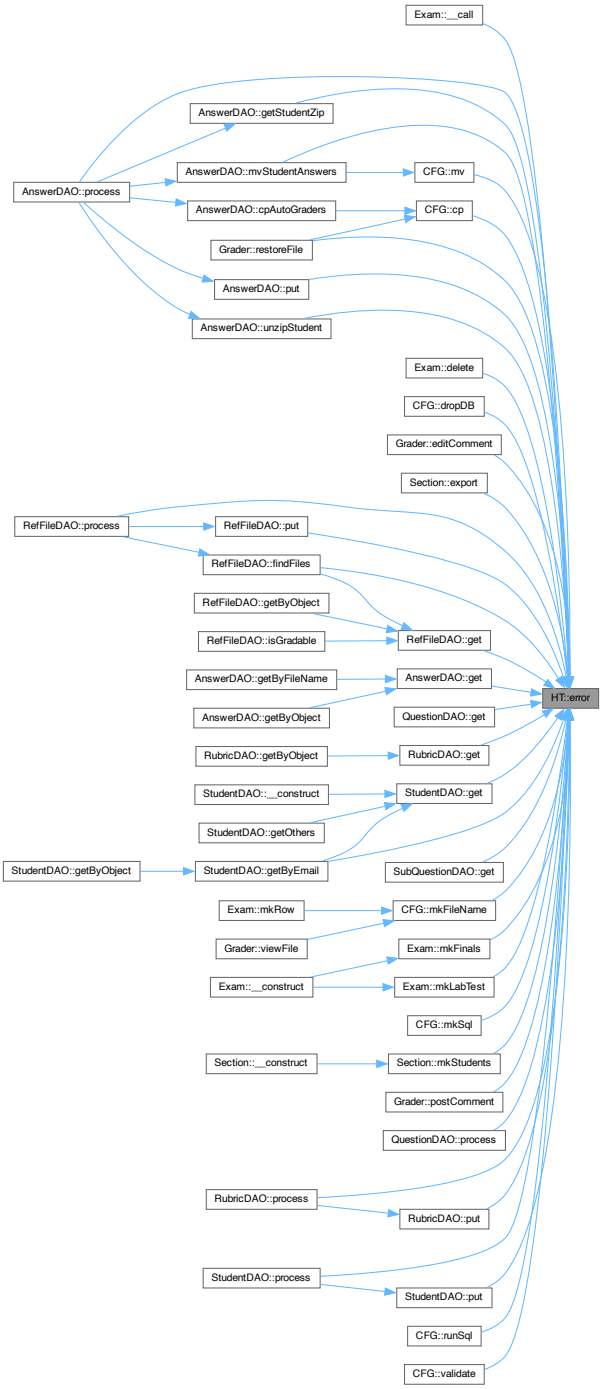
Definition at line 618 of file HT.php.

```
00619   {
00620     return self::say($what, $class = "error", $toStr);
00621   }
```

References $class.

Referenced by Exam::__call(), CFG::cp(), Exam::delete(), CFG::dropDB(), Grader::editComment(), Section::export(), RefFileDAO::findFiles(), AnswerDAO::get(), QuestionDAO::get(), RefFileDAO::get(), RubricDAO::get(), StudentDAO::get(), SubQuestionDAO::get(), StudentDAO::getByEmail(), AnswerDAO::getStudentZip(), CFG::mkFileName(), Exam::mkFinals(), Exam::mkLabTest(), CFG::mkSql(), Section::mkStudents(), CFG::mv(), AnswerDAO::mvStudentAnswers(), Grader::postComment(), AnswerDAO::process(), QuestionDAO::process(), RefFileDAO::process(), RubricDAO::process(), StudentDAO::process(), AnswerDAO::put(), RefFileDAO::put(), RubricDAO::put(), StudentDAO::put(), Grader::restoreFile(), CFG::runSql(), AnswerDAO::unzipStudent(), and CFG::validate().

Here is the caller graph for this function:



foot()

```
static HT::foot (
              $toStr = false)  [static]
```

Render HTML page footer.

Parameters

bool	*$toStr*	

Returns

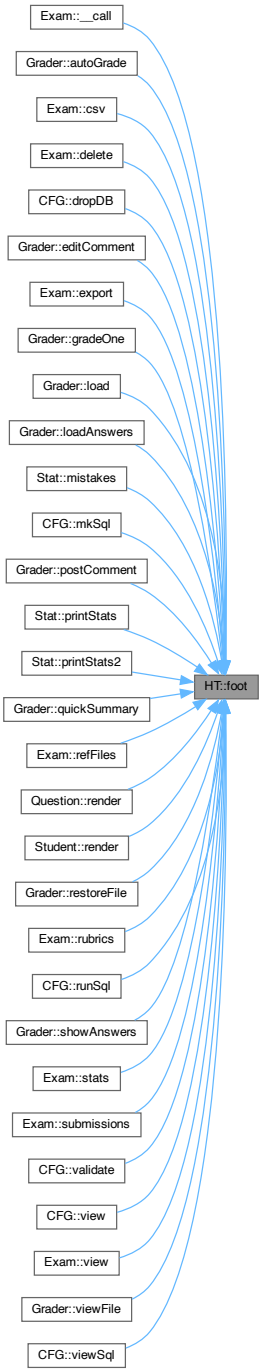
string|void

Definition at line 565 of file HT.php.

```
00566   {
00567     $str = "  </body>
00568     </html>";
00569     if ($toStr) {
00570       return $str;
00571     } else {
00572       echo $str;
00573     }
00574   }
```

Referenced by Exam::__call(), Grader::autoGrade(), Exam::csv(), Exam::delete(), CFG::dropDB(), Grader::editComment(), Exam::export(), Grader::gradeOne(), Grader::load(), Grader::loadAnswers(), Stat::mistakes(), CFG::mkSql(), Grader::postComment(), Stat::printStats(), Stat::printStats2(), Grader::quickSummary(), Exam::refFiles(), Question::render(), Student::render(), Grader::restoreFile(), Exam::rubrics(), CFG::runSql(), Grader::showAnswers(), Exam::stats(), Exam::submissions(), CFG::validate(), CFG::view(), Exam::view(), Grader::viewFile(), and CFG::viewSql().

Here is the caller graph for this function:



getMsgTypes()

```
static HT::getMsgTypes ()  [static]
```

Get available message types.

Returns

array

Definition at line 607 of file HT.php.

```
00608   {
00609     return array_keys(self::$ico);
00610   }
```

head()

```
static HT::head (
              $toStr = false,
              $title = "Grader by Manoj",
              $refresh = '')  [static]
```

Render HTML page header with CSS/JS includes.

Parameters

bool	*$toStr*	
string	*$title*	
string	*$refresh*	

Returns

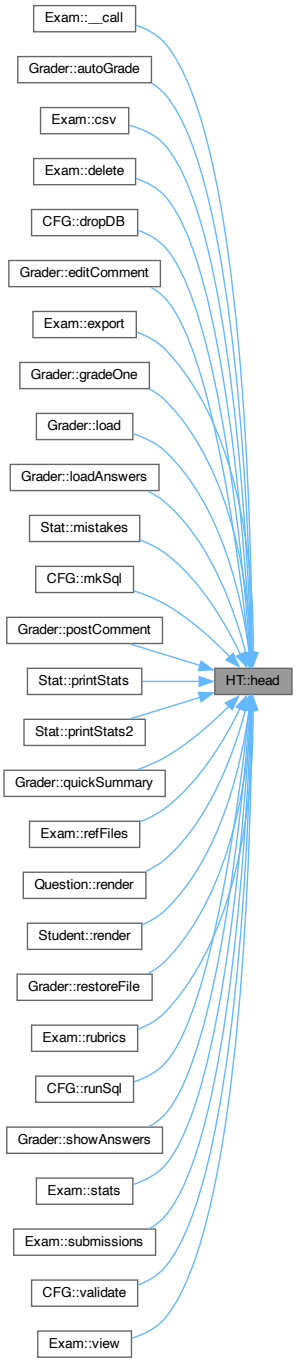
string|void

Definition at line 511 of file HT.php.

```
00515     {
00516     $uiPath = self::$graderPath . "/ui";
00517     $css = "$uiPath/css/error-bar.css";
00518     $js = "$uiPath/js/utils.js";
00519     if (!empty($refresh)) {
00520       $refresh = "<meta http-equiv='refresh' content='$refresh'>";
00521     }
00522     $str = "<!DOCTYPE html>
00523     <html>
00524       <head>
00525         <title>$title</title>
00526         <link href='$css' rel='stylesheet'>
00527         $refresh
00528         <script src='$js'></script>
00529       </head>
00530       <body>
00531       ";
00532     if ($toStr) {
00533       return $str;
00534     } else {
00535       echo $str;
00536     }
00537   }
```

Referenced by Exam::__call(), Grader::autoGrade(), Exam::csv(), Exam::delete(), CFG::dropDB(), Grader::editComment(), Exam::export(), Grader::gradeOne(), Grader::load(), Grader::loadAnswers(), Stat::mistakes(), CFG::mkSql(), Grader::postComment(), Stat::printStats(), Stat::printStats2(), Grader::quickSummary(), Exam::refFiles(), Question::render(), Student::render(), Grader::restoreFile(), Exam::rubrics(), CFG::runSql(), Grader::showAnswers(), Exam::stats(), Exam::submissions(), CFG::validate(), and Exam::view().

Here is the caller graph for this function:



index()

```
static HT::index ()  [static]
```

Main landing page (index.php) handler. Inherited by subpages (in CFG, Exam, Grader, Report)

Returns

void

Definition at line 210 of file HT.php.

```
00211   {
00212     // self::setRoot();
00213     $frame = array_keys($_GET);
00214     if (empty($frame)) {
00215       self::head();
00216       $dbHost = "localhost";
00217       $db = new DB($dbHost, CFG::$dbUser, CFG::$dbPass, CFG::$dbName);
00218       $error = $db->getError();
00219       if ($error) {
00220         self::head();
00221         self::error($error);
00222         self::foot();
00223       } else if (!$db->tablesExist(CFG::$dbPrefix)) {
00224         self::head();
00225         self::warn("No tables in the DB! Please run the DB setup script again.
00226         go to <strong>Setup and Configuration</strong> and set it up.");
00227         self::foot();
00228       }
00229       $top = static::top();
00230       echo "<iframe name='top' srcdoc='$top' id='top'
00231       onload='resizeIframe(this);'
00232       style='border:none;width:96vw;height:90px;' ></iframe>\n";
00233       echo self::mkFrame("left");
00234       echo self::mkFrame("right");
00235       self::foot();
00236     } else {
00237       $action = array_keys($_GET)[0];
00238       if ($action != 'do') {
00239         xdebug_print_function_stack(
00240           "Should not get here!",
00241           XDEBUG_STACK_NO_DESC
00242         );
00243       }
00244       static::$action();
00245     }
00246   }
```

References CFG::$dbName, CFG::$dbPass, CFG::$dbPrefix, and CFG::$dbUser.

info()

```
static HT::info (
              $what,
              $toStr = false)  [static]
```

Display informational message.

Parameters

string	*$what*	
bool	*$toStr*	

Returns

string|void

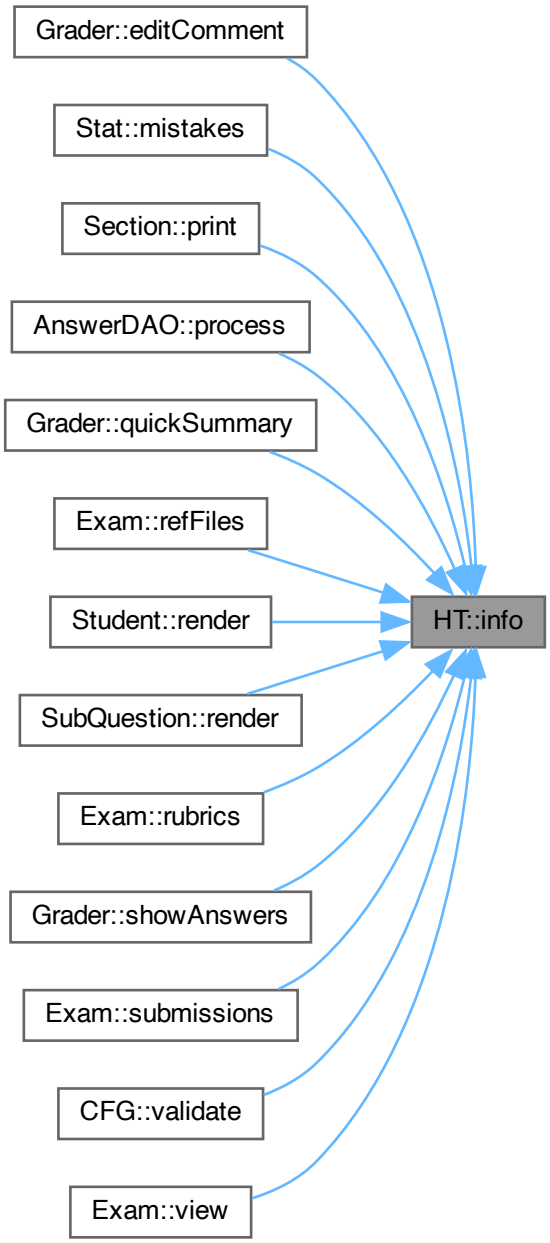
Definition at line 640 of file HT.php.

```
00641   {
00642     return self::say($what, $class = "info", $toStr);
00643   }
```

References $class.

Referenced by Grader::editComment(), Stat::mistakes(), Section::print(), AnswerDAO::process(), Grader::quickSummary(), Exam::refFiles(), Student::render(), SubQuestion::render(), Exam::rubrics(), Grader::showAnswers(), Exam::submissions(), CFG::validate(), and Exam::view().

Here is the caller graph for this function:



init()

```
static HT::init ()  [static]
```

Initializes the DAO instances if applicable.

Returns

void

Definition at line 185 of file HT.php.

```
00186   {
00187     $parent = realpath(__DIR__ . "/..");
00188     self::$graderPath = substr($parent, strlen($_SERVER['DOCUMENT_ROOT']));
00189     // If the display title is not set, we are doing index.php, meaning
00190     // it's too soon to create the DAOs
00191     $dao = !empty(static::$display["title"]);
00192     // die(static::$display["title"]);
00193     if ($dao) {
00194       self::$studentDao = new StudentDAO();
00195       self::$answerDao = new AnswerDAO();
00196       self::$marksDao = new MarksDAO();
00197       self::$rubricDao = new RubricDAO();
00198       self::$refFileDao = new RefFileDAO();
00199       self::$questionDao = new QuestionDAO();
00200       self::$subQuestionDao = new SubQuestionDAO();
00201       self::$exam = new Exam();
00202     }
00203   }
```

mkButtons()

```
static HT::mkButtons ()  [static], [private]
```

Generate Previous/Next button links for navigation.

Returns

array

Definition at line 385 of file HT.php.

```
00386   {
00387     $buttons = [];
00388     $next = $prev = "";
00389     if (empty(static::$display["title"])) {
00390       return $buttons;
00391     }
00392     $action = basename($_SERVER['REQUEST_URI'], ".php");
00393     $actions = array_keys(self::$actions);
00394     $nActions = count($actions);
00395     for ($i = 0; $i < $nActions; $i++) {
00396       if ($action == $actions[$i]) {
00397         if ($i > 0) {
00398           $prev = $actions[$i - 1];
00399         }
00400         if ($i < $nActions - 1) {
00401           $next = $actions[$i + 1];
00402         }
00403         break;
00404       }
00405     }
00406     if ($prev) {
00407       $prevText = self::$actions[$prev];
00408       $buttons['prev'] = "<td>
00409       <a href='$prev.php' target='_parent'><button class='success'><strong>
00410       &Leftarrow; $prevText</strong></button></a></td>";
00411     }
00412     if ($next) {
00413       $nextText = self::$actions[$next];
00414       $buttons['next'] = "<td>
00415       <a href='$next.php' target='_parent'><button class='success'><strong>
00416       $nextText &Rightarrow;</strong></button></a></td>";
00417     }
00418     return $buttons;
00419   }
```

References $actions.

mkFrame()

```
static HT::mkFrame (
              $side = 'left')  [static], [protected]
```

Create left or right iframe content.

Parameters

string	*$side*	

Returns

string

Definition at line 302 of file HT.php.

```
00303   {
00304     $srcdoc = htmlentities(self::quickInfo($side), ENT_QUOTES);
00305     $str = "<iframe name='$side' srcdoc='$srcdoc'
00306     style='border:none;float:left;width:48vw;height:90vh;'></iframe>\n";
00307     return $str;
00308   }
```

mkTop()

```
static HT::mkTop (
              $actions)  [static], [private]
```

Generate main top action buttons.

Parameters

array	*$actions*	

Returns

string

Definition at line 426 of file HT.php.

```
00427   {
00428     $back = !empty(static::$display["title"]);
00429     $what = "<table align='center'><tr>";
00430     $buttons = self::mkButtons();
00431     if ($back) {
00432       $target = "target='left'";
00433     } else {
00434       $target = "target='_parent'";
00435     }
00436     $method = "method='post'";
00437     if ($back) {
00438       if (array_key_exists('prev', $buttons)) {
00439         $what .= $buttons['prev'];
00440       }
00441       $what .= "
00442       <td>
00443       <a href='' target='_parent'>
00444       <button type='submit' class='info'>
00445       <strong>Quick Info</strong>
00446       </button>
00447       </a>
00448       </td>";
00449     }
00450     foreach ($actions as $name => $value) {
00451       $action = "action='?do'";
00452       $class = "class='plain'";
00453       $force = $clearFrames = "";
00454       if (array_key_exists($name, static::$actionProperties)) {
00455         $actionProperties = static::$actionProperties[$name];
00456         if (array_key_exists('target', $actionProperties)) {
00457           $target = $actionProperties['target'];
00458           $target = "target='$target'";
00459         }
```

```
00460         if (array_key_exists('class', $actionProperties)) {
00461           $class = $actionProperties['class'];
00462           $class = "class='$class'";
00463         }
00464         if (array_key_exists('clear', $actionProperties)) {
00465           $clearFrames = "onclick='window.parent.right.document.write();
00466           window.parent.left.document.write();'";
00467         }
00468         if (array_key_exists('force', $actionProperties)) {
00469           $force = "<input type='hidden' name='force' value='force'>";
00470         }
00471       }
00472       $what .= "
00473       <td>
00474       <form $action $method $target style='text-align:center;'>
00475       <input type='hidden' name='do' value='$name'>
00476       $force
00477       <button type='submit' $class $clearFrames>
00478       $value
00479       </button>
00480       </form>
00481       </td>";
00482     }
00483     if ($back) {
00484       $what .= "
00485       <td>
00486       <a href='..' target='_parent'>
00487       <button type='submit' class='success'>
00488       <strong>Home &Uparrow;</strong>
00489       </button>
00490       </a>
00491       </td>";
00492       if (array_key_exists('next', $buttons)) {
00493         $what .= $buttons['next'];;
00494       }
00495     }
00496     $what .= "</tr></table>";
00497     return $what;
00498   }
```

References $actionProperties, $actions, and $class.

plain()

```
static HT::plain (
               $what,
               $toStr = false)  [static]
```

Display plain message without icon.

Parameters

string	*$what*	
bool	*$toStr*	

Returns

string|void

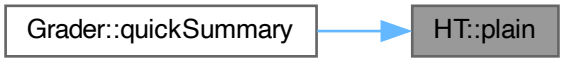
Definition at line 662 of file HT.php.

```
00663   {
00664     return self::say($what, $class = "plain", $toStr);
00665   }
```

References $class.

Referenced by Grader::quickSummary().

Here is the caller graph for this function:



print()

```
HT::print (
              $toStr = true)
```

Collect and render accumulated messages of different types.

Parameters

bool	*$toStr*	

Returns

string

Definition at line 672 of file HT.php.

```
00673   {
00674     $str = "";
00675     $msgTypes = self::getMsgTypes();
00676     foreach ($msgTypes as $fun) {
00677       if ($this->$fun) {
00678         $str .= self::$fun($this->$fun, true);
00679       }
00680     }
00681     return $str;
00682   }
```

quickInfo()

```
static HT::quickInfo (
              $side)  [static]
```

Generate quick information block (intro or action buttons).

Parameters

string	*$side*	

Returns

string

Definition at line 315 of file HT.php.

```
00316   {
00317     $str = self::head(true);
00318     if ($side == 'left') {
00319       $str0 = static::$intro;
00320       if (__CLASS__ != static::$class) {
00321         $str0 .= self::say(self::$intro0, 'plain', true);
00322       }
00323     } elseif ($side = 'right') {
00324       $actions = static::$actions;
00325       $actionProperties = static::$actionProperties;
00326       $actionDesc = static::$actionDesc;
00327       $str0 = "<h4>Action Buttons</h4>";
00328       $str0 .= "<table cellpadding='15'>";
00329       foreach ($actions as $key => $button) {
00330         $desc = "";
00331         if (array_key_exists($key, $actionDesc)) {
00332           $desc = $actionDesc[$key];
00333         }
00334         $class = "plain";
00335         if (array_key_exists($key, $actionProperties)) {
00336           $properties = $actionProperties[$key];
00337           if (array_key_exists('class', $properties)) {
00338             $class = $properties['class'];
00339           }
00340         }
00341         $str0 .= "<tr>
00342           <td width='20%'><button class='$class'>$button</button></td>
00343           <td>$desc</td>
00344           </tr>";
00345       }
00346       $str0 .= "</table>";
00347     }
00348     $display = static::$display["class"];
00349     $str .= self::$display($str0, true);
00350     $str .= self::foot(true);
00351     return "<div id='div$side'>$str</div>";
00352   }
```

References $actionDesc, $actionProperties, $actions, $class, and $display.

say()

```
static HT::say (
              $what,
              $class = 'plain',
              $toStr = false,
              $showIcon = false)  [static]
```

Generic message display with icons and CSS class.

Parameters

string	*$what*	
string	*$class*	
bool	*$toStr*	
bool	*$showIcon*	

Returns

string|void

Definition at line 584 of file HT.php.

```
00585   {
00586     if ($showIcon) {
00587       $icon = self::$ico[$class];
00588       $icon = "<i class='ico'>$icon</i>";
00589     } else {
00590       $icon = "";
00591     }
00592     $str = "    <div class='bar  $class'>
00593     $icon $what
00594   </div>
00595   ";
00596     if ($toStr) {
00597       return $str;
00598     } else {
00599       echo $str;
00600     }
00601   }
```

References $class.

Referenced by Section::grade().

Here is the caller graph for this function:

setRoot()

```
static HT::setRoot ()  [static], [private]
```

Set some global variables $root and $grader. The idea is to load Grader/index.php rather than Grader/Grader8/index.php. That way, Grader8 folder doesn't have to know anything about LT1 or LTx - all configs reside in the LT or FT folder, driven by Grader/index.php

TODO: Doesn't work yet!

Returns

void

Definition at line 696 of file HT.php.

```
00697   {
00698     if (isset($GLOBALS['grader'])) {
00699       $grader = $GLOBALS['grader'];
00700       echo '<script>window.history.pushState({}, "", "/Grader/' . $grader . '/index.php");</script>';
00701     } else {
00702       $GLOBALS['grader'] = $root = basename(__DIR__);
00703     }
00704   }
```

References $root.

success()

```
static HT::success (
              $what,
              $toStr = false)  [static]
```

Display success message.

Parameters

string	*$what*	
bool	*$toStr*	

Returns

string|void

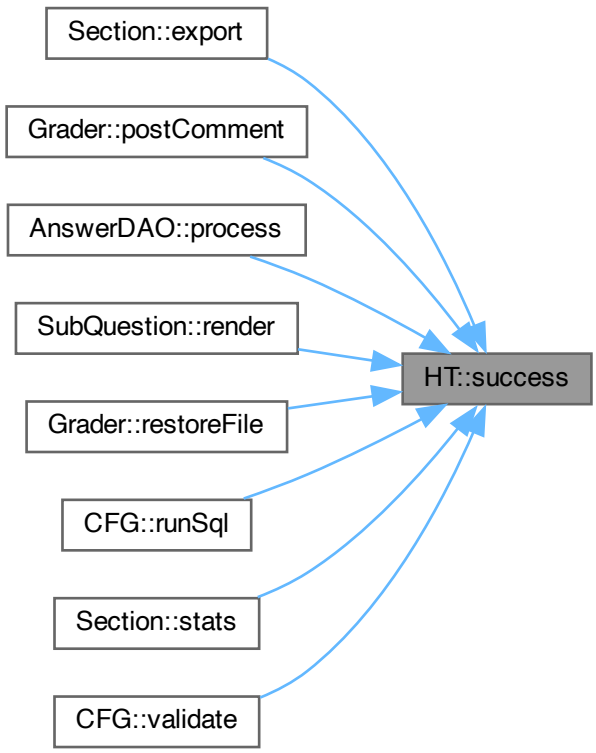
Definition at line 651 of file HT.php.

```
00652   {
00653     return self::say($what, $class = "success", $toStr);
00654   }
```

References $class.

Referenced by Section::export(), Grader::postComment(), AnswerDAO::process(), SubQuestion::render(), Grader::restoreFile(), CFG::runSql(), Section::stats(), and CFG::validate().

Here is the caller graph for this function:



top()

```
static HT::top ()  [static], [private]
```

Generate top frame for navigation buttons.

Returns

string

Definition at line 358 of file HT.php.

```
00359   {
00360     if (empty($_POST)) {
00361       $top = self::head(true);
00362       $title = static::$display['title'];
00363       if ($title) {
00364         $title = ": $title";
00365       }
00366       $what = "<div style='text-align:center;font-size:120%;font-weight:bold'>"
00367         . CFG::$examLongName . $title . "</div>";
00368       $actions = static::$actions;
00369       $what .= static::mkTop($actions);
00370       $display = static::$display["class"];
00371       $top .= self::$display($what, true);
00372       $top .= self::foot(true);
00373       $top = htmlentities($top, ENT_QUOTES);
00374 
00375       // $top = str_replace([ '"', '&' ], [ '&quot;', '&amp;amp;' ], $top);
00376 
00377       return $top;
00378     }
00379   }
```

References $actions, $display, and CFG::$examLongName.

warn()

```
static HT::warn (
              $what,
              $toStr = false)  [static]
```

Display warning message.

Parameters

string	*$what*	
bool	*$toStr*	

Returns

string|void

Definition at line 629 of file HT.php.

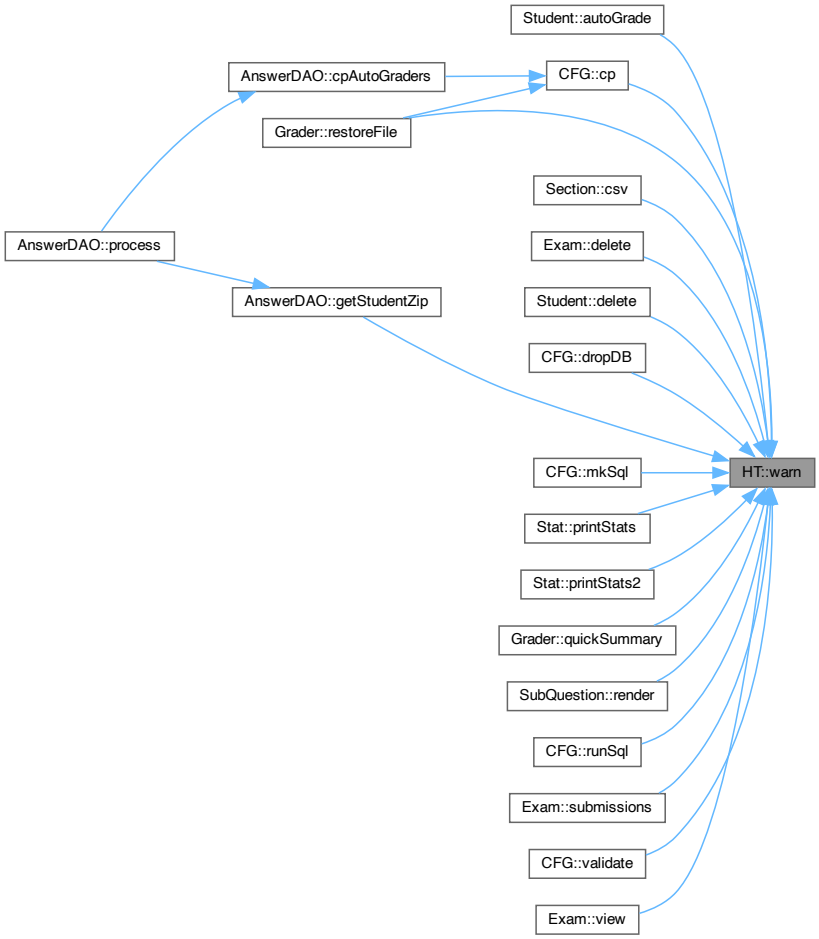
```
00630   {
00631     return self::say($what, $class = "warn", $toStr);
00632   }
```

References $class.

Referenced by Student::autoGrade(), CFG::cp(), Section::csv(), Exam::delete(), Student::delete(), CFG::dropDB(), AnswerDAO::getStudentZip(), CFG::mkSql(), Stat::printStats(), Stat::printStats2(), Grader::quickSummary(),

SubQuestion::render(), Grader::restoreFile(), CFG::runSql(), Exam::submissions(), CFG::validate(), and Exam::view().

Here is the caller graph for this function:



4.8.3 Member Data Documentation

$actionDesc

HT::$actionDesc [static], [protected]

Initial value:

```
= [
    'setup' => 'Set up the database, inspect and validate the configuration file
    <code>config.php</code>. If you see a database setup warning above, click
    on this button to verify your configuration, validate it, examine the SQL
    file and set up your database.',
    'process' => 'Import <code>rubrics.csv</code> to create rubrics, import
    the grade book downloads from eLearn to create questions, subquestions and
    students, and import student submissions to create answers and prepare
    for grading. You can also use Grader to verify that students have submitted
    valid zip files',
    'grade' => 'The next phase (the most time-consuming one) is to grade the
    students. Clicking on this button will make the process as painless
    as possible.',
```

```
    'stats' => 'View Grade Statistics',
    'report' => 'Once the grading process is completed, click on this button
    to export the grades to a CSV file (which can be imported to eLearn
    with minimal modifications). It also has some heuristics to detect
    potential cheating efforts and inconsistencies (such as missing or
    wrong emails in student files as well as similarities among them).'
  ]
```

Definition at line 49 of file HT.php.

Referenced by quickInfo().

$actionProperties

```
HT::$actionProperties = [] [static], [protected]
```

Definition at line 46 of file HT.php.

Referenced by mkTop(), and quickInfo().

$actions

```
HT::$actions [static], [protected]
```

Initial value:

```
= [
    'setup' => 'Setup and Validate',
    'process' => 'Process Submissions, Rubrics etc.',
    'grade' => 'Grade Students',
    'stats' => 'View Grade Statistics',
    'report' => 'Investigate'
  ]
```

Definition at line 37 of file HT.php.

Referenced by mkButtons(), mkTop(), quickInfo(), and top().

$answerDao

```
HT::$answerDao [static], [protected]
```

Definition at line 159 of file HT.php.

$class

```
HT::$class = __CLASS__ [static], [protected]
```

Definition at line 34 of file HT.php.

Referenced by do(), error(), info(), mkTop(), plain(), quickInfo(), SubQuestion::render(), say(), success(), and warn().

$display

```
HT::$display = ["class" => "success", "title" => ""]  [static], [protected]
```

Definition at line 155 of file HT.php.

Referenced by quickInfo(), and top().

$exam

```
HT::$exam  [static], [protected]
```

Definition at line 167 of file HT.php.

Referenced by Grader::autoGrade(), Exam::delete(), Grader::export(), Exam::refFiles(), Answer::render(), Exam::rubrics(), Exam::submissions(), Exam::view(), and Grader::view().

$graderPath

```
HT::$graderPath  [static], [protected]
```

Definition at line 22 of file HT.php.

$ico

```
HT::$ico  [static], [private]
```

Initial value:

```
= [
    "plain" => "",
    "info" => "&#9728;",
    "success" => "&#10004;",
    "warn" => "&#9888;",
    "error" => "&#9747;"
  ]
```

Definition at line 25 of file HT.php.

$intro

```
HT::$intro  [static], [protected]
```

Definition at line 71 of file HT.php.

$intro0

HT::$intro0 [static], [private]

Initial value:

```
= "<p>The normal workflow is expected to be carried out
  by clicking on the buttons from left to right in the top frame. The buttons
  <button class='success'><strong> &Leftarrow; Previous</strong></button> and
  <button class='success'><strong>Next &Rightarrow;</strong></button> can also be used.
  The first
  <button class='info'><strong>Quick Info</strong></button> button will bring up
  this information. The <button class='success'><strong>Home&Uparrow;</strong></button>
  button will take you back to the main menu, the root home page.</p>
  <p>The buttons in <button class='warn'>Yellow</button> are typically not
  needed in the normal workflow, while the ones in <button class='error'>Red
  </button> are dangerous. They may delete information or files or launch
  long, cpu-intensive processes.</p> "
```

Definition at line 141 of file HT.php.

$marksDao

HT::$marksDao [static], [protected]

Definition at line 160 of file HT.php.

$questionDao

HT::$questionDao [static], [protected]

Definition at line 163 of file HT.php.

$refFileDao

HT::$refFileDao [static], [protected]

Definition at line 162 of file HT.php.

$rubricDao

HT::$rubricDao [static], [protected]

Definition at line 161 of file HT.php.

$studentDao

HT::$studentDao [static], [protected]

Definition at line 158 of file HT.php.

$subQuestionDao

```
HT::$subQuestionDao  [static], [protected]
```

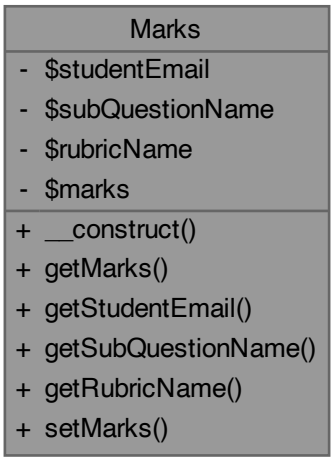
Definition at line 164 of file HT.php.

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

- HT.php

4.9 Marks Class Reference

Collaboration diagram for Marks:



Public Member Functions

- __construct ($studentEmail, $subQuestionName, $rubricName, $marks)
- getMarks ()
- getStudentEmail ()
- getSubQuestionName ()
- getRubricName ()
- setMarks ($marks)

Private Attributes

- $studentEmail
- $subQuestionName
- $rubricName
- $marks

4.9.1 Detailed Description

Class Marks

Represents the marks awarded to a student for a specific rubric in a sub-question of an exam or lab test.

Definition at line 9 of file Marks.php.

4.9.2 Member Function Documentation

__construct()

```
Marks::__construct (
              $studentEmail,
              $subQuestionName,
              $rubricName,
              $marks)
```

Marks constructor.

Parameters

string	*$studentEmail*	The student's email address.
string	*$subQuestionName*	The sub-question name/identifier.
string	*$rubricName*	The rubric name/identifier.
float	*$marks*	The marks awarded.

Definition at line 39 of file Marks.php.

```
00040     {
00041         $this->studentEmail = $studentEmail;
00042         $this->subQuestionName = $subQuestionName;
00043         $this->rubricName = $rubricName;
00044         $this->marks = $marks;
00045     }
```

References $marks, $rubricName, $studentEmail, and $subQuestionName.

getMarks()

```
Marks::getMarks ()
```

Get the marks awarded.

Returns

float The marks awarded for this rubric.

Definition at line 52 of file Marks.php.

```
00053     {
00054         return $this->marks;
00055     }
```

References $marks.

getRubricName()

```
Marks::getRubricName ()
```

Get the rubric name/identifier.

Returns

string The rubric name/identifier.

Definition at line 82 of file Marks.php.

```
00083     {
00084         return $this->rubricName;
00085     }
```

References $rubricName.

getStudentEmail()

```
Marks::getStudentEmail ()
```

Get the student's email address.

Returns

string The student's email address.

Definition at line 62 of file Marks.php.

```
00063     {
00064         return $this->studentEmail;
00065     }
```

References $studentEmail.

getSubQuestionName()

```
Marks::getSubQuestionName ()
```

Get the sub-question name/identifier.

Returns

string The sub-question name (e.g., q1a-2).

Definition at line 72 of file Marks.php.

```
00073     {
00074         return $this->subQuestionName;
00075     }
```

References $subQuestionName.

setMarks()

```
Marks::setMarks (
              $marks)
```

Set the marks awarded.

Parameters

float	*$marks*	The marks to set.

Returns

self Returns the instance for method chaining.

Definition at line 93 of file Marks.php.

```
00093                                     : self
00094     {
00095         $this->marks = $marks;
00096         return $this;
00097     }
```

References $marks.

4.9.3 Member Data Documentation

$marks

```
Marks::$marks  [private]
```

Definition at line 29 of file Marks.php.

Referenced by __construct(), getMarks(), and setMarks().

$rubricName

```
Marks::$rubricName  [private]
```

Definition at line 24 of file Marks.php.

Referenced by __construct(), and getRubricName().

$studentEmail

```
Marks::$studentEmail  [private]
```

Definition at line 14 of file Marks.php.

Referenced by __construct(), and getStudentEmail().

$subQuestionName

```
Marks::$subQuestionName  [private]
```

Definition at line 19 of file Marks.php.

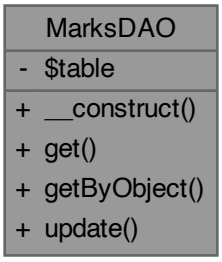
Referenced by __construct(), and getSubQuestionName().

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

- Marks.php

4.10 MarksDAO Class Reference

Collaboration diagram for MarksDAO:



Public Member Functions

- __construct ()
- get ($where=[])
- getByObject ($student, $subQuestion=null, $rubric=null)
- update ($marks)

Static Private Attributes

- static $table

4.10.1 Detailed Description

Class MarksDAO

Handles CRUD operations for student marks.

Definition at line 8 of file MarksDAO.php.

4.10.2 Member Function Documentation

__construct()

```
MarksDAO::__construct ()
```

MarksDAO constructor. Initializes the table name for marks.

Definition at line 16 of file MarksDAO.php.

```
00017     {
00018         $table = CFG::mkTableName("marks");
00019         self::$table = $table;
00020     }
```

References $table, and CFG::mkTableName().

Here is the call graph for this function:

get()

```
MarksDAO::get (
              $where = [])
```

Retrieves Marks records from the database based on given conditions.

Parameters

array	*$where*	Associative array of conditions for SQL WHERE clause.

Returns

Marks[] Array of Marks objects.

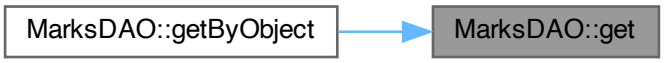
Definition at line 28 of file MarksDAO.php.

```
00029     {
00030         $table = self::$table;
00031         $rows = DB::$db->get(
00032             table: $table,
00033             where: $where
00034         );
00035         $marks = [];
00036         foreach ($rows as $row) {
00037             $marks[] = new Marks(
00038                 $row['student_email'],
00039                 $row['subquestion_name'],
00040                 $row['rubric_name'],
00041                 $row['marks']
00042             );
00043         }
00044         return $marks;
00045     }
```

References DB::$db, and $table.

Referenced by getByObject().

Here is the caller graph for this function:



getByObject()

```
MarksDAO::getByObject (
              $student,
              $subQuestion = null,
              $rubric = null)
```

Retrieves Marks records for a given student, optionally filtered by subquestion or rubric.

Parameters

Student	*$student*	Student object to filter marks.
SubQuestion | null	*$subQuestion*	Optional subquestion filter.
Rubric | null	*$rubric*	Optional rubric filter.

Returns

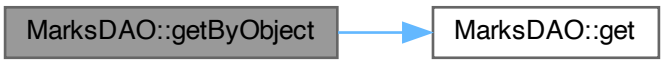
Marks[] Array of Marks objects.

Definition at line 55 of file MarksDAO.php.

```
00056     {
00057         $where = ['student_email' => $student->getEmail()];
00058         if ($subQuestion) {
00059             $where['subquestion_name'] = $subQuestion->getName();
00060         }
00061         if ($rubric) {
00062             $where['rubric_name'] = $rubric->getRubricName();
00063         }
00064         return $this->get($where);
00065     }
```

References get().

Here is the call graph for this function:



update()

```
MarksDAO::update (
              $marks)
```

Updates or inserts a Marks record in the database.

Parameters

Marks	*$marks*	Marks object to update or insert.

Returns

void

Definition at line 73 of file MarksDAO.php.

```
00074     {
00075         $table = self::$table;
00076         $studentEmail = $marks->getStudentEmail();
00077         $subQuestionName = $marks->getSubQuestionName();
00078         $rubricName = $marks->getRubricName();
00079         $studentMarks = $marks->getMarks();
00080         $columns = ['student_email', 'subquestion_name', 'rubric_name', 'marks'];
00081         $row = [$studentEmail, $subQuestionName, $rubricName, $studentMarks];
00082         // DB::$db->put($table, $columns, $row); // Optional insert
00083         DB::$db->update($table, $columns, $row); // Update if exists
00084     }
```

References DB::$db, and $table.

4.10.3 Member Data Documentation

$table

`MarksDAO::$table [static], [private]`

Definition at line 11 of file MarksDAO.php.

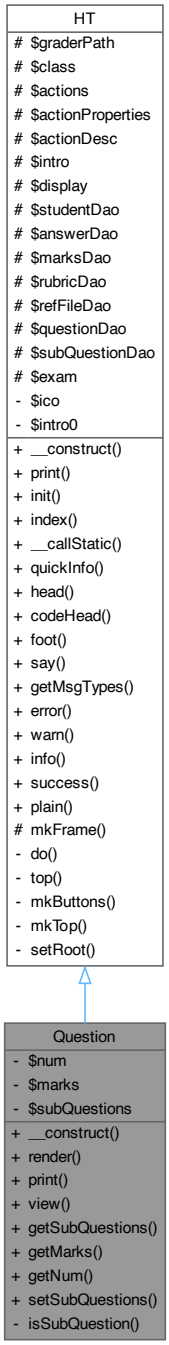
Referenced by __construct(), get(), and update().

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

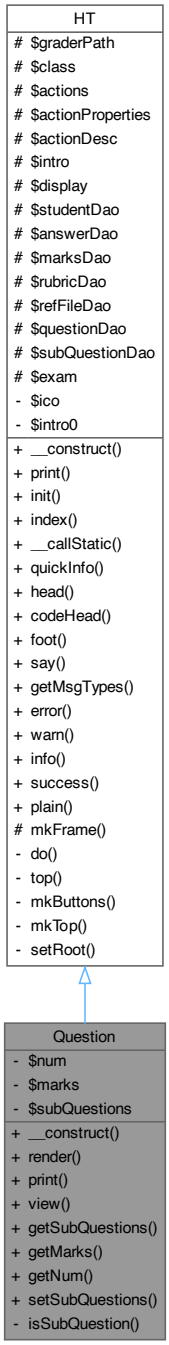
- MarksDAO.php

4.11 Question Class Reference

Inheritance diagram for Question:



Collaboration diagram for Question:



Public Member Functions

- __construct ($num, $marks)
- render ($student)
- print ($toStr=true)
- view ()
- getSubQuestions ()

- getMarks ()
- getNum ()
- setSubQuestions ($subQuestions)

Public Member Functions inherited from HT

- __construct ()
- print ($toStr=true)

Private Member Functions

- isSubQuestion ($sqName)

Private Attributes

- $num
- $marks = 0
- $subQuestions = [ ]

Additional Inherited Members

Static Public Member Functions inherited from HT

- static init ()
- static index ()
- static __callStatic ($method, $args)
- static quickInfo ($side)
- static head ($toStr=false, $title="Grader by Manoj", $refresh='')
- static codeHead ($title="Code Listing by Manoj")
- static foot ($toStr=false)
- static say ($what, $class='plain', $toStr=false, $showIcon=false)
- static getMsgTypes ()
- static error ($what, $toStr=false)
- static warn ($what, $toStr=false)
- static info ($what, $toStr=false)
- static success ($what, $toStr=false)
- static plain ($what, $toStr=false)

Static Protected Member Functions inherited from HT

- static mkFrame ($side='left')

Static Protected Attributes inherited from HT

- static $graderPath
- static $class = __CLASS__
- static $actions
- static $actionProperties = [ ]
- static $actionDesc
- static $intro
- static $display = ["class" => "success", "title" => ""]
- static $studentDao
- static $answerDao
- static $marksDao
- static $rubricDao
- static $refFileDao
- static $questionDao
- static $subQuestionDao
- static $exam

4.11.1 Detailed Description

Class Question

Represents a single question in the exam, containing multiple sub-questions. Each sub-question has associated marks and grading rubric.

Responsibilities:

- Manages question number and its sub-questions.
- Calculates total marks for the question.
- Renders sub-questions for a given student.
- Provides utility methods to access question details.

Inherits from the HT class for rendering HTML and messaging.

Definition at line 17 of file Question.php.

4.11.2 Member Function Documentation

__construct()

```
Question::__construct (
              $num,
              $marks)
```

Constructor for Question.

Parameters

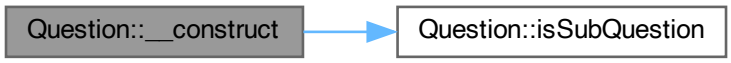
int	*$num*	Question number.
array | float	*$marks*	Array of sub-question names and their marks, or total marks.

Definition at line 40 of file Question.php.

```
00041     {
00042         $this->num = $num;
00043         if (is_array($marks)) {
00044             foreach ($marks as $subQuestionName => $subQuestionMarks) {
00045                 if ($this->isSubQuestion($subQuestionName)) {
00046                     $this->marks += $subQuestionMarks;
00047                     $this->subQuestions[$subQuestionName] =
00048                         new SubQuestion($subQuestionName, $subQuestionMarks, $num);
00049                 }
00050             }
00051         } else {
00052             $this->marks = $marks;
00053         }
00054     }
```

References $marks, $num, and isSubQuestion().

Here is the call graph for this function:



getMarks()

```
Question::getMarks ()
```

Get total marks for this question.

Returns

float Total marks.

Definition at line 124 of file Question.php.

```
00125     {
00126         return $this->marks;
00127     }
```

References $marks.

getNum()

```
Question::getNum ()
```

Get question number.

Returns

int Question number.

Definition at line 134 of file Question.php.

```
00135     {
00136         return $this->num;
00137     }
```

References $num.

getSubQuestions()

```
Question::getSubQuestions ()
```

Get all sub-questions associated with this question.

Returns

SubQuestion[] Array of SubQuestion instances.

Definition at line 114 of file Question.php.

```
00115     {
00116         return $this->subQuestions;
00117     }
```

References $subQuestions.

isSubQuestion()

```
Question::isSubQuestion (
              $sqName)  [private]
```

Helper to check if a given name corresponds to a valid sub-question of this question.

Parameters

string	*$sqName*	Sub-question name (e.g., q1a, q2b-1).

Returns

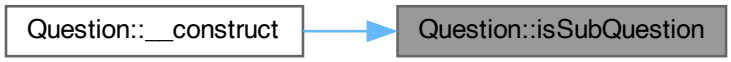
bool True if valid sub-question, false otherwise.

Definition at line 104 of file Question.php.

```
00105     {
00106         return stripos($sqName, "q$this->num") === 0;
00107     }
```

Referenced by __construct().

Here is the caller graph for this function:



print()

```
Question::print (
              $toStr = true)
```

Print a summary of the question and its sub-questions.

Parameters

bool	*$toStr*	If true, returns as string; otherwise, echoes.

Returns

string HTML summary of the question.

Definition at line 78 of file Question.php.

```
00079     {
00080         $str = "<h3> Question: $this->num,  Marks: $this->marks</h3>";
00081         foreach ($this->subQuestions as $subQuestion) {
00082             $str .= $subQuestion->print();
00083         }
00084         return $str;
00085     }
```

render()

```
Question::render (
               $student)
```

Render all sub-questions for a given student.

Parameters

Student	*$student*	The student whose answers are being displayed.

Returns

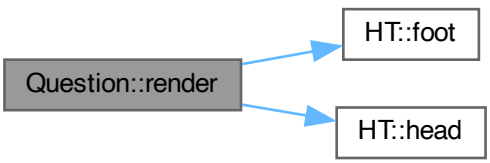
string Rendered HTML content of sub-questions.

Definition at line 62 of file Question.php.

```
00063     {
00064         $srcDoc = HT::head(true);
00065         foreach ($this->subQuestions as $subQuestion) {
00066             $srcDoc .= $subQuestion->render($student);
00067         }
00068         $srcDoc .= HT::foot(true);
00069         return $srcDoc;
00070     }
```

References HT::foot(), and HT::head().

Here is the call graph for this function:



setSubQuestions()

```
Question::setSubQuestions (
              $subQuestions)
```

Set sub-questions manually.

Parameters

SubQuestion[ ]	*$subQuestions*	Array of SubQuestion instances.

Returns

self Fluent setter for chaining.

Definition at line 145 of file Question.php.

```
00145                                                   : self
00146     {
00147         $this->subQuestions = $subQuestions;
00148         return $this;
00149     }
```

References $subQuestions.

view()

```
Question::view ()
```

Placeholder for future functionality to view question details. (Currently empty).

Returns

void

Definition at line 93 of file Question.php.

```
00094     {
00095         // Future implementation
00096     }
```

4.11.3 Member Data Documentation

$marks

```
Question::$marks = 0  [private]
```

Definition at line 27 of file Question.php.

Referenced by __construct(), and getMarks().

$num

```
Question::$num  [private]
```

Definition at line 22 of file Question.php.

Referenced by __construct(), and getNum().

$subQuestions

```
Question::$subQuestions = [ ]  [private]
```

Definition at line 32 of file Question.php.

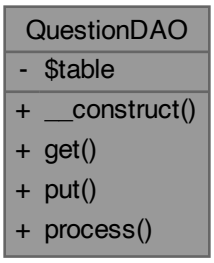
Referenced by getSubQuestions(), and setSubQuestions().

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

- Question.php

4.12 QuestionDAO Class Reference

Collaboration diagram for QuestionDAO:



Public Member Functions

- __construct ()
- get ($where=[ ], $verbose=false)
- put ($questions)
- process ()

Static Private Attributes

- static $table

4.12.1 Detailed Description

Class QuestionDAO

Handles CRUD operations for Questions, including parsing and loading from CSV.

Definition at line 8 of file QuestionDAO.php.

4.12.2 Member Function Documentation

__construct()

```
QuestionDAO::__construct ()
```

QuestionDAO constructor. Initializes the table name for questions.

Definition at line 16 of file QuestionDAO.php.

```
00017     {
00018         $table = CFG::mkTableName("questions");
00019         self::$table = $table;
00020     }
```

References $table, and CFG::mkTableName().

Here is the call graph for this function:

get()

```
QuestionDAO::get (
                $where = [],
                $verbose = false)
```

Retrieves Question records from the database based on given conditions. Filters out non-gradable questions.

Parameters

array	*$where*	Associative array of conditions for SQL WHERE clause.
bool	*$verbose*	Display error if no questions found when true.

Returns

Question[] Array of Question objects.

Definition at line 30 of file QuestionDAO.php.

```
00031     {
00032         $table = self::$table;
00033         $rows = DB::$db->get(
00034             table: $table,
00035             where: $where
00036         );
00037         $got = count($rows);
00038         if ($verbose && $got <= 0) { // Not found in the DB
00039             HT::error("<strong>AnswerDAO</strong>:
00040                 No answers in the DB!<br>
00041                 Run AnswerDAO->process(student) for each student first.");
00042             return;
00043         } else {
00044             $questions = [];
00045             foreach ($rows as $row) {
00046                 $questions[] = new Question(
00047                     $row['question_num'],
00048                     $row['marks']
00049                 );
00050             }
00051         }
00052         // Drop the questions not in the gradable list
00053         foreach ($questions as $key => $question) {
00054             if (!in_array($question->getNum(), CFG::$gradeQuestions)) {
00055                 unset($questions[$key]);
00056             }
00057         }
00058         return $questions;
00059     }
```

References DB::$db, CFG::$gradeQuestions, $table, and HT::error().

Here is the call graph for this function:

process()

`QuestionDAO::process ()`

Processes the CSV file to extract and store Question objects in the database.

Returns

void

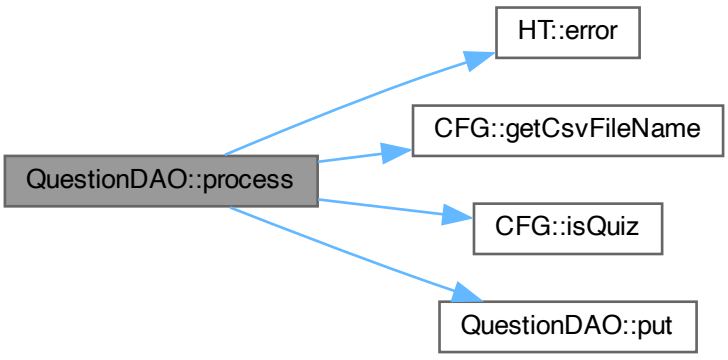
Definition at line 86 of file QuestionDAO.php.

```
00087     {
00088         if (CFG::isQuiz()) {
00089             // Done in Exam.php already.
00090             return;
00091         }
00092         // Section Names
```

```
00093         $secNames = CFG::$sectionNames;
00094         // Locations of subquestion number and marks in the CSV file header
00095         $locSubQ = CFG::$locSubQ;
00096         $locMarks = CFG::$locMarks;
00097         // From the header row of the given CSV file (eLearn grade export),
00098         // construct the question and its subquestions
00099         // $csv = "./CSV/IS113-LT1-{$secNames[0]}.csv";
00100         $csv = CFG::getCsvFileName($secNames[0]);
00101         if (!is_file($csv)) {
00102             HT::error("Error loading $csv");
00103             return;
00104         }
00105         $file = fopen($csv, 'r');
00106         $headers = fgetcsv($file);
00107         fclose($file);
00108         // Filter only the questions column headers
00109         foreach ($headers as $key => $value) {
00110             if (empty($value) || (preg_match('@[qQ].[A-Za-z]@', $value) == 0)) {
00111                 unset($headers[$key]);
00112             }
00113         }
00114         // Get the marks for each subquestion and question numbers
00115         $marks = $questionNum = [];
00116         foreach ($headers as $key => $value) {
00117             $tokens = explode(" ", $value);
00118             $subQuestionName = strtolower(trim($tokens[$locSubQ]));
00119             $marks[$subQuestionName] = explode(":", $tokens[$locMarks])[1];
00120             // Use preg_match() function to check match
00121             preg_match('!\d+!', $subQuestionName, $match);
00122             if (!in_array($match[0], $questionNum)) {
00123                 $questionNum[] = $match[0];
00124             }
00125         }
00126         // We now have question numbers, subquestion names and marks
00127         $questions = [];
00128         foreach ($questionNum as $num) {
00129             $questions[] = new Question($num, $marks);
00130         }
00131         $this->put($questions);
00132     }
```

References CFG::$locMarks, CFG::$locSubQ, CFG::$sectionNames, HT::error(), CFG::getCsvFileName(), CFG::isQuiz(), and put().

Here is the call graph for this function:



put()

```
QuestionDAO::put (
                $questions)
```

Inserts or updates Question records in the database.

Parameters

Question[]	*$questions*	Array of Question objects to be saved.

Returns

void

Definition at line 67 of file QuestionDAO.php.

```
00068     {
00069         $table = self::$table;
00070         $rows = [];
00071         foreach ($questions as $question) {
00072             $num = $question->getNum();
00073             $marks = $question->getMarks();
00074             $columns = ['question_num', 'marks'];
00075             $rows[] = [$num, $marks];
00076         }
00077         // DB::$db->put($table, $columns, $rows); // Uncomment to use insert
00078         DB::$db->update($table, $columns, $rows); // Update existing records
00079     }
```

References DB::$db, and $table.

Referenced by process().

Here is the caller graph for this function:

4.12.3 Member Data Documentation

$table

`QuestionDAO::$table [static], [private]`

Definition at line 11 of file QuestionDAO.php.

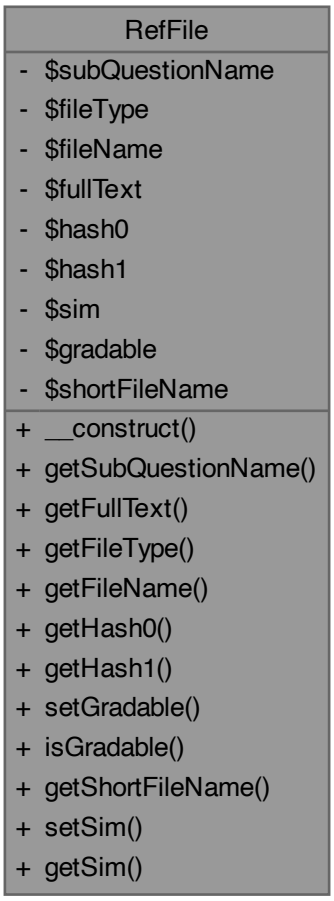
Referenced by __construct(), get(), and put().

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

- QuestionDAO.php

4.13 RefFile Class Reference

Collaboration diagram for RefFile:



Public Member Functions

- __construct ($subQuestionName, $fileType, $fileName, $fullText='', $hash0='', $hash1='')
- getSubQuestionName ()
- getFullText ()
- getFileType ()
- getFileName ()
- getHash0 ()
- getHash1 ()
- setGradable ($gradable)
- isGradable ()
- getShortFileName ()
- setSim ($sim)
- getSim ()

Private Attributes

- $subQuestionName
- $fileType
- $fileName
- $fullText
- $hash0
- $hash1
- $sim
- $gradable = false
- $shortFileName

4.13.1 Detailed Description

Class RefFile

Represents a reference file (resource or solution) associated with a sub-question. It contains metadata such as file type, hashes for similarity checking, and grading flags.

Definition at line 9 of file RefFile.php.

4.13.2 Member Function Documentation

__construct()

```
RefFile::__construct (
              $subQuestionName,
              $fileType,
              $fileName,
              $fullText = '',
              $hash0 = '',
              $hash1 = '')
```

RefFile constructor.

Parameters

string	*$subQuestionName*	Sub-question name.
string	*$fileType*	Type of file ('resources' or 'solutions').
string	*$fileName*	Full file path.
string	*$fullText*	Optional full file content (auto-loaded if empty).
string	*$hash0*	Optional primary hash (auto-generated if empty).
string	*$hash1*	Optional secondary hash (auto-generated if empty).

Definition at line 48 of file RefFile.php.

```
00055       {
00056         $this->subQuestionName = $subQuestionName;
00057         $this->fileType = $fileType;
00058         $this->fileName = $fileName;
00059         if (empty($fullText)) {
00060             $fullText = DB::$db->getFileContent($fileName);
00061         }
00062         $this->fullText = $fullText;
00063         if (empty($hash0)) {
```

```
00064             $hash0 = Report::hash($fileName);
00065         }
00066         $this->hash0 = $hash0;
00067         if (empty($hash1)) {
00068             $hash1 = Report::hash($fileName, true);
00069         }
00070         $this->hash1 = $hash1;
00071         $folders = [
00072             'resources' => CFG::$resourceFolder,
00073             'solutions' => CFG::$solutionFolder,
00074         ];
00075         $this->shortFileName = trim(
00076             str_ireplace($folders[$fileType], "", $fileName),
00077             '/'
00078         );
00079     }
```

References DB::$db, $fileName, $fileType, $fullText, $hash0, $hash1, CFG::$resourceFolder, CFG::$solutionFolder, and $subQuestionName.

getFileName()

```
RefFile::getFileName ()
```

Get the full path of the file.

Returns

string

Definition at line 116 of file RefFile.php.

```
00117     {
00118         return $this->fileName;
00119     }
```

References $fileName.

getFileType()

```
RefFile::getFileType ()
```

Get the type of the file ('resources' or 'solutions').

Returns

string

Definition at line 106 of file RefFile.php.

```
00107     {
00108         return $this->fileType;
00109     }
```

References $fileType.

getFullText()

```
RefFile::getFullText ()
```

Get the full text content of the file.

Returns

string

Definition at line 96 of file RefFile.php.

```
00097     {
00098         return $this->fullText;
00099     }
```

References $fullText.

getHash0()

```
RefFile::getHash0 ()
```

Get the primary hash value.

Returns

string

Definition at line 126 of file RefFile.php.

```
00127     {
00128         return $this->hash0;
00129     }
```

References $hash0.

getHash1()

```
RefFile::getHash1 ()
```

Get the secondary hash value.

Returns

string

Definition at line 136 of file RefFile.php.

```
00137     {
00138         return $this->hash1;
00139     }
```

References $hash1.

getShortFileName()

```
RefFile::getShortFileName ()
```

Get the shortened file name relative to its base folder.

Returns

string

Definition at line 168 of file RefFile.php.

```
00169     {
00170         return $this->shortFileName;
00171     }
```

References $shortFileName.

getSim()

```
RefFile::getSim ()
```

Get the similarity metric value.

Returns

mixed

Definition at line 190 of file RefFile.php.

```
00191     {
00192         return $this->sim;
00193     }
```

References $sim.

getSubQuestionName()

```
RefFile::getSubQuestionName ()
```

Get the sub-question name.

Returns

string

Definition at line 86 of file RefFile.php.

```
00087     {
00088         return $this->subQuestionName;
00089     }
```

References $subQuestionName.

isGradable()

```
RefFile::isGradable ()
```

Check if the file is gradable.

Returns

bool

Definition at line 158 of file RefFile.php.

```
00159     {
00160         return $this->gradable;
00161     }
```

References $gradable.

setGradable()

```
RefFile::setGradable (
              $gradable)
```

Set whether this file is gradable.

Parameters

bool	*$gradable*	

Returns

self

Definition at line 147 of file RefFile.php.

```
00147                                           : self
00148     {
00149         $this->gradable = $gradable;
00150         return $this;
00151     }
```

References $gradable.

setSim()

```
RefFile::setSim (
              $sim)
```

Set the similarity metric value.

Parameters

mixed	*$sim*	

Returns

self

Definition at line 179 of file RefFile.php.

```
00179                                 : self
00180     {
00181         $this->sim = $sim;
00182         return $this;
00183     }
```

References $sim.

4.13.3 Member Data Documentation

$fileName

```
RefFile::$fileName  [private]
```

Definition at line 18 of file RefFile.php.

Referenced by __construct(), and getFileName().

$fileType

```
RefFile::$fileType  [private]
```

Definition at line 15 of file RefFile.php.

Referenced by __construct(), and getFileType().

$fullText

```
RefFile::$fullText  [private]
```

Definition at line 21 of file RefFile.php.

Referenced by __construct(), and getFullText().

$gradable

```
RefFile::$gradable = false  [private]
```

Definition at line 33 of file RefFile.php.

Referenced by isGradable(), and setGradable().

$hash0

```
RefFile::$hash0  [private]
```

Definition at line 24 of file RefFile.php.

Referenced by __construct(), and getHash0().

$hash1

```
RefFile::$hash1  [private]
```

Definition at line 27 of file RefFile.php.

Referenced by __construct(), and getHash1().

$shortFileName

`RefFile::$shortFileName [private]`

Definition at line 36 of file RefFile.php.

Referenced by getShortFileName().

$sim

`RefFile::$sim [private]`

Definition at line 30 of file RefFile.php.

Referenced by getSim(), and setSim().

$subQuestionName

`RefFile::$subQuestionName [private]`

Definition at line 12 of file RefFile.php.

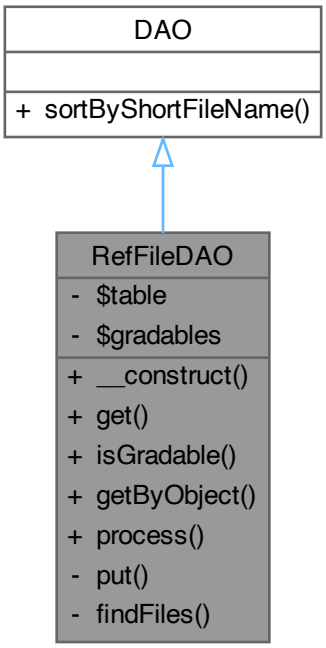
Referenced by __construct(), and getSubQuestionName().

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

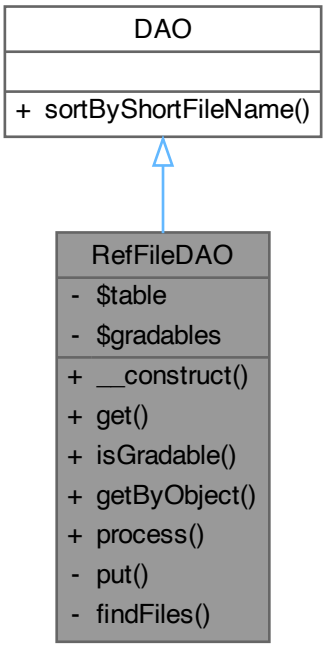
- RefFile.php

4.14 RefFileDAO Class Reference

Inheritance diagram for RefFileDAO:



Collaboration diagram for RefFileDAO:



Public Member Functions

- __construct ()
- get ($where=[ ], $verbose=false)
- isGradable ($shortFileName)
- getByObject ($subQuestion, $fileType='')
- process ($subQuestion)

Private Member Functions

- put ($subQuestion, $refFiles)
- findFiles ($subQuestion, $folder)

Static Private Attributes

- static $table
- static $gradables = [ ]

Additional Inherited Members

Static Public Member Functions inherited from DAO

- static sortByShortFileName ($objects)

4.14.1 Detailed Description

RefFileDAO handles CRUD operations and processing of reference files (solutions and resources) associated with sub-questions. It determines which files are gradable by comparing their similarity.

Definition at line 8 of file RefFileDAO.php.

4.14.2 Member Function Documentation

__construct()

```
RefFileDAO::__construct ()
```

Constructor initializes the table name with proper prefix.

Definition at line 19 of file RefFileDAO.php.

```
00020     {
00021         $table = CFG::mkTableName("ref_files");
00022         self::$table = $table;
00023     }
```

References $table, and CFG::mkTableName().

Here is the call graph for this function:

findFiles()

```
RefFileDAO::findFiles (
              $subQuestion,
              $folder)  [private]
```

Find files related to a sub-question in a specified folder.

Parameters

SubQuestion	*$subQuestion*	The sub-question object
string	*$folder*	Path to search for files

Returns

string[] List of file paths found

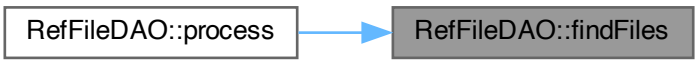
Definition at line 215 of file RefFileDAO.php.

References HT::error(), and get().

Referenced by process().

Here is the call graph for this function:

Here is the caller graph for this function:



get()

```
RefFileDAO::get (
              $where = [],
              $verbose = false)
```

Retrieve reference files from the database.

Parameters

array	*$where*	Optional WHERE clause key-value pairs
bool	*$verbose*	If true, shows error if no rows are found

Returns

RefFile[] List of RefFile objects

Definition at line 32 of file RefFileDAO.php.

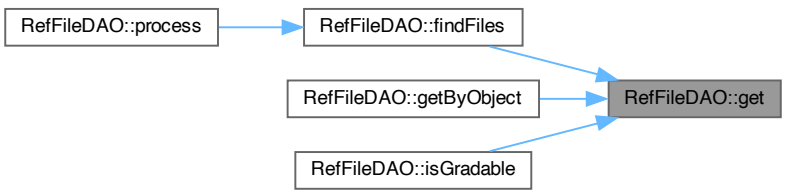
```
00033     {
00034         $table = self::$table;
00035         $rows = DB::$db->get(
00036             table: $table,
00037             where: $where
00038         );
00039         $got = count($rows);
00040         if ($verbose && $got <= 0) { // Not found in the DB
00041             HT::error("<strong>RefFileDAO</strong>:
00042                 No resources or solutions in the DB!<br>
00043                 Run RefFileDAO->process(student) for each student first.");
00044             return;
00045         } else {
00046             $refFiles = [];
00047             foreach ($rows as $row) {
00048                 $refFile = new RefFile(
00049                     $row['subquestion_name'],
00050                     $row['filetype'],
00051                     $row['filename'],
00052                     $row['fulltext'],
00053                     $row['hash0'],
00054                     $row['hash1']
00055                 );
00056                 $refFile->setSim($row['sim']);
00057                 $refFile->setGradable((bool) $row['gradable']);
00058                 $refFiles[] = $refFile;
00059             }
00060         }
00061         $refFiles = self::sortByShortFileName($refFiles);
00062         return $refFiles;
00063     }
```

References DB::$db, $table, and HT::error().

Referenced by findFiles(), getByObject(), and isGradable().

Here is the call graph for this function:

Here is the caller graph for this function:



getByObject()

```
RefFileDAO::getByObject (
              $subQuestion,
              $fileType = '')
```

Retrieve reference files based on a SubQuestion object and optional file type.

Parameters

SubQuestion	*$subQuestion*	The sub-question object
string	*$fileType*	Optional file type ('resources' or 'solutions')

Returns

RefFile[] List of RefFile objects

Definition at line 89 of file RefFileDAO.php.

```
00090     {
00091         $where = ['subquestion_name' => $subQuestion->getName()];
00092         if ($fileType) {
00093             $where['filetype'] = $fileType;
00094         }
00095         return $this->get($where);
00096     }
```

References get().

Here is the call graph for this function:

isGradable()

```
RefFileDAO::isGradable (
              $shortFileName)
```

Check if a given short file name is gradable.

Parameters

string	*$shortFileName*	

Returns

bool True if gradable, false otherwise

Definition at line 71 of file RefFileDAO.php.

```
00072     {
00073         if (empty(self::$gradables)) {
00074             $resources = $this->get(['filetype' => 'resources', 'gradable' => 1]);
00075             foreach ($resources as $resource) {
00076                 self::$gradables[] = $resource->getShortFileName();
00077             }
00078         }
00079         return in_array($shortFileName, self::$gradables);
00080     }
```

References get().

Here is the call graph for this function:

process()

```
RefFileDAO::process (
            $subQuestion)
```

Process solutions and resources for a sub-question, determine gradability based on similarity, and store them in the database.

Parameters

SubQuestion	*$subQuestion*	The sub-question object to process

Returns

void

Definition at line 155 of file RefFileDAO.php.

```
00156     {
00157         $subQuestionName = $subQuestion->getName();
00158         $folders = [
00159             'resources' => CFG::$resourceFolder,
00160             'solutions' => CFG::$solutionFolder,
00161         ];
00162         $solutions = $resources = $refFiles = $objects = [];
00163         $objects['resources'] = $objects['solutions'] = [];
00164         foreach ($folders as $fileType => $folder) {
00165             $$fileType = $this->findFiles($subQuestion, $folder);
00166             if (count($$fileType) <= 0) {
00167                 HT::error("<strong>SubQuestion $subQuestionName</strong>:
00168                  No $fileType found in $folder!");
00169             } else {
00170                 foreach ($$fileType as $fileName) {
00171                     if (is_file($fileName)) {
00172                         // $fileName = CFG::mkFileName($fileName);
```

```
00173                         $refFile = new RefFile($subQuestionName, $fileType, $fileName);
00174                         $refFiles[] = $refFile;
00175                         $objects[$fileType][] = $refFile;
00176                     }
00177                 }
00178             }
00179         }
00180         // We now have all the resource files ($resources) and solutions ($solutions)
00181         // for this question. Find the ones that are different as gradable
00182         foreach ($objects['resources'] as $resource) {
00183             $resourceShort = $resource->getShortFileName();
00184             foreach ($objects['solutions'] as $solution) {
00185                 $solutionShort = $solution->getShortFileName();
00186                 if ($solutionShort == $resourceShort) {
00187                     if ($resource->getHash0() == $solution->getHash0()) {
00188                         $sim = 100;
00189                     } else if ($resource->getHash1() == $solution->getHash1()) {
00190                         $sim = 100;
00191                     } else {
00192                         $sim = Report::similarity(
00193                             $resource->getFileName(),
00194                             $solution->getFileName()
00195                         );
00196                     }
00197                     $gradable = $sim < 99.9;
00198                     $resource->setSim($sim);
00199                     $solution->setSim($sim);
00200                     $resource->setGradable($gradable);
00201                     $solution->setGradable($gradable);
00202                 }
00203             }
00204         }
00205         $this->put($subQuestion, $refFiles);
00206     }
```

References CFG::$resourceFolder, CFG::$solutionFolder, HT::error(), findFiles(), and put().

Here is the call graph for this function:

put()

```
RefFileDAO::put (
              $subQuestion,
              $refFiles)  [private]
```

Store reference files for a sub-question into the database.

Parameters

SubQuestion	*$subQuestion*	The sub-question object
RefFile[]	*$refFiles*	Array of reference file objects to be stored

Returns

void

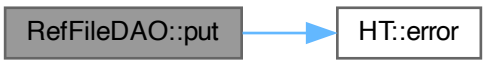
Definition at line 105 of file RefFileDAO.php.

```
00106     {
00107         $table = self::$table;
00108         $subQuestionName = $subQuestion->getName();
00109         if (count($refFiles) <= 0) { // No answers found. An error!
00110             HT::error("<strong>SubQuestion $subQuestionName</strong>:
00111             No solutions or resources found!");
00112         } else {
00113             $columns = [
00114                 'subquestion_name',
00115                 'filetype',
00116                 'filename',
00117                 'fulltext',
00118                 'hash0',
00119                 'hash1',
00120                 'sim',
00121                 'gradable'
00122             ];
00123             $rows = [];
00124             foreach ($refFiles as $refFile) {
00125                 $fileName = $refFile->getFileName();
00126                 $fileType = $refFile->getFileType();
00127                 $fullText = DB::$db->getFileContent($fileName);
00128                 $hash0 = $refFile->getHash0();
00129                 $hash1 = $refFile->getHash1();
00130                 $sim = (float) $refFile->getSim();
00131                 $gradable = (int) $refFile->isGradable();
00132                 $rows[] = [
00133                     $subQuestionName,
00134                     $fileType,
00135                     $fileName,
00136                     $fullText,
00137                     $hash0,
00138                     $hash1,
00139                     $sim,
00140                     $gradable
00141                 ];
00142             }
00143             // DB::$db->put($table, $columns, $rows);
00144             DB::$db->update($table, $columns, $rows);
00145         }
00146     }
```

References DB::$db, $table, and HT::error().

Referenced by process().

Here is the call graph for this function:



Here is the caller graph for this function:

4.14.3 Member Data Documentation

$gradables

```
RefFileDAO::$gradables = [] [static], [private]
```

Definition at line 14 of file RefFileDAO.php.

$table

```
RefFileDAO::$table [static], [private]
```

Definition at line 11 of file RefFileDAO.php.

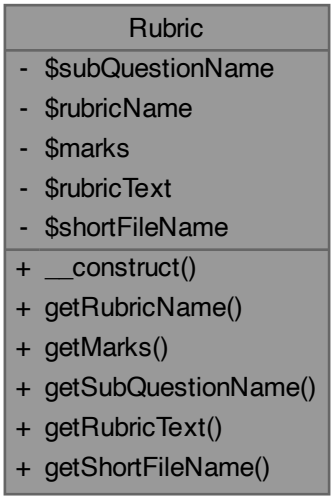
Referenced by __construct(), get(), and put().

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

- RefFileDAO.php

4.15 Rubric Class Reference

Collaboration diagram for Rubric:



Public Member Functions

- __construct ($subQuestionName, $rubricName, $marks, $rubricText, $shortFileName)
- getRubricName ()
- getMarks ()
- getSubQuestionName ()
- getRubricText ()
- getShortFileName ()

Private Attributes

- $subQuestionName
- $rubricName
- $marks
- $rubricText
- $shortFileName

4.15.1 Detailed Description

Class Rubric

Represents a rubric item associated with a sub-question. Each rubric defines a specific grading criterion, the marks associated, and the text description that guides the grading.

Definition at line 10 of file Rubric.php.

4.15.2 Member Function Documentation

__construct()

```
Rubric::__construct (
              $subQuestionName,
              $rubricName,
              $marks,
              $rubricText,
              $shortFileName)
```

Rubric constructor.

Parameters

string	*$subQuestionName*	Name of the sub-question this rubric belongs to.
string	*$rubricName*	Unique name/identifier of the rubric.
float	*$marks*	Marks awarded for this rubric.
string	*$rubricText*	Description of the rubric.
string	*$shortFileName*	Optional short file name associated with this rubric.

Definition at line 36 of file Rubric.php.

```
00042       {
00043         $this->subQuestionName = $subQuestionName;
00044         $this->rubricName = $rubricName;
00045         $this->marks = $marks;
00046         $this->rubricText = $rubricText;
00047         $this->shortFileName = $shortFileName;
00048     }
```

References $marks, $rubricName, $rubricText, $shortFileName, and $subQuestionName.

getMarks()

```
Rubric::getMarks ()
```

Get the marks awarded for this rubric.

Returns

float

Definition at line 65 of file Rubric.php.

```
00066     {
00067         return $this->marks;
00068     }
```

References $marks.

getRubricName()

```
Rubric::getRubricName ()
```

Get the rubric name (unique identifier).

Returns

string

Definition at line 55 of file Rubric.php.

```
00056     {
00057         return $this->rubricName;
00058     }
```

References $rubricName.

getRubricText()

```
Rubric::getRubricText ()
```

Get the descriptive text of the rubric.

Returns

string

Definition at line 85 of file Rubric.php.

```
00086     {
00087         return $this->rubricText;
00088     }
```

References $rubricText.

getShortFileName()

```
Rubric::getShortFileName ()
```

Get the short file name associated with this rubric (if any).

Returns

string

Definition at line 95 of file Rubric.php.

```
00096     {
00097         return $this->shortFileName;
00098     }
```

References $shortFileName.

getSubQuestionName()

```
Rubric::getSubQuestionName ()
```

Get the name of the associated sub-question.

Returns

string

Definition at line 75 of file Rubric.php.

```
00076     {
00077         return $this->subQuestionName;
00078     }
```

References $subQuestionName.

4.15.3 Member Data Documentation

$marks

```
Rubric::$marks  [private]
```

Definition at line 19 of file Rubric.php.

Referenced by __construct(), and getMarks().

$rubricName

```
Rubric::$rubricName  [private]
```

Definition at line 16 of file Rubric.php.

Referenced by __construct(), and getRubricName().

$rubricText

`Rubric::$rubricText [private]`

Definition at line 22 of file Rubric.php.

Referenced by __construct(), and getRubricText().

$shortFileName

`Rubric::$shortFileName [private]`

Definition at line 25 of file Rubric.php.

Referenced by __construct(), and getShortFileName().

$subQuestionName

`Rubric::$subQuestionName [private]`

Definition at line 13 of file Rubric.php.

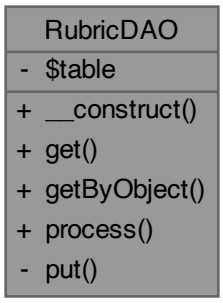
Referenced by __construct(), and getSubQuestionName().

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

- Rubric.php

4.16 RubricDAO Class Reference

Collaboration diagram for RubricDAO:



Public Member Functions

- __construct ()
- get ($where=[ ], $verbose=false)
- getByObject ($subQuestion)
- process ($subQuestion)

Private Member Functions

- put ($subQuestion, $rubrics)

Static Private Attributes

- static $table

4.16.1 Detailed Description

Class RubricDAO

Handles CRUD operations and processing of rubrics for each sub-question. Rubrics include grading criteria, marks, and links to reference files.

Definition at line 9 of file RubricDAO.php.

4.16.2 Member Function Documentation

__construct()

```
RubricDAO::__construct ()
```

Constructor initializes the table name with appropriate prefix.

Definition at line 17 of file RubricDAO.php.

```
00018     {
00019         $table = CFG::mkTableName("rubrics");
00020         self::$table = $table;
00021     }
```

References $table, and CFG::mkTableName().

Here is the call graph for this function:

get()

```
RubricDAO::get (
              $where = [],
              $verbose = false)
```

Retrieve rubrics from the database based on given conditions.

Parameters

array	*$where*	Optional key-value pairs for WHERE clause
bool	*$verbose*	If true, displays an error if no rubrics are found

Returns

Rubric[] List of Rubric objects

Definition at line 30 of file RubricDAO.php.

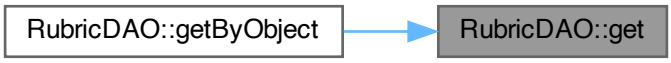
```
00031     {
00032         $rubrics = [];
00033         $table = self::$table;
00034         $rows = DB::$db->get(
00035             table: $table,
00036             where: $where
00037         );
00038         $got = count($rows);
00039         if ($verbose && $got <= 0) { // Not found in the DB
00040             HT::error("<strong>RubricDAO</strong>: No rubrics in the DB!<br>
00041             Run RubricDAO->process(subQuestion) for each subQuestion first.");
00042             return;
00043         } else {
00044             foreach ($rows as $row) {
00045                 $rubrics[] = new Rubric(
00046                     $row['subquestion_name'],
00047                     $row['rubric_name'],
00048                     $row['marks'],
00049                     $row['rubric_text'],
00050                     $row['short_filename'],
00051                 );
00052             }
00053         }
00054         return $rubrics;
00055     }
```

References DB::$db, $table, and HT::error().

Referenced by getByObject().

Here is the call graph for this function:

Here is the caller graph for this function:



getByObject()

```
RubricDAO::getByObject (
              $subQuestion)
```

Retrieve all rubrics associated with a specific sub-question.

Parameters

SubQuestion	*$subQuestion*	The sub-question object

Returns

Rubric[] List of Rubric objects linked to the sub-question

Definition at line 63 of file RubricDAO.php.

```
00064     {
00065         $where = ['subquestion_name' => $subQuestion->getName()];
00066         return $this->get($where);
00067     }
```

References get().

Here is the call graph for this function:

process()

```
RubricDAO::process (
              $subQuestion)
```

Process rubrics for a given sub-question by reading from the rubric CSV file. This method reads the CSV, filters relevant rubrics, and stores them in the database.

Parameters

SubQuestion	*$subQuestion*	The sub-question object to process rubrics for

Returns

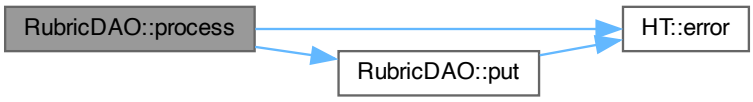
void

Definition at line 118 of file RubricDAO.php.

```
00119     {
00120         $subQuestionName = $subQuestion->getName();
00121         // Rubrics are kept in rubric.csv
00122         // name, marks, rubric text
00123         $rubricFile = glob(CFG::$rubricFile);
00124         $rubricNum = 1;
00125         if (count($rubricFile) != 1) {
00126             HT::error("<strong>SubQuestion $subQuestionName</strong>:
00127         Rubric File not found: " . CFG::$rubricFile);
00128         } else {
00129             $rubricFile = $rubricFile[0];
00130             $lines = file($rubricFile);
00131             $rubrics = [];
00132             foreach ($lines as $line) {
00133                 $lineElements = explode(",", $line);
00134                 $name = strtolower(trim($lineElements[0]));
00135                 if ($name == $subQuestionName) {
00136                     $marks = (float) $lineElements[1];
00137                     $rubricText = trim($lineElements[2]);
00138                     $rubricText = str_replace('""', "'", $rubricText);
00139                     $rubricText = trim($rubricText, '"');
00140                     $rubricText = str_replace("'", '"', $rubricText);
00141                     $shortFileName = trim($lineElements[3]);
00142                     $shortFileName = str_replace("//", '/', $shortFileName);
00143                     $rubricName = $name . "-" . $rubricNum++;
00144                     $rubrics[] = new Rubric(
00145                         $name,
00146                         $rubricName,
00147                         $marks,
00148                         $rubricText,
00149                         $shortFileName
00150                     );
00151                 }
00152             }
00153             $this->put($subQuestion, $rubrics);
00154         }
00155     }
```

References CFG::$rubricFile, HT::error(), and put().

Here is the call graph for this function:



put()

```
RubricDAO::put (
                $subQuestion,
                $rubrics)  [private]
```

Store rubric objects into the database for a given sub-question.

Parameters

SubQuestion	*$subQuestion*	The sub-question object
Rubric[]	*$rubrics*	Array of Rubric objects to be stored

Returns

void

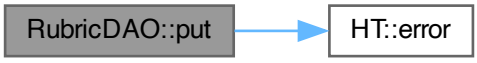
Definition at line 76 of file RubricDAO.php.

```
00077     {
00078         $table = self::$table;
00079         $subQuestionName = $subQuestion->getName();
00080         if (count($rubrics) <= 0) { // No rubrics found. An error!
00081             HT::error("<strong>SubQuestion $subQuestionName</strong>:
00082             No rubrics found. <br>
00083             Something very wrong!");
00084         } else {
00085             $columns = [
00086                 'subquestion_name',
00087                 'rubric_name',
00088                 'marks',
00089                 'rubric_text',
00090                 'short_filename'
00091             ];
00092             $rows = [];
00093             foreach ($rubrics as $rubric) {
00094                 $rubricName = $rubric->getRubricName();
00095                 $marks = $rubric->getMarks();
00096                 $rubricText = $rubric->getRubricText();
00097                 $shortFileName = $rubric->getShortFileName();
00098                 $rows[] = [
00099                     $subQuestionName,
00100                     $rubricName,
00101                     $marks,
00102                     $rubricText,
00103                     $shortFileName
00104                 ];
00105             }
00106             // DB::$db->put($table, $columns, $rows);
00107             DB::$db->update($table, $columns, $rows);
00108         }
00109     }
```

References DB::$db, $table, and HT::error().

Referenced by process().

Here is the call graph for this function:



Here is the caller graph for this function:

4.16.3 Member Data Documentation

$table

`RubricDAO::$table  [static], [private]`

Definition at line 12 of file RubricDAO.php.

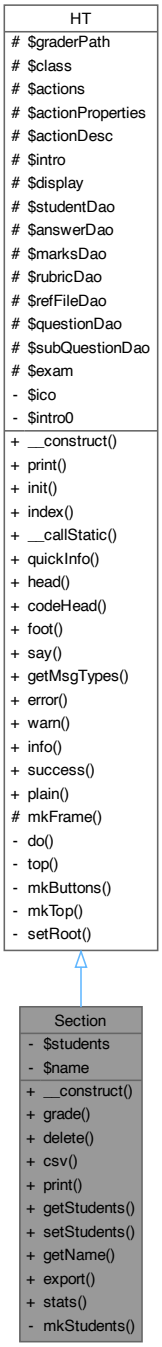
Referenced by __construct(), get(), and put().

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

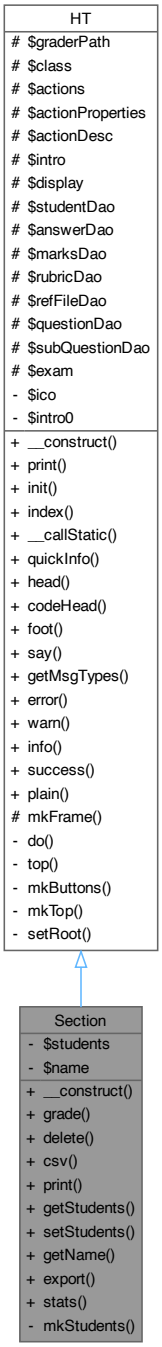
- RubricDAO.php

4.17 Section Class Reference

Inheritance diagram for Section:



Collaboration diagram for Section:



Public Member Functions

- __construct ($name)
- grade ($name='')
- delete ($name='')
- csv ()
- print ($toStr=true)

- getStudents ()
- setStudents ($students)
- getName ()
- export ()
- stats ()

Public Member Functions inherited from HT

- __construct ()
- print ($toStr=true)

Private Member Functions

- mkStudents ()

Private Attributes

- $students = [ ]
- $name

Additional Inherited Members

Static Public Member Functions inherited from HT

- static init ()
- static index ()
- static __callStatic ($method, $args)
- static quickInfo ($side)
- static head ($toStr=false, $title="Grader by Manoj", $refresh='')
- static codeHead ($title="Code Listing by Manoj")
- static foot ($toStr=false)
- static say ($what, $class='plain', $toStr=false, $showIcon=false)
- static getMsgTypes ()
- static error ($what, $toStr=false)
- static warn ($what, $toStr=false)
- static info ($what, $toStr=false)
- static success ($what, $toStr=false)
- static plain ($what, $toStr=false)

Static Protected Member Functions inherited from HT

- static mkFrame ($side='left')

Static Protected Attributes inherited from HT

- static $graderPath
- static $class = __CLASS__
- static $actions
- static $actionProperties = [ ]
- static $actionDesc
- static $intro
- static $display = ["class" => "success", "title" => ""]
- static $studentDao
- static $answerDao
- static $marksDao
- static $rubricDao
- static $refFileDao
- static $questionDao
- static $subQuestionDao
- static $exam

4.17.1 Detailed Description

Class Section Represents a section of students in an exam or lab test, handling their grading.

Inherits from the HT class for rendering HTML and messaging.

Definition at line 9 of file Section.php.

4.17.2 Member Function Documentation

__construct()

```
Section::__construct (
              $name)
```

Section constructor. Retrieves students from the database or CSV file.

Parameters

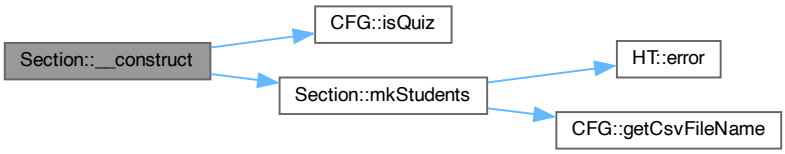
string	*$name*	The name of the section

Definition at line 27 of file Section.php.

```
00028     {
00029         $this->name = $name;
00030         $this->students = self::$studentDao->get(["section" => $name]);
00031 
00032         if (empty($this->students)) { // Make students from the CSV files
00033             if (!CFG::isQuiz()) { // Add students only for lab tests
00034                 $this->mkStudents();
00035             }
00036         }
00037     }
```

References $name, CFG::isQuiz(), and mkStudents().

Here is the call graph for this function:



csv()

```
Section::csv ()
```

Displays student information as a CSV-styled HTML table.

Returns

void

Definition at line 91 of file Section.php.

```
00092     {
00093         $str = "<table class='grader' width='98%'><tr><th colspan='100%'>Section:
      $this->name</th></tr>";
00094         foreach ($this->students as $student) {
00095             $str .= $student->csv();
00096         }
00097         $str .= "</table>";
00098         HT::warn($str);
00099     }
```

References HT::warn().

Here is the call graph for this function:

delete()

```
Section::delete (
              $name = '')
```

Deletes all students' data in the section.

Parameters

string	*$name*	(Optional) Not used in the function

Returns

void

Definition at line 79 of file Section.php.

```
00080     {
00081         foreach ($this->students as $student) {
00082             $student->delete();
00083         }
00084     }
```

References $name.

export()

```
Section::export ()
```

Exports students' marks to a CSV file for eLearn.

Returns

void

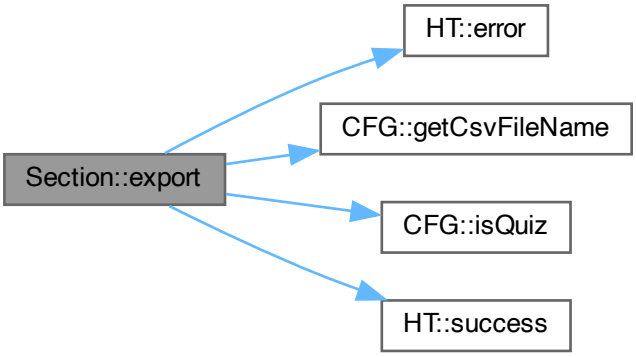
Definition at line 153 of file Section.php.

```
00154     {
00155         $csv = CFG::getCsvFileName($this->name);
00156         if (!is_file($csv)) {
00157             HT::error("Error loading $csv");
00158             return;
00159         }
00160         $exported = dirname($csv) . "/Export-" . basename($csv);
00161         $fileIn = fopen($csv, 'r');
00162         $fileOut = fopen($exported, 'w');
00163         if ($fileOut === false) {
00164             HT::error("Error opening the file $exported");
00165         }
00166 
00167         if (CFG::isQuiz()) {
00168             $headers = ["email", "Username"];
00169             $qNums = [];
00170             foreach (CFG::$autoGradables as $file => $autoGrade) {
00171                 $qNums[] = $autoGrade['subQuestionName'];
00172             }
00173             foreach ($qNums as $qNum) {
00174                 $headers[] = $qNum;
00175             }
00176         } else {
00177             $headers = fgetcsv($fileIn); // copy the header
00178         }
00179         fputcsv($fileOut, $headers);
00180 
00181         foreach ($this->students as $student) {
00182             $line = [];
00183             $line[0] = "#" . $student->getEmail();
00184             $line[1] = $student->getName();
00185             $marks = $student->sumMarksBySubQuestion();
00186 
00187             foreach ($headers as $iCol => $header) {
00188                 if ($iCol < 2) {
00189                     continue;
00190                 }
00191                 $line[$iCol] = "";
00192                 foreach ($marks as $subQuestionName => $m) {
00193                     if (stripos($header, $subQuestionName) !== false) {
00194                         $line[$iCol] = $m;
```

```
00195                         break;
00196                     }
00197                 }
00198             }
00199
00200             if (CFG::isQuiz()) {
00201                 fputcsv($fileOut, $line);
00202             } else {
00203                 // Drop the last element (a spurious blank) in line
00204                 unset($line[$iCol]);
00205                 fputcsv($fileOut, $line, ",", "\"", "\\", ",#\n");
00206             }
00207         }
00208
00209         fclose($fileOut);
00210         $fullText = file_get_contents($exported);
00211         // Remove quotation marks - eLearn doesn't like them
00212         $fullText = str_replace("\"", "", $fullText);
00213         file_put_contents($exported, $fullText);
00214         $exported = realpath($exported);
00215         HT::success("Exported $this->name marks to <code>$exported</code>");
00216     }
```

References CFG::$autoGradables, HT::error(), CFG::getCsvFileName(), CFG::isQuiz(), and HT::success().

Here is the call graph for this function:



getName()

```
Section::getName ()
```

Gets the name of the section.

Returns

string The section name

Definition at line 143 of file Section.php.

```
00144     {
00145         return $this->name;
00146     }
```

References $name.

getStudents()

```
Section::getStudents ()
```

Gets the list of students in the section.

Returns

array List of Student objects

Definition at line 121 of file Section.php.

```
00122     {
00123         return $this->students;
00124     }
```

References $students.

grade()

```
Section::grade (
              $name = '')
```

Grades all students in the section.

Parameters

string	*$name*	(Optional) Not used in the function

Returns

void

Definition at line 65 of file Section.php.

```
00066     {
00067         HT::say("<h4>Section: $this->name</h4>");
00068         foreach ($this->students as $student) {
00069             $student->grade();
00070         }
00071     }
```

References $name, and HT::say().

Here is the call graph for this function:

mkStudents()

```
Section::mkStudents ()  [private]
```

Reads students from the CSV file and prepares their solutions from their eLearn submissions.

Returns

void

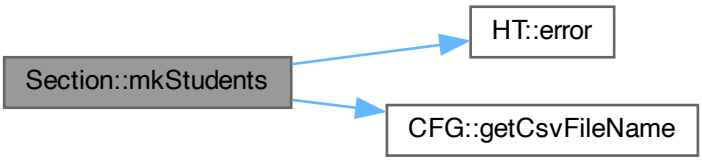
Definition at line 44 of file Section.php.

```
00045     {
00046         $csv = CFG::getCsvFileName($this->name);
00047         if (!is_file($csv)) {
00048             HT::error("Error loading $csv");
00049             return;
00050         }
00051         $file = fopen($csv, 'r');
00052         $line = fgetcsv($file); // Skip the header
00053         while (($line = fgetcsv($file)) !== false) {
00054             $this->students[] = new Student($line[0], $line[1], $this->name);
00055         }
00056         fclose($file);
00057     }
```

References HT::error(), and CFG::getCsvFileName().

Referenced by __construct().

Here is the call graph for this function:



Here is the caller graph for this function:

print()

```
Section::print (
              $toStr = true)
```

Prints section information, including all students.

Parameters

bool	*$toStr*	Whether to return the output as a string

Returns

string|null The formatted output if $toStr is true, otherwise null

Definition at line 107 of file Section.php.

```
00108     {
00109         $str = $this->info("<strong> Section: $this->name</strong>", $toStr);
00110         foreach ($this->students as $std) {
00111             $str .= $std->print();
00112         }
00113         return $str;
00114     }
```

References HT::info().

Here is the call graph for this function:

setStudents()

```
Section::setStudents (
              $students)
```

Sets the list of students in the section.

Parameters

array	*$students*	The students to assign to this section

Returns

self

Definition at line 132 of file Section.php.

```
00132                                         : self
00133     {
00134         $this->students = $students;
00135         return $this;
00136     }
```

References $students.

stats()

```
Section::stats ()
```

Generates statistics for the section. (To be implemented)

Returns

void

Definition at line 223 of file Section.php.

```
00224     {
00225         foreach ($this->students as $student) {
00226         }
00227         HT::success("Statistics for $this->name.<br>To be implemented.");
00228     }
```

References HT::success().

Here is the call graph for this function:

4.17.3 Member Data Documentation

$name

```
Section::$name  [private]
```

Definition at line 19 of file Section.php.

Referenced by __construct(), delete(), getName(), and grade().

$students

```
Section::$students = [ ]  [private]
```

Definition at line 14 of file Section.php.

Referenced by getStudents(), and setStudents().

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

- Section.php

4.18 Stat Class Reference

Inheritance diagram for Stat:

Collaboration diagram for Stat:

Static Public Member Functions

- static mistakes ()
- static mean ($data)
- static median ($data)
- static range ($data)
- static var ($data)

- static varp ($data)
- static stdev ($data)
- static stdevp ($data)
- static simple_regression ($x, $y)

Static Public Member Functions inherited from HT

- static init ()
- static index ()
- static __callStatic ($method, $args)
- static quickInfo ($side)
- static head ($toStr=false, $title="Grader by Manoj", $refresh='')
- static codeHead ($title="Code Listing by Manoj")
- static foot ($toStr=false)
- static say ($what, $class='plain', $toStr=false, $showIcon=false)
- static getMsgTypes ()
- static error ($what, $toStr=false)
- static warn ($what, $toStr=false)
- static info ($what, $toStr=false)
- static success ($what, $toStr=false)
- static plain ($what, $toStr=false)

Static Protected Member Functions

- static getRows ($reduce=["section"=> "s.section"])
- static printStats ($grouped, $byTitle="Grouped by", $toString=false, $drillDown='')
- static printStats2 ($grouped2, $byTitle1, $byTitle2)
- static all ()
- static sections ()
- static questions ()
- static questions2 ()
- static subquestions ()
- static subquestions2 ()

Static Protected Member Functions inherited from HT

- static mkFrame ($side='left')

Static Protected Attributes

- static $actions
- static $actionProperties
- static $actionDesc
- static $class = __CLASS__
- static $intro
- static $display

Static Protected Attributes inherited from HT

- static $graderPath
- static $class = __CLASS__
- static $actions
- static $actionProperties = [ ]
- static $actionDesc
- static $intro
- static $display = ["class" => "success", "title" => ""]
- static $studentDao
- static $answerDao
- static $marksDao
- static $rubricDao
- static $refFileDao
- static $questionDao
- static $subQuestionDao
- static $exam

Static Private Member Functions

- static groupBy ($rows, $by='')
- static groupBy2 ($rows, $by1, $by2)
- static mkSql ($reduce)
- static count ($data, $value, $tolerance=0.01)

Additional Inherited Members

Public Member Functions inherited from HT

- __construct ()
- print ($toStr=true)

4.18.1 Detailed Description

Class Stat

Handles statistics generation for student marks, including means, medians, standard deviations, and analysis of mistakes. Provides various ways to group and visualize statistics.

Inherits from the HT class for rendering HTML and messaging.

Definition at line 12 of file Stat.php.

4.18.2 Member Function Documentation

all()

```
static Stat::all ()  [static], [protected]
```

Displays overall statistics.

Definition at line 214 of file Stat.php.

```
00215     {
00216         $rows = self::getRows();
00217         $grouped = self::groupBy($rows);
00218         self::printStats($grouped);
00219     }
```

count()

```
static Stat::count (
              $data,
              $value,
              $tolerance = 0.01)  [static], [private]
```

Counts occurrences of a value within tolerance.

Parameters

array	*$data*	Data array.
float	*$value*	Value to count.
float	*$tolerance*	Tolerance for comparison.

Returns

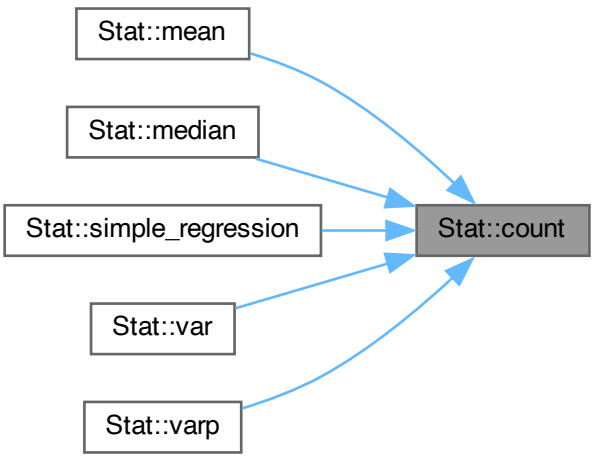
int Count of matching values.

Definition at line 343 of file Stat.php.

```
00344     {
00345         $count = 0;
00346         foreach ($data as $val) {
00347             if (abs($val - $value) <= $tolerance) {
00348                 $count++;
00349             }
00350         }
00351         return $count;
00352     }
```

Referenced by mean(), median(), simple_regression(), var(), and varp().

Here is the caller graph for this function:



getRows()

```
static Stat::getRows (
              $reduce = ["section" => "s.section"])  [static], [protected]
```

Fetches rows from the database with optional reduction/grouping.

Parameters

array	*$reduce*	Associative array for grouping (e.g., section, question).

Returns

array Resulting data rows.

Definition at line 73 of file Stat.php.

```
00074     {
00075         $rows = DB::$db->query(self::mkSql($reduce))->fetchAll();
00076         return $rows;
00077     }
```

References DB::$db.

groupBy()

```
static Stat::groupBy (
              $rows,
              $by = '')  [static], [private]
```

Groups data by a specific key.

Parameters

array	*$rows*	Data rows.
string	*$by*	Grouping key.

Returns

array Grouped data.

Definition at line 170 of file Stat.php.

```
00171     {
00172         $grouped = [];
00173         foreach ($rows as $row) {
00174             if (empty($by)) {
00175                 $key = "All";
00176             } else {
00177                 $key = $row[$by];
00178             }
00179             $grouped[$key][] = (float) $row['marks'];
00180         }
00181         ksort($grouped);
00182         return $grouped;
00183     }
```

groupBy2()

```
static Stat::groupBy2 (
              $rows,
              $by1,
              $by2)  [static], [private]
```

Groups data by two keys (two-dimensional grouping).

Parameters

array	*$rows*	Data rows.
string	*$by1*	First grouping key.
string	*$by2*	Second grouping key.

Returns

array Two-level grouped data.

Definition at line 193 of file Stat.php.

```
00194     {
00195         $grouped = [];
00196         foreach ($rows as $row) {
00197             $key1 = $row[$by1];
00198             $key2 = $row[$by2];
00199             if (empty($grouped[$key1][$key2])) {
00200                 $grouped[$key1][$key2] = [];
00201             }
00202             $grouped[$key1][$key2][] = (float) $row['marks'];
00203         }
00204         ksort($grouped);
00205         foreach ($grouped as $key => $bySubQuestion) {
00206             ksort($grouped[$key]);
00207         }
00208         return $grouped;
00209     }
```

mean()

```
static Stat::mean (
              $data)  [static]
```

Calculates mean of data.

Parameters

array	*$data*	Dataset.

Returns

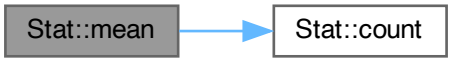
float Mean value.

Definition at line 360 of file Stat.php.

```
00361     {
00362         $n = count($data);
00363         if ($n == 0) {
00364             $mean = 0;
00365         } else {
00366             $mean = array_sum($data) / $n;
00367         }
00368         return $mean;
00369     }
```

References count().

Here is the call graph for this function:



median()

```
static Stat::median (
              $data)  [static]
```

Calculates median of data.

Parameters

array	*$data*	Dataset.

Returns

float Median value.

Definition at line 377 of file Stat.php.

```
00378     {
00379         sort($data);
00380         $elements = count($data);
00381         if (($elements % 2) == 0) {
00382             $i = $elements / 2;
00383             return (($data[$i - 1] + $data[$i]) / 2);
00384         } else {
00385             $i = ($elements - 1) / 2;
00386             return $data[$i];
00387         }
00388     }
```

References count().

Here is the call graph for this function:

mistakes()

```
static Stat::mistakes ()  [static]
```

Lists common mistakes (comments) submitted by students.

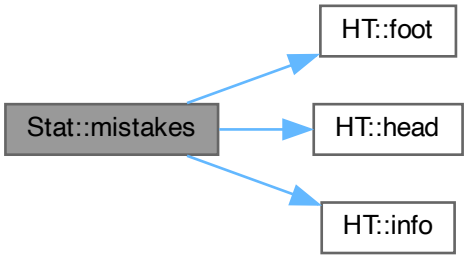
Definition at line 309 of file Stat.php.

```
00310     {
00311         $sql = "SELECT
00312             subquestion_name, comment
00313         FROM
00314             `{dbPrefix}answers`
00315         WHERE
00316             comment <> ''
00317         ORDER BY
00318             subquestion_name";
00319         $sql = str_replace('{dbPrefix}', CFG::$dbPrefix, $sql);
00320         $rows = DB::$db->query($sql)->fetchAll();
```

```
00321         $table = "<table width='80%' align='center' class='warn grader'>
00322                 <tr><th width='20%'>Sub Question Number</th>
00323                 <th width='80%'>Comment</th></tr>";
00324         foreach ($rows as $row) {
00325             $comment = "<pre>" . htmlentities($row["comment"]) . "</pre>";
00326             $table .= "<tr><td>{$row['subquestion_name']}</td>
00327             <td style='text-align:left'>$comment</td></tr>";
00328         }
00329         $table .= "</table><br>";
00330         HT::head();
00331         HT::info($table);
00332         HT::foot();
00333     }
```

References DB::$db, CFG::$dbPrefix, $sql, HT::foot(), HT::head(), and HT::info().

Here is the call graph for this function:



mkSql()

```
static Stat::mkSql (
              $reduce)  [static], [private]
```

Generates SQL query based on reduction fields.

Parameters

array	*$reduce*	Associative array of fields for grouping.

Returns

string SQL query.

Definition at line 283 of file Stat.php.

```
00284     {
00285         $select = $groupBy = "";
00286         foreach ($reduce as $alias => $field) {
00287             $select .= "$field AS '$alias',";
00288             $groupBy .= "`$alias`, ";
00289         }
00290         $groupBy = trim($groupBy, ", ");
00291         $sql = "SELECT
00292             s.student_email AS 'email',
00293             s.student_name AS 'name',
00294             $select
```

```
00295             SUM(m.marks) AS 'marks'
00296         FROM
00297             `{dbPrefix}students` AS s
00298             INNER JOIN `{dbPrefix}marks` AS m ON m.student_email = s.student_email
00299             INNER JOIN `{dbPrefix}subquestions` AS sq ON m.subquestion_name = sq.subquestion_name
00300         GROUP BY
00301            `email`, `name`, $groupBy";
00302         $sql = str_replace('{dbPrefix}', CFG::$dbPrefix, $sql);
00303         return $sql;
00304     }
```

References CFG::$dbPrefix, and $sql.

printStats()

```
static Stat::printStats (
              $grouped,
              $byTitle = "Grouped by",
              $toString = false,
              $drillDown = '')  [static], [protected]
```

Displays grouped statistics in a formatted table.

Parameters

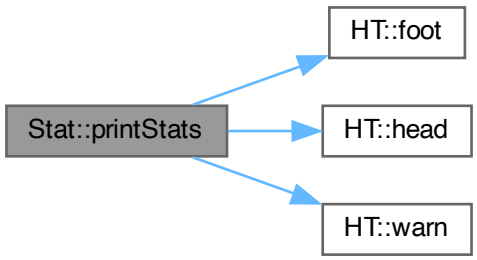
array	*$grouped*	Grouped data.
string	*$byTitle*	Title of the grouping.
bool	*$toString*	Return as string if true.
string	*$drillDown*	Optional drill-down action.

Definition at line 87 of file Stat.php.

```
00092       {
00093         $rows = "";
00094         foreach ($grouped as $by => $marks) {
00095             $rows .= "<tr><th>$by</th>";
00096             $mean = number_format(self::mean($marks), 2);
00097             $std = number_format(self::stdev($marks), 2);
00098             $median = number_format(self::median($marks), 2);
00099             $min = min($marks);
00100             $minCount = self::count($marks, $min);
00101             $max = max($marks);
00102             $maxCount = self::count($marks, $max);
00103             $min = number_format($min, 2) . " <small>[$minCount]</small>";
00104             $max = number_format($max, 2) . " <small>[$maxCount]</small>";
00105             $rows .= "<td>$mean</td>
00106             <td>$std</td>
00107             <td>$median</td>
00108             <td>$min</td>
00109             <td>$max</td></tr>";
00110         }
00111         if ($toString) {
00112             return $rows;
00113         }
00114         $rows .= "<tr><td colspan='10'><small>[count in brackets]</small></td></tr>";
00115         if ($drillDown) {
00116             $rows .= "<tr><th colspan='100%'>
00117             <form method='post' target='right'>
00118             <input type='hidden' name='do' value='$drillDown'>
00119             <button class='info' type='submit'>Drill-down by Section</button>
00120             </form></th></tr>";
00121         }
00122         $str = "<table class='grader success' align='center'>
00123         <tr><th>$byTitle</th><th>Mean</th><th>Std Dev</th>
00124         <th>Median</th><th>Min</th><th>Max</th></tr>
00125         $rows
00126         </table>";
00127         HT::head();
00128         HT::warn($str);
00129         HT::foot();
00130     }
```

References HT::foot(), HT::head(), and HT::warn().

Here is the call graph for this function:



printStats2()

```
static Stat::printStats2 (
               $grouped2,
               $byTitle1,
               $byTitle2)  [static], [protected]
```

Displays two-level grouped statistics in a formatted table.

Parameters

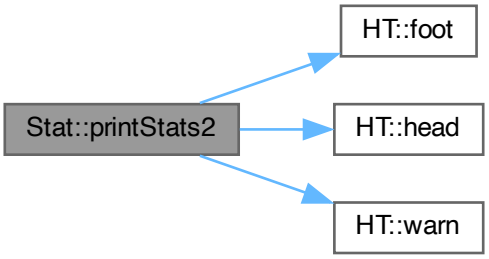
array	*$grouped2*	Two-level grouped data.
string	*$byTitle1*	Primary grouping title.
string	*$byTitle2*	Secondary grouping title.

Definition at line 139 of file Stat.php.

```
00140     {
00141         $rowsGrouped = [];
00142         foreach ($grouped2 as $key => $grouped) {
00143             $rows = self::printStats($grouped, $byTitle2, true);
00144             $rowSpan = substr_count($rows, "<tr>");
00145             $rowsGrouped[] = preg_replace(
00146                 "/<tr>/",
00147                 "<tr><th rowspan='$rowSpan'>$key</th>",
00148                 $rows,
00149                 1
00150             );
00151         }
00152 
00153         $str = "<table class='grader success' align='center'>
00154         <tr><th>$byTitle1</th><th>$byTitle2</th><th>Mean</th><th>Std Dev</th>
00155         <th>Median</th><th>Min</th><th>Max</th></tr>" .
00156             implode("\n", $rowsGrouped) .
00157             "<tr><td colspan='10'><small>[count in brackets]</small></td></tr>" .
00158             "</table>";
00159         HT::head();
00160         HT::warn($str);
00161         HT::foot();
00162     }
```

References HT::foot(), HT::head(), and HT::warn().

Here is the call graph for this function:



questions()

```
static Stat::questions ()  [static], [protected]
```

Displays statistics by question.

Definition at line 234 of file Stat.php.

```
00235     {
00236         $rows = self::getRows(['question' => 'sq.question_num']);
00237         $grouped = self::groupBy($rows, 'question');
00238         self::printStats($grouped, 'Question', false, "questions2");
00239     }
```

questions2()

```
static Stat::questions2 ()  [static], [protected]
```

Drill-down: Statistics by question and section.

Definition at line 244 of file Stat.php.

```
00245     {
00246         $rows = self::getRows([
00247             'section' => 's.section',
00248             'question' => 'sq.question_num'
00249         ]);
00250         $grouped2 = self::groupBy2($rows, 'question', 'section');
00251         self::printStats2($grouped2, 'Question', 'Section');
00252     }
```

range()

```
static Stat::range (
              $data)  [static]
```

Calculates range (max - min) of data.

Parameters

array	*$data*	Dataset.

Returns

float Range value.

Definition at line 396 of file Stat.php.

```
00397     {
00398         return (max($data) - min($data));
00399     }
```

sections()

```
static Stat::sections ()  [static], [protected]
```

Displays statistics by section.

Definition at line 224 of file Stat.php.

```
00225     {
00226         $rows = self::getRows();
00227         $grouped = self::groupBy($rows, 'section');
00228         self::printStats($grouped, 'Section');
00229     }
```

simple_regression()

```
static Stat::simple_regression (
               $x,
               $y)  [static]
```

Runs simple linear regression on two datasets.

Parameters

array	*$x*	Independent variable dataset.
array	*$y*	Dependent variable dataset.

Returns

array Associative array containing regression results:

- a: Intercept
- b: Slope
- s: Standard error of estimate
- r: Correlation coefficient
- r2: Coefficient of determination
- cov: Covariance
- t: t-statistic

Definition at line 481 of file Stat.php.

```
00482     {
00483         $output = array();
00484         $output['a'] = 0;
00485         $n = min(count($x), count($y));
00486         $mean_x = self::mean($x);
00487         $mean_y = self::mean($y);
00488         $SS_x = 0;
00489         foreach ($x as $element) {
00490             $SS_x += pow(($element - $mean_x), 2);
00491         }
00492         $SS_y = 0;
00493         foreach ($y as $element) {
00494             $SS_y += pow(($element - $mean_y), 2);
00495         }
00496         $SS_xy = 0;
00497         for ($i = 0; $i < $n; $i++) {
00498             $SS_xy += ($x[$i] - $mean_x) * ($y[$i] - $mean_y);
00499         }
00500         $output['b'] = $SS_xy / $SS_x;
00501         $output['a'] = $mean_y - $output['b'] * $mean_x;
00502         $output['s'] = sqrt(($SS_y - $output['b'] * $SS_xy) / ($n - 2));
00503         $output['r'] = $SS_xy / sqrt($SS_x * $SS_y);
00504         $output['r2'] = pow($output['r'], 2);
00505         $output['cov'] = $SS_xy / ($n - 1);
00506         $output['t'] = $output['r'] / sqrt((1 - $output['r2']) / ($n - 2));
00507 
00508         return $output;
00509     }
```

References count().

Here is the call graph for this function:

stdev()

```
static Stat::stdev (
              $data)  [static]
```

Calculates sample standard deviation of data.

Parameters

array	*$data*	Dataset.

Returns

float Sample standard deviation.

Definition at line 451 of file Stat.php.

```
00452     {
00453         return sqrt(self::var($data));
00454     }
```

stdevp()

```
static Stat::stdevp (
              $data)  [static]
```

Calculates population standard deviation of data.

Parameters

array	*$data*	Dataset.

Returns

float Population standard deviation.

Definition at line 462 of file Stat.php.

```
00463     {
00464         return sqrt(self::varp($data));
00465     }
```

subquestions()

```
static Stat::subquestions ()  [static], [protected]
```

Displays statistics by subquestion.

Definition at line 257 of file Stat.php.

```
00258     {
00259         $rows = self::getRows(['subquestion' => 'sq.subquestion_name']);
00260         $grouped = self::groupBy($rows, 'subquestion');
00261         self::printStats($grouped, 'Subquestion', false, "subquestions2");
00262     }
```

subquestions2()

```
static Stat::subquestions2 ()  [static], [protected]
```

Drill-down: Statistics by subquestion and section.

Definition at line 267 of file Stat.php.

```
00268     {
00269         $rows = self::getRows([
00270             'section' => 's.section',
00271             'subquestion' => 'sq.subquestion_name'
00272         ]);
00273         $grouped2 = self::groupBy2($rows, 'subquestion', 'section');
00274         self::printStats2($grouped2, 'Subquestion', 'Section');
00275     }
```

var()

```
static Stat::var (
              $data)  [static]
```

Calculates sample variance of data.

Parameters

array	*$data*	Dataset.

Returns

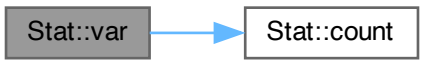
float Sample variance.

Definition at line 407 of file Stat.php.

```
00408     {
00409         $n = count($data);
00410         if ($n <= 1) {
00411             $var = 0;
00412         } else {
00413             $mean = self::mean($data);
00414             $sum = 0;
00415             foreach ($data as $element) {
00416                 $sum += pow(($element - $mean), 2);
00417             }
00418             $var = $sum / ($n - 1);
00419         }
00420         return $var;
00421     }
```

References count().

Here is the call graph for this function:



varp()

```
static Stat::varp (
              $data)  [static]
```

Calculates population variance of data.

Parameters

array	*$data*	Dataset.

Returns

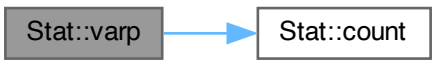
float Population variance.

Definition at line 429 of file Stat.php.

```
00430     {
00431         $n = count($data);
00432         if ($n <= 1) {
00433             $var = 0;
00434         } else {
00435             $mean = self::mean($data);
00436             $sum = 0;
00437             foreach ($data as $element) {
00438                 $sum += pow(($element - $mean), 2);
00439             }
00440             $var = $sum / $n;
00441         }
00442         return $var;
00443     }
```

References count().

Here is the call graph for this function:



4.18.3 Member Data Documentation

$actionDesc

Stat::$actionDesc [static], [protected]

Initial value:

```
= [
        'all' => 'Overall statistics for all your sections.',
        'sections' => 'Statistics broken down by section.',
        'questions' => 'Statistics by question, with option to drill down by
        sections.',
        'subquestions' => 'Statistics by subquestion, with option to drill down by
        sections.',
        'mistakes' => 'Display comments grouped by sub-questions.',
    ]
```

Definition at line 44 of file Stat.php.

$actionProperties

Stat::$actionProperties [static], [protected]

Initial value:

```
= [
        'all' => [
            'target' => 'left',
            'clear' => 'yes'
        ],
```

```
        'sections' => [
            'target' => 'right'
        ],
        'questions' => [
            'target' => 'left'
        ],
        'subquestions' => [
            'target' => 'left'
        ],
        'mistakes' => [
            'target' => 'left'
        ]
    ]
```

Definition at line 24 of file Stat.php.

$actions

```
Stat::$actions  [static], [protected]
```

Initial value:

```
= [
        'all' => 'Overall',
        'sections' => 'By Section',
        'questions' => 'By Question',
        'subquestions' => 'By Subquestion',
        'mistakes' => 'Common Mistakes'
    ]
```

Definition at line 15 of file Stat.php.

$class

```
Stat::$class = __CLASS__  [static], [protected]
```

Definition at line 55 of file Stat.php.

$display

```
Stat::$display  [static], [protected]
```

Initial value:

```
= [
        "class" => "info",
        "title" => "View Grade Statistics"
    ]
```

Definition at line 62 of file Stat.php.

$intro

```
Stat::$intro  [static], [protected]
```

Initial value:

```
= "<h4>Statistics</h4>
      <p>Obvious, isn't it? This page lets you generate marks statistics.</p>"
```

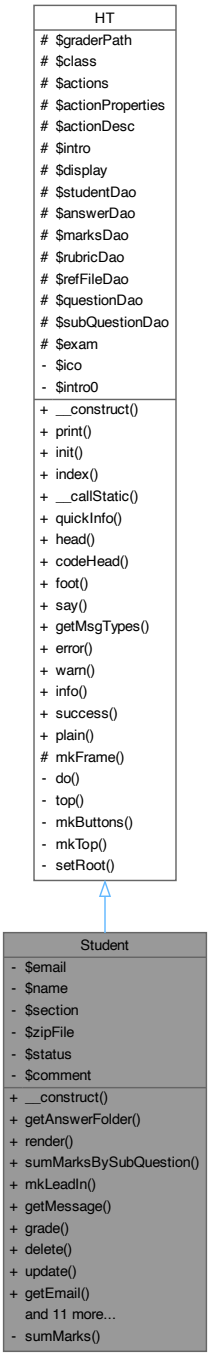
Definition at line 58 of file Stat.php.

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

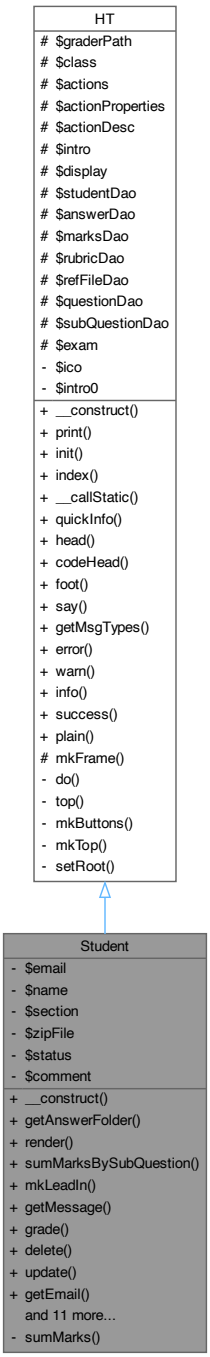
- Stat.php

4.19 Student Class Reference

Inheritance diagram for Student:



Collaboration diagram for Student:



Public Member Functions

- __construct ($email, $name, $section)
- getAnswerFolder ()
- render ()
- sumMarksBySubQuestion ()
- mkLeadIn ()

- getMessage ()
- grade ()
- delete ($name='')
- update ()
- getEmail ()
- getSection ()
- getStatus ()
- print ($toStr=true)
- autoGrade ()
- getZipFile ()
- getName ()
- setZipFile ($zipFile)
- getComment ()
- setComment ($comment)
- setStatus ($status)
- csv ()

Public Member Functions inherited from HT

- __construct ()
- print ($toStr=true)

Private Member Functions

- sumMarks ()

Private Attributes

- $email
- $name
- $section
- $zipFile
- $status = "New"
- $comment = ""

Additional Inherited Members

Static Public Member Functions inherited from HT

- static init ()
- static index ()
- static __callStatic ($method, $args)
- static quickInfo ($side)
- static head ($toStr=false, $title="Grader by Manoj", $refresh='')
- static codeHead ($title="Code Listing by Manoj")
- static foot ($toStr=false)
- static say ($what, $class='plain', $toStr=false, $showIcon=false)
- static getMsgTypes ()
- static error ($what, $toStr=false)
- static warn ($what, $toStr=false)
- static info ($what, $toStr=false)
- static success ($what, $toStr=false)
- static plain ($what, $toStr=false)

Static Protected Member Functions inherited from HT

- static mkFrame ($side='left')

Static Protected Attributes inherited from HT

- static $graderPath
- static $class = __CLASS__
- static $actions
- static $actionProperties = [ ]
- static $actionDesc
- static $intro
- static $display = ["class" => "success", "title" => ""]
- static $studentDao
- static $answerDao
- static $marksDao
- static $rubricDao
- static $refFileDao
- static $questionDao
- static $subQuestionDao
- static $exam

4.19.1 Detailed Description

Class Student

Represents a student with grading-related functionalities, including grading, summary, statistics, CSV export, and auto-grading.

Inherits from the HT class for rendering HTML and messaging.

Definition at line 11 of file Student.php.

4.19.2 Member Function Documentation

__construct()

```
Student::__construct (
              $email,
              $name,
              $section)
```

Student constructor.

Parameters

string	*$email*	Student's email address
string	*$name*	Student's name
string	*$section*	Section name

Definition at line 38 of file Student.php.

```
00039     {
00040         $this->email = trim(trim($email), "#");
00041         $this->name = $name;
00042         $this->section = $section;
00043     }
```

References $email, $name, and $section.

autoGrade()

`Student::autoGrade ()`

Autograde student answers based on configured auto-gradable questions.

Returns

void

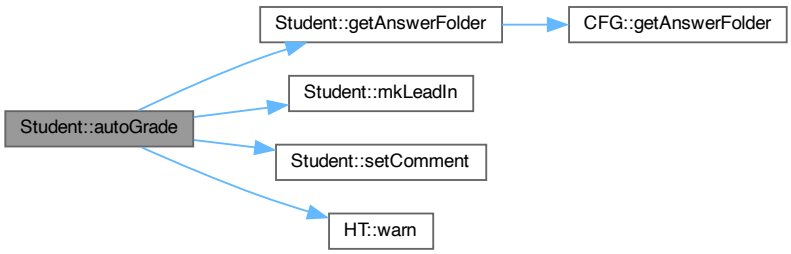
Definition at line 361 of file Student.php.

```
00362     {
00363         if ($this->status == 'Absent') {
00364             return;
00365         }
00366         // $options = CFG::$autoGradeOptions;
00367         // $subQuestionNames = $options['subQuestionNames'];
00368         $autoGradables = CFG::$autoGradables;
00369         $leadIn = $this->mkLeadIn();
00370         $what = "<p><strong>{$this->getName()}</strong></p>
00371         <table class='grader' width='80%' align='center'>
00372         <tr><th>Question</th><th>File</th><th>Marks</th></tr>";
00373         $isGraded = false;
00374         foreach ($autoGradables as $file => $autoGraded) {
00375             $autoGraderBase = $autoGraded['grader'];
00376             $autoGrader = realpath($this->getAnswerFolder() . $autoGraderBase);
00377 
00378             // This snippet doesn't work because urlencode is too aggressive
00379             // $autoGrader = str_replace($_SERVER['DOCUMENT_ROOT'], '', $autoGrader);
00380             // $autoGrader = urlencode($autoGrader);
00381             // $url = "http://localhost" . $autoGrader;
00382 
00383             // Just replace the spaces for now
00384             $url0 = str_replace(realpath($_SERVER['DOCUMENT_ROOT']), "http://localhost", $autoGrader);
00385             $url = str_replace(' ', '%20', $url0);
00386             $ch = curl_init();
00387             curl_setopt($ch, CURLOPT_URL, $url);
00388             curl_setopt($ch, CURLOPT_HEADER, 0);
00389             // In case there are infinite loops in the student code, timeout in 15 seconds
00390             // Fatal errors also seem to hang the CURL connection
00391             $timeout = 16;
00392             curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);
00393             curl_setopt($ch, CURLOPT_CONNECTTIMEOUT, $timeout); // Timeout for connection
00394             curl_setopt($ch, CURLOPT_TIMEOUT, $timeout); // Timeout for full request
00395             curl_setopt($ch, CURLOPT_FAILONERROR, true); // Catch HTTP errors
00396             $output = curl_exec($ch);
00397             curl_close($ch);
00398             $subQuestionName = $autoGraded['subQuestionName'];
00399             $marker = "$subQuestionName: ";
00400             $rubricName = "$subQuestionName-1";
00401             $lines = explode($marker, $output);
00402             // Mark is the first token in $line[1]
00403             $mark = 0;
00404             if (isset($lines[1])) {
00405                 $isGraded = true;
00406                 $mark = floatval($lines[1]);
00407                 // str_pad is to bust out of output buffering
00408                 if ($mark > 0) {
00409                     $what .=  "
00410                     <tr class='success'>
00411                     <td> $subQuestionName</td>
00412                     <td> $file</td>
00413                     <td> $mark</td>
00414                     </tr>";
00415                     // $what .= HT::warn(str_pad($what, 4096), true);
00416                 } else {
00417                     $what .=  "
00418                     <tr class='info'>
00419                     <td> $subQuestionName</td>
00420                     <td> $file</td>
00421                     <td> $mark</td>
00422                     </tr>";
00423                     // $what .= HT::error(str_pad("$leadIn: $mark [File: $file]", 4096));
00424                 }
00425             } else {
00426                 $what .=  "
00427                     <tr class='error'>
00428                     <td> $subQuestionName</td>
00429                     <td> $file</td>
00430                     <td> Failed</td>
```

```
00431                     </tr>
00432                     <tr class='error'><td colspan='2'>Marker '$marker', as specified in
    <code>CFG::\$autoGradables['$subQuestionName']</code>, is missing in the output!</td><td>
00433                     <table align='right' border='0'><tr>
00434                     <td><a href='$url' target='right'><button>View</button></a></td>
00435                     <td><form method='post' target='left'>
00436                     <input type='submit' name='gradeOne' value='Grade'>
00437                     <input type='hidden' name='do' value='gradeOne'>
00438                     <input type='hidden' name='email' value='$this->email'>
00439                     </form></td>
00440                     </tr></table>
00441                     </td></tr>";
00442             }
00443             $marksObj = new Marks($this->email, $subQuestionName, $rubricName, $mark);
00444             self::$marksDao->update($marksObj);
00445             $this->setComment(htmlentities($output));
00446             self::$studentDao->update($this);
00447         }
00448         $what .= "</table>";
00449         if ($isGraded) {
00450             HT::warn(str_pad($what, 4096));
00451         }
00452     }
```

References CFG::$autoGradables, getAnswerFolder(), mkLeadIn(), setComment(), and HT::warn().

Here is the call graph for this function:



csv()

```
Student::csv ()
```

Render student marks summary for CSV table display.

Returns

string CSV-formatted HTML row (

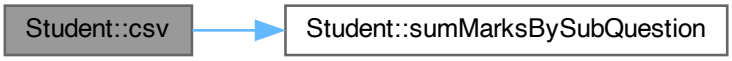
)

Definition at line 528 of file Student.php.

```
00529     {
00530         $str = "<tr><td>$this->name</td>";
00531         $marks = $this->sumMarksBySubQuestion();
00532         foreach ($marks as $subQuestionName => $m) {
00533             $str .= "<td>$subQuestionName &rarr; $m</td>";
00534         }
00535         $str .= "</tr>";
00536         return $str;
00537     }
```

References sumMarksBySubQuestion().

Here is the call graph for this function:



delete()

```
Student::delete (
              $name = '')
```

Delete student's extracted submission files.

Parameters

string	*$name*	Optional parameter (unused)

Returns

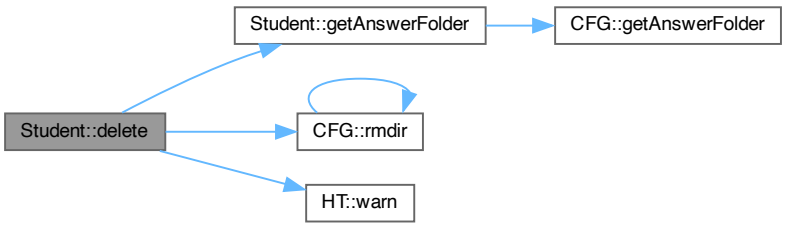
void

Definition at line 296 of file Student.php.

```
00297     {
00298         $submissionFolder = realpath($this->getAnswerFolder());
00299         CFG::rmdir($submissionFolder);
00300         HT::warn("<strong>$this->name</strong>: Deleted unzipped files in <br>
00301         <code>$submissionFolder</code>");
00302     }
```

References $name, getAnswerFolder(), CFG::rmdir(), and HT::warn().

Here is the call graph for this function:



getAnswerFolder()

```
Student::getAnswerFolder ()
```

Get path to student's answer folder.

Returns

string Path to answer folder

Definition at line 50 of file Student.php.

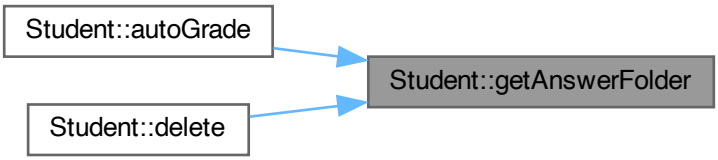
```
00051     {
00052         $sectionName = $this->section;
00053         // $studentName = $this->name;
00054         // $answerFolder = CFG::getAnswerFolder($sectionName, $studentName);
00055         $studentEmail = $this->email;
00056         $answerFolder = CFG::getAnswerFolder($sectionName, $studentEmail);
00057         return $answerFolder;
00058     }
```

References $email, $section, and CFG::getAnswerFolder().

Referenced by autoGrade(), and delete().

Here is the call graph for this function:

Here is the caller graph for this function:



getComment()

```
Student::getComment ()
```

Get comment/remarks for the student.

Returns

string Comments

Definition at line 491 of file Student.php.

```
00492     {
00493         return $this->comment;
00494     }
```

References $comment.

getEmail()

```
Student::getEmail ()
```

Get student's email.

Returns

string Student's email

Definition at line 319 of file Student.php.

```
00320     {
00321         return $this->email;
00322     }
```

References $email.

getMessage()

```
Student::getMessage ()
```

Prepare grading status message with marks summary.

Returns

string Grading status message

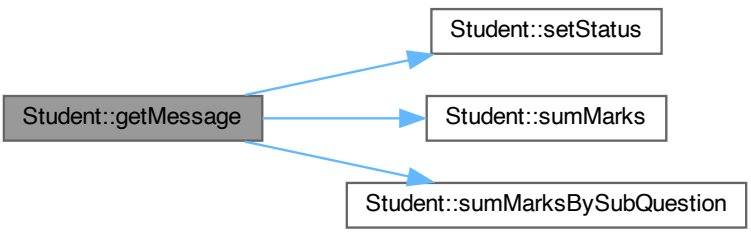
Definition at line 212 of file Student.php.

```
00213     {
00214         $marksAll = self::$marksDao->getByObject($this);
00215         $total = $this->sumMarks();
00216         $rubricTotal = 0;
00217         $rubricsAll = [];
00218         foreach (self::$exam->getQuestions() as $question) {
00219             $subQuestions = $question->getSubquestions();
00220             foreach ($subQuestions as $subQuestion) {
00221                 $rubrics = self::$rubricDao->getByObject($subQuestion);
00222                 foreach ($rubrics as $rubric) {
00223                     $rubricMark = $rubric->getMarks();
00224                     $rubricsAll[] = $rubric;
00225                     if ($rubricMark > 0) {
00226                         $rubricTotal += $rubricMark;
00227                     }
00228                 }
00229             }
00230         }
00231         $numMarks = count($marksAll);
00232         $numRubrics = count($rubricsAll);
00233         if ($numMarks == $numRubrics) {
00234             $this->setStatus("Graded")->update();
00235             $msg = "[Grading Complete:  Marks: $total/$rubricTotal]";
00236         } else if (count($marksAll) > 0) {
00237             $this->setStatus("Grading")->update();
00238             $msg = "[Grading in Progress:  Marks: $total/$rubricTotal ($numMarks of $numRubrics
      rubrics done)]";
00239         } else {
00240             $msg = "";
00241         }
00242         $this->sumMarksBySubQuestion();
00243         return $msg;
00244     }
```

References setStatus(), sumMarks(), and sumMarksBySubQuestion().

Referenced by grade().

Here is the call graph for this function:



Here is the caller graph for this function:

getName()

```
Student::getName ()
```

Get student's name.

Returns

string Student's name

Definition at line 469 of file Student.php.

```
00470     {
00471         return $this->name;
00472     }
```

References $name.

getSection()

```
Student::getSection ()
```

Get student's section name.

Returns

string Section name

Definition at line 329 of file Student.php.

```
00330     {
00331         return $this->section;
00332     }
```

References $section.

getStatus()

```
Student::getStatus ()
```

Get student's grading status.

Returns

string Grading status

Definition at line 339 of file Student.php.

```
00340     {
00341         return $this->status;
00342     }
```

References $status.

getZipFile()

```
Student::getZipFile ()
```

Get student's zip file name.

Returns

string Zip file path

Definition at line 459 of file Student.php.

```
00460     {
00461         return $this->zipFile;
00462     }
```

References $zipFile.

grade()

```
Student::grade ()
```

Handle grading action for the student, showing grading UI.

Returns

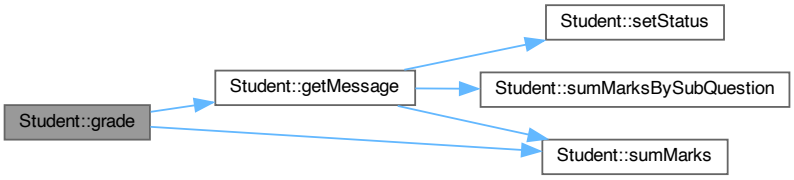
void

Definition at line 251 of file Student.php.

```
00252     {
00253         $msg = $this->getMessage();
00254         $echo = "info";
00255         $what = "";
00256         $disabled = "";
00257         if ($this->status == "Absent") {
00258             $value = "Ignore";
00259             $echo = "error";
00260             $disabled = "disabled";
00261         } else if ($this->status == "Grading") {
00262             $value = "Continue";
00263             $echo = ($this->sumMarks() > 0) ? "warn" : "plain";
00264         } elseif ($this->status == "Graded") {
00265             $value = "Regrade";
00266             $echo = "success";
00267         } else {
00268             $value = "Grade";
00269             $echo = "info";
00270         }
00271         $nStudents = self::$studentDao->getNStudents();
00272         $iStudent = self::$studentDao->getIStudent($this->email);
00273         $what = "<strong>$this->name</strong> <small>$msg 
00274             ($iStudent of " . $nStudents . ")</small>
00275         <form method='post'  target='quickSummary'>
00276         <input type='submit' name='quickSummary' value='Summary' $disabled
00277         style='position:absolute;top:10px;right:90px;'>
00278         <input type='hidden' name='do' value='quickSummary'>
00279         <input type='hidden' name='email' value='$this->email'>
00280         </form>
00281         <form method='post'  target='left'>
00282         <input type='submit' name='gradeOne' value='$value' $disabled
00283         style='position:absolute;top:10px;right:10px;'>
00284         <input type='hidden' name='do' value='gradeOne'>
00285         <input type='hidden' name='email' value='$this->email'>
00286         </form>";
00287         $this->$echo($what);
00288     }
```

References getMessage(), and sumMarks().

Here is the call graph for this function:



mkLeadIn()

```
Student::mkLeadIn ()
```

Prepare leading display string for student information.

Returns

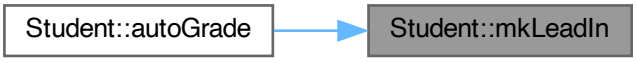
string Lead-in string

Definition at line 201 of file Student.php.

```
00202     {
00203         $leadIn = "<strong>$this->name</strong> [$this->section]: ";
00204         return $leadIn;
00205     }
```

Referenced by autoGrade().

Here is the caller graph for this function:



print()

```
Student::print (
              $toStr = true)
```

Render printable view of student information.

Parameters

bool	*$toStr*	Whether to return as string

Returns

string Rendered content

Definition at line 350 of file Student.php.

```
00351     {
00352         $str = parent::print($toStr);
00353         return $str;
00354     }
```

render()

```
Student::render ()
```

Render student grading navigation and summary UI.

Returns

string Rendered HTML content

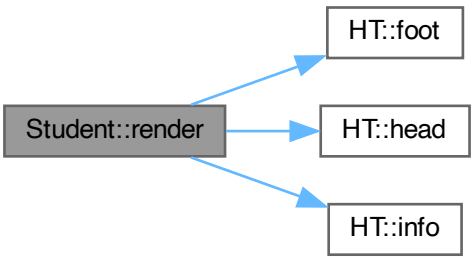
Definition at line 65 of file Student.php.

```
00066     {
00067         list($prev, $prevUngraded, $nextUngraded, $next)
00068             = self::$studentDao->getOthers($this);
00069         $srcDoc = HT::head(true);
00070         $prevForm =
00071             $prevUngradedForm =
00072             $nextForm =
00073             $nextUngradedForm = "<td style='width:1%'></td>";
00074         $clearRight = "onclick='var right1 = window.parent.parent.right.right1;
00075         right1.document.write(); var right2 = window.parent.parent.right.right2;
00076         right2.document.write();'";
00077         if ($prev) {
00078             $prevForm = "<td style='width:1%'>
00079             <form method='post' target='left' action='?do'>
00080             <input type='hidden' name='do' value='gradeOne'>
00081             <input type='hidden' name='email' value='$prev->email'>
00082             <input type='submit' value='Prev' title='$prev->name' $clearRight >
00083             </form></td>";
00084         }
00085         if ($prevUngraded) {
00086             $prevUngradedForm = "<td style='width:1%'>
00087             <form method='post' target='left' action='?do'>
00088             <input type='hidden' name='do' value='gradeOne'>
00089             <input type='hidden' name='email' value='$prevUngraded->email'>
00090             <input type='submit' value='Prev Ungraded'
00091             title='$prevUngraded->name' $clearRight>
00092             </form></td>";
00093         }
00094         if ($next) {
00095             $nextForm = "<td style='width:1%'>
00096             <form method='post' target='left' action='?do'>
00097             <input type='hidden' name='do' value='gradeOne' >
00098             <input type='hidden' name='email' value='$next->email'>
00099             <input type='submit' value='Next' title='$next->name' $clearRight>
00100             </form></td>";
00101         }
00102         if ($nextUngraded) {
00103             $nextUngradedForm = "<td style='width:1%'>
00104             <form method='post' target='left' action='?do'>
00105             <input type='hidden' name='do' value='gradeOne'>
00106             <input type='hidden' name='email' value='$nextUngraded->email'>
00107             <input type='submit' value='Next Ungraded'
00108             title='$nextUngraded->name' $clearRight>
00109             </form></td>";
00110         }
```

```
00111         $nStudents = self::$studentDao->getNStudents();
00112         $iStudent = self::$studentDao->getIStudent($this->email);
00113         $srcDoc .= HT::info("<div style='text-align:center;font-size:120%;'>
00114         <table align='center' style='width:100%'><tr>$prevForm $prevUngradedForm
00115         <td><strong title='Student::render()'>$this->name</strong> 
00116         <small style='font-size:80%;'> [<strong>$this->section</strong>:
00117         $iStudent of $nStudents]</small></td>
00118         $nextUngradedForm $nextForm </tr></table></div>
00119         <table align='center'><tr>
00120         <td><form method='post' target='left' id='reload' action='?do'>
00121         <input type='submit' value='Reload'>
00122         <input type='hidden' name='do' value='gradeOne'>
00123         <input type='hidden' name='name' value='$this->name'>
00124         <input type='hidden' name='email' value='$this->email'>
00125         </form></td>
00126         <td><form method='post' target='right' action='?do'>
00127         <input type='submit' id='loadAnswers' value='Show Answers'>
00128         <input type='hidden' name='do' value='loadAnswers'>
00129         <input type='hidden' name='name' value='$this->name'>
00130         <input type='hidden' name='email' value='$this->email'>
00131         </form></td>
00132         <td><form method='post' target='quickSummary' action='?do' id='$this->email'>
00133         <input type='submit' name='summary' value='Quick Summary' id='quickSummary'
00134         onclick=
00135         \"var h = screen.height * 0.8;
00136         var t = screen.height * 0.05;
00137         var w = screen.width * 0.4;
00138         var l = screen.width * 0.05;
00139         window.open('', 'quickSummary', 'width='+ w +',height=' + h + ',left='+ l +',top='+ t
      +',resizable=yes,scrollbars=yes,toolbar=yes,menubar=no,location=no,directories=no,status=yes');
00140           document.getElementById('$this->email').submit();
00141           return true;\">
00142         <input type='hidden' name='do' value='quickSummary'>
00143         <input type='hidden' name='name' value='$this->name'>
00144         <input type='hidden' name='email' value='$this->email'>
00145         </form></td>
00146         </tr></table>
00147         ", true);
00148         $loadAnswers = "<script type='text/javascript'>
00149         window.onload=function(){document.getElementById('loadAnswers').click();}
00150         </script>";
00151         $srcDoc .= $loadAnswers;
00152         $srcDoc .= HT::foot(true);
00153         return $srcDoc;
00154     }
```

References HT::foot(), HT::head(), and HT::info().

Here is the call graph for this function:



setComment()

```
Student::setComment (
              $comment)
```

Set or append a comment for the student.

Parameters

string	*$comment*	Comment to add

Returns

self

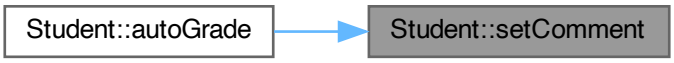
Definition at line 503 of file Student.php.

```
00503                                       : self
00504     {
00505         if (strpos($this->comment, $comment) === false) {
00506             $this->comment .= $comment;
00507         }
00508         return $this;
00509     }
```

References $comment.

Referenced by autoGrade().

Here is the caller graph for this function:



setStatus()

```
Student::setStatus (
              $status)
```

Set grading status.

Parameters

string	*$status*	Status to set

Returns

self

Definition at line 517 of file Student.php.

```
00517                                       : self
00518     {
00519         $this->status = $status;
00520         return $this;
00521     }
```

References $status.

Referenced by getMessage().

Here is the caller graph for this function:

setZipFile()

```
Student::setZipFile (
              $zipFile)
```

Set zip file path.

Parameters

string	*$zipFile*	Path to zip file

Returns

self

Definition at line 480 of file Student.php.

```
00480                                       : self
00481     {
00482         $this->zipFile = $zipFile;
00483         return $this;
00484     }
```

References $zipFile.

sumMarks()

```
Student::sumMarks ()  [private]
```

Calculate total marks awarded to the student.

Returns

float Total marks

Definition at line 161 of file Student.php.

```
00162     {
00163         $marksAll = self::$marksDao->getByObject($this);
00164         $total = 0;
00165         foreach ($marksAll as $marks) {
00166             $total += $marks->getMarks();
00167         }
00168         return $total;
00169     }
```

Referenced by getMessage(), and grade().

Here is the caller graph for this function:

sumMarksBySubQuestion()

```
Student::sumMarksBySubQuestion ()
```

Calculate total marks awarded to the student grouped by sub-question.

Returns

array Associative array of sub-question names and their marks

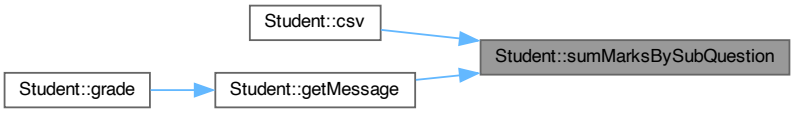
Definition at line 176 of file Student.php.

```
00177     {
00178         $sums = [];
00179         foreach (self::$exam->getQuestions() as $question) {
00180             $subQuestions = $question->getSubquestions();
00181             foreach ($subQuestions as $subQuestion) {
00182                 $marks = self::$marksDao->getByObject($this, $subQuestion);
00183                 $subQuestionName = $subQuestion->getName();
00184                 foreach ($marks as $m) {
00185                     if (isset($sums[$subQuestionName])) {
00186                         $sums[$subQuestionName] += $m->getMarks();
00187                     } else {
00188                         $sums[$subQuestionName] = $m->getMarks();
00189                     }
00190                 }
00191             }
00192         }
00193         return $sums;
```

```
00194     }
```

Referenced by csv(), and getMessage().

Here is the caller graph for this function:



update()

```
Student::update ()
```

Update student record in the database.

Returns

void

Definition at line 309 of file Student.php.

```
00310     {
00311         self::$studentDao->update($this);
00312     }
```

4.19.3 Member Data Documentation

$comment

```
Student::$comment = ""  [private]
```

Definition at line 29 of file Student.php.

Referenced by getComment(), and setComment().

$email

```
Student::$email  [private]
```

Definition at line 14 of file Student.php.

Referenced by __construct(), getAnswerFolder(), and getEmail().

$name

```
Student::$name  [private]
```

Definition at line 17 of file Student.php.

Referenced by __construct(), delete(), and getName().

$section

```
Student::$section  [private]
```

Definition at line 20 of file Student.php.

Referenced by __construct(), getAnswerFolder(), and getSection().

$status

```
Student::$status = "New"  [private]
```

Definition at line 26 of file Student.php.

Referenced by getStatus(), and setStatus().

$zipFile

```
Student::$zipFile  [private]
```

Definition at line 23 of file Student.php.

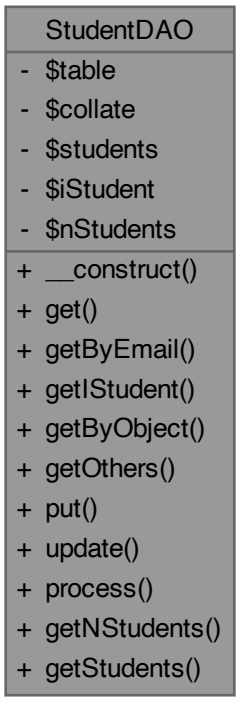
Referenced by getZipFile(), and setZipFile().

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

- Student.php

4.20 StudentDAO Class Reference

Collaboration diagram for StudentDAO:



Public Member Functions

- __construct ()
- get ($where=[ ], $verbose=false)
- getByEmail ($studentEmail)
- getIStudent ($studentEmail)
- getByObject ($answer)
- getOthers ($student)
- put ($section, $students)
- update ($student)
- process ($section)

Static Public Member Functions

- static getNStudents ()
- static getStudents ()

Static Private Attributes

- static $table
- static $collate = 'student_email'
- static $students = [ ]
- static $iStudent = [ ]
- static $nStudents = 0

4.20.1 Detailed Description

Class StudentDAO

Handles the CRUD operations, indexing, and retrieval of student records. Maintains a static list of students and their ordering to enable navigation between students for grading workflows.

Definition at line 10 of file StudentDAO.php.

4.20.2 Member Function Documentation

__construct()

```
StudentDAO::__construct ()
```

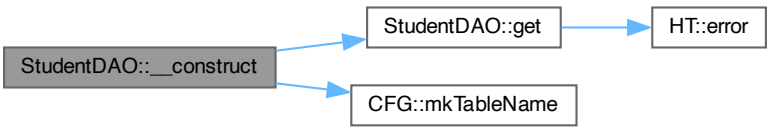
Initializes the table name and loads all student records into static cache.

Definition at line 30 of file StudentDAO.php.

```
00031     {
00032         $table = CFG::mkTableName("students");
00033         self::$table = $table;
00034         if (empty(self::$students)) {
00035             $students = $this->get();
00036             self::$students = $students;
00037             self::$nStudents = count($students);
00038             $i = 0;
00039             foreach ($students as $student) {
00040                 $i++;
00041                 self::$iStudent[$student->getEmail()] = $i;
00042             }
00043             if (self::$nStudents != $i) {
00044                 xdebug_print_function_stack();
00045                 die("Error in student arrays");
00046             }
00047             ksort(self::$iStudent);
00048         }
00049     }
```

References $students, $table, get(), and CFG::mkTableName().

Here is the call graph for this function:



get()

```
StudentDAO::get (
                $where = [],
                $verbose = false)
```

Retrieves student records based on given conditions.

Parameters

array	*$where*	Optional associative array for SQL WHERE clause.
bool	*$verbose*	Display error if no students are found.

Returns

Student[] List of Student objects.

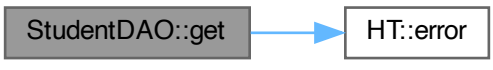
Definition at line 58 of file StudentDAO.php.

```
00059     {
00060         $table = self::$table;
00061         $rows = DB::$db->get(
00062             table: $table,
00063             where: $where
00064         );
00065         $got = count($rows);
00066         $students = [];
00067         if ($verbose && $got <= 0) { // Not found in the DB
00068             HT::error("<strong>StudentDAO</strong>: No students in the DB!<br>
00069             Run StudentDAO->process(section) for each section first.");
00070         } else {
00071             foreach ($rows as $row) {
00072                 $student = new Student(
00073                     $row['student_email'],
00074                     $row['student_name'],
00075                     $row['section']
00076                 );
00077                 $student->setZipFile($row['zip_filename']);
00078                 $student->setStatus($row['status']);
00079                 $student->setComment($row['comment']);
00080                 $students[] = $student;
00081             }
00082         }
00083         return $students;
00084     }
```

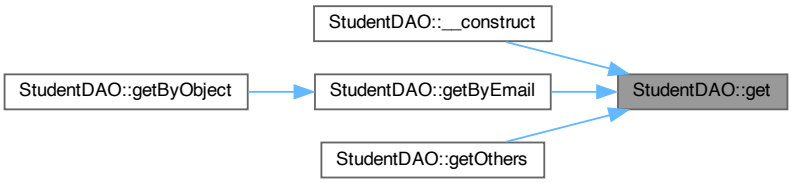
References DB::$db, $students, $table, and HT::error().

Referenced by __construct(), getByEmail(), and getOthers().

Here is the call graph for this function:



Here is the caller graph for this function:



getByEmail()

```
StudentDAO::getByEmail (
              $studentEmail)
```

Retrieves a student object by their email address.

Parameters

string	*$studentEmail*	Student's email.

Returns

Student[]|Student Single student object or list (if erroneous duplicates found).

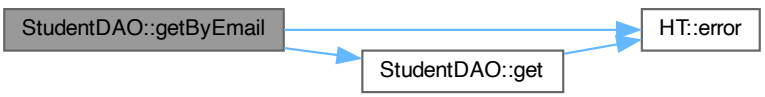
Definition at line 92 of file StudentDAO.php.

```
00093     {
00094         $student = $this->get(['student_email' => $studentEmail]);
00095         if (count($student) > 1) {
00096             HT::error("Too many students with email id $studentEmail!");
00097             $student = $student[0];
00098         }
00099         return $student;
00100     }
```

References HT::error(), and get().

Referenced by getByObject().

Here is the call graph for this function:



Here is the caller graph for this function:

getByObject()

```
StudentDAO::getByObject (
            $answer)
```

Retrieves a student associated with a specific Answer object.

Parameters

Answer	*$answer*	Answer object containing the student email.

Returns

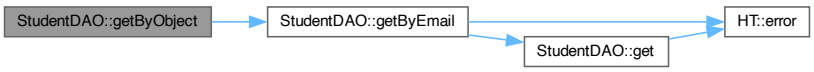
Student[]|Student

Definition at line 119 of file StudentDAO.php.

```
00120     {
00121         return $this->getByEmail($answer->getStudentEmail());
00122     }
```

References getByEmail().

Here is the call graph for this function:



getIStudent()

```
StudentDAO::getIStudent (
            $studentEmail)
```

Retrieves the 1-based index (position) of a student for grading navigation.

Parameters

string	*$studentEmail*	Student's email.

Returns

int Index of the student.

Definition at line 108 of file StudentDAO.php.

```
00109     {
00110         return self::$iStudent[$studentEmail];
00111     }
```

getNStudents()

```
static StudentDAO::getNStudents ()  [static]
```

Returns the total number of students.

Returns

int Total number of students.

Definition at line 248 of file StudentDAO.php.

```
00249     {
00250         return self::$nStudents;
00251     }
```

getOthers()

```
StudentDAO::getOthers (
              $student)
```

Returns neighboring students for navigation: previous, previous ungraded, next ungraded, next.

Parameters

Student	*$student*	Current student.

Returns

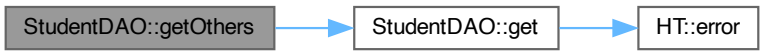
array Array of [prev, prevUngraded, nextUngraded, next] students.

Definition at line 131 of file StudentDAO.php.

```
00132     {
00133         $others = $this->get();
00134         $needPrevious = true;
00135         $needNext = false;
00136         $prev = $prevUngraded = $nextUngraded = $next = "";
00137         foreach ($others as $other) {
00138             $status = $other->getStatus();
00139             if ($status == 'Absent') {
00140                 continue;
00141             }
00142             if ($other->getEmail() == $student->getEmail()) {
00143                 $needPrevious = false;
00144                 $needNext = true;
00145                 continue;
00146             }
00147             if ($needPrevious) {
00148                 $prev = $other;
00149                 if ($status == "New") {
00150                     $prevUngraded = $other;
00151                 }
00152             }
00153             if ($needNext) {
00154                 if ($next == "") {
00155                     $next = $other;
00156                 }
00157                 if ($status == "New") {
00158                     $nextUngraded = $other;
00159                     break;
00160                 }
00161             }
00162         }
00163         return [$prev, $prevUngraded, $nextUngraded, $next];
00164     }
```

References get().

Here is the call graph for this function:



getStudents()

```
static StudentDAO::getStudents ()  [static]
```

Returns the cached list of Student objects.

Returns

Student[] List of students.

Definition at line 258 of file StudentDAO.php.

```
00259     {
00260         return self::$students;
00261     }
```

process()

```
StudentDAO::process (
              $section)
```

Processes students from a section's CSV file and stores them in the database.

Parameters

Section	*$section*	Section object.

Returns

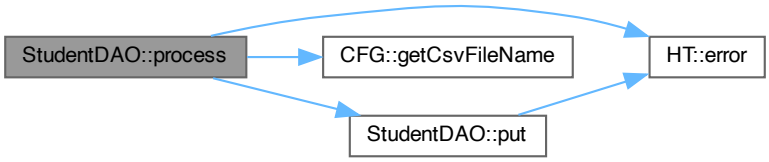
void

Definition at line 224 of file StudentDAO.php.

```
00225     {
00226         $sectionName = $section->getName();
00227         $csv = CFG::getCsvFileName($sectionName);
00228         if (!is_file($csv)) {
00229             HT::error("Error loading $csv");
00230             return;
00231         }
00232         $file = fopen($csv, 'r');
00233         $line = fgetcsv($file); // Skip the header
00234         $students = [];
00235         while (($line = fgetcsv($file)) !== FALSE) {
00236             //$line is an array of the csv elements
00237             $students[] = new Student($line[0], $line[1], $sectionName);
00238         }
00239         fclose($file);
00240         $this->put($section, $students);
00241     }
```

References $students, HT::error(), CFG::getCsvFileName(), and put().

Here is the call graph for this function:



put()

```
StudentDAO::put (
              $section,
              $students)
```

Inserts or updates all students of a given section into the database.

Parameters

Section	*$section*	Section object.
Student[ ]	*$students*	Array of Student objects to be saved.

Returns

void

Definition at line 173 of file StudentDAO.php.

```
00174     {
00175         $table = self::$table;
00176         $sectionName = $section->getName();
00177         if (count($students) <= 0) { // No students found. An error!
00178             HT::error("<strong>Section $sectionName</strong>:
00179             No students found. <br>
00180             Something very wrong!");
00181         } else {
00182             $columns = ['student_email', 'student_name', 'section', 'status', 'zip_filename',
    'comment'];
00183             $rows = [];
00184             foreach ($students as $student) {
00185                 $studentEmail = $student->getEmail();
00186                 $studentName = $student->getName();
00187                 $status = $student->getStatus();
00188                 $zipFileName = $student->getZipFile();
00189                 $comment = $student->getComment();
00190                 $rows[] = [$studentEmail, $studentName, $sectionName, $status, $zipFileName,
    $comment];
00191             }
00192             // DB::$db->put($table, $columns, $rows);
00193             DB::$db->update($table, $columns, $rows);
00194         }
00195     }
```

References DB::$db, $students, $table, and HT::error().

Referenced by process().

Here is the call graph for this function:

Here is the caller graph for this function:

update()

```
StudentDAO::update (
              $student)
```

Updates an individual student's record in the database.

Parameters

Student	*$student*	Student object to be updated.

Returns

void

Definition at line 203 of file StudentDAO.php.

```
00204     {
00205         $table = self::$table;
00206         $columns = ['student_email', 'student_name', 'section', 'status', 'zip_filename', 'comment'];
00207         $studentEmail = $student->getEmail();
00208         $studentName = $student->getName();
00209         $section = $student->getSection();
00210         $status = $student->getStatus();
00211         $zipFileName = $student->getZipFile();
00212         $comment = $student->getComment();
00213         $rows = [[$studentEmail, $studentName, $section, $status, $zipFileName, $comment]];
00214         // DB::$db->put($table, $columns, $rows);
00215         DB::$db->update($table, $columns, $rows);
00216     }
```

References DB::$db, and $table.

4.20.3 Member Data Documentation

$collate

```
StudentDAO::$collate = 'student_email'  [static], [private]
```

Definition at line 16 of file StudentDAO.php.

$iStudent

```
StudentDAO::$iStudent = []  [static], [private]
```

Definition at line 22 of file StudentDAO.php.

$nStudents

```
StudentDAO::$nStudents = 0  [static], [private]
```

Definition at line 25 of file StudentDAO.php.

$students

```
StudentDAO::$students = []  [static], [private]
```

Definition at line 19 of file StudentDAO.php.

Referenced by __construct(), get(), process(), and put().

$table

```
StudentDAO::$table  [static], [private]
```

Definition at line 13 of file StudentDAO.php.

Referenced by __construct(), get(), put(), and update().

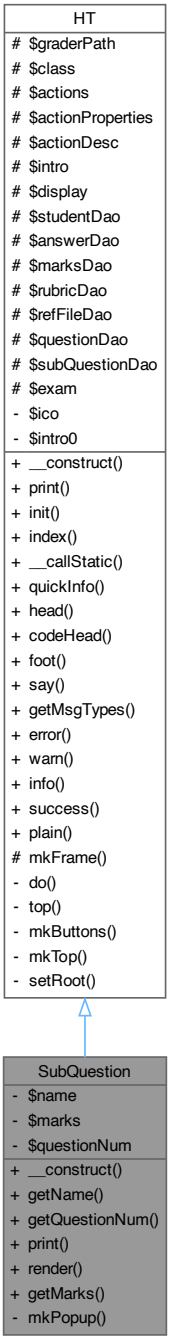
The documentation for this class was generated from the following file:

- StudentDAO.php

4.21 SubQuestion Class Reference

Inheritance diagram for SubQuestion:

Collaboration diagram for SubQuestion:



Public Member Functions

- __construct ($name, $mark, $questionNum)
- getName ()
- getQuestionNum ()
- print ($toStr=true)
- render ($student)
- getMarks ()

Public Member Functions inherited from HT

- __construct ()
- print ($toStr=true)

Static Private Member Functions

- static mkPopup ($fileName)

Private Attributes

- $name
- $marks
- $questionNum

Additional Inherited Members

Static Public Member Functions inherited from HT

- static init ()
- static index ()
- static __callStatic ($method, $args)
- static quickInfo ($side)
- static head ($toStr=false, $title="Grader by Manoj", $refresh='')
- static codeHead ($title="Code Listing by Manoj")
- static foot ($toStr=false)
- static say ($what, $class='plain', $toStr=false, $showIcon=false)
- static getMsgTypes ()
- static error ($what, $toStr=false)
- static warn ($what, $toStr=false)
- static info ($what, $toStr=false)
- static success ($what, $toStr=false)
- static plain ($what, $toStr=false)

Static Protected Member Functions inherited from HT

- static mkFrame ($side='left')

Static Protected Attributes inherited from HT

- static $graderPath
- static $class = __CLASS__
- static $actions
- static $actionProperties = [ ]
- static $actionDesc
- static $intro
- static $display = ["class" => "success", "title" => ""]
- static $studentDao
- static $answerDao
- static $marksDao
- static $rubricDao
- static $refFileDao
- static $questionDao
- static $subQuestionDao
- static $exam

4.21.1 Detailed Description

Class SubQuestion

Represents a sub-question in an exam or assignment, including marks, rubrics, reference solutions, and resources. Provides rendering and grading interfaces for students.

Definition at line 10 of file SubQuestion.php.

4.21.2 Member Function Documentation

__construct()

```
SubQuestion::__construct (
              $name,
              $mark,
              $questionNum)
```

SubQuestion constructor.

Parameters

string	*$name*	Sub-question name
float	*$mark*	Marks assigned to the sub-question
string	*$questionNum*	Parent question number

Definition at line 28 of file SubQuestion.php.

```
00029     {
00030         $this->name = strtolower($name);
00031         $this->marks = $mark;
00032         $this->questionNum = strtolower($questionNum);
00033     }
```

References $name, and $questionNum.

getMarks()

```
SubQuestion::getMarks ()
```

Get marks assigned to this sub-question.

Returns

float Marks for sub-question

Definition at line 242 of file SubQuestion.php.

```
00243     {
00244         return $this->marks;
00245     }
```

References $marks.

getName()

```
SubQuestion::getName ()
```

Get sub-question name.

Returns

string Sub-question name

Definition at line 40 of file SubQuestion.php.

```
00041     {
00042         return $this->name;
00043     }
```

References $name.

getQuestionNum()

```
SubQuestion::getQuestionNum ()
```

Get parent question number.

Returns

string Parent question number

Definition at line 50 of file SubQuestion.php.

```
00051     {
00052         return $this->questionNum;
00053     }
```

References $questionNum.

mkPopup()

```
static SubQuestion::mkPopup (
              $fileName)  [static], [private]
```

Create a popup for file viewing in grading interface.

Parameters

string	*$fileName*	Name of the file to open in popup

Returns

string Javascript popup code for HTML element

Definition at line 74 of file SubQuestion.php.

```
00075     {
00076         return "onclick=
00077         \"var h = screen.height * 0.8;
00078         var t = screen.height * 0.05;
00079         var w = screen.width * 0.4;
00080         var l = screen.width * 0.05;
00081         window.open('', '$fileName', 'width='+ w +',height=' + h + ',left='+ l +',top='+ t
      +',resizable=yes,scrollbars=yes,toolbar=yes,menubar=no,location=no,directories=no,status=yes');
00082           document.getElementById('$fileName').submit();
00083           return false;\"";
00084     }
```

print()

```
SubQuestion::print (
              $toStr = true)
```

Render a printable view of the sub-question.

Parameters

bool	*$toStr*	Whether to return the output as string

Returns

string Rendered string of sub-question and marks

Definition at line 61 of file SubQuestion.php.

```
00062     {
00063         $str = "<strong>  SubQuestion: $this->name
00064           Marks: $this->marks</strong><br>";
00065         return $str;
00066     }
```

render()

```
SubQuestion::render (
              $student)
```

Render sub-question information and rubric grading interface.

Parameters

Student	*$student*	Student object for whom grading is being done

Returns

string Rendered HTML content

Definition at line 92 of file SubQuestion.php.

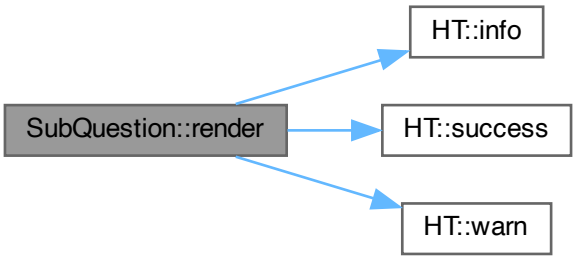
```
00093     {
00094         $subQuestionName = $this->name;
00095         $resources = self::$refFileDao->getByObject($this, 'resources');
00096         $solutions = self::$refFileDao->getByObject($this, 'solutions');
00097         $srcDoc = " <div style='font-size:110%;font-weight:bold'
00098             title='SubQuestion::render()'>
00099             SubQuestion: $subQuestionName</div>";
00100         $tabRes = "<table title='SubQuestion::render()'>
00101             <tr><th colspan='100%'>Resources</th></tr><tr>";
00102         $tabSoln = "<table title='SubQuestion::render()'>
00103             <tr><th colspan='100%'>Solutions</th></tr><tr>";
00104         // Get the rubric and get the filename so that the right resource/solution
00105         // can be highlighted
00106         $rubrics = self::$rubricDao->getByObject($this);
00107         $rubricFiles = [];
00108         foreach ($rubrics as $rubric) {
00109             $rubricFile = basename($rubric->getShortFileName());
00110             if (!in_array($rubricFile, $rubricFiles)) {
00111                 $rubricFiles[] = $rubricFile;
00112             }
00113         }
00114         // Assumes resources and solutions are sorted in the same order.
00115         // If not, sort them in the RefFilesDAO class
00116         foreach ($resources as $key => $resource) {
00117             $shortFIleName = $resource->getShortFileName();
00118             if (!self::$refFileDao->isGradable($shortFIleName)) {
00119                 continue;
00120             }
00121             $fileName = $resource->getFileName();
00122             $basename = basename($fileName);
00123             $fileClass = "plain";
00124             if (in_array($basename, $rubricFiles)) {
00125                 $fileClass = "error";
00126             }
```

```
00127             $fullText = htmlentities($resource->getFullText(), ENT_QUOTES);
00128             $popup = self::mkPopup($fileName);
00129             $tabRes .= "<td>
00130             <form method='post' action='?do'
00131             target='$fileName' id='$fileName'>
00132             <input type='submit' name='viewfile' title='$fileName'
00133             value='$basename' $popup class='$fileClass'>
00134             <input type='hidden' name='do' value='viewFile'>
00135             <input type='hidden' name='filename' value='$fileName'>
00136             <input type='hidden' name='fulltext' value='$fullText'>
00137             </form></td>
00138             ";
00139             // TODO: Investigate: How come $solnKey wasn't necessary before?
00140             $solnKey = str_replace("resources", "solutions", $key);
00141             $solution = $solutions[$solnKey];
00142             $fileName = $solution->getFileName();
00143             $basename = basename($fileName);
00144             $fullText = htmlentities($solution->getFullText(), ENT_QUOTES);
00145             $popup = self::mkPopup($fileName);
00146             $tabSoln .= "<td>
00147             <form method='post' action='?do'
00148             target='$fileName' id='$fileName'>
00149             <input type='submit' name='viewfile' title='$fileName'
00150             value='$basename' $popup class='$fileClass'>
00151             <input type='hidden' name='do' value='viewFile'>
00152             <input type='hidden' name='filename' value='$fileName'>
00153             <input type='hidden' name='fulltext' value='$fullText'>
00154             </form></td>
00155             ";
00156         }
00157         $closeTable = "</td></tr></table>";
00158         $tabRes .= $closeTable;
00159         $tabSoln .= $closeTable;
00160         $srcDoc .= HT::warn("<table align='center'>
00161         <tr><td align='center'>$tabRes</td><td> </td></tr>
00162         <tr><td colspan='100%'> </td></tr>
00163         <tr><td align='center'>$tabSoln</td></tr>
00164         </table>", true);
00165         $srcDoc = HT::success($srcDoc, true);
00166         $rubSrc = "<form method='post' action='?do' target='right2'
00167         id='$subQuestionName'>
00168         <table width='90%' align='center' class='grader'>
00169         <tr><th>Award</th><th>Ref</th><th>Rubric</th></tr>";
00170         // $rubrics = self::$rubricDao->getByObject($this);
00171         $awarded = self::$marksDao->getByObject($student);
00172         // var_dump($awarded);
00173         $totAwarded = $totQuestion = 0.0;
00174         $showAwarded = false;
00175         foreach ($rubrics as $k => $rubric) {
00176             $step = 0.025;
00177             $marks = $rubric->getMarks();
00178             $rubricName = $rubric->getRubricName();
00179             $rubricText = $rubric->getRubricText();
00180             $min = min([$marks, 0]);
00181             $max = max([$marks, 0]);
00182             $totQuestion += $max; // Do not count negative marks
00183             if ($min < 0) {
00184                 $step = -$min / 4;
00185                 $min = -5 * $step;
00186                 $max = -$min;
00187             }
00188             $rubricText = htmlentities($rubricText, ENT_QUOTES);
00189             $value = "";
00190             foreach ($awarded as $a) {
00191                 if ($a->getRubricName() == $rubricName) {
00192                     $showAwarded = true;
00193                     $value = number_format($a->getMarks(), 3);
00194                     $totAwarded += $value;
00195                 }
00196             }
00197             if ($value == $max) {
00198                 $class = "success";
00199             } else if ($value > 0) {
00200                 $class = "warn";
00201             } else if ($value === '') {
00202                 $class = "plain";
00203             } else {
00204                 $class = "error";
00205             }
00206             $rubSrc .= "<tr class='$class'><td>
00207             <input name='$rubricName' type='number'
00208             value='$value' min='$min' max='$max' step='$step'>
00209             <input name='full-$rubricName' type='hidden' value='$max'>
00210             </td>
00211             <td> $marks </td>
00212             <td style='text-align:left;'> $rubricName:
00213             <code>$rubricText</code></td>
```

```
00214             </tr>";
00215         }
00216         if ($showAwarded) {
00217             $rubSrc .= "<tr><th>$totAwarded</th><th>$totQuestion</th><th>";
00218         } else {
00219             $rubSrc .= "<tr><th colspan='100%'>";
00220         }
00221         $rubSrc .= "<input type='hidden' name='do' value='putMarks'>
00222         <input type='submit' value='Update Marks'
00223         name='putMarks' align='center'>
00224         <input type='submit' value='Award Full Marks'
00225         name='putFullMarks' align='center'>
00226         </th></tr></table>
00227         <input type='hidden' name='name' value='{$student->getName()}'>
00228         <input type='hidden' name='student_email' value='{$student->getEmail()}'>
00229         <input type='hidden' name='subquestion_name' value='$subQuestionName'>
00230         </form></div>";
00231         // $rubSrc = htmlentities($rubSrc);
00232         $srcDoc .= HT::success($rubSrc, true);
00233         $srcDoc = HT::info($srcDoc, true);
00234         return $srcDoc;
00235     }
```

References HT::$class, $marks, $name, HT::info(), HT::success(), and HT::warn().

Here is the call graph for this function:



4.21.3 Member Data Documentation

$marks

`SubQuestion::$marks [private]`

Definition at line 16 of file SubQuestion.php.

Referenced by getMarks(), and render().

$name

`SubQuestion::$name [private]`

Definition at line 13 of file SubQuestion.php.

Referenced by __construct(), getName(), and render().

$questionNum

`SubQuestion::$questionNum [private]`

Definition at line 19 of file SubQuestion.php.

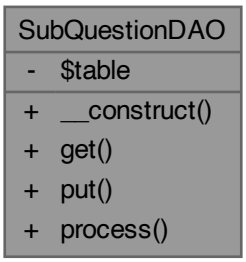
Referenced by __construct(), and getQuestionNum().

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

- SubQuestion.php

4.22 SubQuestionDAO Class Reference

Collaboration diagram for SubQuestionDAO:



Public Member Functions

- __construct ()
- get ($where=[ ], $verbose=false)
- put ($subQuestions)
- process ($question)

Static Private Attributes

- static $table

4.22.1 Detailed Description

Class SubQuestionDAO

Handles the CRUD operations for subquestions associated with each question. Typically used to insert and retrieve subquestion metadata such as marks and names.

NOTE: This class may not be actively used but is kept for possible future extensions.

Definition at line 11 of file SubQuestionDAO.php.

4.22.2 Member Function Documentation

__construct()

```
SubQuestionDAO::__construct ()
```

Initializes the table name for subquestions.

Definition at line 19 of file SubQuestionDAO.php.

```
00020     {
00021         self::$table = CFG::mkTableName("subquestions");
00022     }
```

References CFG::mkTableName().

Here is the call graph for this function:

get()

```
SubQuestionDAO::get (
              $where = [],
              $verbose = false)
```

Retrieves subquestions based on given conditions.

Parameters

array	*$where*	Optional associative array for SQL WHERE clause.
bool	*$verbose*	Display an error if no subquestions are found.

Returns

SubQuestion[] List of SubQuestion objects.

Definition at line 31 of file SubQuestionDAO.php.

```
00032     {
00033         $table = self::$table;
00034         $subQuestions = [];
00035         $rows = DB::$db->get(
00036             table: $table,
00037             where: $where
00038         );
00039         $got = count($rows);
00040         if ($verbose && $got <= 0) { // Not found in the DB
00041             HT::error("<strong>SubQuestionDAO</strong>:
00042             No subQuestions in the DB!<br>
00043             Run SubQuestionDAO->process(question) for each question first.");
```

```
00044             return;
00045         } else {
00046             foreach ($rows as $row) {
00047                 $subQuestions[] = new SubQuestion(
00048                     $row['subquestion_name'],
00049                     $row['marks'],
00050                     $row['question_num']
00051                 );
00052             }
00053         }
00054         return $subQuestions;
00055     }
```

References DB::$db, $table, and HT::error().

Here is the call graph for this function:

process()

```
SubQuestionDAO::process (
              $question)
```

Processes and stores subquestions associated with a given question.

Parameters

Question	*$question*	Question object containing subquestions.

Returns

void

Definition at line 84 of file SubQuestionDAO.php.

```
00085     {
00086         $subQuestions = $question->getSubQuestions();
00087         $this->put($subQuestions);
00088     }
```

References put().

Here is the call graph for this function:

put()

```
SubQuestionDAO::put (
            $subQuestions)
```

Inserts or updates subquestions in the database.

Parameters

SubQuestion[ ]	*$subQuestions*	Array of SubQuestion objects to be saved.

Returns

void

Definition at line 63 of file SubQuestionDAO.php.

```
00064     {
00065         $table = self::$table;
00066         $rows = [];
00067         foreach ($subQuestions as $subQuestion) {
00068             $subQuestionName = $subQuestion->getName();
00069             $questionNum = $subQuestion->getQuestionNum();
00070             $marks = $subQuestion->getMarks();
00071             $columns = ['subquestion_name', 'question_num', 'marks'];
00072             $rows[] = [$subQuestionName, $questionNum, $marks];
00073         }
00074         // DB::$db->put($table, $columns, $rows);
00075         DB::$db->update($table, $columns, $rows);
00076     }
```

References DB::$db, and $table.

Referenced by process().

Here is the caller graph for this function:

4.22.3 Member Data Documentation

$table

```
SubQuestionDAO::$table  [static], [private]
```

Definition at line 14 of file SubQuestionDAO.php.

Referenced by get(), and put().

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

- SubQuestionDAO.php

5 File Documentation

5.1 config.php File Reference

Variables

- $root = realpath(__DIR__ . "/../FT0")

5.1.1 Variable Documentation

$root

```
$root = realpath(__DIR__ .  "/../FT0")
```

Definition at line 8 of file config.php.

Referenced by HT::setRoot().

5.2 config.php

Go to the documentation of this file.

```
00001 <?php
00002 ini_set('max_execution_time', '600');
00003 ini_set('memory_limit', '1024M'); // 1GB!
00004 ini_set('xdebug.var_display_max_data', 4024);
00005
00006 // Folder where all your files for this lab/final test are.
00007 // Grader expects to find a config.php there.
00008 $root = realpath(__DIR__ . "/../FT0");
00009
00010 // Load the config from the folder being graded
00011 require_once "$root/config.php";
```

5.3 index.php File Reference

5.4 index.php

Go to the documentation of this file.

```
00001 <?php
00002 require_once("model/autoload.php");
00003 require_once("config.php");
00004
00005 HT::index();
```

5.5 Answer.php File Reference

Classes

- class Answer

5.6 Answer.php

Go to the documentation of this file.

```
00001 <?php
00002
00014 class Answer extends HT
00015 {
00019     private $studentEmail;
00020
00024     private $subQuestionName;
00025
00029     private $fileName;
00030
00034     private $fullText;
00035
00039     private $hash0;
00040
00044     private $hash1;
00045
00049     private $comment;
00050
00054     private $shortFileName;
00055
00067     public function __construct(
00068         $studentEmail,
00069         $subQuestionName,
00070         $fileName,
00071         $fullText = '',
00072         $hash0 = '',
00073         $hash1 = '',
00074         $comment = ''
00075     ) {
00076         $this->studentEmail = $studentEmail;
00077         $this->subQuestionName = $subQuestionName;
00078         $this->fileName = $fileName;
00079         if ($fullText) {
00080             $this->fullText = $fullText;
00081         } else {
00082             $this->fullText = file_get_contents($this->fileName);
00083         }
00084         if ($hash0) {
00085             $this->hash0 = $hash0;
00086         } else {
00087             $this->hash0 = Report::hash($this->fileName);
00088         }
00089         if ($hash1) {
00090             $this->hash1 = $hash1;
00091         } else {
00092             $this->hash1 = Report::hash($this->fileName, false);
00093         }
00094         $this->comment = $comment;
00095         $student = self::$studentDao->getByEmail($studentEmail)[0];
00096         $studentFolder = $student->getAnswerFolder();
00097         $this->shortFileName = trim(
00098             str_replace($studentFolder, "", $fileName),
00099             '/'
00100         );
00101     }
00102
00108     private static function mkCommentPopup()
00109     {
00110         $file = "editComment";
00111         return "onclick=
00112         \"var h = screen.height * 0.3;
00113         var t = screen.height * 0.05;
00114         var w = screen.width * 0.25;
00115         var l = screen.width * 0.65;
00116         window.open('', '$file', 'width='+ w +',height=' + h + ',left='+ l +',top='+ t
    +',resizable=yes,scrollbars=yes,toolbar=yes,menubar=no,location=no,directories=no,status=yes');
00117           document.getElementById('$file').submit();
00118           return false;\"";
00119     }
00120
00128     public function render($firstAnswer = "")
00129     {
00130         // var_dump($this); return;
00131         $subQuestionName = $this->subQuestionName;
00132         $exam = self::$exam;
00133         $gradable = false;
00134         // Grade the question only if it is in CFG::$gradeQuestions
00135         foreach ($exam->getQuestions() as $question) {
00136             if (in_array($question->getNum(), CFG::$gradeQuestions)) {
00137                 foreach ($question->getSubQuestions() as $subQuestion) {
00138                     if ($subQuestionName == $subQuestion->getName()) {
00139                         $gradable = true;
```

```
00140                         break 2;
00141                     }
00142                 }
00143             }
00144         }
00145         if (!$gradable) {
00146             return "";
00147         }
00148         $autoGraders = [];
00149         $autoFiles = [];
00150         static $autoDone = [];
00151         $student = self::$studentDao->getByObject($this)[0];
00152         // $submissionFolder = $student->getAnswerFolder();
00153         // foreach (CFG::$autoGrade as $grader => $files) {
00154         //  $autoGrader = $submissionFolder . $grader;
00155         //  $autoGraders[] = $autoGrader;
00156         //  $autoFiles = array_merge($autoFiles, $files);
00157         // }
00158 
00159         $answerFolder = $student->getAnswerFolder();
00160         foreach (CFG::$autoGradables as $answerFile => $autoGrade) {
00161             $autoGrader = $autoGrade['grader'];
00162             $autoGraders[] = $answerFolder . $autoGrader;
00163             $autoFiles[] = $answerFile;
00164         }
00165 
00166         $fileName = $this->fileName;
00167         $shortFileName = $this->shortFileName;
00168         $basename = basename($fileName);
00169         if (!self::$refFileDao->isGradable($shortFileName)) {
00170             return "";
00171         }
00172         $fullText = htmlentities($this->fullText, ENT_QUOTES);
00173         $popup = htmlentities(self::mkCommentPopup(), ENT_QUOTES);
00174         $td = "";
00175         $tdFileDB = "<td align='center'>
00176         <form method='post' action='?do' target='right2'>
00177         <input type='submit' $firstAnswer name='viewfile'
00178         title='$fileName' value='$basename: DB' >
00179         <input type='hidden' name='do' value='viewFile'>
00180         <input type='hidden' name='student_email' value='$this->studentEmail'>
00181         <input type='hidden' name='subquestion_name'
00182         value='$subQuestionName'>
00183         <input type='hidden' name='filename' value='$fileName'>
00184         <input type='hidden' name='showDB' value='Yes'>
00185         <input type='hidden' name='fulltext' value='$fullText'>
00186         </form></td>
00187         ";
00188         $tdFileHDD = "<td align='center'>
00189         <form method='post' action='?do' target='right2'>
00190         <input type='submit' $firstAnswer name='viewfile'
00191         title='$fileName' value='$basename' >
00192         <input type='hidden' name='do' value='viewFile'>
00193         <input type='hidden' name='commentable' value='$fileName'>
00194         <input type='hidden' name='popup' value='$popup'>
00195         <input type='hidden' name='student_email' value='$this->studentEmail'>
00196         <input type='hidden' name='subquestion_name'
00197         value='$subQuestionName'>
00198         <input type='hidden' name='filename' value='$fileName'>
00199         <input type='hidden' name='fulltext' value='$fullText'>
00200         </form></td>
00201         ";
00202         if (in_array($basename, $autoFiles)) {
00203             foreach ($autoGraders as $autoGrader) {
00204                 if (!in_array($autoGrader, $autoDone)) {
00205                     $td .= $tdFileHDD;
00206                     $autoDone[] = $autoGrader;
00207                     $graderURL = str_replace(realpath($_SERVER['DOCUMENT_ROOT']), "", $autoGrader);
00208                     $td .= "<td align='center'>
00209                     <a href='$graderURL' target='right2'><input type='button'
00210                     $firstAnswer value='$basename: Auto Grade'></a></td>";
00211                     break;
00212                 }
00213             }
00214             if (CFG::isQuiz()) {
00215                 $td .= $tdFileDB . $tdFileHDD;
00216             }
00217         } else {
00218             $td .= $tdFileHDD;
00219         }
00220         return $td;
00221     }
00222 
00228     public function getStudentEmail()
00229     {
00230         return $this->studentEmail;
00231     }
```

```
00232
00238     public function getSubQuestionName()
00239     {
00240         return $this->subQuestionName;
00241     }
00242
00248     public function getFullText()
00249     {
00250         return $this->fullText;
00251     }
00252
00258     public function getFileName()
00259     {
00260         return $this->fileName;
00261     }
00262
00268     public function getComment()
00269     {
00270         return $this->comment;
00271     }
00272
00279     public function setComment($comment): self
00280     {
00281         $this->comment = $comment;
00282         return $this;
00283     }
00284
00290     public function getHash0()
00291     {
00292         return $this->hash0;
00293     }
00299     public function getHash1()
00300     {
00301         return $this->hash1;
00302     }
00303
00309     public function getShortFileName()
00310     {
00311         return $this->shortFileName;
00312     }
00313 }
```

5.7 autoload.php File Reference

Functions

- dbg ()
- vd (... $args)

5.7.1 Function Documentation

dbg()

```
dbg ()
```

Outputs a stack trace using XDebug's stack printer if available, otherwise falls back to PHP's debug_backtrace.

Returns

void

Definition at line 38 of file autoload.php.

```
00039 {
00040     if (function_exists('xdebug_print_function_stack')) {
00041         xdebug_print_function_stack("Stack Trace", XDEBUG_STACK_NO_DESC);
00042     } else {
00043         debug_backtrace();
00044     }
00045 }
```

vd()

```
vd (
              ...)
```

Alias for var_dump that accepts multiple arguments.

Example: vd($var1, $var2, ...);

Parameters

mixed	...$args One or more variables to dump.

Returns

void

Definition at line 55 of file autoload.php.

```
00056 {
00057     var_dump(...$args);
00058 }
```

5.8 autoload.php

Go to the documentation of this file.

```
00001 <?php
00002
00012 spl_autoload_register(
00013     function ($class) {
00014         $folders = [
00015             "model",
00016             "model/DAO",
00017             "../model",
00018             "../model/DAO",
00019             "../../model",
00020             "../../model/DAO"
00021         ];
00022         foreach ($folders as $folder) {
00023             $file = "$folder/$class.php";
00024             if (file_exists($file)) {
00025                 require_once($file);
00026                 return; // Stop after first match
00027             }
00028         }
00029     }
00030 );
00031
00038 function dbg()
00039 {
00040     if (function_exists('xdebug_print_function_stack')) {
00041         xdebug_print_function_stack("Stack Trace", XDEBUG_STACK_NO_DESC);
00042     } else {
00043         debug_backtrace();
00044     }
00045 }
00046
00055 function vd(...$args)
00056 {
00057     var_dump(...$args);
00058 }
```

5.9 CFG.php File Reference

Classes

- class CFG

5.10 CFG.php

Go to the documentation of this file.

```
00001 <?php
00002
00014 class CFG extends HT
00015 {
00016   // ------------ Imported from config.php ------------
00017   // Exam details
00019   public static $examShortName;
00020
00022   public static $examLongName;
00023
00025   public static $mode = "LT";
00026
00028   public static $sectionNames;
00029
00031   private static $csvFileNamePattern;
00032
00034   public static $locSubQ;
00035
00037   public static $locMarks;
00038
00040   public static $rubricFile = "./rubric.csv";
00041
00043   private static $submissionFolderPattern;
00044
00046   public static $emailFolder;
00047
00049   private static $answerFolderPattern;
00050
00052   public static $solutionFolder;
00053
00055   public static $resourceFolder;
00056
00057   // ------------ End of imports from config.php ------------
00059   private static $vars = [
00060     "examShortName",
00061     "examLongName",
00062     "mode",
00063     "sectionNames",
00064     "csvFileNamePattern",
00065     "locSubQ",
00066     "locMarks",
00067     "rubricFile",
00068     "submissionFolderPattern",
00069     "answerFolderPattern",
00070     "emailFolder",
00071     "solutionFolder",
00072     "resourceFolder",
00073     "gradeQuestions",
00074     "autoGradables",
00075     "dbUser",
00076     "dbPass",
00077     "dbName"
00078   ];
00079
00081   public static $configFile = "../config.php";
00082
00084   private static $sqlTemplate = "../model/setup.template.sql";
00085
00087   private static $sqlFile = 'setup.sql';
00088
00090   private static $root;
00091
00093   public static $gradeQuestions;
00094
00096   public static $autoGradables;
00097
00099   public static $dbUser;
00100
00102   public static $dbPass;
00103
00105   public static $dbName;
00106
00108   public static $dbPrefix;
00109
00111   public static $numQuestions;
00112
00113   protected static $class = __CLASS__;
00114
00115   // Do not modify below this line
00116   protected static $actions = [
00117     'view' => 'View Configuration',
00118     'validate' => 'Validate Config and Folders',
```

```
00119     'viewSql' => 'View SQL File',
00120     'runSql' => 'Setup Database',
00121     'dropDB' => 'Drop Database',
00122   ];
00123   protected static $actionProperties = [
00124     'view' => [
00125       'target' => 'left',
00126       'clear' => 'yes'
00127     ],
00128     'validate' => [
00129       'target' => 'right'
00130     ],
00131     'viewSql' => [
00132       'target' => 'left',
00133       'clear' => 'yes'
00134     ],
00135     'runSql' => [
00136       'target' => 'right'
00137     ],
00138     'dropDB' => [
00139       'class' => 'error',
00140       'target' => 'right'
00141     ]
00142   ];
00143   protected static $actionDesc = [
00144     'view' => 'View (not edit) the current settings in your <code>config.php</code> file.',
00145     'validate' => 'Validate your configuration in <code>config.php</code> file and check for the
      existence an contents of the folders specified.',
00146     'viewSql' => 'Examine the SQL file that will be run to set up your database. Do not edit this
      file. It is generated based on your <code>config.php</code> and a template',
00147     'runSql' => 'Run the generated SQL file to set up the database. If you see a red box about errors
      in the database, try this button. Clicking the button again is harmless: It does not reset the
      database.',
00148     'dropDB' => 'Drop all database tables and restart: This button will really reset the database. If
      you have graded some students, all the entered grades will really disappear.'
00149   ];
00150   protected static $intro = "<h4>Configuration Menu</h4>
00151   This subPage allows you to view and validate your configuration file.
00152   Validation means checking for the existence of the necessary folders and files.
00153   This is also the place where you will set up the database tables. If necessary,
00154   you can reset your DB also here. See the descriptions of what the buttons do
00155   on the right.";
00156   protected static $display = [
00157     "class" => "plain",
00158     "title" => "Setup and Configuration"
00159   ];
00160
00167   public static function init()
00168   {
00169     $vars = self::$vars;
00170     foreach ($vars as $v) {
00171       if (isset($GLOBALS[$v])) { // Necessary for $mode = "ft"
00172         self::$$v = $GLOBALS[$v];
00173       }
00174     }
00175     self::$dbName = strtolower(self::$dbName);
00176     $dbHost = "localhost";
00177     $db = new DB($dbHost, self::$dbUser, self::$dbPass, self::$dbName);
00178     self::$dbPrefix = self::getDbPrefix();
00179     self::$configFile = realpath(self::$configFile);
00180     self::$root = $GLOBALS['root'];
00181   }
00182
00189   public static function setNumQuestions($numQuestions)
00190   {
00191     self::$numQuestions = $numQuestions;
00192   }
00193
00201   public static function getCsvFileName($sectionName, $studentName = "")
00202   {
00203     return str_replace(
00204       ["{sectionName}", "{studentName}"],
00205       [trim($sectionName), trim($studentName)],
00206       self::$csvFileNamePattern
00207     );
00208   }
00209
00217   public static function getSubmissionFolder($sectionName, $studentName = "")
00218   {
00219     return str_replace(
00220       ["{sectionName}", "{studentName}"],
00221       [trim($sectionName), trim($studentName)],
00222       self::$submissionFolderPattern
00223     );
00224   }
00225
00233   public static function getAnswerFolder($sectionName, $studentName = "")
```

```
00234   {
00235     return str_replace(
00236       ["{sectionName}", "{studentName}"],
00237       [trim($sectionName), trim($studentName)],
00238       self::$answerFolderPattern
00239     );
00240   }
00241
00247   public static function isQuiz()
00248   {
00249     return trim(strtolower(self::$mode)) == "ft";
00250   }
00251
00257   public static function getQDirPattern()
00258   {
00259     // Create a regular expression like "@/q[123]/..@
00260     $pattern = "@/[qQ][";
00261     for ($i = 1; $i <= self::$numQuestions; $i++) {
00262       $pattern .= $i;
00263     }
00264     $pattern .= "]/..@";
00265     return $pattern;
00266   }
00267
00268
00269   // Dispatch handlers
00270
00271
00277   public static function view()
00278   {
00279     $configFile = self::$configFile;
00280     $ext = pathinfo($configFile, PATHINFO_EXTENSION);
00281     $configFileContents = file_get_contents($configFile);
00282     try {
00283       $includedConfigFile = explode("require_once ", $configFileContents)[1];
00284       $includedConfigFile = explode('"', $includedConfigFile)[1];
00285       $includedConfigFile = str_replace('$root', self::$root, $includedConfigFile);
00286       $includedConfigFileContent = file_get_contents($includedConfigFile);
00287     } catch (Exception $e) {
00288       echo $e->getMessage();
00289     }
00290     $fullText0 = htmlentities($configFileContents);
00291     $fullText1 = htmlentities($includedConfigFileContent);
00292     $realPath0 = "<div
00293       style='color: #373;
00294       background: #ecffd6;
00295       border: 1px solid #617c42;
00296       padding:4px;
00297       font-weight:bold;
00298       text-align:left;'><code>" .
00299       realpath($configFile) . "</code></div>";
00300     $realPath1 = "<div
00301       style='color: #373;
00302       background: #ecffd6;
00303       border: 1px solid #617c42;
00304       padding:4px;
00305       font-weight:bold;
00306       text-align:left;'><code>" .
00307       realpath($includedConfigFile) . "</code></div>";
00308     HT::codeHead($configFile);
00309     echo "$realPath0<pre><code class='language-$ext'>$fullText0</code></pre>";
00310     echo "$realPath1<pre><code class='language-$ext'>$fullText1</code></pre>";
00311     HT::foot();
00312   }
00313
00319   public static function validate()
00320   {
00321     HT::head();
00322     $error = false;
00323     $configFile = self::$configFile;
00324     $str = HT::info(
00325       "Validating <strong>" . self::$examShortName . ": " .
00326         self::$examLongName . "</strong><br>
00327       Config file: <a href='?do=view'><code>$configFile</code></a>",
00328       true
00329     );
00330     $root = self::$root;
00331     $realpath = realpath($root);
00332     if (!is_dir($root)) {
00333       $str .= HT::error("Exam folder (<code>$realpath</code>) not found!<br>
00334       Edit <code>$configFile</code> to set <code>\$root.</code>", true);
00335       $error = true;
00336       goto end;
00337     } else {
00338       $str .= HT::success("Exam folder (<code>$realpath</code>) located.", true);
00339     }
00340     if (self::isQuiz()) {
```

```
00341       $str .= HT::success(
00342         "Grading a finals (an eLearn quiz, as opposed to a Lab Test)!",
00343         true
00344       );
00345       self::$sectionNames = ['All'];
00346     }
00347     if (empty(self::$sectionNames)) {
00348       $str .= HT::error(
00349         "Section names (<code>\$sectionNames</code>) not specified!<br>
00350         Edit <code>$configFile</code> to set it as an array.",
00351         true
00352       );
00353       $error = true;
00354       goto end;
00355     } else {
00356       $str1 = implode(", ", self::$sectionNames);
00357       $str .= HT::success(
00358         "Found these section[s]: <strong>$str1</strong>.",
00359         true
00360       );
00361       $str0 = "";
00362       foreach (self::$sectionNames as $sectionName) {
00363         $csvFileName = self::getCsvFileName($sectionName);
00364         if (is_file($csvFileName)) {
00365           $csvFileName = realpath($csvFileName);
00366           $str0 .= HT::success(
00367             " &rarr; eLearn Grade Export file located:<br>
00368              <code>$csvFileName</code>",
00369             true
00370           );
00371         } else {
00372           $str0 .= HT::error(
00373             " &rarr; eLearn Grade Export file  not found.<br>
00374              <code>$csvFileName</code><br>
00375             Please create it: Export the grade for the section $sectionName as CSV.",
00376             true
00377           );
00378           $error = true;
00379         }
00380       }
00381       if ($error) {
00382         $str .= HT::warn($str0, true);
00383       } else {
00384         $str .= HT::info($str0, true);
00385       }
00386       $str0 = "";
00387       foreach (self::$sectionNames as $sectionName) {
00388         $submissionFolder = self::getSubmissionFolder($sectionName);
00389         if (is_dir($submissionFolder)) {
00390           $submissionFolder = realpath($submissionFolder);
00391           $nZips = count(glob($submissionFolder . "/*.zip"));
00392           if ($nZips > 0) {
00393             $str0 .= HT::success(
00394               " &rarr; Found $nZips student submissions in<br>
00395                <code>$submissionFolder</code>",
00396               true
00397             );
00398           } else {
00399             $str0 .= HT::warn(
00400               " &rarr; No zip files in the student submission folder
00401                <code>$submissionFolder</code>",
00402               true
00403             );
00404             $error = true;
00405           }
00406         } else {
00407           $str0 .= HT::error(
00408             " &rarr; Student Submission folder not found:<br>
00409              <code>$submissionFolder</code> <br>
00410             Please download the student submissions from $sectionName
00411             and copy them there.",
00412             true
00413           );
00414           $error = true;
00415         }
00416       }
00417       foreach (self::$sectionNames as $sectionName) {
00418         $answerFolder = self::getanswerFolder($sectionName);
00419         if (is_dir($answerFolder)) {
00420           $answerFolder = realpath($answerFolder);
00421         } else {
00422           $str0 .= HT::error(
00423             " &rarr; Student answer folder not found:<br>
00424              <code>$answerFolder</code> <br>
00425             Please create it so that their zip files can be unzipped.",
00426             true
00427           );
```

```
00428           $error = true;
00429         }
00430       }
00431       if ($error) {
00432         $str .= HT::warn($str0, true);
00433       } else {
00434         $str .= HT::info($str0, true);
00435       }
00436       $rubricFile = self::$rubricFile;
00437       if (!is_file($rubricFile)) {
00438         $str .= HT::error(
00439           "Rubric file not found: <br>
00440            <code>$rubricFile</code><br>
00441           Create it before proceeding.",
00442           true
00443         );
00444         $error = true;
00445       } else {
00446         $rubricFile = realpath($rubricFile);
00447         $str .= HT::success(
00448           "Rubric File located:<br>
00449            <code>$rubricFile</code><br>",
00450           true
00451         );
00452       }
00453       $emailFolder = self::$emailFolder;
00454       if (empty($emailFolder) || !is_dir($emailFolder)) {
00455         $str .= HT::warn(
00456           "Email Submission folder not found. <br>
00457            <code>$emailFolder</code><br>
00458           Create it and copy the emailed zip files there.",
00459           true
00460         );
00461       } else {
00462         $emailFolder = realpath(self::$emailFolder);
00463         $nZips = count(glob($emailFolder . "/*.zip"));
00464         $str .= HT::success(
00465           "Email Submission folder located with $nZips files:<br>
00466            <code>$emailFolder</code>",
00467           true
00468         );
00469       }
00470     }
00471     $solutionFolder = self::$solutionFolder;
00472     if (!is_dir($solutionFolder)) {
00473       $str .= HT::error(
00474         "Solution folder not found: <br>
00475          <code>$solutionFolder</code><br>
00476         Create it and copy the reference solutions there.",
00477         true
00478       );
00479       $error = true;
00480     } else {
00481       $solutionFolder = realpath($solutionFolder);
00482       $gPattern = "$solutionFolder/{*,*/*,*/*/*}.*";
00483       $nFiles = count(glob($gPattern, GLOB_BRACE));
00484       if ($nFiles <= 0) {
00485         $str .= HT::warn(
00486           "Solution folder is empty: <br>
00487          <code>$solutionFolder</code><br>
00488         Copy the reference solutions there.",
00489           true
00490         );
00491       } else {
00492         $str .= HT::success(
00493           "Solution folder located with $nFiles files:<br>
00494          <code>$solutionFolder</code>",
00495           true
00496         );
00497       }
00498     }
00499     $resourceFolder = self::$resourceFolder;
00500     if (!is_dir($resourceFolder)) {
00501       $str .= HT::error(
00502         "Resource folder not found: <br>
00503          <code>$resourceFolder</code><br>
00504         Create it and copy the resource files there.",
00505         true
00506       );
00507       $error = true;
00508     } else {
00509       $resourceFolder = realpath($resourceFolder);
00510       $gPattern = "$resourceFolder/{*,*/*,*/*/*}.*";
00511       $nFiles = count(glob($gPattern, GLOB_BRACE));
00512       if ($nFiles <= 0) {
00513         $str .= HT::warn(
00514           "Resource folder is empty: <br>
```

```
00515          <code>$resourceFolder</code><br>
00516         Copy the resource files there.",
00517           true
00518         );
00519       } else {
00520         $str .= HT::success(
00521           "Resource folder located with $nFiles files:<br>
00522          <code>$resourceFolder</code>",
00523           true
00524         );
00525       }
00526     }
00527     $gradeQuestions = self::$gradeQuestions;
00528     if (empty($gradeQuestions)) {
00529       $str .= HT::error(
00530         "No questions are marked gradable (<code>\$gradeQuestions</code>)! <br>
00531         Edit <code>$configFile</code> to set it as an array.",
00532         true
00533       );
00534       $error = true;
00535     } else {
00536       $str .= HT::success(
00537         "Will assist in grading questions <strong>"
00538           . implode(", ", $gradeQuestions) . "</strong>",
00539         true
00540       );
00541     }
00542     $autoGradables = self::$autoGradables;
00543     if (empty($autoGradables)) {
00544       $str .= HT::warn(
00545         "No questions are marked for autograding (<code>\$autoGradables</code>)!<br>
00546         Edit <code>$configFile</code> to set it if needed.",
00547         true
00548       );
00549     } else {
00550       foreach ($autoGradables as $file => $autoGraded) {
00551         $autoGraderBase = $autoGraded['grader'];
00552         $autoGrader = "$solutionFolder/$autoGraderBase";
00553         $subQuestionName = $autoGraded['subQuestionName'];
00554         if (file_exists($autoGrader)) {
00555           $str .= HT::success(
00556             "Autograder <code>$autoGraderBase</code> will grade <code>$subQuestionName
      [$file]</code>",
00557             true
00558           );
00559         } else {
00560           $str .= HT::error(
00561             "Autograder <code>$autoGraderBase</code> for <code>$subQuestionName [$file]</code> not
      found.<br>
00562             Copy it to <code>$solutionFolder</code>",
00563             true
00564           );
00565           $error = true;
00566         }
00567         $autoHelpers = $autoGraded['helpers'] ?? [];
00568         foreach ($autoHelpers as $helper) {
00569           $helperFull = "$solutionFolder/$helper";
00570           if (file_exists($helperFull)) {
00571             $str .= HT::success(
00572               "Helper file <code>$helper</code> for <code>$subQuestionName [$file]</code> located",
00573               true
00574             );
00575           } else {
00576             $str .= HT::error(
00577               "Helper file <code>$helper</code> for <code>$subQuestionName [$file]</code> not
      found.<br>
00578             Copy it to <code>$solutionFolder</code>",
00579               true
00580             );
00581             $error = true;
00582           }
00583         }
00584       }
00585     }
00586     end:
00587     if ($error) {
00588       $configFile = self::$configFile;
00589       $str .= HT::error(
00590         "Fix the errors (red boxes) and validate again.<br>
00591          The comments in <a href='?do=view'><code>$configFile</code></a> may help.",
00592         true
00593       );
00594     }
00595     HT::warn($str);
00596     HT::foot();
00597   }
00598
```

```
00604   public static function viewSql()
00605   {
00606     $fileName = self::$root . DIRECTORY_SEPARATOR . self::$sqlFile;
00607     self::mkSql();
00608     $ext = pathinfo($fileName, PATHINFO_EXTENSION);
00609     $fullText = htmlentities(file_get_contents($fileName), ENT_QUOTES);
00610     HT::codeHead($fileName);
00611     echo "<pre><code class='language-$ext'>$fullText</code></pre>";
00612     HT::foot();
00613   }
00614
00621   public static function rmdir($dir)
00622   {
00623     if (is_dir($dir)) {
00624       $files = array_diff(scandir($dir), array('.', '..'));
00625       foreach ($files as $file) {
00626         (is_dir("$dir/$file")) ? self::rmdir("$dir/$file") : unlink("$dir/$file");
00627       }
00628       return rmdir($dir);
00629     }
00630   }
00631
00639   public static function mv($from, $to)
00640   {
00641     $toFolder = dirname($to);
00642     if (!is_dir($toFolder)) {
00643       mkdir($toFolder, 0777, true);
00644     }
00645     if (is_dir($to)) {
00646       $to .= "/" . $from;
00647     }
00648     // Make a subfolder if needed
00649     $error = "Error moving <code>$from</code> to <code>$to</code>:";
00650     if (is_dir($from)) {
00651       HT::error("$error Source is a folder.");
00652       return false;
00653     }
00654     if (!is_readable($from)) {
00655       HT::error("$error Source not found.");
00656       return false;
00657     }
00658     if (!rename($from, $to)) {
00659       if (copy($from, $to)) {
00660         if (!unlink($from)) {
00661           HT::error("Error moving $from to $to: Could not remove source.");
00662           return false;
00663         }
00664       } else {
00665         HT::error("Error moving $from to $to: Could not copy source.");
00666         return false;
00667       }
00668     }
00669     return true;
00670   }
00671
00679   public static function cpr($from, $to)
00680   {
00681     if (is_dir($from)) {
00682       $dir_handle = opendir($from);
00683       while ($file = readdir($dir_handle)) {
00684         if ($file != "." && $file != "..") {
00685           if (is_dir($from . "/" . $file)) {
00686             if (!is_dir($to . "/" . $file)) {
00687               mkdir($to . "/" . $file);
00688             }
00689             self::cpr($from . "/" . $file, $to . "/" . $file);
00690           } else {
00691             copy($from . "/" . $file, $to . "/" . $file);
00692           }
00693         }
00694       }
00695       closedir($dir_handle);
00696     } else {
00697       copy($from, $to);
00698     }
00699   }
00700
00709   public static function cp($from, $to, $mkDir = false)
00710   {
00711     $error = "Error copying <code>$from</code> to <code>$to</code>:";
00712     $error = str_replace('//', '/', $error);
00713     if (is_dir($from)) {
00714       // HT::error("$error Source is a folder.");
00715       // try cpr after creating the $to folder: Dicey!
00716       if (!is_dir($to)) {
00717         mkdir($to, 0777, true);
00718       }
```

```
00719       self::cpr($from, $to);
00720       return false;
00721     }
00722     if (!is_readable($from)) {
00723       HT::error("$error Source not found/readable.");
00724       return false;
00725     }
00726     if (is_dir($to)) {
00727       if (!file_exists($to)) {
00728         HT::error("$error Target not found/writable.");
00729         return false;
00730       }
00731       $to .= "/" . basename($from);
00732     }
00733     $toDir = dirname($to);
00734     if (!file_exists($toDir)) {
00735       if ($mkDir && mkdir($toDir)) {
00736         HT::warn("$error Target directory was not found. Created it.");
00737       } else {
00738         HT::error("$error Target directory not found/writable.");
00739         return false;
00740       }
00741     }
00742     copy($from, $to);
00743     return true;
00744   }
00745
00752   public static function mkFileName($fileName)
00753   {
00754     $root = $_SERVER['DOCUMENT_ROOT'];
00755     $fileName0 = realpath($fileName);
00756     if (stripos($fileName, $root) !== 0) { // Does not start with DocRoot
00757       HT::error("$fileName: not found under $root.");
00758     }
00759     return substr($fileName0, strlen($root));
00760   }
00761
00767   public static function getDbPrefix()
00768   {
00769     $dbPrefix = preg_replace("/\s+/", "", self::$examShortName);
00770     $dbPrefix = strtolower($dbPrefix) . "_";
00771     return $dbPrefix;
00772   }
00773
00780   public static function mkSql($toString = false)
00781   {
00782     $template = self::$sqlTemplate;
00783     $dbPrefix = self::getDbPrefix();
00784     $dbName = strtolower(self::$dbName);
00785     $search = ["{dbName}", "{dbPrefix}"];
00786     $replace = [$dbName, $dbPrefix];
00787     $sql = file_get_contents($template);
00788     $sql = str_replace($search, $replace, $sql);
00789     $sqlFile = self::$root . DIRECTORY_SEPARATOR . self::$sqlFile;
00790     HT::head($toString);
00791     if (!file_put_contents($sqlFile, $sql)) {
00792       HT::error("Error creating $sqlFile while setting up the database.<br>
00793       Cannot safely continue without fixing this issue!");
00794       return "";
00795     }
00796     HT::warn("Created $sqlFile.");
00797     HT::foot($toString);
00798     return $sql;
00799   }
00800
00806   private static function mkDropDB()
00807   {
00808     $dbPrefix = self::getDbPrefix();
00809     $dbName = strtolower(self::$dbName);
00810     $search = ["{dbName}", "{dbPrefix}"];
00811     $replace = [$dbName, $dbPrefix];
00812     $sql = 'DROP DATABASE IF EXISTS `{dbName}`;';
00813     $sql = str_replace($search, $replace, $sql);
00814
00815
00816     // Pattern for matching table names (SQL LIKE syntax)
00817     $pattern = "$dbPrefix%";
00818
00819     // SQL to get the list of tables that match the wildcard pattern
00820     $sql0 = "SELECT TABLE_NAME FROM INFORMATION_SCHEMA.TABLES WHERE TABLE_SCHEMA = '$dbName' AND
      TABLE_NAME LIKE '$pattern'";
00821
00822     $sql = "";
00823     if (DB::$db->query($sql0)) {
00824       $tables = DB::$db->fetchAll();
00825       foreach ($tables as $row) {
00826         $table = $row["TABLE_NAME"];
```

```
00827         $sql .= "DROP TABLE `$table`;\n";
00828       }
00829     } else {
00830       echo "No tables found";
00831     }
00832     // var_dump($sql); die;
00833     return $sql;
00834   }
00835
00841   public static function runSql()
00842   {
00843     $template = self::$sqlTemplate;
00844     $sql = self::mkSql();
00845     if (empty($sql)) {
00846       HT::error("Error running $template!");
00847       return false;
00848     }
00849     $sqlFile = self::$root . DIRECTORY_SEPARATOR . self::$sqlFile;
00850     $db = new DB(dbPass: self::$dbPass);
00851     $status = $db->importSQL($sqlFile);
00852     if ($status) {
00853       $msg = HT::success("Executed {$status['success']} successfully.", true);
00854       $msg .= HT::error("{$status['errors']} statements failed.", true);
00855     } else {
00856       $msg = HT::error("Error setting up the database [CFG::runSql fails].", true);
00857     }
00858     HT::head();
00859     HT::warn("<h4>Result from <code>CFG::runSQL():</code></h4>" . $msg);
00860     HT::foot();
00861   }
00862
00868
00869   public static function dropDB()
00870   {
00871     HT::head();
00872     $sql = self::mkDropDB();
00873     if (empty($sql)) {
00874       HT::error("Error creating drop DB script!");
00875       return false;
00876     }
00877     $dbName = self::$dbName;
00878     if (!isset($_POST['confirmDropDB'])) {
00879       $str = "Dropping DB ($dbName)!<br><pre><code>$sql</code></pre><br>
00880           Really drop it? <table align='right'><tr>
00881           <td><form method='post' action='?do'>
00882           <input type='hidden' name='do' value='dropDB'>
00883           <input type='hidden' name='confirmDropDB' value='dummy'>
00884           <button type='submit' class='error'>
00885           <strong>Proceed to drop it</strong>
00886           </button></form></td>
00887           </tr></table>";
00888       HT::warn($str);
00889     } else {
00890       DB::$db->multiQuery($sql);
00891       $error = DB::$db->getError();
00892       if ($error) {
00893         $error = "<br>$error";
00894       }
00895       HT::error("Dropping DB ($dbName)!<br><pre><code>$sql</code></pre>$error");
00896       HT::warn("Please run the DB setup script again.
00897       <a href='?do=runSql'>Run it!</a>");
00898       // self::runSql();
00899     }
00900     HT::foot();
00901   }
00902
00909   public static function mkTableName($table)
00910   {
00911     $pf = self::getDbPrefix();
00912     if (strpos($table, $pf) !== 0) {
00913       $table = $pf . $table;
00914     }
00915     return $table;
00916   }
00917
00925   public static function toSnake($input)
00926   {
00927     return strtolower(preg_replace('/(?<!^)[A-Z]/', '_$0', $input));
00928   }
00929
00937   public static function toCamel($input)
00938   {
00939     return lcfirst(str_replace(' ', '', ucwords(str_replace('_', ' ', $input))));
00940   }
00941 }
```

5.11 AnswerDAO.php File Reference

Classes

- class AnswerDAO

5.12 AnswerDAO.php

Go to the documentation of this file.

```
00001 <?php
00002
00009 class AnswerDAO extends DAO
00010 {
00012     private static $table;
00013
00015     private $processErrors = "";
00016
00018     private $processSummary = "";
00019
00023     public function __construct()
00024     {
00025         $table = CFG::mkTableName("answers");
00026         self::$table = $table;
00027     }
00028
00036     public function get($where = [], $verbose = false)
00037     {
00038         $table = self::$table;
00039         $answers = [];
00040         $rows = DB::$db->get(
00041             table: $table,
00042             where: $where
00043         );
00044         $got = count($rows);
00045         if ($verbose && $got <= 0) { // Not found in the DB
00046             HT::error("<strong>AnswerDAO</strong>:
00047                 No answers in the DB!<br>
00048                 Run AnswerDAO->process(student) for each student first.");
00049         } else {
00050             foreach ($rows as $row) {
00051                 $answers[] = new Answer(
00052                     $row['student_email'],
00053                     $row['subquestion_name'],
00054                     $row['filename'],
00055                     $row['fulltext'],
00056                     $row['hash0'],
00057                     $row['hash1'],
00058                     $row['comment']
00059                 );
00060             }
00061         }
00062         return $answers;
00063     }
00064
00073     public function getByObject($student, $subQuestion = null, $answer = null)
00074     {
00075         $where = ['student_email' => $student->getEmail()];
00076         if ($subQuestion) {
00077             $where['subquestion_name'] = $subQuestion->getName();
00078         }
00079         $got = $this->get($where);
00080         if ($answer) { // Use the short file name to filter, not
00081             $answers = [];
00082             foreach ($got as $a) {
00083                 if ($answer->getShortFileName() == $a->getShortFileName()) {
00084                     $answers[] = $a;
00085                 }
00086             }
00087         } else {
00088             $answers = $got;
00089         }
00090         // Sort the answers by short file name, for aesthetics
00091         $answers = self::sortByShortFileName($answers);
00092         return $answers;
00093     }
00094
00101     public function getByFileName($fileName)
00102     {
00103         $where = ['filename' => $fileName];
```

```
00104         return $this->get($where);
00105     }
00106
00114     private function getStudentZip($student)
00115     {
00116         $zipFile = "";
00117         // Try the emailed zips first
00118         $email = $student->getEmail();
00119         $section = $student->getSection();
00120         $name = $student->getName();
00121         $leadIn = $student->mkLeadIn();
00122         $gPat = CFG::$emailFolder . "*$email*.zip";
00123         $fNames = glob($gPat);
00124         $nFiles = count($fNames);
00125         if ($nFiles == 0) { // Try the section folder
00126             $gPat = CFG::getSubmissionFolder($section) . "*$email*.zip";
00127             $fNames = glob($gPat);
00128             $nFiles = count($fNames);
00129         }
00130         if ($nFiles == 0) { // Try the section folder with student name
00131             $gPat = CFG::getSubmissionFolder($section) . "*$name*.zip";
00132             $fNames = glob($gPat);
00133             $nFiles = count($fNames);
00134         }
00135         if ($nFiles > 1) {
00136             $this->processErrors .= HT::warn(
00137                 "$leadIn Found $nFiles Zip files. Will use the latest one.",
00138                 true
00139             );
00140             // Ref: https://stackoverflow.com/a/35925596
00141             usort($fNames, function ($a, $b) {
00142                 return filemtime($b) - filemtime($a);
00143             });
00144             $nFiles = 1;
00145         }
00146         if ($nFiles != 1) {
00147             $this->processErrors .= HT::error(
00148                 "$leadIn Found $nFiles Zip files. Please investigate!",
00149                 true
00150             );
00151             $student->setStatus("Absent")->update();
00152         } else {
00153             $zipFile = $fNames[0];
00154             $student->setZipFile($zipFile);
00155         }
00156         return $zipFile;
00157     }
00158
00165     private function unzipStudent($student)
00166     {
00167         $zipFile = $student->getZipFile();
00168         $zip = new ZipArchive;
00169         $res = $zip->open($zipFile);
00170         if ($res === TRUE) {
00171             CFG::rmdir("foo");
00172             $zip->extractTo("foo");
00173         } else {
00174             $leadIn = $student->mkLeadIn();
00175             $this->processErrors .= HT::error(
00176                 "$leadIn Error unzipping to <var>foo</var>!<br>
00177              [<var>$zipFile</var>]",
00178                 true
00179             );
00180             return false;
00181         }
00182         $zip->close();
00183         return true;
00184     }
00185
00192     private function cpAutoGraders($student)
00193     { // No error handling for now!
00194         $answerFolder = $student->getAnswerFolder();
00195         foreach (CFG::$autoGradables as $file => $autoGrade) {
00196             $grader = $autoGrade['grader'];
00197             $from = CFG::$solutionFolder . $grader;
00198             $to = $answerFolder . "/" . $grader;
00199             CFG::cp($from, $to, $mkDir = true);
00200             $helpers = $autoGrade['helpers'] ?? [];
00201             foreach ($helpers as $helper) {
00202                 $from = CFG::$solutionFolder . $helper;
00203                 $to = $answerFolder . "/" . $helper;
00204                 CFG::cp($from, $to, $mkDir = true);
00205             }
00206         }
00207         return true;
00208     }
00209
```

```
00216     private static function stripCommon($strings)
00217     {
00218         if (CFG::isQuiz()) {
00219             $stripped = str_replace("foo/", "", $strings);
00220         } else {
00221             // need to remove silly paths students have used
00222             $difPosition = 10000; // just a large number
00223             $count = count($strings);
00224             for ($i = 1; $i < $count; $i++) {
00225                 $position = strspn($strings[$i - 1] ^ $strings[$i], "\0");
00226                 $difPosition = min([$difPosition, $position]);
00227             }
00228             if ($strings[0][$difPosition - 1] == 'q') {
00229                 $difPosition--;
00230             }
00231             $stripped = [];
00232             foreach ($strings as $string) {
00233                 $stripped[] = substr($string, $difPosition);
00234             }
00235         }
00236         return $stripped;
00237     }
00238
00245     private function mvStudentAnswers($student)
00246     {
00247         // Iterate over the files in unzipped (in foo)
00248         $refFileDAO = new RefFileDAO();
00249         $answers = glob(
00250             "foo/{*,*/*,*/*/*,*/*/*/*,*/*/*/*/*}/*.{php,html}",
00251             GLOB_BRACE
00252         );
00253         if (empty($answers)) {
00254             return [];
00255         }
00256         $shortNames = self::stripCommon($answers);
00257         $answerFolder = $student->getAnswerFolder($student);
00258         if (realpath($answerFolder)) {
00259             $student->delete();
00260         }
00261         @mkdir($answerFolder, 0777, true);
00262         $moved = [];
00263         foreach ($answers as $key => $a) {
00264             $shortFileName = $shortNames[$key];
00265             $refFile = $refFileDAO->getRefFile($shortFileName, 'resources');
00266             if (empty($refFile)) {
00267                 $leadIn = $student->mkLeadIn();
00268                 $this->processErrors .= HT::error(
00269                     "$leadIn Created an unknown file <code>$shortFileName</code>",
00270                     true
00271                 );
00272                 continue;
00273             }
00274             $refShortFileName = $refFile->getShortFileName();
00275             $target = $answerFolder . $refShortFileName;
00276             CFG::mv($a, $target);
00277             $moved[$refShortFileName] = $target;
00278         }
00279         CFG::rmdir("foo");
00280         return $moved;
00281     }
00282
00283     // This feels like bad OO design: Need the subQuestion name in the answers
00284     // table (to facilitate grading). But the only way to get it is to use
00285     // the filename as a key in the ref_files table.
00292     private static function getSubQuestionName($shortFileName)
00293     {
00294         $rubricDao = new RubricDAO();
00295         $rubrics = $rubricDao->get([
00296             'short_filename' => $shortFileName
00297         ]);
00298         $subQuestionName = "";
00299         if (count($rubrics) > 0) {
00300             $subQuestionName = $rubrics[0]->getSubQuestionName();
00301         }
00302         return $subQuestionName;
00303     }
00304
00312     private function put($student, $answers)
00313     {
00314         $leadIn = $student->mkLeadIn();
00315         if (count($answers) <= 0) { // No answers found. An error!
00316             $this->processErrors .= HT::error(
00317                 "$leadIn No answers found.
00318             Probably absent or alias-zipping.",
00319                 true
00320             );
00321         } else {
```

```
00322             $table = self::$table;
00323             $columns = [
00324                 'student_email',
00325                 'subquestion_name',
00326                 'filename',
00327                 'fulltext',
00328                 'hash0',
00329                 'hash1',
00330             ];
00331             $rows = [];
00332             foreach ($answers as $answer) {
00333                 $studentEmail = $student->getEmail();
00334                 $subQuestionName = $answer->getSubQuestionName();
00335                 $fileName = $answer->getFileName();
00336                 $fullText = DB::$db->getFileContent($fileName);
00337                 $hash0 = Report::hash($fileName);
00338                 $hash1 = Report::hash($fileName, false);
00339                 $rows[] = [
00340                     $studentEmail,
00341                     $subQuestionName,
00342                     $fileName,
00343                     $fullText,
00344                     $hash0,
00345                     $hash1,
00346                 ];
00347             }
00348             DB::$db->update($table, $columns, $rows);
00349         }
00350     }
00351
00358     public function process($student)
00359     {
00360         $name = $student->getName();
00361         $studentEmail = $student->getEmail();
00362         $section = $student->getSection();
00363         $msg = $student->mkLeadIn();
00364         if ($zipFile = $this->getStudentZip($student)) {
00365             $dirName = realpath(dirname($zipFile));
00366             $baseName = basename($zipFile);
00367             $msg .= HT::success("Located the zip file: <br>
00368              Folder: <code>$dirName</code><br>
00369              File: <code>$baseName</code>", true);
00370             if ($this->unzipStudent($student)) {
00371                 $answerFiles = $this->mvStudentAnswers($student);
00372                 $count = count($answerFiles);
00373                 if ($count) {
00374                     $msg .= HT::success("$count files successfully unzipped.", true);
00375                 } else {
00376                     $msg .= HT::error("No files unzipped.", true);
00377                 }
00378                 $answers = [];
00379                 foreach ($answerFiles as $shortFileName => $fileName) {
00380                     $subQuestionName = self::getSubQuestionName($shortFileName);
00381                     if ($subQuestionName) {
00382                         $answers[] = new Answer(
00383                             $studentEmail,
00384                             $subQuestionName,
00385                             $fileName
00386                         );
00387                     }
00388                 }
00389                 $count = count($answers);
00390                 if ($count) {
00391                     $msg .= HT::success("$count Answer objects successfully created.", true);
00392                 } else {
00393                     $msg .= HT::error("No Answer objects created.", true);
00394                 }
00395                 $this->put($student, $answers);
00396                 $count = DB::$db->affectedRows();
00397                 if ($count) {
00398                     $msg .= HT::success("$count records went into the database.", true);
00399                 } else {
00400                     $msg .= HT::error("No database rows affected.", true);
00401                 }
00402                 $count = $this->cpAutoGraders($student);
00403                 if ($count) {
00404                     $msg .= HT::success("Attempted autograded question. Copied.", true);
00405                 } else {
00406                     $msg .= HT::error("No autograders copied.", true);
00407                 }
00408             }
00409         }
00410         $msg .= $this->processErrors;
00411         $this->processSummary .= $this->processErrors;
00412         $this->processErrors = "";
00413         HT::info($msg);
00414     }
```

```
00415
00422     public function update($answer)
00423     {
00424         $table = self::$table;
00425         $columns = [
00426             'student_email',
00427             'subquestion_name',
00428             'filename',
00429             'fulltext',
00430             'hash0',
00431             'hash1',
00432             'comment'
00433         ];
00434         $studentEmail = $answer->getStudentEmail();
00435         $subQuestionName = $answer->getSubQuestionName();
00436         $fileName = $answer->getFileName();
00437         $fulltext = DB::$db->sanitize($answer->getFullText());
00438         $hash0 = $answer->getHash0();
00439         $hash1 = $answer->getHash1();
00440         $comment = $answer->getComment();
00441         $rows = [
00442             $studentEmail,
00443             $subQuestionName,
00444             $fileName,
00445             $fulltext,
00446             $hash0,
00447             $hash1,
00448             $comment
00449         ];
00450         // DB::$db->put($table, $columns, $rows);
00451         DB::$db->update($table, $columns, $rows);
00452     }
00453
00459     public function getProcessSummary()
00460     {
00461         return $this->processSummary;
00462     }
00463
00470     public function setProcessSummary($processSummary): self
00471     {
00472         $this->processSummary = $processSummary;
00473         return $this;
00474     }
00475 }
```

5.13 DAO.php File Reference

Classes

- class DAO

5.14 DAO.php

Go to the documentation of this file.

```
00001 <?php
00002
00009 class DAO
00010 {
00017     public static function sortByShortFileName($objects)
00018     {
00019         $toSort = [];
00020         foreach ($objects as $o) {
00021             $toSort[$o->getShortFileName()] = $o;
00022         }
00023         ksort($toSort);
00024         return $toSort;
00025     }
00026 }
```

5.15 MarksDAO.php File Reference

Classes

- class MarksDAO

5.16 MarksDAO.php

Go to the documentation of this file.

```
00001 <?php
00002
00008 class MarksDAO
00009 {
00011     private static $table;
00012
00016     public function __construct()
00017     {
00018         $table = CFG::mkTableName("marks");
00019         self::$table = $table;
00020     }
00021
00028     public function get($where = [])
00029     {
00030         $table = self::$table;
00031         $rows = DB::$db->get(
00032             table: $table,
00033             where: $where
00034         );
00035         $marks = [];
00036         foreach ($rows as $row) {
00037             $marks[] = new Marks(
00038                 $row['student_email'],
00039                 $row['subquestion_name'],
00040                 $row['rubric_name'],
00041                 $row['marks']
00042             );
00043         }
00044         return $marks;
00045     }
00046
00055     public function getByObject($student, $subQuestion = null, $rubric = null)
00056     {
00057         $where = ['student_email' => $student->getEmail()];
00058         if ($subQuestion) {
00059             $where['subquestion_name'] = $subQuestion->getName();
00060         }
00061         if ($rubric) {
00062             $where['rubric_name'] = $rubric->getRubricName();
00063         }
00064         return $this->get($where);
00065     }
00066
00073     public function update($marks)
00074     {
00075         $table = self::$table;
00076         $studentEmail = $marks->getStudentEmail();
00077         $subQuestionName = $marks->getSubQuestionName();
00078         $rubricName = $marks->getRubricName();
00079         $studentMarks = $marks->getMarks();
00080         $columns = ['student_email', 'subquestion_name', 'rubric_name', 'marks'];
00081         $row = [$studentEmail, $subQuestionName, $rubricName, $studentMarks];
00082         // DB::$db->put($table, $columns, $row); // Optional insert
00083         DB::$db->update($table, $columns, $row); // Update if exists
00084     }
00085 }
```

5.17 QuestionDAO.php File Reference

Classes

- class QuestionDAO

5.18 QuestionDAO.php

Go to the documentation of this file.

```
00001 <?php
00002
00008 class QuestionDAO
00009 {
00011     private static $table;
```

```
00012
00016     public function __construct()
00017     {
00018         $table = CFG::mkTableName("questions");
00019         self::$table = $table;
00020     }
00021
00030     public function get($where = [], $verbose = false)
00031     {
00032         $table = self::$table;
00033         $rows = DB::$db->get(
00034             table: $table,
00035             where: $where
00036         );
00037         $got = count($rows);
00038         if ($verbose && $got <= 0) { // Not found in the DB
00039             HT::error("<strong>AnswerDAO</strong>:
00040                 No answers in the DB!<br>
00041                 Run AnswerDAO->process(student) for each student first.");
00042             return;
00043         } else {
00044             $questions = [];
00045             foreach ($rows as $row) {
00046                 $questions[] = new Question(
00047                     $row['question_num'],
00048                     $row['marks']
00049                 );
00050             }
00051         }
00052         // Drop the questions not in the gradable list
00053         foreach ($questions as $key => $question) {
00054             if (!in_array($question->getNum(), CFG::$gradeQuestions)) {
00055                 unset($questions[$key]);
00056             }
00057         }
00058         return $questions;
00059     }
00060
00067     public function put($questions)
00068     {
00069         $table = self::$table;
00070         $rows = [];
00071         foreach ($questions as $question) {
00072             $num = $question->getNum();
00073             $marks = $question->getMarks();
00074             $columns = ['question_num', 'marks'];
00075             $rows[] = [$num, $marks];
00076         }
00077         // DB::$db->put($table, $columns, $rows); // Uncomment to use insert
00078         DB::$db->update($table, $columns, $rows); // Update existing records
00079     }
00080
00086     public function process()
00087     {
00088         if (CFG::isQuiz()) {
00089             // Done in Exam.php already.
00090             return;
00091         }
00092         // Section Names
00093         $secNames = CFG::$sectionNames;
00094         // Locations of subquestion number and marks in the CSV file header
00095         $locSubQ = CFG::$locSubQ;
00096         $locMarks = CFG::$locMarks;
00097         // From the header row of the given CSV file (eLearn grade export),
00098         // construct the question and its subquestions
00099         // $csv = "./CSV/IS113-LT1-{$secNames[0]}.csv";
00100         $csv = CFG::getCsvFileName($secNames[0]);
00101         if (!is_file($csv)) {
00102             HT::error("Error loading $csv");
00103             return;
00104         }
00105         $file = fopen($csv, 'r');
00106         $headers = fgetcsv($file);
00107         fclose($file);
00108         // Filter only the questions column headers
00109         foreach ($headers as $key => $value) {
00110             if (empty($value) || (preg_match('@[qQ].[A-Za-z]@', $value) == 0)) {
00111                 unset($headers[$key]);
00112             }
00113         }
00114         // Get the marks for each subquestion and question numbers
00115         $marks = $questionNum = [];
00116         foreach ($headers as $key => $value) {
00117             $tokens = explode(" ", $value);
00118             $subQuestionName = strtolower(trim($tokens[$locSubQ]));
00119             $marks[$subQuestionName] = explode(":", $tokens[$locMarks])[1];
00120             // Use preg_match() function to check match
```

```
00121             preg_match('!\d+!', $subQuestionName, $match);
00122             if (!in_array($match[0], $questionNum)) {
00123                 $questionNum[] = $match[0];
00124             }
00125         }
00126         // We now have question numbers, subquestion names and marks
00127         $questions = [];
00128         foreach ($questionNum as $num) {
00129             $questions[] = new Question($num, $marks);
00130         }
00131         $this->put($questions);
00132     }
00133 }
```

5.19 RefFileDAO.php File Reference

Classes

- class RefFileDAO

5.20 RefFileDAO.php

Go to the documentation of this file.

```
00001 <?php
00002
00008 class RefFileDAO extends DAO
00009 {
00011     private static $table;
00012
00014     private static $gradables = [];
00015
00019     public function __construct()
00020     {
00021         $table = CFG::mkTableName("ref_files");
00022         self::$table = $table;
00023     }
00024
00032     public function get($where = [], $verbose = false)
00033     {
00034         $table = self::$table;
00035         $rows = DB::$db->get(
00036             table: $table,
00037             where: $where
00038         );
00039         $got = count($rows);
00040         if ($verbose && $got <= 0) { // Not found in the DB
00041             HT::error("<strong>RefFileDAO</strong>:
00042                 No resources or solutions in the DB!<br>
00043                 Run RefFileDAO->process(student) for each student first.");
00044             return;
00045         } else {
00046             $refFiles = [];
00047             foreach ($rows as $row) {
00048                 $refFile = new RefFile(
00049                     $row['subquestion_name'],
00050                     $row['filetype'],
00051                     $row['filename'],
00052                     $row['fulltext'],
00053                     $row['hash0'],
00054                     $row['hash1']
00055                 );
00056                 $refFile->setSim($row['sim']);
00057                 $refFile->setGradable((bool) $row['gradable']);
00058                 $refFiles[] = $refFile;
00059             }
00060         }
00061         $refFiles = self::sortByShortFileName($refFiles);
00062         return $refFiles;
00063     }
00064
00071     public function isGradable($shortFileName)
00072     {
00073         if (empty(self::$gradables)) {
00074             $resources = $this->get(['filetype' => 'resources', 'gradable' => 1]);
00075             foreach ($resources as $resource) {
00076                 self::$gradables[] = $resource->getShortFileName();
```

```
00077             }
00078         }
00079         return in_array($shortFileName, self::$gradables);
00080     }
00081
00089     public function getByObject($subQuestion, $fileType = '')
00090     {
00091         $where = ['subquestion_name' => $subQuestion->getName()];
00092         if ($fileType) {
00093             $where['filetype'] = $fileType;
00094         }
00095         return $this->get($where);
00096     }
00097
00105     private function put($subQuestion, $refFiles)
00106     {
00107         $table = self::$table;
00108         $subQuestionName = $subQuestion->getName();
00109         if (count($refFiles) <= 0) { // No answers found. An error!
00110             HT::error("<strong>SubQuestion $subQuestionName</strong>:
00111             No solutions or resources found!");
00112         } else {
00113             $columns = [
00114                 'subquestion_name',
00115                 'filetype',
00116                 'filename',
00117                 'fulltext',
00118                 'hash0',
00119                 'hash1',
00120                 'sim',
00121                 'gradable'
00122             ];
00123             $rows = [];
00124             foreach ($refFiles as $refFile) {
00125                 $fileName = $refFile->getFileName();
00126                 $fileType = $refFile->getFileType();
00127                 $fullText = DB::$db->getFileContent($fileName);
00128                 $hash0 = $refFile->getHash0();
00129                 $hash1 = $refFile->getHash1();
00130                 $sim = (float) $refFile->getSim();
00131                 $gradable = (int) $refFile->isGradable();
00132                 $rows[] = [
00133                     $subQuestionName,
00134                     $fileType,
00135                     $fileName,
00136                     $fullText,
00137                     $hash0,
00138                     $hash1,
00139                     $sim,
00140                     $gradable
00141                 ];
00142             }
00143             // DB::$db->put($table, $columns, $rows);
00144             DB::$db->update($table, $columns, $rows);
00145         }
00146     }
00155     public function process($subQuestion)
00156     {
00157         $subQuestionName = $subQuestion->getName();
00158         $folders = [
00159             'resources' => CFG::$resourceFolder,
00160             'solutions' => CFG::$solutionFolder,
00161         ];
00162         $solutions = $resources = $refFiles = $objects = [];
00163         $objects['resources'] = $objects['solutions'] = [];
00164         foreach ($folders as $fileType => $folder) {
00165             $$fileType = $this->findFiles($subQuestion, $folder);
00166             if (count($$fileType) <= 0) {
00167                 HT::error("<strong>SubQuestion $subQuestionName</strong>:
00168                  No $fileType found in $folder!");
00169             } else {
00170                 foreach ($$fileType as $fileName) {
00171                     if (is_file($fileName)) {
00172                         // $fileName = CFG::mkFileName($fileName);
00173                         $refFile = new RefFile($subQuestionName, $fileType, $fileName);
00174                         $refFiles[] = $refFile;
00175                         $objects[$fileType][] = $refFile;
00176                     }
00177                 }
00178             }
00179         }
00180         // We now have all the resource files ($resources) and solutions ($solutions)
00181         // for this question. Find the ones that are different as gradable
00182         foreach ($objects['resources'] as $resource) {
00183             $resourceShort = $resource->getShortFileName();
00184             foreach ($objects['solutions'] as $solution) {
00185                 $solutionShort = $solution->getShortFileName();
```

```
00186                 if ($solutionShort == $resourceShort) {
00187                     if ($resource->getHash0() == $solution->getHash0()) {
00188                         $sim = 100;
00189                     } else if ($resource->getHash1() == $solution->getHash1()) {
00190                         $sim = 100;
00191                     } else {
00192                         $sim = Report::similarity(
00193                             $resource->getFileName(),
00194                             $solution->getFileName()
00195                         );
00196                     }
00197                     $gradable = $sim < 99.9;
00198                     $resource->setSim($sim);
00199                     $solution->setSim($sim);
00200                     $resource->setGradable($gradable);
00201                     $solution->setGradable($gradable);
00202                 }
00203             }
00204         }
00205         $this->put($subQuestion, $refFiles);
00206     }
00207
00215     private function findFiles($subQuestion, $folder)
00216     {
00217         # Look for files like folder/q1/* and /q1/*/* and q1/*/*/*
00218         $questionNum = $subQuestion->getQuestionNum();
00219         $pattern = $folder . "q$questionNum/{*.*,*/*.*,*/*/*.*}";
00220         // $found = glob(strtolower($pattern), GLOB_BRACE);
00221         $found = glob($pattern, GLOB_BRACE);
00222         $files = [];
00223         $allowed = ['php', 'html'];
00224         foreach ($found as $f) {
00225             $ext = pathinfo($f, PATHINFO_EXTENSION);
00226             if (in_array($ext, $allowed)) {
00227                 $files[] = $f;
00228             }
00229         }
00230         return $files;
00231     }
00232
00240     public function getRefFile($shortName, $fileType)
00241     {
00242         $refFiles = $this->get(['filetype' => $fileType]);
00243         $found = [];
00244         foreach ($refFiles as $refFile) {
00245             if (stripos($refFile->getFileName(), "/$shortName") !== false) {
00246                 $found[] = $refFile;
00247             }
00248         }
00249         $count = count($found);
00250         if ($count > 0) {
00251             for ($i = 1; $i < $count; $i++) {
00252                 $prevName = $found[$i - 1]->getFileName();
00253                 $curName = $found[$i]->getFileName();
00254                 if ($prevName != $curName) {
00255                     HT::error("Confusion with $fileType for $shortName:
00256                         $prevName != $curName");
00257                     die;
00258                 }
00259             }
00260             return $found[0];
00261         }
00262         return false;
00263     }
00264 }
```

5.21 RubricDAO.php File Reference

Classes

- class RubricDAO

5.22 RubricDAO.php

Go to the documentation of this file.

```
00001 <?php
```

```
00002
00009 class RubricDAO
00010 {
00012     private static $table;
00013
00017     public function __construct()
00018     {
00019         $table = CFG::mkTableName("rubrics");
00020         self::$table = $table;
00021     }
00022
00030     public function get($where = [], $verbose = false)
00031     {
00032         $rubrics = [];
00033         $table = self::$table;
00034         $rows = DB::$db->get(
00035             table: $table,
00036             where: $where
00037         );
00038         $got = count($rows);
00039         if ($verbose && $got <= 0) { // Not found in the DB
00040             HT::error("<strong>RubricDAO</strong>: No rubrics in the DB!<br>
00041             Run RubricDAO->process(subQuestion) for each subQuestion first.");
00042             return;
00043         } else {
00044             foreach ($rows as $row) {
00045                 $rubrics[] = new Rubric(
00046                     $row['subquestion_name'],
00047                     $row['rubric_name'],
00048                     $row['marks'],
00049                     $row['rubric_text'],
00050                     $row['short_filename'],
00051                 );
00052             }
00053         }
00054         return $rubrics;
00055     }
00056
00063     public function getByObject($subQuestion)
00064     {
00065         $where = ['subquestion_name' => $subQuestion->getName()];
00066         return $this->get($where);
00067     }
00068
00076     private function put($subQuestion, $rubrics)
00077     {
00078         $table = self::$table;
00079         $subQuestionName = $subQuestion->getName();
00080         if (count($rubrics) <= 0) { // No rubrics found. An error!
00081             HT::error("<strong>SubQuestion $subQuestionName</strong>:
00082             No rubrics found. <br>
00083             Something very wrong!");
00084         } else {
00085             $columns = [
00086                 'subquestion_name',
00087                 'rubric_name',
00088                 'marks',
00089                 'rubric_text',
00090                 'short_filename'
00091             ];
00092             $rows = [];
00093             foreach ($rubrics as $rubric) {
00094                 $rubricName = $rubric->getRubricName();
00095                 $marks = $rubric->getMarks();
00096                 $rubricText = $rubric->getRubricText();
00097                 $shortFileName = $rubric->getShortFileName();
00098                 $rows[] = [
00099                     $subQuestionName,
00100                     $rubricName,
00101                     $marks,
00102                     $rubricText,
00103                     $shortFileName
00104                 ];
00105             }
00106             // DB::$db->put($table, $columns, $rows);
00107             DB::$db->update($table, $columns, $rows);
00108         }
00109     }
00110
00118     public function process($subQuestion)
00119     {
00120         $subQuestionName = $subQuestion->getName();
00121         // Rubrics are kept in rubric.csv
00122         // name, marks, rubric text
00123         $rubricFile = glob(CFG::$rubricFile);
00124         $rubricNum = 1;
00125         if (count($rubricFile) != 1) {
```

```
00126             HT::error("<strong>SubQuestion $subQuestionName</strong>:
00127         Rubric File not found: " . CFG::$rubricFile);
00128         } else {
00129             $rubricFile = $rubricFile[0];
00130             $lines = file($rubricFile);
00131             $rubrics = [];
00132             foreach ($lines as $line) {
00133                 $lineElements = explode(",", $line);
00134                 $name = strtolower(trim($lineElements[0]));
00135                 if ($name == $subQuestionName) {
00136                     $marks = (float) $lineElements[1];
00137                     $rubricText = trim($lineElements[2]);
00138                     $rubricText = str_replace('""', "'", $rubricText);
00139                     $rubricText = trim($rubricText, '"');
00140                     $rubricText = str_replace("'", '"', $rubricText);
00141                     $shortFileName = trim($lineElements[3]);
00142                     $shortFileName = str_replace("//", '/', $shortFileName);
00143                     $rubricName = $name . "-" . $rubricNum++;
00144                     $rubrics[] = new Rubric(
00145                         $name,
00146                         $rubricName,
00147                         $marks,
00148                         $rubricText,
00149                         $shortFileName
00150                     );
00151                 }
00152             }
00153             $this->put($subQuestion, $rubrics);
00154         }
00155     }
00156 }
```

5.23 StudentDAO.php File Reference

Classes

- class StudentDAO

5.24 StudentDAO.php

Go to the documentation of this file.

```
00001 <?php
00002
00010 class StudentDAO
00011 {
00013     private static $table;
00014
00016     private static $collate = 'student_email';
00017
00019     private static $students = [];
00020
00022     private static $iStudent = [];
00023
00025     private static $nStudents = 0;
00026
00030     public function __construct()
00031     {
00032         $table = CFG::mkTableName("students");
00033         self::$table = $table;
00034         if (empty(self::$students)) {
00035             $students = $this->get();
00036             self::$students = $students;
00037             self::$nStudents = count($students);
00038             $i = 0;
00039             foreach ($students as $student) {
00040                 $i++;
00041                 self::$iStudent[$student->getEmail()] = $i;
00042             }
00043             if (self::$nStudents != $i) {
00044                 xdebug_print_function_stack();
00045                 die("Error in student arrays");
00046             }
00047             ksort(self::$iStudent);
00048         }
00049     }
00050
```

```
00058     public function get($where = [], $verbose = false)
00059     {
00060         $table = self::$table;
00061         $rows = DB::$db->get(
00062             table: $table,
00063             where: $where
00064         );
00065         $got = count($rows);
00066         $students = [];
00067         if ($verbose && $got <= 0) { // Not found in the DB
00068             HT::error("<strong>StudentDAO</strong>: No students in the DB!<br>
00069             Run StudentDAO->process(section) for each section first.");
00070         } else {
00071             foreach ($rows as $row) {
00072                 $student = new Student(
00073                     $row['student_email'],
00074                     $row['student_name'],
00075                     $row['section']
00076                 );
00077                 $student->setZipFile($row['zip_filename']);
00078                 $student->setStatus($row['status']);
00079                 $student->setComment($row['comment']);
00080                 $students[] = $student;
00081             }
00082         }
00083         return $students;
00084     }
00085
00092     public function getByEmail($studentEmail)
00093     {
00094         $student = $this->get(['student_email' => $studentEmail]);
00095         if (count($student) > 1) {
00096             HT::error("Too many students with email id $studentEmail!");
00097             $student = $student[0];
00098         }
00099         return $student;
00100     }
00101
00108     public function getIStudent($studentEmail)
00109     {
00110         return self::$iStudent[$studentEmail];
00111     }
00112
00119     public function getByObject($answer)
00120     {
00121         return $this->getByEmail($answer->getStudentEmail());
00122     }
00123
00131     public function getOthers($student)
00132     {
00133         $others = $this->get();
00134         $needPrevious = true;
00135         $needNext = false;
00136         $prev = $prevUngraded = $nextUngraded = $next = "";
00137         foreach ($others as $other) {
00138             $status = $other->getStatus();
00139             if ($status == 'Absent') {
00140                 continue;
00141             }
00142             if ($other->getEmail() == $student->getEmail()) {
00143                 $needPrevious = false;
00144                 $needNext = true;
00145                 continue;
00146             }
00147             if ($needPrevious) {
00148                 $prev = $other;
00149                 if ($status == "New") {
00150                     $prevUngraded = $other;
00151                 }
00152             }
00153             if ($needNext) {
00154                 if ($next == "") {
00155                     $next = $other;
00156                 }
00157                 if ($status == "New") {
00158                     $nextUngraded = $other;
00159                     break;
00160                 }
00161             }
00162         }
00163         return [$prev, $prevUngraded, $nextUngraded, $next];
00164     }
00165
00173     public function put($section, $students)
00174     {
00175         $table = self::$table;
00176         $sectionName = $section->getName();
```

```
00177         if (count($students) <= 0) { // No students found. An error!
00178             HT::error("<strong>Section $sectionName</strong>:
00179             No students found. <br>
00180             Something very wrong!");
00181         } else {
00182             $columns = ['student_email', 'student_name', 'section', 'status', 'zip_filename',
    'comment'];
00183             $rows = [];
00184             foreach ($students as $student) {
00185                 $studentEmail = $student->getEmail();
00186                 $studentName = $student->getName();
00187                 $status = $student->getStatus();
00188                 $zipFileName = $student->getZipFile();
00189                 $comment = $student->getComment();
00190                 $rows[] = [$studentEmail, $studentName, $sectionName, $status, $zipFileName,
    $comment];
00191             }
00192             // DB::$db->put($table, $columns, $rows);
00193             DB::$db->update($table, $columns, $rows);
00194         }
00195     }
00196
00203     public function update($student)
00204     {
00205         $table = self::$table;
00206         $columns = ['student_email', 'student_name', 'section', 'status', 'zip_filename', 'comment'];
00207         $studentEmail = $student->getEmail();
00208         $studentName = $student->getName();
00209         $section = $student->getSection();
00210         $status = $student->getStatus();
00211         $zipFileName = $student->getZipFile();
00212         $comment = $student->getComment();
00213         $rows = [[$studentEmail, $studentName, $section, $status, $zipFileName, $comment]];
00214         // DB::$db->put($table, $columns, $rows);
00215         DB::$db->update($table, $columns, $rows);
00216     }
00217
00224     public function process($section)
00225     {
00226         $sectionName = $section->getName();
00227         $csv = CFG::getCsvFileName($sectionName);
00228         if (!is_file($csv)) {
00229             HT::error("Error loading $csv");
00230             return;
00231         }
00232         $file = fopen($csv, 'r');
00233         $line = fgetcsv($file); // Skip the header
00234         $students = [];
00235         while (($line = fgetcsv($file)) !== FALSE) {
00236             //$line is an array of the csv elements
00237             $students[] = new Student($line[0], $line[1], $sectionName);
00238         }
00239         fclose($file);
00240         $this->put($section, $students);
00241     }
00242
00248     public static function getNStudents()
00249     {
00250         return self::$nStudents;
00251     }
00252
00258     public static function getStudents()
00259     {
00260         return self::$students;
00261     }
00262 }
```

5.25 SubQuestionDAO.php File Reference

Classes

- class SubQuestionDAO

5.26 SubQuestionDAO.php

Go to the documentation of this file.

```php
00001 <?php
00002
00011 class SubQuestionDAO
00012 {
00013     /* @var string $table Holds the database table name for subquestions */
00014     private static $table;
00015
00019     public function __construct()
00020     {
00021         self::$table = CFG::mkTableName("subquestions");
00022     }
00023
00031     public function get($where = [], $verbose = false)
00032     {
00033         $table = self::$table;
00034         $subQuestions = [];
00035         $rows = DB::$db->get(
00036             table: $table,
00037             where: $where
00038         );
00039         $got = count($rows);
00040         if ($verbose && $got <= 0) { // Not found in the DB
00041             HT::error("<strong>SubQuestionDAO</strong>:
00042             No subQuestions in the DB!<br>
00043             Run SubQuestionDAO->process(question) for each question first.");
00044             return;
00045         } else {
00046             foreach ($rows as $row) {
00047                 $subQuestions[] = new SubQuestion(
00048                     $row['subquestion_name'],
00049                     $row['marks'],
00050                     $row['question_num']
00051                 );
00052             }
00053         }
00054         return $subQuestions;
00055     }
00056
00063     public function put($subQuestions)
00064     {
00065         $table = self::$table;
00066         $rows = [];
00067         foreach ($subQuestions as $subQuestion) {
00068             $subQuestionName = $subQuestion->getName();
00069             $questionNum = $subQuestion->getQuestionNum();
00070             $marks = $subQuestion->getMarks();
00071             $columns = ['subquestion_name', 'question_num', 'marks'];
00072             $rows[] = [$subQuestionName, $questionNum, $marks];
00073         }
00074         // DB::$db->put($table, $columns, $rows);
00075         DB::$db->update($table, $columns, $rows);
00076     }
00077
00084     public function process($question)
00085     {
00086         $subQuestions = $question->getSubQuestions();
00087         $this->put($subQuestions);
00088     }
00089 }
```

5.27 DB.php File Reference

Classes

- class DB

5.28 DB.php

Go to the documentation of this file.

```php
00001 <?php
00002
00023 class DB
00024 {
00026   protected $connection;
00027
00029   protected $query;
```

```
00030
00032   protected $show_errors = true;
00033
00035   protected $query_closed = true;
00036
00038   public $query_count = 0;
00039
00041   public static $db;
00042
00044   private $error = "";
00045
00047   private static $orderBy = ["section", "student_email"];
00048
00049   // ---------- Constructor ----------
00050
00061   public function __construct(
00062     $dbHost = 'localhost',
00063     $dbUser = 'root',
00064     $dbPass = '',
00065     $dbName = '',
00066     $charset = 'utf8'
00067   ) {
00068     try {
00069       mysqli_report(MYSQLI_REPORT_ERROR | MYSQLI_REPORT_STRICT);
00070       $configFile = realpath("config.php");
00071       $setupMsg = "Edit the config file <code>$configFile</code>
00072       and go to <strong>Setup and Configuration</strong> and set it up.";
00073       $this->connection = new mysqli($dbHost, $dbUser, $dbPass, $dbName);
00074       if ($this->connection->connect_error) { // For WAMP
00075         $error = $this->connection->connect_error;
00076         $this->error = "WAMP: Database connection error: $error.
00077         $setupMsg";
00078       } else {
00079         $this->connection->set_charset($charset);
00080         self::$db = $this;
00081       }
00082     } catch (Exception $e) { // for MAMP
00083       $error = $e->getMessage();
00084       $configFile = realpath("config.php");
00085       $this->error = "MAMP: Database connection error: $error.
00086       $setupMsg";
00087     }
00088   }
00089   // Manoj's utility methods
00090   // ---------- CRUD Operations ----------
00091
00100   public function get($table, $what = '', $where = [])
00101   {
00102     if (is_array($what)) {
00103       if (in_array("*", $what)) {
00104         $what = [];
00105       }
00106       $columns = $what;
00107     } else {
00108       if (empty($what) || $what == "*") {
00109         $columns = [];
00110       } else {
00111         $columns = [$what];
00112       }
00113     }
00114     if (empty($columns)) {
00115       $what = "*";
00116     } else {
00117       $what = "`" . implode("`, `", $columns) . "`";
00118     }
00119     $keys = $where;
00120     if (empty($keys)) {
00121       $where = "";
00122     } else {
00123       $where = [];
00124       foreach ($keys as $key => $val) {
00125         $where[] = "`$key` = '$val'";
00126       }
00127       $where = "WHERE " . implode(" AND ", $where);
00128     }
00129     $sql = "SELECT $what FROM $table $where";
00130     $orderBy = $this->getOrderBy($table, self::$orderBy);
00131     $sql .= "$orderBy";
00132     $rows = $this->query($sql)->fetchAll();
00133     return $rows;
00134   }
00135
00144   public function put($table, $columns, $rows)
00145   {
00146     $sql = self::mkSql($table, $columns, $rows, "REPLACE");
00147     $this->query($sql);
00148   }
```

```
00149
00158   public function update($table, $columns, $rows)
00159   {
00160     $sql = self::mkSql($table, $columns, $rows, "INSERT");
00161     $update = "";
00162     foreach ($columns as $column) {
00163       $update .= "`$column`=VALUES(`$column`), ";
00164     }
00165     $update = trim($update, ", ");
00166     $sql .= " ON DUPLICATE KEY UPDATE $update";
00167     $this->query($sql);
00168   }
00169
00177   public function getOrderBy($table, $columns)
00178   {
00179     if (!is_array($columns)) {
00180       $columns = [$columns];
00181     }
00182     foreach ($columns as $key => $column) {
00183       if (!$this->colExists($table, $column)) {
00184         unset($columns[$key]);
00185       }
00186     }
00187     if (count($columns) > 0) {
00188       $orderBy = "ORDER BY `" . implode("`, `", $columns) . "`";
00189     } else {
00190       $orderBy = "";
00191     }
00192     return $orderBy;
00193   }
00194
00195   // ---------- Column and Table Utilities ----------
00196
00204   public function colExists($table, $column)
00205   {
00206     $sql = "SELECT *
00207       FROM `information_schema`.COLUMNS
00208       WHERE TABLE_NAME = '$table'
00209       AND COLUMN_NAME = '$column'";
00210     return !empty($this->query($sql)->fetchAll());
00211   }
00212
00219   public function tablesExist($dbPrefix)
00220   {
00221     $sql = "SHOW TABLES LIKE '$dbPrefix%'";
00222     $results = $this->query($sql);
00223     $results = $results->fetchAll();
00224     return !empty($results);
00225   }
00226
00234   public function getEnum($table, $field)
00235   {
00236     $sql = "SHOW COLUMNS FROM {$table} WHERE Field = '{$field}'";
00237     $result = self::$db->query($sql);
00238     $row = $result->fetchAll();
00239     $type = $row[0]['Type'];
00240     preg_match('/enum\((.*)\)$/', $type, $matches);
00241     $vals = explode(',', $matches[1]);
00242     return $vals;
00243   }
00244
00251   public function sanitize($str)
00252   {
00253     return $this->connection->real_escape_string($str);
00254   }
00255
00262   public function getFileContent($file)
00263   {
00264     return $this->sanitize(file_get_contents($file));
00265   }
00266
00268
00275   public function query($query)
00276   {
00277     if (!$this->query_closed) {
00278       $this->query->close();
00279     }
00280     if ($this->query = $this->connection->prepare($query)) {
00281       if (func_num_args() > 1) {
00282         $x = func_get_args();
00283         $args = array_slice($x, 1);
00284         $types = '';
00285         $args_ref = array();
00286         foreach ($args as $k => &$arg) {
00287           if (is_array($args[$k])) {
00288             foreach ($args[$k] as $j => &$a) {
00289               $types .= $this->_gettype($args[$k][$j]);
```

```
00290               $args_ref[] = &$a;
00291             }
00292           } else {
00293             $types .= $this->_gettype($args[$k]);
00294             $args_ref[] = &$arg;
00295           }
00296         }
00297         array_unshift($args_ref, $types);
00298         call_user_func_array(array($this->query, 'bind_param'), $args_ref);
00299       }
00300       $this->query->execute();
00301       if ($this->query->errno) {
00302         $this->error('Unable to process MySQL query (check your params) - ' . $this->query->error);
00303       }
00304       $this->query_closed = FALSE;
00305       $this->query_count++;
00306     } else {
00307       $this->error('Unable to prepare MySQL statement (check your syntax) - ' .
      $this->connection->error);
00308     }
00309     return $this;
00310   }
00311
00318   public function fetchAll($callback = null)
00319   {
00320     $params = array();
00321     $row = array();
00322     $meta = $this->query->result_metadata();
00323     while ($field = $meta->fetch_field()) {
00324       $params[] = &$row[$field->name];
00325     }
00326     call_user_func_array(array($this->query, 'bind_result'), $params);
00327     $result = array();
00328     while ($this->query->fetch()) {
00329       $r = array();
00330       foreach ($row as $key => $val) {
00331         $r[$key] = $val;
00332       }
00333       if ($callback != null && is_callable($callback)) {
00334         $value = call_user_func($callback, $r);
00335         if ($value == 'break')
00336           break;
00337       } else {
00338         $result[] = $r;
00339       }
00340     }
00341     $this->query->close();
00342     $this->query_closed = TRUE;
00343     return $result;
00344   }
00345
00351   public function fetchArray()
00352   {
00353     $params = array();
00354     $row = array();
00355     $meta = $this->query->result_metadata();
00356     while ($field = $meta->fetch_field()) {
00357       $params[] = &$row[$field->name];
00358     }
00359     call_user_func_array(array($this->query, 'bind_result'), $params);
00360     $result = array();
00361     while ($this->query->fetch()) {
00362       foreach ($row as $key => $val) {
00363         $result[$key] = $val;
00364       }
00365     }
00366     $this->query->close();
00367     $this->query_closed = TRUE;
00368     return $result;
00369   }
00370
00376   public function close()
00377   {
00378     return $this->connection->close();
00379   }
00380
00386   public function numRows()
00387   {
00388     $this->query->store_result();
00389     return $this->query->num_rows;
00390   }
00391
00397   public function affectedRows()
00398   {
00399     return $this->query->affected_rows;
00400   }
00401
```

```
00407   public function lastInsertID()
00408   {
00409     return $this->connection->insert_id;
00410   }
00411
00418   public function error($error)
00419   {
00420     if ($this->show_errors) {
00421       exit($error);
00422     }
00423   }
00424
00430   public function getError()
00431   {
00432     return $this->error;
00433   }
00434
00435
00436   // ---------- SQL Import & Processing ----------
00437
00444   public static function remove_comments(&$output)
00445   {
00446     $lines = explode("\n", $output);
00447     $output = "";
00448
00449     $linecount = count($lines);
00450
00451     $in_comment = false;
00452     for ($i = 0; $i < $linecount; $i++) {
00453       if (preg_match("/^\/\*/", preg_quote($lines[$i]))) {
00454         $in_comment = true;
00455       }
00456
00457       if (!$in_comment) {
00458         $output .= $lines[$i] . "\n";
00459       }
00460
00461       if (preg_match("/\*\/$/", preg_quote($lines[$i]))) {
00462         $in_comment = false;
00463       }
00464     }
00465
00466     unset($lines);
00467     return $output;
00468   }
00469
00478   public static function remove_remarks($sql)
00479   {
00480     $lines = explode("\n", $sql);
00481     $linecount = count($lines);
00482     $output = "";
00483
00484     for ($i = 0; $i < $linecount; $i++) {
00485       if (($i != ($linecount - 1)) || (strlen($lines[$i]) > 0)) {
00486         if (isset($lines[$i][0]) && $lines[$i][0] != "#") {
00487           $output .= $lines[$i] . "\n";
00488         } else {
00489           $output .= "\n";
00490         }
00491       }
00492     }
00493
00494     return $output;
00495   }
00496
00503   public static function remove_inline($sql)
00504   {
00505     $regex = array('/\/\*.*\*\//U');
00506     $sql = preg_replace($regex, "", $sql);
00507     $sql = trim($sql);
00508     return $sql;
00509   }
00510
00518   public static function split_sql_file($sql, $delimiter = ";")
00519   {
00520     $sql = trim($sql);
00521     $tokens = explode($delimiter, $sql);
00522     $output = array();
00523     $matches = array();
00524
00525     // this is faster than calling count($oktens) every time thru the loop.
00526     $token_count = count($tokens);
00527     for ($i = 0; $i < $token_count; $i++) {
00528       $tokens[$i] = self::remove_inline($tokens[$i]);
00529       // Don't wanna add an empty string as the last thing in the array.
00530       if (($i != ($token_count - 1)) || (strlen($tokens[$i] > 0))) {
00531         // This is the total number of single quotes in the token.
```

```
00532         $total_quotes = preg_match_all("/'/", $tokens[$i], $matches);
00533         // Counts single quotes that are preceded by an odd number of backslashes,
00534         // which means they're escaped quotes.
00535         $escaped_quotes = preg_match_all("/(?<!\\\\)(\\\\\\\\)*\\\\'/", $tokens[$i], $matches);
00536
00537         $unescaped_quotes = $total_quotes - $escaped_quotes;
00538
00539         // If the number of unescaped quotes is even, then the delimiter did NOT occur inside a string
      literal.
00540         if (($unescaped_quotes % 2) == 0) {
00541           // It's a complete sql statement.
00542           $output[] = $tokens[$i];
00543         } else {
00544           // incomplete sql statement. keep adding tokens until we have a complete one.
00545           // $temp will hold what we have so far.
00546           $temp = $tokens[$i] . $delimiter;
00547
00548           // Do we have a complete statement yet?
00549           $complete_stmt = false;
00550
00551           for ($j = $i + 1; (!$complete_stmt && ($j < $token_count)); $j++) {
00552             // This is the total number of single quotes in the token.
00553             $total_quotes = preg_match_all("/'/", $tokens[$j], $matches);
00554             // Counts single quotes that are preceded by an odd number of backslashes,
00555             // which means they're escaped quotes.
00556             $escaped_quotes = preg_match_all("/(?<!\\\\)(\\\\\\\\)*\\\\'/", $tokens[$j], $matches);
00557
00558             $unescaped_quotes = $total_quotes - $escaped_quotes;
00559
00560             if (($unescaped_quotes % 2) == 1) {
00561               // odd number of unescaped quotes. In combination with the previous incomplete
00562               // statement(s), we now have a complete statement. (2 odds always make an even)
00563               $output[] = $temp . $tokens[$j];
00564
00565               // exit the loop.
00566               $complete_stmt = true;
00567               // make sure the outer loop continues at the right point.
00568               $i = $j;
00569             } else {
00570               // even number of unescaped quotes. We still don't have a complete statement.
00571               // (1 odd and 1 even always make an odd)
00572               $temp .= $tokens[$j] . $delimiter;
00573             }
00574           } // for..
00575         } // else
00576       }
00577     }
00578     foreach ($output as $k => $o) {
00579       $o = trim($o);
00580       if (empty($o)) {
00581         unset($output[$k]);
00582       } else {
00583         $output[$k] = $o;
00584       }
00585     }
00586     return $output;
00587   }
00588
00595   public function multiQuery($mSql)
00596   {
00597     $mSql = self::remove_comments($mSql);
00598     $mSql = self::remove_remarks($mSql);
00599     $lines = self::split_sql_file($mSql);
00600     $errors = $success = 0;
00601     $message = "";
00602     foreach ($lines as $sql) {
00603       try {
00604         $this->connection->query($sql, MYSQLI_STORE_RESULT);
00605         ++$success;
00606       } catch (Exception $e) {
00607         ++$errors;
00608         $message .= "\n$errors: " . $e->getMessage();
00609       }
00610     }
00611     $message = trim($message);
00612     return compact('success', 'errors', 'message');
00613   }
00614
00622   public function importSQL($sqlFile, $gzip = false)
00623   {
00624     if (!is_readable($sqlFile)) {
00625       return false;
00626     }
00627     $sql = file_get_contents($sqlFile);
00628     if ($gzip) {
00629       $sql = gzdecode($sql);
00630     }
```

```
00631     return $this->multiQuery($sql);
00632   }
00633
00634
00635   // ---------- Private Helpers ----------
00636
00646   private static function mkSql($table, $columns, $rows, $insert = "REPLACE")
00647   {
00648     $values = [];
00649     foreach ($rows as $key => $row) {
00650       if (is_array($row)) { // Multiple rows
00651         $values[$key] = "('" . implode("', '", $row) . "')";
00652       } else { // Single row
00653         $values = "('" . implode("', '", $rows) . "')";
00654       }
00655     }
00656     if (is_array($values)) { // Multiple rows
00657       $values = implode(",\n ", $values);
00658     }
00659     $sql = "$insert INTO $table
00660         ( `" . implode("`, `", $columns) . "`)
00661         VALUES $values";
00662     return $sql;
00663   }
00664
00671   private function _gettype($var)
00672   {
00673     if (is_string($var))
00674       return 's';
00675     if (is_float($var))
00676       return 'd';
00677     if (is_int($var))
00678       return 'i';
00679     return 'b';
00680   }
00681 }
```

5.29 Exam.php File Reference

Classes

- class Exam

5.30 Exam.php

Go to the documentation of this file.

```
00001 <?php
00002
00015 class Exam extends HT
00016 {
00018     private $name;
00019
00021     private $questions = [];
00022
00024     private $sections = [];
00025
00027     protected static $class = __CLASS__;
00028
00029     // -------------------- Static UI Action Definitions --------------------
00030
00032     protected static $actions = [
00033         'view' => 'Exam Details',
00034         'process' => 'Process Everything!',
00035         'rubrics' => 'Questions and Rubrics',
00036         'refFiles' => 'Resources and Solutions',
00037         'submissions' => 'Submissions',
00038         'delete' => 'Delete Submissions'
00039     ];
00040
00042     protected static $actionProperties = [
00043         'view' => [
00044             'force' => 'force',
00045             'class' => 'error'
00046         ],
00047         'rubrics' => [
00048             'class' => 'warn'
```

```
00049         ],
00050         'refFiles' => [
00051             'class' => 'warn',
00052             'target' => 'right'
00053         ],
00054         'submissions' => [
00055             'class' => 'warn'
00056         ],
00057         'delete' => [
00058             'class' => 'error',
00059             'target' => 'right'
00060         ]
00061     ];
00062
00064     protected static $intro = "<h4>Process Menu</h4>
00065     Here, you can process the files to set up Grader:
00066     <ul>
00067     <li>Process grade book downloads and <code>rubrics.csv</code> to create
00068     questions, subQuestions and rubrics.</li>
00069     <li>Process the reference resource files and solutions.</li>
00070     <li>Process student submissions, as downloaded from eLearn.</li>
00071     </ul>";
00072
00074     protected static $actionDesc = [
00075         'view' => 'Examine the questions, subQuestions and their marks as
00076         processed from the eLearn grade book exported (CSV) files.
00077         <span style="color:red;">In Finals Mode (as opposed to Lab-Test Mode),
00078         this button reprocesses the eLearn output file, and repopulates
00079         the DB. Use with care!</span>',
00080         'process' => 'Does what the next three buttons do:
00081         Import <code>rubrics.csv</code> to create rubrics, import
00082         the grade book downloads from eLearn to create questions, subQuestions
00083         and students, and import student submissions to create answers and
00084         prepare for grading. You can also use Grader to verify that students
00085         have submitted valid zip files: Grader will produce color-coded status
00086         messages per student.',
00087         'rubrics' => 'Import <code>rubrics.csv</code> to create questions,
00088         subQuestions and their rubrics in the database.',
00089         'refFiles' => 'Process the resource and solution files into the database',
00090         'submissions' => 'Import student submissions to create answers and prepare
00091         for grading. This button can be used to validate the submitted zip files.',
00092         'delete' => 'Delete the student submissions: The uncompressed files, not
00093         the zip files downloaded from eLearn.'
00094     ];
00095
00097     protected static $display = [
00098         "class" => "info",
00099         "title" => "Preprocess Files"
00100     ];
00101
00102     // -------------------- Constructor --------------------
00103
00108     public function __construct()
00109     {
00110         $this->name = CFG::$examShortName;
00111         if (CFG::isQuiz()) {
00112             $this->mkFinals();
00113         } else {
00114             $this->mkLabTest();
00115         }
00116     }
00117
00118     // -------------------- Internal Setup Methods --------------------
00119
00125     private function mkLabTest()
00126     {
00127         // Section Names
00128         $sectionNames = CFG::$sectionNames;
00129         // Locations of subQuestion number and marks in the CSV file header
00130         $locSubQ = CFG::$locSubQ;
00131         $locMarks = CFG::$locMarks;
00132         // From the header row of the given CSV file (eLearn grade export),
00133         // construct the question and its subQuestions
00134         // $csv = "./CSV/IS113-LT1-{$sectionNames[0]}.csv";
00135         $csv = CFG::getCsvFileName($sectionNames[0]);
00136         if (!is_file($csv)) {
00137             HT::error("Error loading $csv");
00138             return;
00139         }
00140         $file = fopen($csv, 'r');
00141         $line = fgetcsv($file);
00142         fclose($file);
00143         // Filter only the questions column headers
00144         foreach ($line as $key => $value) {
00145             if (empty($value) || (preg_match('@[qQ].[ A-Za-z]@', $value) == 0)) {
00146                 unset($line[$key]);
00147             }
```

```
00148         }
00149         // Get the marks for each subQuestion and question numbers
00150         $marks = $questionNumbers = [];
00151         foreach ($line as $key => $value) {
00152             $tokens = explode(" ", $value);
00153             $subQuestionName = strtolower(trim($tokens[$locSubQ]));
00154             $marks[$subQuestionName] = explode(":", $tokens[$locMarks])[1];
00155             // Use preg_match() function to check match
00156             preg_match('!\d+!', $subQuestionName, $match);
00157             if (!in_array($match[0], $questionNumbers)) {
00158                 $questionNumbers[] = $match[0];
00159             }
00160         }
00161         // We now have question numbers, subQuestion names and marks
00162         // Add questions (which will add subQuestions too)
00163         // Add only gradeable questions
00164         foreach ($questionNumbers as $qNum) {
00165             if (in_array($qNum, CFG::$gradeQuestions)) {
00166                 $this->questions[$qNum] = new Question($qNum, $marks);
00167             }
00168         }
00169         // Add sections (which will add students too)
00170         foreach ($sectionNames as $sectionName) {
00171             $this->sections[] = new Section($sectionName);
00172         }
00173     }
00174
00180     private function mkFinals()
00181     {
00182         $sectionName = "All";
00183         if (empty($_POST['force'])) { // Use DB
00184             $students = self::$studentDao->getStudents();
00185             $section = new Section($sectionName);
00186             $this->sections[] = $section->setStudents($students);
00187             $questions = self::$questionDao->get();
00188             foreach ($questions as $question) {
00189                 $qNum = $question->getNum();
00190                 $subQuestions = self::$subQuestionDao->get(['question_num' => $qNum]);
00191                 $question->setSubQuestions($subQuestions);
00192                 $this->questions[$qNum] = $question;
00193             }
00194         } else {
00195             $csv = CFG::getCsvFileName("");
00196             if (!is_file($csv)) {
00197                 HT::error("Error loading eLearn report $csv<br>
00198             Please run an <strong>Attempt Details</strong> report.");
00199                 return;
00200             }
00201             // $lockFile = CFG::getSubmissionFolder($sectionName) . "mkFinal.lock";
00202             // if (file_exists($lockFile)) {
00203             //     HT::error("Students have been processed already!<br>
00204             //     If you need to reprocess, first remove the lock file:<br>
00205             //     <code>$lockFile</code>.");
00206             //     return;
00207             // }
00208             // touch($lockFile);
00209             $file = fopen($csv, 'r');
00210             $headers = fgetcsv($file);
00211             $answerIdx = array_search('Answer', $headers);
00212             $questionIdx = array_search('Q #', $headers);
00213             $nameIdx = array_search('FirstName', $headers);
00214             $emailIdx = array_search('Username', $headers);
00215             $marksIdx = array_search('Out Of', $headers);
00216             $qNum = CFG::$gradeQuestions[0];
00217             $qName = "q$qNum";
00218             $zip = new ZipArchive;
00219             $students = [];
00220             while ($line = fgetcsv($file)) {
00221                 if ($line[$questionIdx] == $qNum) {
00222                     $email = $line[$emailIdx];
00223                     $name = $line[$nameIdx];
00224                     $student = new Student($email, $name, $sectionName);
00225                     $zipFile = CFG::getSubmissionFolder($sectionName) . $email . ".zip";
00226                     $student->setZipFile($zipFile);
00227                     $res = $zip->open($zipFile, ZipArchive::CREATE);
00228                     if ($res === TRUE) {
00229                         // Trying to restore line-breaks gobbled by eLearn CSV export
00230                         // TODO: Need to find the right logic rather than hardcode it
00231                         $answer = str_replace("   ", "\n   ", $line[$answerIdx]);
00232                         // https://stackoverflow.com/a/710967
00233                         $answer = preg_replace('/^\h*\v+/m', '', $answer);
00234                         $zip->addFromString("$qName/Olympics.php", $answer);
00235                         $zip->close();
00236                     }
00237                     $marks[$qName] = $line[$marksIdx];
00238                     $students[] = $student;
00239                 }
```

```
00240             }
00241             fclose($file);
00242             $section = new Section($sectionName);
00243             $this->sections[] = $section->setStudents($students);
00244             self::$studentDao->put($section, $students);
00245             $this->questions[$qNum] = new Question($qNum, $marks);
00246             self::$questionDao->put($this->questions);
00247             $subQuestion = new SubQuestion($qName, $marks[$qName], $qNum);
00248             self::$subQuestionDao->put([$subQuestion]);
00249         }
00250     }
00251
00252     // -------------------- Public Operations --------------------
00253
00260     public function render($student)
00261     {
00262         $srcDoc = "";
00263         foreach ($this->questions as $question) {
00264             $srcDoc .= $question->render($student);
00265         }
00266         return $srcDoc;
00267     }
00268
00274     public function grade()
00275     {
00276         foreach ($this->sections as $section) {
00277             $section->grade();
00278         }
00279     }
00280
00286     public function csv()
00287     {
00288         HT::head();
00289         foreach ($this->sections as $section) {
00290             $section->csv();
00291         }
00292         HT::foot();
00293     }
00294
00300     public function export()
00301     {
00302         HT::head();
00303         foreach ($this->sections as $section) {
00304             $section->export();
00305         }
00306         HT::foot();
00307     }
00308
00314     public function stats()
00315     {
00316         HT::head();
00317         foreach ($this->sections as $section) {
00318             $section->stats();
00319         }
00320         HT::foot();
00321     }
00322
00323     // -------------------- Static Action Handlers --------------------
00324     // Dispatch handlers
00325
00331     public static function view()
00332     {
00333         self::init();
00334         $str = "";
00335         $exam = self::$exam;
00336         $marks = 0;
00337         foreach ($exam->questions as $question) {
00338             $marks += $question->getMarks();
00339             $str .= HT::info($question->print(), true);
00340         }
00341         $str .= HT::info("<h3>Total Marks = $marks</h3>", true);
00342         HT::head();
00343         HT::warn($str);
00344         HT::foot();
00345     }
00346
00352     public static function process()
00353     {
00354         self::init();
00355         self::rubrics();
00356         self::refFiles();
00357         self::submissions();
00358     }
00359
00365     public static function submissions()
00366     {
00367         self::init();
```

```
00368         HT::head();
00369         $exam = self::$exam;
00370         foreach ($exam->sections as $section) {
00371             HT::warn($section->getName());
00372             if (CFG::isQuiz()) {
00373             } else {
00374                 self::$studentDao->process($section);
00375             }
00376             foreach ($section->getStudents() as $student) {
00377                 self::$answerDao->process($student);
00378             }
00379         }
00380         HT::info("<h2>Error Message Summary of Submissions</h2>" .
00381             self::$answerDao->getProcessSummary());
00382         HT::foot();
00383     }
00384
00390     public static function rubrics()
00391     {
00392         self::init();
00393         HT::head();
00394         $exam = self::$exam;
00395         $qtnSrc = "<table width='60%' align='center' class='warn grader'>
00396                 <tr><th width='30%'>Question Number</th><th width='10%'></th>
00397                 <th width='60%'>SubQuestion</th></tr>";
00398         $rubSrc = "";
00399         foreach ($exam->questions as $question) {
00400             self::$questionDao->process();
00401             $qtnSrc .= "<tr><td>" . $question->getNum() . " (Marks:" .
00402                 $question->getMarks() .
00403                 ")</td><td></td><td></td></tr>";
00404             foreach ($question->getSubQuestions() as $subQuestion) {
00405                 self::$subQuestionDao->process($question);
00406                 self::$rubricDao->process($subQuestion);
00407                 $subQuestionName = $subQuestion->getName();
00408                 $qtnSrc .= "<tr><td></td><td>&rarr;</td>
00409                 <td>$subQuestionName  (Marks:" . $subQuestion->getMarks() .
00410                     ")</td></tr>";
00411                 $class = "success";
00412                 $rubSrc .= "<table width='95%' align='center' class='grader $class'>
00413                 <tr><th>Marks</th><th>Rubric (for $subQuestionName)</th>
00414                 <th>File Name</th></tr>";
00415                 $rubrics = self::$rubricDao->getByObject($subQuestion);
00416                 foreach ($rubrics as $k => $rubric) {
00417                     $marks = $rubric->getMarks();
00418                     $rubricName = $rubric->getRubricName();
00419                     $rubricText = $rubric->getRubricText();
00420                     $rubricText = htmlentities($rubricText);
00421                     $shortFileName = $rubric->getShortFileName();
00422                     $rubSrc .= "<tr><td> $marks </td>
00423                     <td style='text-align:left;'> $rubricName:
00424                     <code>$rubricText</code></td>
00425                     <td><code>$shortFileName</code></td></tr>";
00426                 }
00427                 $rubSrc .= "</table><br>";
00428             }
00429         }
00430         $qtnSrc .= "</table><br>";
00431         HT::info($qtnSrc . $rubSrc);
00432         HT::foot();
00433     }
00434
00440     public static function refFiles()
00441     {
00442         self::init();
00443         HT::head();
00444         $exam = self::$exam;
00445         $resSrc = "<table width='98%' align='center' class='grader warn'>
00446                 <tr><th colspan='100%'>Resources</th></tr>
00447                 <tr><th width='10%'>SubQ</th><th width='10%'>Show?</th>
00448                 <th width='80%'>File Name</th></tr>";
00449         $solSrc = "<table width='98%' align='center' class='grader warn'>
00450                 <tr><th colspan='100%'>Solutions</th></tr>
00451                 <tr><th width='10%'>SubQ</th><th width='10%'>Show?</th>
00452                 <th width='80%'>File Name</th></tr>";
00453         foreach ($exam->questions as $question) {
00454             foreach ($question->getSubQuestions() as $subQuestion) {
00455                 self::$refFileDao->process($subQuestion);
00456                 $resources = self::$refFileDao->getByObject($subQuestion, 'resources');
00457                 $solutions = self::$refFileDao->getByObject($subQuestion, 'solutions');
00458                 foreach ($resources as $resource) {
00459                     $resSrc .= self::mkRow($subQuestion, $resource);
00460                 }
00461                 foreach ($solutions as $solution) {
00462                     $solSrc .= self::mkRow($subQuestion, $solution);
00463                 }
00464             }
```

```
00465         }
00466         $resSrc .= "</table><br>";
00467         $solSrc .= "</table><br>";
00468         HT::info($resSrc . $solSrc);
00469         HT::foot();
00470     }
00471
00477     protected static function delete()
00478     {
00479         HT::head();
00480         if (!isset($_POST['confirmDelete'])) {
00481             $str = "Deleting all unzipped student files!<br>
00482               Really delete them? <table align='right'><tr>
00483               <td><form method='post' action='?do'>
00484               <input type='hidden' name='do' value='delete'>
00485               <input type='hidden' name='confirmDelete' value='dummy'>
00486               <button type='submit' class='error'>
00487               <strong>Proceed to delete</strong>
00488               </button></form></td>
00489               </tr></table>";
00490             HT::warn($str);
00491         } else {
00492             HT::error("Deleting all student submissions...");
00493             self::init();
00494             $exam = self::$exam;
00495             foreach ($exam->getSections() as $section) {
00496                 $section->delete();
00497             }
00498         }
00499         HT::foot();
00500     }
00501
00502     // -------------------- Helper and Utility Methods --------------------
00503
00511     private static function mkRow($subQuestion, $refFile)
00512     {
00513         if ($refFile->isGradable()) {
00514             $shown = "<td class='success'>Yes</td>";
00515         } else {
00516             $shown = "<td class='error'>No</td>";
00517         }
00518         $refFileName = CFG::mkFileName($refFile->getFileName());
00519         $row = "<tr><td>" . $subQuestion->getName() .
00520             $shown .
00521             "</td><td><code>" . $refFileName .
00522             "</code></td></tr>";
00523         return $row;
00524     }
00525
00533     public function __call($method, $args)
00534     {
00535         HT::head();
00536         HT::error("Got an unknown dispatch request: " . __CLASS__ .
00537             "->$method!");
00538         $this->grade();
00539         HT::foot();
00540     }
00541
00542     // -------------------- Getters --------------------
00543
00549     public function getQuestions()
00550     {
00551         return $this->questions;
00552     }
00553
00559     public function getSections()
00560     {
00561         return $this->sections;
00562     }
00563 }
```

5.31 Grader.php File Reference

Classes

- class Grader

5.32 Grader.php

Go to the documentation of this file.

```
00001 <?php
00002
00014 class Grader extends HT
00015 {
00017   protected static $class = __CLASS__;
00018
00020   protected static $actions = [
00021     'load' => 'Load Student List',
00022     'view' => 'View Marks',
00023     'export' => 'Export Grades (CSV)',
00024     'autoGrade' => 'Auto-Grade',
00025   ];
00026
00028   protected static $actionProperties = [
00029     'load' => [
00030       'target' => 'left',
00031       'clear' => 'yes'
00032     ],
00033     'view' => [
00034       'target' => 'right'
00035     ],
00036     'export' => [
00037       'target' => 'right'
00038     ],
00039     'autoGrade' => [
00040       'target' => 'left',
00041       'class' => 'error',
00042       'clear' => 'yes'
00043     ]
00044   ];
00045
00047   protected static $actionDesc = [
00048     'load' => 'Load Student List so that you can grade each one. The list is
00049     color-coded to indicate the students who were absent for the test.
00050     Once loaded, a student can be graded by clicking on their
00051     <button>Grade</button> or <button>Continue</button> button.
00052     <span style="color:red">Note that the answer files viewed and the output shown
00053     are from the current/edited versions (from the disk).</span>',
00054     'export' => 'After the grading is complete, click on this button to
00055     generate CSV files that can be imported to eLearn. The file names will
00056     be <code>Export-{gradebook}.csv</code> where <code>{gradebook}</code>
00057     is the name of the grade book exported from eLearn. The files will be
00058     in the same  folder as the grade book exports.',
00059     'view' => 'View the marks that will be exported when the next button
00060     is clicked.',
00061     '$autoGradables' => 'Run the autograder (typically for Q3, but as specified in
00062     <code>config.php</code>) for all students. It will update the database
00063     with the marks computed. The autograder file should output the total marks
00064     in some recognizable form, as specified in  <code>config.php &rarr;
00065     $autoGrade[subQuestionName]</code>, which specifies the subquestion
00066     name (usually q3) and the rubric name (usually q3-1), so that the marks
00067     can be inserted into the database. For those students whose file crashes,
00068     Grader will put zero in the database, but will give you buttons to view
00069     and manually grade q3.'
00070   ];
00071
00073   protected static $intro = "<h4>Grade Students</h4>
00074     <p>The most time-consuming process, grading the students, is triggered here.
00075     Click on the load button to get a list of students, and click on the
00076     buttons in the list to grade/regrade etc. Grader can export the marks that
00077     can be directly imported to eLearn.</p>
00078   ";
00079
00081   protected static $display = [
00082     "class" => "warn",
00083     "title" => "Grade Students"
00084   ];
00085
00086   // -------------------- Core Grading Methods --------------------
00087
00094   private static function showAnswers($student)
00095   {
00096     $wrap = 5;
00097     $srcDoc = HT::head(true);
00098     $answers = self::$answerDao->getByObject($student);
00099     $studentName = $student->getName();
00100     $studentEmail = $student->getEmail();
00101     $tabAns = "<table align='center' title='Grader::showAnswers()'><tr>
00102       <td colspan='100%' style='text-align:center;font-size:120%;'>
00103       Answers: <strong title='Grader::showAnswers()'>$studentName</strong>
00104       </td></tr><tr>";
00105     $loadAnswers = "";
```

```
00106     $iAns = 0;
00107     if (count($answers) > 0) { // Student attempted
00108       $firstAnswer = "id='firstAnswer'";
00109       foreach ($answers as $answer) {
00110         $td = $answer->render($firstAnswer);
00111         if ($td) {
00112           $tabAns .= $td;
00113           $firstAnswer = "";
00114           $iAns++;
00115           if ($iAns % $wrap == 0) {
00116             $tabAns .= "</tr><tr>";
00117           }
00118         }
00119       }
00120       $loadAnswers = "<script type='text/javascript'>
00121           window.onload=function()
00122           {document.getElementById('firstAnswer').click();
00123
      window.parent.parent.frames['left'].frames['left1'].document.getElementById('$studentEmail').submit();}

00124           </script>";
00125     } else { // Absent?
00126       $tabAns .= "<td>Not Attempted</td>";
00127     }
00128     $colSpan = $wrap - $iAns % $wrap;
00129     if ($colSpan) {
00130       $colSpan = "colspan='$colSpan'";
00131     } else {
00132       $colSpan = "";
00133     }
00134     $closeTable = "<th $colSpan><button onclick='history.back()' class='success'>
00135     <strong>Go Back</strong></button></th></tr></table>";
00136     $tabAns .= $closeTable;
00137     $srcDoc .= HT::info($tabAns, true);
00138     $srcDoc .= $loadAnswers;
00139     $srcDoc .= HT::foot(true);
00140     return $srcDoc;
00141   }
00142
00148   protected static function putMarks()
00149   {
00150     $full = isset($_POST['putFullMarks']);
00151     self::init();
00152     $row = $_POST;
00153     $subQuestionName = $row['subquestion_name'];
00154     $subQuestion = self::$subQuestionDao->get(
00155       ['subquestion_name' => $subQuestionName]
00156     )[0];
00157     $rubrics = self::$rubricDao->getByObject($subQuestion);
00158     $studentEmail = $row['student_email'];
00159     $student = self::$studentDao->getByEmail($studentEmail)[0];
00160     $rubSrc = "<table class='grader' width='80%' align='center'>
00161       <tr><th>Rubric</th><th>Award</th><th>Ref</th></tr>";
00162     $totAwarded = $totQuestion = 0.0;
00163     foreach ($rubrics as $rubric) {
00164       $rubricName = $rubric->getRubricName();
00165       if (isset($row[$rubricName])) {
00166         $marks = self::$marksDao->getByObject($student, $subQuestion, $rubric);
00167         if (empty($marks)) {
00168           $marks = new Marks($studentEmail, $subQuestionName, $rubricName, 0);
00169         } else {
00170           $marks = $marks[0];
00171         }
00172         $rubricName = $rubric->getRubricName();
00173         if ($full) {
00174           $awarded = (float) $row["full-$rubricName"];
00175         } else {
00176           $awarded = (float) $row[$rubricName];
00177         }
00178         $marks->setMarks($awarded);
00179         self::$marksDao->update($marks);
00180         $refMarks = $rubric->getMarks();
00181         $rubricText = $rubric->getRubricText();
00182         $max = max([$refMarks, 0]);
00183         if ($awarded == $max) {
00184           $class = "success";
00185         } else if ($awarded > 0) {
00186           $class = "warn";
00187         } else {
00188           $class = "error";
00189         }
00190         $totQuestion += $max;
00191         $totAwarded += $awarded;
00192         $rubricText = htmlentities($rubricText, ENT_QUOTES);
00193         $rubSrc .= "<tr class='$class'>
00194         <td style='text-align:left;'> $rubricName: <var>$rubricText</var></td>
00195         <td>$awarded</td><td> $awarded </td> </tr>";
```

```
00196       }
00197     }
00198     $rubSrc .= "<tr><th>Total</th><th>$totAwarded</th>
00199     <th>$totQuestion</th></tr></table>";
00200     $reload = "<script type='text/javascript'>var done='no';
00201       window.onload=function()
00202       {window.parent.parent.left.left1.document.getElementById('reload').submit();
00203       /* alert('Marks Updated: [$subQuestionName = $totAwarded/$totQuestion]');*/}
00204       </script>";
00205     echo $reload;
00206     // HT::success($rubSrc);
00207     // HT::foot();
00208   }
00209
00216   private static function getDefaultComment($answer)
00217   {
00218     self::init();
00219     $subQuestionName = $answer->getSubQuestionName();
00220     $shortFileName = $answer->getShortFileName();
00221     $rubrics = self::$rubricDao->get([
00222       'subquestion_name' => $subQuestionName,
00223       'short_filename' => $shortFileName
00224     ]);
00225     $comment = "";
00226     foreach ($rubrics as $rubric) {
00227       $comment .= $rubric->getRubricName() . " [" .
00228         $rubric->getRubricText() . "]: \n\n";
00229     }
00230     return $comment;
00231   }
00232
00238   public static function restoreFile()
00239   {
00240     self::init();
00241     $fileName = $_POST['filename'];
00242     $answer = self::$answerDao->getByFileName($fileName);
00243     $nAnswer = count($answer);
00244     if ($nAnswer == 0) {
00245       HT::error("Sorry, could not find the answer in the DB!<br>
00246               File: $fileName. This should never happen.");
00247       return;
00248     } elseif ($nAnswer > 1) {
00249       HT::error("Too many answers for $fileName!<br>
00250               This should never happen.");
00251       return;
00252     }
00253     $answer = $answer[0];
00254     $fileText = file_get_contents($fileName);
00255     $fileDB = $answer->getFullText();
00256     $str = "";
00257     if ($fileText == $fileDB) {
00258       $str .= HT::warn("Text in the file is identical to the one in the DB. Nothing to restore!",
    true);
00259     } else {
00260       $pathInfo = pathinfo($fileName);
00261       $to = "{$pathInfo['dirname']}/{$pathInfo['filename']}-edited.{$pathInfo['extension']}";
00262       if (file_exists($to)) {
00263         $str .= HT::error("Target file $to exists. Please keep it safe before trying to overwrite
    it!", true);
00264       } else {
00265         if (CFG::cp($fileName, $to)) {
00266           $str .= HT::success("Copied $fileName to $to successfully!", true);
00267         }
00268       }
00269       if (file_put_contents($fileName, $fileDB)) {
00270         $str .= HT::warn("Restored student update from DB to $fileName. (Overwrote it!)", true);
00271       }
00272     }
00273     HT::head();
00274     echo $str;
00275     HT::foot();
00276     return;
00277   }
00278
00284   public static function editComment()
00285   {
00286     self::init();
00287     $fileName = $_POST['filename'];
00288     $answer = self::$answerDao->getByFileName($fileName);
00289     $nAnswer = count($answer);
00290     if ($nAnswer == 0) {
00291       HT::error("Sorry, could not find the answer in the DB!<br>
00292               File: $fileName. This should never happen.");
00293       return;
00294     } elseif ($nAnswer > 1) {
00295       HT::error("Too many answers for $fileName!<br>
00296               This should never happen.");
```

```
00297       return;
00298     }
00299     $answer = $answer[0];
00300     $studentEmail = $answer->getStudentEmail();
00301     $student = self::$studentDao->getByEmail($studentEmail)[0];
00302     $studentName = $student->getName();
00303     $comment = $answer->getComment();
00304     if (empty($comment)) {
00305       $comment = self::getDefaultComment($answer);
00306     }
00307     $msg = $student->getMessage();
00308     $str = HT::info("<form method='post'>
00309       <table width='90%' align='center'>
00310       <tr><td width='20%'>Name: </td>
00311       <td><input type='text' value='$studentName $msg'
00312       style='width:90%' disabled></td></tr>
00313       <tr><td>File: </td>
00314       <td><input type='text' value='$fileName' style='width:90%' disabled></td>
00315       </tr><tr>
00316       <td valign='top'>Comment:</td>
00317       <td>
00318       <textarea style='width:90%;height:70vh;' name='comment'>$comment</textarea>
00319       </td>
00320       </tr>
00321       <tr>
00322       <th colspan='100%'><input type='submit' value='Update Comment' name='postComment'
00323       align='center'></th>
00324       </tr>
00325       </table>
00326       <input type='hidden' name='do' value='postComment'>
00327       <input type='hidden' name='student_email' value='$studentEmail'>
00328       <input type='hidden' name='filename' value='$fileName'>
00329       </form>", true);
00330     HT::head();
00331     echo $str;
00332     HT::foot();
00333     return;
00334   }
00335
00341   public static function postComment()
00342   {
00343     self::init();
00344     $fileName = $_POST['filename'];
00345     $answer = self::$answerDao->getByFileName($fileName);
00346     $nAnswer = count($answer);
00347     if ($nAnswer == 0) {
00348       HT::error("Sorry, could not find the answer in the DB!<br>
00349               File: $fileName. This should never happen.");
00350       return;
00351     } elseif ($nAnswer > 1) {
00352       HT::error("Too many answers for $fileName!<br>
00353               This should never happen.");
00354       return;
00355     }
00356     $answer = $answer[0];
00357     $studentEmail = $answer->getStudentEmail();
00358     $student = self::$studentDao->getByEmail($studentEmail)[0];
00359     $studentName = $student->getName();
00360     $comment = $_POST['comment'];
00361     $str = HT::success("
00362       <table width='90%' align='center'>
00363       <tr><td width='20%'>Name: </td><td><input type='text' value='$studentName'
00364       style='width:90%' disabled></td></tr>
00365       <tr><td>File: </td><td><input type='text' value='$fileName'
00366       style='width:90%' disabled></td></tr>
00367       <tr><td valign='top'>Comment:</td><td>
00368       <textarea style='width:90%;height:70vh;' disabled>$comment</textarea>
00369       </td></tr>
00370       <tr><th colspan='100%'>Comment Posted to the DB</th></tr>
00371       </table>");
00372     $comment = DB::$db->sanitize($_POST['comment']);
00373     $answer->setComment($comment);
00374     self::$answerDao->update($answer);
00375     HT::head();
00376     echo $str;
00377     HT::foot();
00378     return;
00379   }
00380
00386   protected static function viewFile()
00387   {
00388     $fileName = $_POST['filename'];
00389     if (isset($_POST['showDB'])) {
00390       $fulltext = $_POST['fulltext'];
00391       $output = "<form method='post' id='restoreFile'>
00392       <input type='hidden' name='do' value='restoreFile'>
00393       <input type='hidden' name='filename' value='$fileName'>
```

```
00394       <input type='submit' name='comment' value='Restore File'>
00395       </form>";
00396     } else {
00397       $fulltext = file_get_contents($fileName);
00398       $fileName0 = CFG::mkFileName($fileName);
00399       $output = "<a href='$fileName0'>
00400     <input type='submit' name='run' value='View Output'></a>";
00401     }
00402     $fulltext = htmlentities($fulltext);
00403     $ext = pathinfo($fileName, PATHINFO_EXTENSION);
00404     $commentable = isset($_POST['commentable']);
00405     HT::codeHead($fileName);
00406     $comment = "";
00407     if ($commentable) {
00408       $popup = $_POST['popup'];
00409       $student_email = $_POST['student_email'];
00410       $subquestion_name = $_POST['subquestion_name'];
00411       $comment = "<form method='post' id='editComment' target='editComment'>
00412       <input type='hidden' name='do' value='editComment'>
00413       <input type='hidden' name='filename' value='$fileName'>
00414       <input type='hidden' name='popup' value='$popup'>
00415       <input type='hidden' name='student_email' value='$student_email'>
00416       <input type='hidden' name='subquestion_name' value='$subquestion_name'>
00417       <input type='submit' name='comment' value='Edit Comment' $popup>
00418       </form>";
00419     }
00420     $realPath = "<div
00421     title='Grader::viewFile()'
00422     style='color: #373;
00423     background: #ecffd6;
00424     border: 1px solid #617c42;
00425     padding:4px;
00426     font-weight:bold;
00427     text-align:left;'><code>" .
00428       CFG::mkFileName($fileName) . "</code></div>";
00429     echo "<table padding='20' style='position:absolute;top:60px;right:15px;'>
00430     <tr><td>$comment</td><td>$output</td></tr>
00431     </table>";
00432     echo "$realPath<pre><code class='language-$ext'>$fulltext</code></pre>";
00433     HT::foot();
00434   }
00435
00436   // -------------------- Dispatch Handlers and Grading Actions --------------------
00437
00443   protected static function export()
00444   {
00445     self::init();
00446     $exam = self::$exam;
00447     $exam->export();
00448   }
00449
00455   protected static function view()
00456   {
00457     self::init();
00458     $exam = self::$exam;
00459     $exam->csv();
00460   }
00461
00467   public static function load()
00468   {
00469     self::init();
00470     HT::head();
00471     self::$exam->grade();
00472     HT::foot();
00473   }
00474
00480   public static function gradeOne()
00481   {
00482     $email = $_POST['email'];
00483     self::init();
00484     $student = self::$studentDao->getByEmail($email)[0];
00485     $left1 = htmlentities($student->render(), ENT_QUOTES);
00486     $questions = self::$exam->getQuestions();
00487     $left2 = "";
00488     foreach ($questions as $question) {
00489       $left2 .= htmlentities($question->render($student), ENT_QUOTES);
00490     }
00491     HT::head();
00492     echo "<iframe name='left1' srcdoc='$left1'
00493     style='border:none;width:98vw;height:100px;min-height:100px;' ></iframe>";
00494     echo "<iframe name='left2' srcdoc='$left2'
00495     style='border:none;width:98vw;height:90vh;' ></iframe>";
00496     HT::foot();
00497   }
00498
00504   public static function autoGrade()
00505   {
```

```
00506     self::init();
00507     HT::head();
00508     $exam = self::$exam;
00509     foreach ($exam->getSections() as $section) {
00510       foreach ($section->getStudents() as $student) {
00511         $student->autoGrade();
00512       }
00513     }
00514     HT::foot();
00515   }
00516
00517   // -------------------- Comment and Summary Helpers --------------------
00518
00525   private static function tabulateAnswerComments($answer)
00526   {
00527     $data = $answer->getComment();
00528     $comment0 = "<h3>Question {$answer->getSubQuestionName()}:
00529       <code>{$answer->getShortFileName()}</code> </h3>";
00530     $rows = [];
00531     if (!empty($data)) {
00532       // This is ugly. For the third question, we take the whole entry as comment
00533       if (strpos($data, 'q3-1') !== false) {
00534         $comment1 =
00535           "<table align='center' width='80%' class='grader warn'>
00536               <tr><th>Rerunning the Auto-Grader</th> </tr>
00537               <tr><td style='text-align: left;'><pre>$data</pre></td></tr>
00538             </table>";
00539       } else {
00540         // Convert data into paragraphs (split by double newlines)
00541         $data = str_replace(["\r\n", "\r"], "\n", $data);
00542         $entries = preg_split("/\r?\n\s*\r?\n/", trim($data));
00543         foreach ($entries as $entry) {
00544           // Match rubric number and rubric text
00545           if (preg_match('/^(q\d+[a-zA-Z]*-\d+) \[(.*?)\]:\s*(.*)$/s', trim($entry), $matches)) {
00546             $rubricNumber = $matches[1]; // e.g., q1a-6
00547
00548             $rubricText = htmlspecialchars(trim($matches[2])); // e.g., Missing or wrong <label>
    </label>
00549             $comment = htmlspecialchars(trim($matches[3])); // The rest is the comment
00550
00551             // Extract marks if present
00552             $marks = '';
00553             if (preg_match('/= (-?[\d.]+)/', $comment, $markMatches)) {
00554               $marks = $markMatches[1];
00555               $comment = trim(str_replace($markMatches[0], '', $comment));
00556             }
00557
00558             if (!empty($comment)) {
00559               // Add only if there's a valid comment
00560               $comment = $trimmed = preg_replace('/\s*marks\s*$/i', '', $comment);
00561               $commentTD = "<td style='text-align: left;'>" . nl2br($comment) . "</td>";
00562               // $commentTD = "<td style='text-align: left;'>$comment</td>";
00563               $class = $marks > 0 ? "success" : "error";
00564               $rubric = self::$rubricDao->get(["rubric_name" => $rubricNumber]);
00565               if (is_array($rubric)) {
00566                 $rubric = $rubric[0];
00567                 $outOf = $rubric->getMarks();
00568                 if ($marks >= 0) {
00569                   $outOf = abs($outOf);
00570                 }
00571               }
00572               $rows[] = "<tr class='$class'>
00573                       <td>{$rubricNumber}</td>
00574                       <td style='text-align: left;'>{$rubricText}</td>
00575                       $commentTD
00576                       <td>{$marks}</td>
00577                       <td>{$outOf}</td>
00578                   </tr>";
00579             }
00580           }
00581         }
00582
00583         // Generate the HTML table
00584         $comment1 =  "<table align='center' width='80%' class='grader warn'>
00585           <tr>
00586               <th>Rubric Number</th>
00587               <th>Rubric Text</th>
00588               <th>Comment</th>
00589               <th>Marks</th>
00590               <th>Out Of</th>
00591           </tr>
00592           " . implode("\n", $rows) . "
00593           </table>";
00594       }
00595       $comment0 .= $comment1;
00596       // Debug: Append the raw grading data
00597       // $comment0 .= "<div class='error'><pre><code>" . htmlspecialchars($data) .
```

```
    "</code></pre></div>";
00598       // var_dump($comment);
00599       return $comment0;
00600     }
00601   }
00602
00609   public static function mkMarksTable($student)
00610   {
00611     // Make a table of marks
00612     $marksTable = "<table class='grader success' align='center'>
00613     <tr><th>Question</th><th>Marks</th><th>Out Of</th></tr>";
00614     $subQuestions = self::$subQuestionDao->get();
00615     $total = $totalOutOF = 0;
00616     foreach ($subQuestions as $subQuestion) {
00617       $marks = self::$marksDao->getByObject($student, $subQuestion);
00618       $subQuestionName = $subQuestion->getName();
00619       $subTotal = 0;
00620       foreach ($marks as $mark) {
00621         $subTotal += $mark->getMarks();
00622       }
00623       $total += $subTotal;
00624       $outOf = $subQuestion->getMarks();
00625       $totalOutOF += $outOf;
00626       $subTotal = number_format($subTotal, 3);
00627       $outOf = number_format($outOf, 0);
00628       $marksTable .= "<tr>
00629       <td>$subQuestionName</td>
00630       <td>$subTotal</td>
00631       <td>$outOf</td>
00632       </tr>";
00633     }
00634     $total = number_format($total, 3);
00635     $totalOutOF = number_format($totalOutOF, 0);
00636     $marksTable .= "<tr><th>Total</th><th>$total</th><th>$totalOutOF</th></tr></table>";
00637     return $marksTable;
00638   }
00639
00645   public static function quickSummary()
00646   {
00647     $email = $_POST['email'];
00648     self::init();
00649     $student = self::$studentDao->getByEmail($email)[0];
00650     $answers = self::$answerDao->getByObject($student);
00651
00652     $comment = "";
00653     foreach ($answers as $answer) {
00654       $comment .= self::tabulateAnswerComments($answer);
00655     }
00656     if (!empty($comment)) {
00657       $comment = "<h2>Manual Grading (Overrides or Augments Auto-Grading)</h2>$comment";
00658     } else {
00659       $comment = "<h2>No Manual Grading Needed!</h2>";
00660     }
00661
00662     $subject = CFG::$examLongName . " Feedback";
00663
00664     $studentName = $student->getName();
00665     $marksTable = self::mkMarksTable($student);
00666     $intro = "<p>Dear " . ucwords(strtolower($studentName)) . ",</p>" .
00667       "<p>Here is the $subject for questions 1 and 3.
00668     The grading is done first through automated code, in two passes.
00669     Then, the auto-graded results are carefully reviewed and adjusted
00670     as necessary.</p>" .
00671       "<p>You will find the rubrics and results of the auto-grading part in the
00672     first yellow box, followed by the manual adjustments (if any)
00673     in the second grey box.</p>" .
00674       "<p>If you feel that the rubrics are not correctly applied to your
00675     answers, do get in touch with me. Note that we will not discuss
00676     the merits of the rubrics (because they are applied to all students),
00677     but only their correct application.</p>
00678     <p>Here's the top-line summary:</p>
00679     $marksTable
00680     ";
00681
00682     $emailSmu = "$email@smu.edu.sg";
00683     $msg =
00684       HT::head(true) .
00685       HT::info("<h2>$studentName</h2>" .
00686         "<span onclick='emailThis(\"$emailSmu\", \"$subject\")'
00687         title='Click to email this'>
00688         <h3>Email $subject</h3></span>
00689         <p>{$student->getMessage()}</p>", true) .
00690       HT::info($intro, true) .
00691       HT::warn(html_entity_decode($student->getComment()), true) .
00692       HT::plain($comment, true) .
00693       HT::foot(true);
00694     // echo "<iframe name='left2' srcdoc='$msg'
```

```
00695     // style='border:none;width:98vw;height:90vh;'  ></iframe>";
00696     echo $msg;
00697     return;
00698   }
00699
00700   // -------------------- Student Answer Handling --------------------
00701
00707   public static function loadAnswers()
00708   {
00709     $email = $_POST['email'];
00710     self::init();
00711     $student = self::$studentDao->getByEmail($email)[0];
00712     // $right1 = htmlentities(self::showSummary($student));
00713     $right1 = htmlentities(self::showAnswers($student), ENT_QUOTES);
00714     HT::head();
00715     echo "<iframe name='right1' srcdoc='$right1'
00716     style='border:none;width:98vw;height:150px;min-height:150px;' ></iframe>";
00717     echo "<iframe name='right2' srcdoc=''
00718     style='border:none;width:98vw;height:85vh;' ></iframe>";
00719     HT::foot();
00720   }
00721 }
```

5.33 HT.php File Reference

Classes

- class HT

5.34 HT.php

Go to the documentation of this file.

```
00001 <?php
00002
00019 class HT
00020 {
00022   protected static $graderPath;
00023
00025   private static $ico = [
00026     "plain" => "",
00027     "info" => "&#9728;",
00028     "success" => "&#10004;",
00029     "warn" => "&#9888;",
00030     "error" => "&#9747;"
00031   ];
00032
00034   protected static $class = __CLASS__;
00035
00037   protected static $actions = [
00038     'setup' => 'Setup and Validate',
00039     'process' => 'Process Submissions, Rubrics etc.',
00040     'grade' => 'Grade Students',
00041     'stats' => 'View Grade Statistics',
00042     'report' => 'Investigate'
00043   ];
00044
00046   protected static $actionProperties = [];
00047
00049   protected static $actionDesc = [
00050     'setup' => 'Set up the database, inspect and validate the configuration file
00051     <code>config.php</code>. If you see a database setup warning above, click
00052     on this button to verify your configuration, validate it, examine the SQL
00053     file and set up your database.',
00054     'process' => 'Import <code>rubrics.csv</code> to create rubrics, import
00055     the grade book downloads from eLearn to create questions, subquestions and
00056     students, and import student submissions to create answers and prepare
00057     for grading. You can also use Grader to verify that students have submitted
00058     valid zip files',
00059     'grade' => 'The next phase (the most time-consuming one) is to grade the
00060     students. Clicking on this button will make the process as painless
00061     as possible.',
00062     'stats' => 'View Grade Statistics',
00063     'report' => 'Once the grading process is completed, click on this button
00064     to export the grades to a CSV file (which can be imported to eLearn
00065     with minimal modifications). It also has some heuristics to detect
00066     potential cheating efforts and inconsistencies (such as missing or
```

```
00067     wrong emails in student files as well as similarities among them).'
00068   ];
00069
00071   protected static $intro = "<h4>Grader V2</h4>
00072   <p>Grader is a grading assistant for IS113. It follows a menu system using
00073   buttons. Each button above brings up another page with a bunch of action
00074   buttons. On each page (including this one), you click the buttons roughly
00075   from left to right to complete your grading process. And each page has its
00076   own help text, like this one.</p>
00077   <p>It can do the following:</p>
00078   <ul>
00079       <li>
00080           Read the grade book files (downloaded from eLearn as CSV) and
00081           autogenerate questions, subquestions and their marks.
00082       </li>
00083       <li>
00084           Read the same grade books to generate a list of students to be graded.
00085       </li>
00086       <li>
00087           Unpack and validate the answer zip files the students (as downloaded
00088           from eLearn assignment folders), identifying students who are absent
00089           or had issues zipping/uploading.
00090       </li>
00091       <li>
00092           Generate rubrics based on <code>rubrics.csv</code>
00093           (to be created by hand).
00094       </li>
00095       <li>
00096           Let you view and test the students' answers and their outputs and
00097           compare against reference solutions.
00098       </li>
00099       <li>
00100           Autograde specified questions using the test cases provided.
00101       </li>
00102       <li>
00103           Store the marks entered against each rubric in the database.
00104       </li>
00105       <li>
00106           Create CSV export files to be uploaded back to eLearn.
00107       </li>
00108       <li>
00109           Compare student answers and identify possible cheating attempts.
00110       </li>
00111       <li>
00112           And more that I may have forgotten.
00113       </li>
00114   </ul>
00115   <p>
00116       <strong>
00117           Preparation
00118       </strong>
00119   </p>
00120   <ul>
00121       <li>
00122         Create a folder for your lab test. Example: <code>./LT2</code> within
00123         the Grader folder, but it can be anywhere.
00124       </li>
00125       <li>
00126         Create a <code>rubrics.csv</code> file in <code>./LT2</code>. Copy and
00127         paste the rubrics from the rubrics Word file to create it.
00128       </li>
00129       <li>
00130         Download all student submissions to, e.g, <code>./LT2/submissions/</code>.
00131       </li>
00132       <li>
00133           In case there are email submissions, download them to, for instance,
00134         <code>./LT2/submissions/email/</code>.
00135       <li>
00136         Download Grade Items for the lab test into one CSV each per section.
00137       </li>
00138     </ul>";
00139
00141   private static $intro0 = "<p>The normal workflow is expected to be carried out
00142   by clicking on the buttons from left to right in the top frame. The buttons
00143   <button class='success'><strong> &Leftarrow; Previous</strong></button> and
00144   <button class='success'><strong>Next &Rightarrow;</strong></button> can also be used.
00145   The first
00146   <button class='info'><strong>Quick Info</strong></button> button will bring up
00147   this information. The <button class='success'><strong>Home&Uparrow;</strong></button>
00148   button will take you back to the main menu, the root home page.</p>
00149   <p>The buttons in <button class='warn'>Yellow</button> are typically not
00150   needed in the normal workflow, while the ones in <button class='error'>Red
00151   </button> are dangerous. They may delete information or files or launch
00152   long, cpu-intensive processes.</p> ";
00153
00155   protected static $display = ["class" => "success", "title" => ""];
00156
```

```
00157   // Data Access Objects
00158   protected static $studentDao;
00159   protected static $answerDao;
00160   protected static $marksDao;
00161   protected static $rubricDao;
00162   protected static $refFileDao;
00163   protected static $questionDao;
00164   protected static $subQuestionDao;
00165
00167   protected static $exam;
00168
00169   public function __construct()
00170   {
00171     self::head();
00172     self::error("Attempt to instantiate static class: " . static::$class);
00173     self::foot();
00174     die;
00175   }
00176
00177   // ------------------------------------------------
00178   // Initialization and Dispatcher
00179   // ------------------------------------------------
00180
00185   public static function init()
00186   {
00187     $parent = realpath(__DIR__ . "/..");
00188     self::$graderPath = substr($parent, strlen($_SERVER['DOCUMENT_ROOT']));
00189     // If the display title is not set, we are doing index.php, meaning
00190     // it's too soon to create the DAOs
00191     $dao = !empty(static::$display["title"]);
00192     // die(static::$display["title"]);
00193     if ($dao) {
00194       self::$studentDao = new StudentDAO();
00195       self::$answerDao = new AnswerDAO();
00196       self::$marksDao = new MarksDAO();
00197       self::$rubricDao = new RubricDAO();
00198       self::$refFileDao = new RefFileDAO();
00199       self::$questionDao = new QuestionDAO();
00200       self::$subQuestionDao = new SubQuestionDAO();
00201       self::$exam = new Exam();
00202     }
00203   }
00204
00210   public static function index()
00211   {
00212     // self::setRoot();
00213     $frame = array_keys($_GET);
00214     if (empty($frame)) {
00215       self::head();
00216       $dbHost = "localhost";
00217       $db = new DB($dbHost, CFG::$dbUser, CFG::$dbPass, CFG::$dbName);
00218       $error = $db->getError();
00219       if ($error) {
00220         self::head();
00221         self::error($error);
00222         self::foot();
00223       } else if (!$db->tablesExist(CFG::$dbPrefix)) {
00224         self::head();
00225         self::warn("No tables in the DB! Please run the DB setup script again.
00226         go to <strong>Setup and Configuration</strong> and set it up.");
00227         self::foot();
00228       }
00229       $top = static::top();
00230       echo "<iframe name='top' srcdoc='$top' id='top'
00231       onload='resizeIframe(this);'
00232       style='border:none;width:96vw;height:90px;' ></iframe>\n";
00233       echo self::mkFrame("left");
00234       echo self::mkFrame("right");
00235       self::foot();
00236     } else {
00237       $action = array_keys($_GET)[0];
00238       if ($action != 'do') {
00239         xdebug_print_function_stack(
00240           "Should not get here!",
00241           XDEBUG_STACK_NO_DESC
00242         );
00243       }
00244       static::$action();
00245     }
00246   }
00247
00254   public static function __callStatic($method, $args)
00255   { // Handle static dispatch request: $class::$method()
00256     static::do($method);
00257     exit();
00258   }
00259
```

```
00265   private static function do($method = '')
00266   {
00267     $class = static::$class;
00268     if (array_key_exists('do', $_POST)) {
00269       $action = $_POST['do'];
00270     } else if (array_key_exists('do', $_GET)) {
00271       $action = $_GET['do'];
00272     } else {
00273       self::head();
00274       self::error("Got an unknown static dispatch request: $class::$method()!<br>
00275         Dispatched to HT::do(), but failed.");
00276       var_dump("POST:", $_POST, "GET:", $_GET);
00277       xdebug_print_function_stack(
00278         "Should not be here!",
00279         XDEBUG_STACK_NO_DESC
00280       );
00281       self::foot();
00282       exit();
00283     }
00284     if (method_exists($class, $action)) {
00285       $class::$action();
00286       exit();
00287     } else { // In the HT class, it's just a redirection to subPages
00288       header("Location: ui/$action.php");
00289       exit();
00290     }
00291   }
00292
00293   // ------------------------------------------------
00294   // UI Frames and Navigation Helpers
00295   // ------------------------------------------------
00296
00302   protected static function mkFrame($side = 'left')
00303   {
00304     $srcdoc = htmlentities(self::quickInfo($side), ENT_QUOTES);
00305     $str = "<iframe name='$side' srcdoc='$srcdoc'
00306     style='border:none;float:left;width:48vw;height:90vh;'></iframe>\n";
00307     return $str;
00308   }
00309
00315   public static function quickInfo($side)
00316   {
00317     $str = self::head(true);
00318     if ($side == 'left') {
00319       $str0 = static::$intro;
00320       if (__CLASS__ != static::$class) {
00321         $str0 .= self::say(self::$intro0, 'plain', true);
00322       }
00323     } elseif ($side = 'right') {
00324       $actions = static::$actions;
00325       $actionProperties = static::$actionProperties;
00326       $actionDesc = static::$actionDesc;
00327       $str0 = "<h4>Action Buttons</h4>";
00328       $str0 .= "<table cellpadding='15'>";
00329       foreach ($actions as $key => $button) {
00330         $desc = "";
00331         if (array_key_exists($key, $actionDesc)) {
00332           $desc = $actionDesc[$key];
00333         }
00334         $class = "plain";
00335         if (array_key_exists($key, $actionProperties)) {
00336           $properties = $actionProperties[$key];
00337           if (array_key_exists('class', $properties)) {
00338             $class = $properties['class'];
00339           }
00340         }
00341         $str0 .= "<tr>
00342           <td width='20%'><button class='$class'>$button</button></td>
00343           <td>$desc</td>
00344           </tr>";
00345       }
00346       $str0 .= "</table>";
00347     }
00348     $display = static::$display["class"];
00349     $str .= self::$display($str0, true);
00350     $str .= self::foot(true);
00351     return "<div id='div$side'>$str</div>";
00352   }
00353
00358   private static function top()
00359   {
00360     if (empty($_POST)) {
00361       $top = self::head(true);
00362       $title = static::$display['title'];
00363       if ($title) {
00364         $title = ": $title";
00365       }
```

```
00366       $what = "<div style='text-align:center;font-size:120%;font-weight:bold'>"
00367         . CFG::$examLongName . $title . "</div>";
00368       $actions = static::$actions;
00369       $what .= static::mkTop($actions);
00370       $display = static::$display["class"];
00371       $top .= self::$display($what, true);
00372       $top .= self::foot(true);
00373       $top = htmlentities($top, ENT_QUOTES);
00374
00375       // $top = str_replace([ '"', '&' ], [ '&quot;', '&amp;amp;' ], $top);
00376
00377       return $top;
00378     }
00379   }
00380
00385   private static function mkButtons()
00386   {
00387     $buttons = [];
00388     $next = $prev = "";
00389     if (empty(static::$display["title"])) {
00390       return $buttons;
00391     }
00392     $action = basename($_SERVER['REQUEST_URI'], ".php");
00393     $actions = array_keys(self::$actions);
00394     $nActions = count($actions);
00395     for ($i = 0; $i < $nActions; $i++) {
00396       if ($action == $actions[$i]) {
00397         if ($i > 0) {
00398           $prev = $actions[$i - 1];
00399         }
00400         if ($i < $nActions - 1) {
00401           $next = $actions[$i + 1];
00402         }
00403         break;
00404       }
00405     }
00406     if ($prev) {
00407       $prevText = self::$actions[$prev];
00408       $buttons['prev'] = "<td>
00409       <a href='$prev.php' target='_parent'><button class='success'><strong>
00410       &Leftarrow; $prevText</strong></button></a></td>";
00411     }
00412     if ($next) {
00413       $nextText = self::$actions[$next];
00414       $buttons['next'] = "<td>
00415       <a href='$next.php' target='_parent'><button class='success'><strong>
00416       $nextText &Rightarrow;</strong></button></a></td>";
00417     }
00418     return $buttons;
00419   }
00420
00426   private static function mkTop($actions)
00427   {
00428     $back = !empty(static::$display["title"]);
00429     $what = "<table align='center'><tr>";
00430     $buttons = self::mkButtons();
00431     if ($back) {
00432       $target = "target='left'";
00433     } else {
00434       $target = "target='_parent'";
00435     }
00436     $method = "method='post'";
00437     if ($back) {
00438       if (array_key_exists('prev', $buttons)) {
00439         $what .= $buttons['prev'];
00440       }
00441       $what .= "
00442       <td>
00443       <a href='' target='_parent'>
00444       <button type='submit' class='info'>
00445       <strong>Quick Info</strong>
00446       </button>
00447       </a>
00448       </td>";
00449     }
00450     foreach ($actions as $name => $value) {
00451       $action = "action='?do'";
00452       $class = "class='plain'";
00453       $force = $clearFrames = "";
00454       if (array_key_exists($name, static::$actionProperties)) {
00455         $actionProperties = static::$actionProperties[$name];
00456         if (array_key_exists('target', $actionProperties)) {
00457           $target = $actionProperties['target'];
00458           $target = "target='$target'";
00459         }
00460         if (array_key_exists('class', $actionProperties)) {
00461           $class = $actionProperties['class'];
```

```
00462           $class = "class='$class'";
00463         }
00464         if (array_key_exists('clear', $actionProperties)) {
00465           $clearFrames = "onclick='window.parent.right.document.write();
00466           window.parent.left.document.write();'";
00467         }
00468         if (array_key_exists('force', $actionProperties)) {
00469           $force = "<input type='hidden' name='force' value='force'>";
00470         }
00471       }
00472       $what .= "
00473       <td>
00474       <form $action $method $target style='text-align:center;'>
00475       <input type='hidden' name='do' value='$name'>
00476       $force
00477       <button type='submit' $class $clearFrames>
00478       $value
00479       </button>
00480       </form>
00481       </td>";
00482     }
00483     if ($back) {
00484       $what .= "
00485       <td>
00486       <a href='..' target='_parent'>
00487       <button type='submit' class='success'>
00488       <strong>Home &Uparrow;</strong>
00489       </button>
00490       </a>
00491       </td>";
00492       if (array_key_exists('next', $buttons)) {
00493         $what .= $buttons['next'];;
00494       }
00495     }
00496     $what .= "</tr></table>";
00497     return $what;
00498   }
00499
00500   // ------------------------------------------
00501   // HTML and Message Rendering Helpers
00502   // ------------------------------------------
00503
00511   public static function head(
00512     $toStr = false,
00513     $title = "Grader by Manoj",
00514     $refresh = ''
00515   ) {
00516     $uiPath = self::$graderPath . "/ui";
00517     $css = "$uiPath/css/error-bar.css";
00518     $js = "$uiPath/js/utils.js";
00519     if (!empty($refresh)) {
00520       $refresh = "<meta http-equiv='refresh' content='$refresh'>";
00521     }
00522     $str = "<!DOCTYPE html>
00523     <html>
00524       <head>
00525         <title>$title</title>
00526         <link href='$css' rel='stylesheet'>
00527         $refresh
00528         <script src='$js'></script>
00529       </head>
00530       <body>
00531       ";
00532     if ($toStr) {
00533       return $str;
00534     } else {
00535       echo $str;
00536     }
00537   }
00538
00544   public static function codeHead($title = "Code Listing by Manoj")
00545   {
00546     $path = ".";
00547     $str = "<!DOCTYPE html>
00548     <html>
00549       <head>
00550         <title>$title</title>
00551         <link rel='stylesheet' href='$path/highlight/styles/agate.min.css'>
00552         <script src='$path/highlight/highlight.min.js'></script>
00553         <script>hljs.highlightAll();</script>
00554       </head>
00555       <body>
00556       ";
00557     echo $str;
00558   }
00559
00565   public static function foot($toStr = false)
```

```
00566   {
00567     $str = "  </body>
00568     </html>";
00569     if ($toStr) {
00570       return $str;
00571     } else {
00572       echo $str;
00573     }
00574   }
00575
00584   public static function say($what, $class = 'plain', $toStr = false, $showIcon = false)
00585   {
00586     if ($showIcon) {
00587       $icon = self::$ico[$class];
00588       $icon = "<i class='ico'>$icon</i>";
00589     } else {
00590       $icon = "";
00591     }
00592     $str = "    <div class='bar  $class'>
00593     $icon $what
00594   </div>
00595   ";
00596     if ($toStr) {
00597       return $str;
00598     } else {
00599       echo $str;
00600     }
00601   }
00602
00607   public static function getMsgTypes()
00608   {
00609     return array_keys(self::$ico);
00610   }
00611
00618   public static function error($what, $toStr = false)
00619   {
00620     return self::say($what, $class = "error", $toStr);
00621   }
00622
00629   public static function warn($what, $toStr = false)
00630   {
00631     return self::say($what, $class = "warn", $toStr);
00632   }
00633
00640   public static function info($what, $toStr = false)
00641   {
00642     return self::say($what, $class = "info", $toStr);
00643   }
00644
00651   public static function success($what, $toStr = false)
00652   {
00653     return self::say($what, $class = "success", $toStr);
00654   }
00655
00662   public static function plain($what, $toStr = false)
00663   {
00664     return self::say($what, $class = "plain", $toStr);
00665   }
00666
00672   public function print($toStr = true)
00673   {
00674     $str = "";
00675     $msgTypes = self::getMsgTypes();
00676     foreach ($msgTypes as $fun) {
00677       if ($this->$fun) {
00678         $str .= self::$fun($this->$fun, true);
00679       }
00680     }
00681     return $str;
00682   }
00683
00696   private static function setRoot()
00697   {
00698     if (isset($GLOBALS['grader'])) {
00699       $grader = $GLOBALS['grader'];
00700       echo '<script>window.history.pushState({}, "", "/Grader/' . $grader . '/index.php");</script>';
00701     } else {
00702       $GLOBALS['grader'] = $root = basename(__DIR__);
00703     }
00704   }
00705 }
```

5.35 Marks.php File Reference

Classes

- class Marks

5.36 Marks.php

Go to the documentation of this file.

```
00001 <?php
00002
00009 class Marks
00010 {
00014     private $studentEmail;
00015
00019     private $subQuestionName;
00020
00024     private $rubricName;
00025
00029     private $marks;
00030
00039     public function __construct($studentEmail, $subQuestionName, $rubricName, $marks)
00040     {
00041         $this->studentEmail = $studentEmail;
00042         $this->subQuestionName = $subQuestionName;
00043         $this->rubricName = $rubricName;
00044         $this->marks = $marks;
00045     }
00046
00052     public function getMarks()
00053     {
00054         return $this->marks;
00055     }
00056
00062     public function getStudentEmail()
00063     {
00064         return $this->studentEmail;
00065     }
00066
00072     public function getSubQuestionName()
00073     {
00074         return $this->subQuestionName;
00075     }
00076
00082     public function getRubricName()
00083     {
00084         return $this->rubricName;
00085     }
00086
00093     public function setMarks($marks): self
00094     {
00095         $this->marks = $marks;
00096         return $this;
00097     }
00098 }
```

5.37 Question.php File Reference

Classes

- class Question

5.38 Question.php

Go to the documentation of this file.

```
00001 <?php
00002
00017 class Question extends HT
00018 {
00022     private $num;
00023
00027     private $marks = 0;
00028
00032     private $subQuestions = [];
00033
00040     public function __construct($num, $marks)
00041     {
00042         $this->num = $num;
00043         if (is_array($marks)) {
00044             foreach ($marks as $subQuestionName => $subQuestionMarks) {
00045                 if ($this->isSubQuestion($subQuestionName)) {
00046                     $this->marks += $subQuestionMarks;
00047                     $this->subQuestions[$subQuestionName] =
00048                         new SubQuestion($subQuestionName, $subQuestionMarks, $num);
00049                 }
00050             }
00051         } else {
00052             $this->marks = $marks;
00053         }
00054     }
00055
00062     public function render($student)
00063     {
00064         $srcDoc = HT::head(true);
00065         foreach ($this->subQuestions as $subQuestion) {
00066             $srcDoc .= $subQuestion->render($student);
00067         }
00068         $srcDoc .= HT::foot(true);
00069         return $srcDoc;
00070     }
00071
00078     public function print($toStr = true)
00079     {
00080         $str = "<h3> Question: $this->num,  Marks: $this->marks</h3>";
00081         foreach ($this->subQuestions as $subQuestion) {
00082             $str .= $subQuestion->print();
00083         }
00084         return $str;
00085     }
00086
00093     public function view()
00094     {
00095         // Future implementation
00096     }
00097
00104     private function isSubQuestion($sqName)
00105     {
00106         return stripos($sqName, "q$this->num") === 0;
00107     }
00108
00114     public function getSubQuestions()
00115     {
00116         return $this->subQuestions;
00117     }
00118
00124     public function getMarks()
00125     {
00126         return $this->marks;
00127     }
00128
00134     public function getNum()
00135     {
00136         return $this->num;
00137     }
00138
00145     public function setSubQuestions($subQuestions): self
00146     {
00147         $this->subQuestions = $subQuestions;
00148         return $this;
00149     }
00150 }
```

5.39 RefFile.php File Reference

Classes

- class RefFile

5.40 RefFile.php

Go to the documentation of this file.

```
00001 <?php
00002
00009 class RefFile
00010 {
00012     private $subQuestionName;
00013
00015     private $fileType;
00016
00018     private $fileName;
00019
00021     private $fullText;
00022
00024     private $hash0;
00025
00027     private $hash1;
00028
00030     private $sim;
00031
00033     private $gradable = false;
00034
00036     private $shortFileName;
00037
00048     public function __construct(
00049         $subQuestionName,
00050         $fileType,
00051         $fileName,
00052         $fullText = '',
00053         $hash0 = '',
00054         $hash1 = ''
00055     ) {
00056         $this->subQuestionName = $subQuestionName;
00057         $this->fileType = $fileType;
00058         $this->fileName = $fileName;
00059         if (empty($fullText)) {
00060             $fullText = DB::$db->getFileContent($fileName);
00061         }
00062         $this->fullText = $fullText;
00063         if (empty($hash0)) {
00064             $hash0 = Report::hash($fileName);
00065         }
00066         $this->hash0 = $hash0;
00067         if (empty($hash1)) {
00068             $hash1 = Report::hash($fileName, true);
00069         }
00070         $this->hash1 = $hash1;
00071         $folders = [
00072             'resources' => CFG::$resourceFolder,
00073             'solutions' => CFG::$solutionFolder,
00074         ];
00075         $this->shortFileName = trim(
00076             str_ireplace($folders[$fileType], "", $fileName),
00077             '/'
00078         );
00079     }
00080
00086     public function getSubQuestionName()
00087     {
00088         return $this->subQuestionName;
00089     }
00090
00096     public function getFullText()
00097     {
00098         return $this->fullText;
00099     }
00100
00106     public function getFileType()
00107     {
00108         return $this->fileType;
00109     }
00110
```

```
00116     public function getFileName()
00117     {
00118         return $this->fileName;
00119     }
00120
00126     public function getHash0()
00127     {
00128         return $this->hash0;
00129     }
00130
00136     public function getHash1()
00137     {
00138         return $this->hash1;
00139     }
00140
00147     public function setGradable($gradable): self
00148     {
00149         $this->gradable = $gradable;
00150         return $this;
00151     }
00152
00158     public function isGradable()
00159     {
00160         return $this->gradable;
00161     }
00162
00168     public function getShortFileName()
00169     {
00170         return $this->shortFileName;
00171     }
00172
00179     public function setSim($sim): self
00180     {
00181         $this->sim = $sim;
00182         return $this;
00183     }
00184
00190     public function getSim()
00191     {
00192         return $this->sim;
00193     }
00194 }
```

5.41 Rubric.php File Reference

Classes

- class Rubric

5.42 Rubric.php

Go to the documentation of this file.

```
00001 <?php
00002
00010 class Rubric
00011 {
00013     private $subQuestionName;
00014
00016     private $rubricName;
00017
00019     private $marks;
00020
00022     private $rubricText;
00023
00025     private $shortFileName;
00026
00036     public function __construct(
00037         $subQuestionName,
00038         $rubricName,
00039         $marks,
00040         $rubricText,
00041         $shortFileName
00042     ) {
00043         $this->subQuestionName = $subQuestionName;
00044         $this->rubricName = $rubricName;
00045         $this->marks = $marks;
```

```
00046         $this->rubricText = $rubricText;
00047         $this->shortFileName = $shortFileName;
00048     }
00049
00055     public function getRubricName()
00056     {
00057         return $this->rubricName;
00058     }
00059
00065     public function getMarks()
00066     {
00067         return $this->marks;
00068     }
00069
00075     public function getSubQuestionName()
00076     {
00077         return $this->subQuestionName;
00078     }
00079
00085     public function getRubricText()
00086     {
00087         return $this->rubricText;
00088     }
00089
00095     public function getShortFileName()
00096     {
00097         return $this->shortFileName;
00098     }
00099 }
```

5.43 Section.php File Reference

Classes

- class Section

5.44 Section.php

Go to the documentation of this file.

```
00001 <?php
00002
00009 class Section extends HT
00010 {
00014     private $students = [];
00015
00019     private $name;
00020
00027     public function __construct($name)
00028     {
00029         $this->name = $name;
00030         $this->students = self::$studentDao->get(["section" => $name]);
00031
00032         if (empty($this->students)) { // Make students from the CSV files
00033             if (!CFG::isQuiz()) { // Add students only for lab tests
00034                 $this->mkStudents();
00035             }
00036         }
00037     }
00038
00044     private function mkStudents()
00045     {
00046         $csv = CFG::getCsvFileName($this->name);
00047         if (!is_file($csv)) {
00048             HT::error("Error loading $csv");
00049             return;
00050         }
00051         $file = fopen($csv, 'r');
00052         $line = fgetcsv($file); // Skip the header
00053         while (($line = fgetcsv($file)) !== false) {
00054             $this->students[] = new Student($line[0], $line[1], $this->name);
00055         }
00056         fclose($file);
00057     }
00058
00065     public function grade($name = '')
00066     {
```

```
00067         HT::say("<h4>Section: $this->name</h4>");
00068         foreach ($this->students as $student) {
00069             $student->grade();
00070         }
00071     }
00072
00079     public function delete($name = '')
00080     {
00081         foreach ($this->students as $student) {
00082             $student->delete();
00083         }
00084     }
00085
00091     public function csv()
00092     {
00093         $str = "<table class='grader' width='98%'><tr><th colspan='100%'>Section:
      $this->name</th></tr>";
00094         foreach ($this->students as $student) {
00095             $str .= $student->csv();
00096         }
00097         $str .= "</table>";
00098         HT::warn($str);
00099     }
00100
00107     public function print($toStr = true)
00108     {
00109         $str = $this->info("<strong> Section: $this->name</strong>", $toStr);
00110         foreach ($this->students as $std) {
00111             $str .= $std->print();
00112         }
00113         return $str;
00114     }
00115
00121     public function getStudents()
00122     {
00123         return $this->students;
00124     }
00125
00132     public function setStudents($students): self
00133     {
00134         $this->students = $students;
00135         return $this;
00136     }
00137
00143     public function getName()
00144     {
00145         return $this->name;
00146     }
00147
00153     public function export()
00154     {
00155         $csv = CFG::getCsvFileName($this->name);
00156         if (!is_file($csv)) {
00157             HT::error("Error loading $csv");
00158             return;
00159         }
00160         $exported = dirname($csv) . "/Export-" . basename($csv);
00161         $fileIn = fopen($csv, 'r');
00162         $fileOut = fopen($exported, 'w');
00163         if ($fileOut === false) {
00164             HT::error("Error opening the file $exported");
00165         }
00166
00167         if (CFG::isQuiz()) {
00168             $headers = ["email", "Username"];
00169             $qNums = [];
00170             foreach (CFG::$autoGradables as $file => $autoGrade) {
00171                 $qNums[] = $autoGrade['subQuestionName'];
00172             }
00173             foreach ($qNums as $qNum) {
00174                 $headers[] = $qNum;
00175             }
00176         } else {
00177             $headers = fgetcsv($fileIn); // copy the header
00178         }
00179         fputcsv($fileOut, $headers);
00180
00181         foreach ($this->students as $student) {
00182             $line = [];
00183             $line[0] = "#" . $student->getEmail();
00184             $line[1] = $student->getName();
00185             $marks = $student->sumMarksBySubQuestion();
00186
00187             foreach ($headers as $iCol => $header) {
00188                 if ($iCol < 2) {
00189                     continue;
00190                 }
```

```
00191                 $line[$iCol] = "";
00192                 foreach ($marks as $subQuestionName => $m) {
00193                     if (stripos($header, $subQuestionName) !== false) {
00194                         $line[$iCol] = $m;
00195                         break;
00196                     }
00197                 }
00198             }
00199
00200             if (CFG::isQuiz()) {
00201                 fputcsv($fileOut, $line);
00202             } else {
00203                 // Drop the last element (a spurious blank) in line
00204                 unset($line[$iCol]);
00205                 fputcsv($fileOut, $line, ",", "\"", "\\", ",#\n");
00206             }
00207         }
00208
00209         fclose($fileOut);
00210         $fullText = file_get_contents($exported);
00211         // Remove quotation marks - eLearn doesn't like them
00212         $fullText = str_replace("\"", "", $fullText);
00213         file_put_contents($exported, $fullText);
00214         $exported = realpath($exported);
00215         HT::success("Exported $this->name marks to <code>$exported</code>");
00216     }
00217
00223     public function stats()
00224     {
00225         foreach ($this->students as $student) {
00226         }
00227         HT::success("Statistics for $this->name.<br>To be implemented.");
00228     }
00229 }
```

5.45 Stat.php File Reference

Classes

- class Stat

5.46 Stat.php

Go to the documentation of this file.

```
00001 <?php
00002
00012 class Stat extends HT
00013 {
00015     protected static $actions = [
00016         'all' => 'Overall',
00017         'sections' => 'By Section',
00018         'questions' => 'By Question',
00019         'subquestions' => 'By Subquestion',
00020         'mistakes' => 'Common Mistakes'
00021     ];
00022
00024     protected static $actionProperties = [
00025         'all' => [
00026             'target' => 'left',
00027             'clear' => 'yes'
00028         ],
00029         'sections' => [
00030             'target' => 'right'
00031         ],
00032         'questions' => [
00033             'target' => 'left'
00034         ],
00035         'subquestions' => [
00036             'target' => 'left'
00037         ],
00038         'mistakes' => [
00039             'target' => 'left'
00040         ]
00041     ];
00042
00044     protected static $actionDesc = [
```

```
00045         'all' => 'Overall statistics for all your sections.',
00046         'sections' => 'Statistics broken down by section.',
00047         'questions' => 'Statistics by question, with option to drill down by
00048         sections.',
00049         'subquestions' => 'Statistics by subquestion, with option to drill down by
00050         sections.',
00051         'mistakes' => 'Display comments grouped by sub-questions.',
00052     ];
00053
00055     protected static $class = __CLASS__;
00056
00058     protected static $intro = "<h4>Statistics</h4>
00059       <p>Obvious, isn't it? This page lets you generate marks statistics.</p>";
00060
00062     protected static $display = [
00063         "class" => "info",
00064         "title" => "View Grade Statistics"
00065     ];
00066
00073     protected static function getRows($reduce = ["section" => "s.section"])
00074     {
00075         $rows = DB::$db->query(self::mkSql($reduce))->fetchAll();
00076         return $rows;
00077     }
00078
00087     protected static function printStats(
00088         $grouped,
00089         $byTitle = "Grouped by",
00090         $toString = false,
00091         $drillDown = ''
00092     ) {
00093         $rows = "";
00094         foreach ($grouped as $by => $marks) {
00095             $rows .= "<tr><th>$by</th>";
00096             $mean = number_format(self::mean($marks), 2);
00097             $std = number_format(self::stdev($marks), 2);
00098             $median = number_format(self::median($marks), 2);
00099             $min = min($marks);
00100             $minCount = self::count($marks, $min);
00101             $max = max($marks);
00102             $maxCount = self::count($marks, $max);
00103             $min = number_format($min, 2) . " <small>[$minCount]</small>";
00104             $max = number_format($max, 2) . " <small>[$maxCount]</small>";
00105             $rows .= "<td>$mean</td>
00106             <td>$std</td>
00107             <td>$median</td>
00108             <td>$min</td>
00109             <td>$max</td></tr>";
00110         }
00111         if ($toString) {
00112             return $rows;
00113         }
00114         $rows .= "<tr><td colspan='10'><small>[count in brackets]</small></td></tr>";
00115         if ($drillDown) {
00116             $rows .= "<tr><th colspan='100%'>
00117             <form method='post' target='right'>
00118             <input type='hidden' name='do' value='$drillDown'>
00119             <button class='info' type='submit'>Drill-down by Section</button>
00120             </form></th></tr>";
00121         }
00122         $str = "<table class='grader success' align='center'>
00123         <tr><th>$byTitle</th><th>Mean</th><th>Std Dev</th>
00124         <th>Median</th><th>Min</th><th>Max</th></tr>
00125         $rows
00126         </table>";
00127         HT::head();
00128         HT::warn($str);
00129         HT::foot();
00130     }
00131
00139     protected static function printStats2($grouped2, $byTitle1, $byTitle2)
00140     {
00141         $rowsGrouped = [];
00142         foreach ($grouped2 as $key => $grouped) {
00143             $rows = self::printStats($grouped, $byTitle2, true);
00144             $rowSpan = substr_count($rows, "<tr>");
00145             $rowsGrouped[] = preg_replace(
00146                 "/<tr>/",
00147                 "<tr><th rowspan='$rowSpan'>$key</th>",
00148                 $rows,
00149                 1
00150             );
00151         }
00152
00153         $str = "<table class='grader success' align='center'>
00154         <tr><th>$byTitle1</th><th>$byTitle2</th><th>Mean</th><th>Std Dev</th>
00155         <th>Median</th><th>Min</th><th>Max</th></tr>" .
```

```
00156             implode("\n", $rowsGrouped) .
00157             "<tr><td colspan='10'><small>[count in brackets]</small></td></tr>" .
00158             "</table>";
00159         HT::head();
00160         HT::warn($str);
00161         HT::foot();
00162     }
00170     private static function groupBy($rows, $by = '')
00171     {
00172         $grouped = [];
00173         foreach ($rows as $row) {
00174             if (empty($by)) {
00175                 $key = "All";
00176             } else {
00177                 $key = $row[$by];
00178             }
00179             $grouped[$key][] = (float) $row['marks'];
00180         }
00181         ksort($grouped);
00182         return $grouped;
00183     }
00184
00193     private static function groupBy2($rows, $by1, $by2)
00194     {
00195         $grouped = [];
00196         foreach ($rows as $row) {
00197             $key1 = $row[$by1];
00198             $key2 = $row[$by2];
00199             if (empty($grouped[$key1][$key2])) {
00200                 $grouped[$key1][$key2] = [];
00201             }
00202             $grouped[$key1][$key2][] = (float) $row['marks'];
00203         }
00204         ksort($grouped);
00205         foreach ($grouped as $key => $bySubQuestion) {
00206             ksort($grouped[$key]);
00207         }
00208         return $grouped;
00209     }
00210
00214     protected static function all()
00215     {
00216         $rows = self::getRows();
00217         $grouped = self::groupBy($rows);
00218         self::printStats($grouped);
00219     }
00220
00224     protected static function sections()
00225     {
00226         $rows = self::getRows();
00227         $grouped = self::groupBy($rows, 'section');
00228         self::printStats($grouped, 'Section');
00229     }
00230
00234     protected static function questions()
00235     {
00236         $rows = self::getRows(['question' => 'sq.question_num']);
00237         $grouped = self::groupBy($rows, 'question');
00238         self::printStats($grouped, 'Question', false, "questions2");
00239     }
00240
00244     protected static function questions2()
00245     {
00246         $rows = self::getRows([
00247             'section' => 's.section',
00248             'question' => 'sq.question_num'
00249         ]);
00250         $grouped2 = self::groupBy2($rows, 'question', 'section');
00251         self::printStats2($grouped2, 'Question', 'Section');
00252     }
00253
00257     protected static function subquestions()
00258     {
00259         $rows = self::getRows(['subquestion' => 'sq.subquestion_name']);
00260         $grouped = self::groupBy($rows, 'subquestion');
00261         self::printStats($grouped, 'Subquestion', false, "subquestions2");
00262     }
00263
00267     protected static function subquestions2()
00268     {
00269         $rows = self::getRows([
00270             'section' => 's.section',
00271             'subquestion' => 'sq.subquestion_name'
00272         ]);
00273         $grouped2 = self::groupBy2($rows, 'subquestion', 'section');
00274         self::printStats2($grouped2, 'Subquestion', 'Section');
00275     }
```

```
00276
00283     private static function mkSql($reduce)
00284     {
00285         $select = $groupBy = "";
00286         foreach ($reduce as $alias => $field) {
00287             $select .= "$field AS 'alias',";
00288             $groupBy .= "`$alias`, ";
00289         }
00290         $groupBy = trim($groupBy, ", ");
00291         $sql = "SELECT
00292             s.student_email AS 'email',
00293             s.student_name AS 'name',
00294             $select
00295             SUM(m.marks) AS 'marks'
00296         FROM
00297             `{dbPrefix}students` AS s
00298             INNER JOIN `{dbPrefix}marks` AS m ON m.student_email = s.student_email
00299             INNER JOIN `{dbPrefix}subquestions` AS sq ON m.subquestion_name = sq.subquestion_name
00300         GROUP BY
00301            `email`, `name`, $groupBy";
00302         $sql = str_replace('{dbPrefix}', CFG::$dbPrefix, $sql);
00303         return $sql;
00304     }
00305
00309     public static function mistakes()
00310     {
00311         $sql = "SELECT
00312             subquestion_name, comment
00313         FROM
00314             `{dbPrefix}answers`
00315         WHERE
00316             comment <> ''
00317         ORDER BY
00318             subquestion_name";
00319         $sql = str_replace('{dbPrefix}', CFG::$dbPrefix, $sql);
00320         $rows = DB::$db->query($sql)->fetchAll();
00321         $table = "<table width='80%' align='center' class='warn grader'>
00322                 <tr><th width='20%'>Sub Question Number</th>
00323                 <th width='80%'>Comment</th></tr>";
00324         foreach ($rows as $row) {
00325             $comment = "<pre>" . htmlentities($row["comment"]) . "</pre>";
00326             $table .= "<tr><td>{$row['subquestion_name']}</td>
00327             <td style='text-align:left'>$comment</td></tr>";
00328         }
00329         $table .= "</table><br>";
00330         HT::head();
00331         HT::info($table);
00332         HT::foot();
00333     }
00334
00343     private static function count($data, $value, $tolerance = 0.01)
00344     {
00345         $count = 0;
00346         foreach ($data as $val) {
00347             if (abs($val - $value) <= $tolerance) {
00348                 $count++;
00349             }
00350         }
00351         return $count;
00352     }
00353
00360     public static function mean($data)
00361     {
00362         $n = count($data);
00363         if ($n == 0) {
00364             $mean = 0;
00365         } else {
00366             $mean = array_sum($data) / $n;
00367         }
00368         return $mean;
00369     }
00370
00377     public static function median($data)
00378     {
00379         sort($data);
00380         $elements = count($data);
00381         if (($elements % 2) == 0) {
00382             $i = $elements / 2;
00383             return (($data[$i - 1] + $data[$i]) / 2);
00384         } else {
00385             $i = ($elements - 1) / 2;
00386             return $data[$i];
00387         }
00388     }
00389
00396     public static function range($data)
00397     {
```

```
00398         return (max($data) - min($data));
00399     }
00400
00407     public static function var($data)
00408     {
00409         $n = count($data);
00410         if ($n <= 1) {
00411             $var = 0;
00412         } else {
00413             $mean = self::mean($data);
00414             $sum = 0;
00415             foreach ($data as $element) {
00416                 $sum += pow(($element - $mean), 2);
00417             }
00418             $var = $sum / ($n - 1);
00419         }
00420         return $var;
00421     }
00422
00429     public static function varp($data)
00430     {
00431         $n = count($data);
00432         if ($n <= 1) {
00433             $var = 0;
00434         } else {
00435             $mean = self::mean($data);
00436             $sum = 0;
00437             foreach ($data as $element) {
00438                 $sum += pow(($element - $mean), 2);
00439             }
00440             $var = $sum / $n;
00441         }
00442         return $var;
00443     }
00444
00451     public static function stdev($data)
00452     {
00453         return sqrt(self::var($data));
00454     }
00455
00462     public static function stdevp($data)
00463     {
00464         return sqrt(self::varp($data));
00465     }
00466
00481     public static function simple_regression($x, $y)
00482     {
00483         $output = array();
00484         $output['a'] = 0;
00485         $n = min(count($x), count($y));
00486         $mean_x = self::mean($x);
00487         $mean_y = self::mean($y);
00488         $SS_x = 0;
00489         foreach ($x as $element) {
00490             $SS_x += pow(($element - $mean_x), 2);
00491         }
00492         $SS_y = 0;
00493         foreach ($y as $element) {
00494             $SS_y += pow(($element - $mean_y), 2);
00495         }
00496         $SS_xy = 0;
00497         for ($i = 0; $i < $n; $i++) {
00498             $SS_xy += ($x[$i] - $mean_x) * ($y[$i] - $mean_y);
00499         }
00500         $output['b'] = $SS_xy / $SS_x;
00501         $output['a'] = $mean_y - $output['b'] * $mean_x;
00502         $output['s'] = sqrt(($SS_y - $output['b'] * $SS_xy) / ($n - 2));
00503         $output['r'] = $SS_xy / sqrt($SS_x * $SS_y);
00504         $output['r2'] = pow($output['r'], 2);
00505         $output['cov'] = $SS_xy / ($n - 1);
00506         $output['t'] = $output['r'] / sqrt((1 - $output['r2']) / ($n - 2));
00507
00508         return $output;
00509     }
00510 }
```

5.47 Student.php File Reference

Classes

- class Student

5.48 Student.php

Go to the documentation of this file.

```
00001 <?php
00002
00011 class Student extends HT
00012 {
00014     private $email;
00015
00017     private $name;
00018
00020     private $section;
00021
00023     private $zipFile;
00024
00026     private $status = "New";
00027
00029     private $comment = "";
00030
00038     public function __construct($email, $name, $section)
00039     {
00040         $this->email = trim(trim($email), "#");
00041         $this->name = $name;
00042         $this->section = $section;
00043     }
00044
00050     public function getAnswerFolder()
00051     {
00052         $sectionName = $this->section;
00053         // $studentName = $this->name;
00054         // $answerFolder = CFG::getAnswerFolder($sectionName, $studentName);
00055         $studentEmail = $this->email;
00056         $answerFolder = CFG::getAnswerFolder($sectionName, $studentEmail);
00057         return $answerFolder;
00058     }
00059
00065     public function render()
00066     {
00067         list($prev, $prevUngraded, $nextUngraded, $next)
00068             = self::$studentDao->getOthers($this);
00069         $srcDoc = HT::head(true);
00070         $prevForm =
00071             $prevUngradedForm =
00072             $nextForm =
00073             $nextUngradedForm = "<td style='width:1%'></td>";
00074         $clearRight = "onclick='var right1 = window.parent.parent.right.right1;
00075         right1.document.write(); var right2 = window.parent.parent.right.right2;
00076         right2.document.write();'";
00077         if ($prev) {
00078             $prevForm = "<td style='width:1%'>
00079             <form method='post' target='left' action='?do'>
00080             <input type='hidden' name='do' value='gradeOne'>
00081             <input type='hidden' name='email' value='$prev->email'>
00082             <input type='submit' value='Prev' title='$prev->name' $clearRight >
00083             </form></td>";
00084         }
00085         if ($prevUngraded) {
00086             $prevUngradedForm = "<td style='width:1%'>
00087             <form method='post' target='left' action='?do'>
00088             <input type='hidden' name='do' value='gradeOne'>
00089             <input type='hidden' name='email' value='$prevUngraded->email'>
00090             <input type='submit' value='Prev Ungraded'
00091             title='$prevUngraded->name' $clearRight>
00092             </form></td>";
00093         }
00094         if ($next) {
00095             $nextForm = "<td style='width:1%'>
00096             <form method='post' target='left' action='?do'>
00097             <input type='hidden' name='do' value='gradeOne' >
00098             <input type='hidden' name='email' value='$next->email'>
00099             <input type='submit' value='Next' title='$next->name' $clearRight>
00100             </form></td>";
00101         }
00102         if ($nextUngraded) {
00103             $nextUngradedForm = "<td style='width:1%'>
00104             <form method='post' target='left' action='?do'>
00105             <input type='hidden' name='do' value='gradeOne'>
00106             <input type='hidden' name='email' value='$nextUngraded->email'>
00107             <input type='submit' value='Next Ungraded'
00108             title='$nextUngraded->name' $clearRight>
00109             </form></td>";
00110         }
00111         $nStudents = self::$studentDao->getNStudents();
00112         $iStudent = self::$studentDao->getIStudent($this->email);
00113         $srcDoc .= HT::info("<div style='text-align:center;font-size:120%;'>
```

```
00114         <table align='center' style='width:100%'><tr>$prevForm $prevUngradedForm
00115         <td><strong title='Student::render()'>$this->name</strong> 
00116         <small style='font-size:80%;'> [<strong>$this->section</strong>:
00117         $iStudent of $nStudents]</small></td>
00118         $nextUngradedForm $nextForm </tr></table></div>
00119         <table align='center'><tr>
00120         <td><form method='post' target='left' id='reload' action='?do'>
00121         <input type='submit' value='Reload'>
00122         <input type='hidden' name='do' value='gradeOne'>
00123         <input type='hidden' name='name' value='$this->name'>
00124         <input type='hidden' name='email' value='$this->email'>
00125         </form></td>
00126         <td><form method='post' target='right' action='?do'>
00127         <input type='submit' id='loadAnswers' value='Show Answers'>
00128         <input type='hidden' name='do' value='loadAnswers'>
00129         <input type='hidden' name='name' value='$this->name'>
00130         <input type='hidden' name='email' value='$this->email'>
00131         </form></td>
00132         <td><form method='post' target='quickSummary' action='?do' id='$this->email'>
00133         <input type='submit' name='summary' value='Quick Summary' id='quickSummary'
00134         onclick=
00135         \"var h = screen.height * 0.8;
00136         var t = screen.height * 0.05;
00137         var w = screen.width * 0.4;
00138         var l = screen.width * 0.05;
00139         window.open('', 'quickSummary', 'width='+ w +',height=' + h + ',left='+ l +',top='+ t
      +',resizable=yes,scrollbars=yes,toolbar=yes,menubar=no,location=no,directories=no,status=yes');
00140           document.getElementById('$this->email').submit();
00141           return true;\">
00142         <input type='hidden' name='do' value='quickSummary'>
00143         <input type='hidden' name='name' value='$this->name'>
00144         <input type='hidden' name='email' value='$this->email'>
00145         </form></td>
00146         </tr></table>
00147         ", true);
00148         $loadAnswers = "<script type='text/javascript'>
00149         window.onload=function(){document.getElementById('loadAnswers').click();}
00150         </script>";
00151         $srcDoc .= $loadAnswers;
00152         $srcDoc .= HT::foot(true);
00153         return $srcDoc;
00154     }
00155
00161     private function sumMarks()
00162     {
00163         $marksAll = self::$marksDao->getByObject($this);
00164         $total = 0;
00165         foreach ($marksAll as $marks) {
00166             $total += $marks->getMarks();
00167         }
00168         return $total;
00169     }
00170
00176     public function sumMarksBySubQuestion()
00177     {
00178         $sums = [];
00179         foreach (self::$exam->getQuestions() as $question) {
00180             $subQuestions = $question->getSubquestions();
00181             foreach ($subQuestions as $subQuestion) {
00182                 $marks = self::$marksDao->getByObject($this, $subQuestion);
00183                 $subQuestionName = $subQuestion->getName();
00184                 foreach ($marks as $m) {
00185                     if (isset($sums[$subQuestionName])) {
00186                         $sums[$subQuestionName] += $m->getMarks();
00187                     } else {
00188                         $sums[$subQuestionName] = $m->getMarks();
00189                     }
00190                 }
00191             }
00192         }
00193         return $sums;
00194     }
00195
00201     public function mkLeadIn()
00202     {
00203         $leadIn = "<strong>$this->name</strong> [$this->section]: ";
00204         return $leadIn;
00205     }
00206
00212     public function getMessage()
00213     {
00214         $marksAll = self::$marksDao->getByObject($this);
00215         $total = $this->sumMarks();
00216         $rubricTotal = 0;
00217         $rubricsAll = [];
00218         foreach (self::$exam->getQuestions() as $question) {
00219             $subQuestions = $question->getSubquestions();
```

```
00220             foreach ($subQuestions as $subQuestion) {
00221                 $rubrics = self::$rubricDao->getByObject($subQuestion);
00222                 foreach ($rubrics as $rubric) {
00223                     $rubricMark = $rubric->getMarks();
00224                     $rubricsAll[] = $rubric;
00225                     if ($rubricMark > 0) {
00226                         $rubricTotal += $rubricMark;
00227                     }
00228                 }
00229             }
00230         }
00231         $numMarks = count($marksAll);
00232         $numRubrics = count($rubricsAll);
00233         if ($numMarks == $numRubrics) {
00234             $this->setStatus("Graded")->update();
00235             $msg = "[Grading Complete:  Marks: $total/$rubricTotal]";
00236         } else if (count($marksAll) > 0) {
00237             $this->setStatus("Grading")->update();
00238             $msg = "[Grading in Progress:  Marks: $total/$rubricTotal ($numMarks of $numRubrics
      rubrics done)]";
00239         } else {
00240             $msg = "";
00241         }
00242         $this->sumMarksBySubQuestion();
00243         return $msg;
00244     }
00245
00251     public function grade()
00252     {
00253         $msg = $this->getMessage();
00254         $echo = "info";
00255         $what = "";
00256         $disabled = "";
00257         if ($this->status == "Absent") {
00258             $value = "Ignore";
00259             $echo = "error";
00260             $disabled = "disabled";
00261         } else if ($this->status == "Grading") {
00262             $value = "Continue";
00263             $echo = ($this->sumMarks() > 0) ? "warn" : "plain";
00264         } elseif ($this->status == "Graded") {
00265             $value = "Regrade";
00266             $echo = "success";
00267         } else {
00268             $value = "Grade";
00269             $echo = "info";
00270         }
00271         $nStudents = self::$studentDao->getNStudents();
00272         $iStudent = self::$studentDao->getIStudent($this->email);
00273         $what = "<strong>$this->name</strong> <small>$msg 
00274             ($iStudent of " . $nStudents . ")</small>
00275         <form method='post'  target='quickSummary'>
00276         <input type='submit' name='quickSummary' value='Summary' $disabled
00277         style='position:absolute;top:10px;right:90px;'>
00278         <input type='hidden' name='do' value='quickSummary'>
00279         <input type='hidden' name='email' value='$this->email'>
00280         </form>
00281         <form method='post'  target='left'>
00282         <input type='submit' name='gradeOne' value='$value' $disabled
00283         style='position:absolute;top:10px;right:10px;'>
00284         <input type='hidden' name='do' value='gradeOne'>
00285         <input type='hidden' name='email' value='$this->email'>
00286         </form>";
00287         $this->$echo($what);
00288     }
00289
00296     public function delete($name = '')
00297     {
00298         $submissionFolder = realpath($this->getAnswerFolder());
00299         CFG::rmdir($submissionFolder);
00300         HT::warn("<strong>$this->name</strong>: Deleted unzipped files in <br>
00301         <code>$submissionFolder</code>");
00302     }
00303
00309     public function update()
00310     {
00311         self::$studentDao->update($this);
00312     }
00313
00319     public function getEmail()
00320     {
00321         return $this->email;
00322     }
00323
00329     public function getSection()
00330     {
00331         return $this->section;
```

```
00332     }
00333
00339     public function getStatus()
00340     {
00341         return $this->status;
00342     }
00343
00350     public function print($toStr = true)
00351     {
00352         $str = parent::print($toStr);
00353         return $str;
00354     }
00355
00361     public function autoGrade()
00362     {
00363         if ($this->status == 'Absent') {
00364             return;
00365         }
00366         // $options = CFG::$autoGradeOptions;
00367         // $subQuestionNames = $options['subQuestionNames'];
00368         $autoGradables = CFG::$autoGradables;
00369         $leadIn = $this->mkLeadIn();
00370         $what = "<p><strong>{$this->getName()}</strong></p>
00371         <table class='grader' width='80%' align='center'>
00372         <tr><th>Question</th><th>File</th><th>Marks</th></tr>";
00373         $isGraded = false;
00374         foreach ($autoGradables as $file => $autoGraded) {
00375             $autoGraderBase = $autoGraded['grader'];
00376             $autoGrader = realpath($this->getAnswerFolder() . $autoGraderBase);
00377
00378             // This snippet doesn't work because urlencode is too aggressive
00379             // $autoGrader = str_replace($_SERVER['DOCUMENT_ROOT'], '', $autoGrader);
00380             // $autoGrader = urlencode($autoGrader);
00381             // $url = "http://localhost" . $autoGrader;
00382
00383             // Just replace the spaces for now
00384             $url0 = str_replace(realpath($_SERVER['DOCUMENT_ROOT']), "http://localhost", $autoGrader);
00385             $url = str_replace(' ', '%20', $url0);
00386             $ch = curl_init();
00387             curl_setopt($ch, CURLOPT_URL, $url);
00388             curl_setopt($ch, CURLOPT_HEADER, 0);
00389             // In case there are infinite loops in the student code, timeout in 15 seconds
00390             // Fatal errors also seem to hang the CURL connection
00391             $timeout = 16;
00392             curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);
00393             curl_setopt($ch, CURLOPT_CONNECTTIMEOUT, $timeout); // Timeout for connection
00394             curl_setopt($ch, CURLOPT_TIMEOUT, $timeout); // Timeout for full request
00395             curl_setopt($ch, CURLOPT_FAILONERROR, true); // Catch HTTP errors
00396             $output = curl_exec($ch);
00397             curl_close($ch);
00398             $subQuestionName = $autoGraded['subQuestionName'];
00399             $marker = "$subQuestionName: ";
00400             $rubricName = "$subQuestionName-1";
00401             $lines = explode($marker, $output);
00402             // Mark is the first token in $line[1]
00403             $mark = 0;
00404             if (isset($lines[1])) {
00405                 $isGraded = true;
00406                 $mark = floatval($lines[1]);
00407                 // str_pad is to bust out of output buffering
00408                 if ($mark > 0) {
00409                     $what .=  "
00410                     <tr class='success'>
00411                     <td> $subQuestionName</td>
00412                     <td> $file</td>
00413                     <td> $mark</td>
00414                     </tr>";
00415                     // $what .= HT::warn(str_pad($what, 4096), true);
00416                 } else {
00417                     $what .=  "
00418                     <tr class='info'>
00419                     <td> $subQuestionName</td>
00420                     <td> $file</td>
00421                     <td> $mark</td>
00422                     </tr>";
00423                     // $what .= HT::error(str_pad("$leadIn: $mark [File: $file]", 4096));
00424                 }
00425             } else {
00426                 $what .=  "
00427                     <tr class='error'>
00428                     <td> $subQuestionName</td>
00429                     <td> $file</td>
00430                     <td> Failed</td>
00431                     </tr>
00432                     <tr class='error'><td colspan='2'>Marker '$marker', as specified in
      <code>CFG::\$autoGradables['$subQuestionName']</code>, is missing in the output!</td><td>
00433                     <table align='right' border='0'><tr>
```

```
00434                     <td><a href='$url' target='right'><button>View</button></a></td>
00435                     <td><form method='post' target='left'>
00436                     <input type='submit' name='gradeOne' value='Grade'>
00437                     <input type='hidden' name='do' value='gradeOne'>
00438                     <input type='hidden' name='email' value='$this->email'>
00439                     </form></td>
00440                     </tr></table>
00441                     </td></tr>";
00442             }
00443             $marksObj = new Marks($this->email, $subQuestionName, $rubricName, $mark);
00444             self::$marksDao->update($marksObj);
00445             $this->setComment(htmlentities($output));
00446             self::$studentDao->update($this);
00447         }
00448         $what .= "</table>";
00449         if ($isGraded) {
00450             HT::warn(str_pad($what, 4096));
00451         }
00452     }
00453
00459     public function getZipFile()
00460     {
00461         return $this->zipFile;
00462     }
00463
00469     public function getName()
00470     {
00471         return $this->name;
00472     }
00473
00480     public function setZipFile($zipFile): self
00481     {
00482         $this->zipFile = $zipFile;
00483         return $this;
00484     }
00485
00491     public function getComment()
00492     {
00493         return $this->comment;
00494     }
00495
00502
00503     public function setComment($comment): self
00504     {
00505         if (strpos($this->comment, $comment) === false) {
00506             $this->comment .= $comment;
00507         }
00508         return $this;
00509     }
00510
00517     public function setStatus($status): self
00518     {
00519         $this->status = $status;
00520         return $this;
00521     }
00522
00528     public function csv()
00529     {
00530         $str = "<tr><td>$this->name</td>";
00531         $marks = $this->sumMarksBySubQuestion();
00532         foreach ($marks as $subQuestionName => $m) {
00533             $str .= "<td>$subQuestionName &rarr; $m</td>";
00534         }
00535         $str .= "</tr>";
00536         return $str;
00537     }
00538 }
```

5.49 SubQuestion.php File Reference

Classes

- class SubQuestion

5.50 SubQuestion.php

Go to the documentation of this file.

```
00001 <?php
00002
00010 class SubQuestion extends HT
00011 {
00013     private $name;
00014
00016     private $marks;
00017
00019     private $questionNum;
00020
00028     public function __construct($name, $mark, $questionNum)
00029     {
00030         $this->name = strtolower($name);
00031         $this->marks = $mark;
00032         $this->questionNum = strtolower($questionNum);
00033     }
00034
00040     public function getName()
00041     {
00042         return $this->name;
00043     }
00044
00050     public function getQuestionNum()
00051     {
00052         return $this->questionNum;
00053     }
00054
00061     public function print($toStr = true)
00062     {
00063         $str = "<strong>  SubQuestion: $this->name
00064           Marks: $this->marks</strong><br>";
00065         return $str;
00066     }
00067
00074     private static function mkPopup($fileName)
00075     {
00076         return "onclick=
00077         \"var h = screen.height * 0.8;
00078         var t = screen.height * 0.05;
00079         var w = screen.width * 0.4;
00080         var l = screen.width * 0.05;
00081         window.open('', '$fileName', 'width='+ w +',height=' + h + ',left='+ l +',top='+ t
      +',resizable=yes,scrollbars=yes,toolbar=yes,menubar=no,location=no,directories=no,status=yes');
00082           document.getElementById('$fileName').submit();
00083           return false;\"";
00084     }
00085
00092     public function render($student)
00093     {
00094         $subQuestionName = $this->name;
00095         $resources = self::$refFileDao->getByObject($this, 'resources');
00096         $solutions = self::$refFileDao->getByObject($this, 'solutions');
00097         $srcDoc = " <div style='font-size:110%;font-weight:bold'
00098             title='SubQuestion::render()'>
00099             SubQuestion: $subQuestionName</div>";
00100         $tabRes = "<table title='SubQuestion::render()'>
00101             <tr><th colspan='100%'>Resources</th></tr><tr>";
00102         $tabSoln = "<table title='SubQuestion::render()'>
00103             <tr><th colspan='100%'>Solutions</th></tr><tr>";
00104         // Get the rubric and get the filename so that the right resource/solution
00105         // can be highlighted
00106         $rubrics = self::$rubricDao->getByObject($this);
00107         $rubricFiles = [];
00108         foreach ($rubrics as $rubric) {
00109             $rubricFile = basename($rubric->getShortFileName());
00110             if (!in_array($rubricFile, $rubricFiles)) {
00111                 $rubricFiles[] = $rubricFile;
00112             }
00113         }
00114         // Assumes resources and solutions are sorted in the same order.
00115         // If not, sort them in the RefFilesDAO class
00116         foreach ($resources as $key => $resource) {
00117             $shortFIleName = $resource->getShortFileName();
00118             if (!self::$refFileDao->isGradable($shortFIleName)) {
00119                 continue;
00120             }
00121             $fileName = $resource->getFileName();
00122             $basename = basename($fileName);
00123             $fileClass = "plain";
00124             if (in_array($basename, $rubricFiles)) {
00125                 $fileClass = "error";
00126             }
00127             $fullText = htmlentities($resource->getFullText(), ENT_QUOTES);
00128             $popup = self::mkPopup($fileName);
00129             $tabRes .= "<td>
00130             <form method='post' action='?do'
00131             target='$fileName' id='$fileName'>
```

```
00132             <input type='submit' name='viewfile' title='$fileName'
00133             value='$basename' $popup class='$fileClass'>
00134             <input type='hidden' name='do' value='viewFile'>
00135             <input type='hidden' name='filename' value='$fileName'>
00136             <input type='hidden' name='fulltext' value='$fullText'>
00137             </form></td>
00138             ";
00139             // TODO: Investigate: How come $solnKey wasn't necessary before?
00140             $solnKey = str_replace("resources", "solutions", $key);
00141             $solution = $solutions[$solnKey];
00142             $fileName = $solution->getFileName();
00143             $basename = basename($fileName);
00144             $fullText = htmlentities($solution->getFullText(), ENT_QUOTES);
00145             $popup = self::mkPopup($fileName);
00146             $tabSoln .= "<td>
00147             <form method='post' action='?do'
00148             target='$fileName' id='$fileName'>
00149             <input type='submit' name='viewfile' title='$fileName'
00150             value='$basename' $popup class='$fileClass'>
00151             <input type='hidden' name='do' value='viewFile'>
00152             <input type='hidden' name='filename' value='$fileName'>
00153             <input type='hidden' name='fulltext' value='$fullText'>
00154             </form></td>
00155             ";
00156         }
00157         $closeTable = "</td></tr></table>";
00158         $tabRes .= $closeTable;
00159         $tabSoln .= $closeTable;
00160         $srcDoc .= HT::warn("<table align='center'>
00161         <tr><td align='center'>$tabRes</td><td> </td></tr>
00162         <tr><td colspan='100%'> </td></tr>
00163         <tr><td align='center'>$tabSoln</td></tr>
00164         </table>", true);
00165         $srcDoc = HT::success($srcDoc, true);
00166         $rubSrc = "<form method='post' action='?do' target='right2'
00167         id='$subQuestionName'>
00168         <table width='90%' align='center' class='grader'>
00169         <tr><th>Award</th><th>Ref</th><th>Rubric</th></tr>";
00170         // $rubrics = self::$rubricDao->getByObject($this);
00171         $awarded = self::$marksDao->getByObject($student);
00172         // var_dump($awarded);
00173         $totAwarded = $totQuestion = 0.0;
00174         $showAwarded = false;
00175         foreach ($rubrics as $k => $rubric) {
00176             $step = 0.025;
00177             $marks = $rubric->getMarks();
00178             $rubricName = $rubric->getRubricName();
00179             $rubricText = $rubric->getRubricText();
00180             $min = min([$marks, 0]);
00181             $max = max([$marks, 0]);
00182             $totQuestion += $max; // Do not count negative marks
00183             if ($min < 0) {
00184                 $step = -$min / 4;
00185                 $min = -5 * $step;
00186                 $max = -$min;
00187             }
00188             $rubricText = htmlentities($rubricText, ENT_QUOTES);
00189             $value = "";
00190             foreach ($awarded as $a) {
00191                 if ($a->getRubricName() == $rubricName) {
00192                     $showAwarded = true;
00193                     $value = number_format($a->getMarks(), 3);
00194                     $totAwarded += $value;
00195                 }
00196             }
00197             if ($value == $max) {
00198                 $class = "success";
00199             } else if ($value > 0) {
00200                 $class = "warn";
00201             } else if ($value === '') {
00202                 $class = "plain";
00203             } else {
00204                 $class = "error";
00205             }
00206             $rubSrc .= "<tr class='$class'><td>
00207             <input name='$rubricName' type='number'
00208             value='$value' min='$min' max='$max' step='$step'>
00209             <input name='full-$rubricName' type='hidden' value='$max'>
00210             </td>
00211             <td> $marks </td>
00212             <td style='text-align:left;'> $rubricName:
00213             <code>$rubricText</code></td>
00214             </tr>";
00215         }
00216         if ($showAwarded) {
00217             $rubSrc .= "<tr><th>$totAwarded</th><th>$totQuestion</th><th>";
00218         } else {
```

```
00219             $rubSrc .= "<tr><th colspan='100%'>";
00220         }
00221         $rubSrc .= "<input type='hidden' name='do' value='putMarks'>
00222         <input type='submit' value='Update Marks'
00223         name='putMarks' align='center'>
00224         <input type='submit' value='Award Full Marks'
00225         name='putFullMarks' align='center'>
00226         </th></tr></table>
00227         <input type='hidden' name='name' value='{$student->getName()}'>
00228         <input type='hidden' name='student_email' value='{$student->getEmail()}'>
00229         <input type='hidden' name='subquestion_name' value='$subQuestionName'>
00230         </form></div>";
00231         // $rubSrc = htmlentities($rubSrc);
00232         $srcDoc .= HT::success($rubSrc, true);
00233         $srcDoc = HT::info($srcDoc, true);
00234         return $srcDoc;
00235     }
00236
00242     public function getMarks()
00243     {
00244         return $this->marks;
00245     }
00246 }
```

5.51 grade.php File Reference

5.52 grade.php

Go to the documentation of this file.

```
00001 <?php
00002 require_once("../model/autoload.php");
00003 require_once("../config.php");
00004
00005 Grader::index();
```

5.53 process.php File Reference

5.54 process.php

Go to the documentation of this file.

```
00001 <?php
00002 require_once("../model/autoload.php");
00003 require_once("../config.php");
00004
00005 Exam::index();
```

5.55 model/report.php File Reference

5.56 model/report.php

Go to the documentation of this file.

```
00001 <?php
00002
00018 class Report extends HT
00019 {
00021     static $algo = "sha1";
00022
00024     static $simLimit = 98;
00025
00027     private static $maxExecutionTime = 3600;
00028
00030     private static $fbiFile = 'temp.fbi.html';
00031
00033     protected static $actions = [
00034         // 'stats' => 'Marks Statistics',
```

```
00035         'verify' => 'Verify Student email IDs',
00036         'load' => 'Load Student List',
00037         'start' => 'Build Database Tables',
00038     ];
00039
00041     protected static $actionProperties = [
00042         'start' => [
00043             'class' => 'error',
00044             'target' => 'left'
00045         ],
00046         'stop' => [
00047             'class' => 'error',
00048             'target' => 'left'
00049         ]
00050     ];
00051
00053     protected static $actionDesc = [
00054         'stats' => 'Marks Statistics: Not implemented yet.',
00055         'verify' => 'Compare the student email IDs from the comments in
00056         the uploaded files to the ones from eLearn submission',
00057         'load' => 'Load Student List for in-depth analysis for copying/cheating
00058         attempts.',
00059         'start' => 'Build FBI Table: The cheating investigation is done by
00060         comparing each submitted file against all other files from all other
00061         students. Launch this (one-time) processing to create the necessary
00062         database tables. It will take a while. But it can be interrupted and
00063         restarted without losing what is computed.',
00064     ];
00065
00067     protected static $intro = "<h4>Reports and Investigation</h4>
00068     <h5> Statistics and Detection of Academic Dishonesty</h5>
00069   <p> Grader can help you detect cheating attempts using heuristics: Missing or
00070   wrong emails in the files, content similarity, number of submitted files that
00071   look similar to the same student etc.</p>";
00072
00074     protected static $display = [
00075         "class" => "warn",
00076         "title" => "Investigate"
00077     ];
00078
00079
00085     private static $cutoffs = [
00086         100 => [
00087             'class' => 'error',
00088             'msg' => [
00089                 'solution' => 'Possible exam leak!',
00090                 'resource' => 'Question not attempted',
00091                 'other' => 'High probability of cheating!'
00092             ]
00093         ],
00094         98 => [
00095             'class' => 'error',
00096             'msg' => [
00097                 'solution' => 'Possible leak!',
00098                 'other' => 'Possible cheating!'
00099             ]
00100         ],
00101         90 => [
00102             'class' => 'warn',
00103             'msg' => [
00104                 'other' => 'Check for cheating!'
00105             ]
00106         ],
00107         80 => [
00108             'class' => 'plain',
00109             'msg' => [
00110                 'other' => 'Slight chance of cheating'
00111             ]
00112         ],
00113         0 => [
00114             'class' => 'success',
00115             'msg' => [
00116                 'other' => 'No issues detected'
00117             ]
00118         ],
00119     ];
00120
00122     protected static $class = __CLASS__;
00123
00131     public static function hash($file, $raw = true)
00132     {
00133         if ($raw) {
00134             $data = file_get_contents($file);
00135         } else {
00136             $data = self::trimFile($file);
00137         }
00138         return hash(self::$algo, $data);
```

```
00139     }
00140
00147     public static function trimFile($file)
00148     {
00149         $lines = file($file);
00150         $data = "";
00151         for ($i = 0; $i < count($lines); $i++) {
00152             $data .= self::trim($lines[$i]);
00153         }
00154         return $data;
00155     }
00156
00163     private static function trim($line)
00164     {
00165         $trimmed = strtolower(preg_replace("/\s+/", "", $line));
00166         if (strpos($trimmed, "user:") === 0) {
00167             $trimmed = "";
00168         }
00169         if (strpos($trimmed, "name:") === 0) {
00170             $trimmed = "";
00171         }
00172         if (strpos($trimmed, "email:") === 0) {
00173             $trimmed = "";
00174         }
00175         if (strpos($trimmed, "<?php") === 0) {
00176             $trimmed = "";
00177         }
00178         if (strpos($trimmed, "?>") === 0) {
00179             $trimmed = "";
00180         }
00181         return $trimmed;
00182     }
00183
00191     public static function similarity($f1, $f2)
00192     {
00193         $d1 = self::trimFile($f1);
00194         $d2 = self::trimFile($f2);
00195         $similarity = 0.0;
00196         similar_text($d1, $d2, $similarity);
00197         return $similarity;
00198     }
00199
00207     public static function getSimilarity($answer, $reference)
00208     {
00209         // var_dump(date("h:m:ss"), $answer->getFileName());
00210         $sim = [];
00211         if ($answer->getHash0() == $reference->getHash0()) {
00212             $sim[0] = 100;
00213         }
00214         if ($answer->getHash1() == $reference->getHash1()) {
00215             $sim[1] = 100;
00216         }
00217         if (!isset($sim[0])) {
00218             $similarity = 0.0;
00219             similar_text(
00220                 $answer->getFullText(),
00221                 $reference->getFullText(),
00222                 $similarity
00223             );
00224             $sim[0] = round($similarity, 2);
00225         }
00226         if (!isset($sim[1])) {
00227             $sim[1] = round(self::similarity(
00228                 $answer->getFileName(),
00229                 $reference->getFileName()
00230             ), 2);
00231         }
00232         return $sim;
00233     }
00234
00238     protected static function showSummary()
00239     {
00240         self::init();
00241         $email = $_POST['email'];
00242         $student = self::$studentDao->getByEmail($email)[0];
00243         $studentName = $student->getName();
00244         $srcDoc = HT::head(true);
00245         $answers = self::$answerDao->getByObject($student);
00246         $table1 = "<table class='grader' align='center'><tr>
00247         <th colspan='100%' style='text-align:center;font-size:120%;'>
00248         Summary: $studentName [$email]</th></tr>
00249         <tr><th>Suspect (email)</th><th>Number of Repeats</th>
00250         <th>Max. Similarity</th><th>Comment</th></tr>";
00251         $table2 = "<table class='grader' align='center'><tr>
00252         <th colspan='100%' style='text-align:center;font-size:120%;'>
00253         Details: $studentName [$email]</th></tr>
00254         <tr><th>Subquestion</th><th>File Name</th>
```

```
00255         <th>Suspect (email)</th><th>Similarity<br>Sim Resource</th><th>Comment</th></tr>";
00256         if (count($answers) > 0) { // Student attempted
00257             $allEmails = [];
00258             $allSims = [];
00259             $simRes = [];
00260             foreach ($answers as $answer) {
00261                 $shortFileName = $answer->getShortFileName();
00262                 if (!self::$refFileDao->isGradable($shortFileName)) {
00263                     continue;
00264                 }
00265                 $resource = self::$refFileDao->getRefFile($shortFileName, 'resources');
00266                 if ($resource->getSim() > self::$simLimit) {
00267                     continue;
00268                 }
00269                 $simRes[$shortFileName] = max(Report::getSimilarity($answer, $resource));
00270                 if ($simRes[$shortFileName] < 100) {
00271                     $row = self::getFBI($student, $answer);
00272                     if (count($row) == 0) {
00273                         $srcDoc .= HT::error("Student " . $student->getName() .
00274                             " not processed yet. No info in the Database!", true);
00275                         $srcDoc .= HT::foot(true);
00276                         echo $srcDoc;
00277                         return;
00278                     }
00279                     $r = $row[0];
00280                     for ($i = 0; $i < 5; $i++) {
00281                         $key = "sim$i";
00282                         $otherEmail = $r[$key . "_email"];
00283                         if (array_key_exists($otherEmail, $allSims)) {
00284                             if ($allSims[$otherEmail] < $r[$key]) { // Get max value
00285                                 $allSims[$otherEmail] = $r[$key];
00286                             }
00287                         } else {
00288                             $allSims[$otherEmail] = $r[$key];
00289                         }
00290                         $allEmails[] = $otherEmail;
00291                     }
00292                 }
00293             }
00294             // sort($allEmails);
00295             // ksort($allSims);
00296             // var_dump($suspects, $allEmails, $allSims);
00297             $suspects = array_count_values($allEmails);
00298             arsort($suspects);
00299             foreach ($suspects as $otherEmail => $repeats) {
00300                 if ($repeats < 2) {
00301                     continue;
00302                 }
00303                 $maxSim = $allSims[$otherEmail];
00304                 if ($maxSim < 80) {
00305                     continue;
00306                 }
00307                 [$class, $comment] = self::simToMsg($maxSim, 'other');
00308                 $table1 .= "<tr class='$class'><td>$otherEmail</td><td>$repeats</td>
00309                 <td>$maxSim</td><td>$comment</td></tr>";
00310                 foreach ($answers as $answer) {
00311                     $shortFileName = $answer->getShortFileName();
00312                     if (!self::$refFileDao->isGradable($shortFileName)) {
00313                         continue;
00314                     }
00315                     $resource = self::$refFileDao->getRefFile($shortFileName, 'resources');
00316                     if ($resource->getSim() > self::$simLimit) {
00317                         continue;
00318                     }
00319                     if ($simRes[$shortFileName] > self::$simLimit) {
00320                         continue;
00321                     }
00322                     $subQuestionName = $answer->getSubQuestionName();
00323                     $sim = self::getSimFromDB([
00324                         'email1' => $email,
00325                         'email2' => $otherEmail,
00326                         'short_filename' => $shortFileName
00327                     ]);
00328                     [$class, $comment] = self::simToMsg($sim, 'other');
00329                     $table2 .= "<tr class='$class'><td>$subQuestionName</td>
00330                     <td>$shortFileName</td><td>$otherEmail</td>
00331                     <td>$sim<br>$simRes[$shortFileName]</td><td>$comment</td></tr>";
00332                 }
00333             }
00334         } else {
00335             $table1 .= "<tr><td colspan='100%'>Not Attempted</td></tr>";
00336             $table2 .= "<tr><td colspan='100%'>Not Attempted</td></tr>";
00337         }
00338         $closeTable = "</table>";
00339         $table1 .= $closeTable;
00340         $table2 .= $closeTable;
00341         $srcDoc .= HT::warn($table1, true);
```

```
00342         $srcDoc .= HT::warn($table2, true);
00343         $srcDoc .= HT::foot(true);
00344         echo $srcDoc;
00345         return $srcDoc;
00346     }
00347 
00348     // Dispatch handlers
00349 
00353     protected static function stats()
00354     {
00355         self::init();
00356         $exam = self::$exam;
00357         $exam->stats();
00358     }
00359 
00363     protected static function verify()
00364     {
00365         self::init();
00366         $exam = self::$exam;
00367         HT::head();
00368         foreach ($exam->getSections() as $section) {
00369             HT::success("<h3>Section: " . $section->getName() . "</h3>");
00370             foreach ($section->getStudents() as $student) {
00371                 if ($student->getStatus() != "Absent") {
00372                     self::verifyStudent($student);
00373                 }
00374             }
00375         }
00376         $exam->stats();
00377     }
00378 
00384     private static function verifyStudent($student)
00385     {
00386         $email = $student->getEmail();
00387         $what = "<p><strong>{$student->getName()}</strong></p>
00388         <table class='grader' width='80%' align='center'>
00389         <tr><th>File</th><th>Email</th><th>File Email</th></tr>";
00390         $isValid = true;
00391         $answers = self::$answerDao->getByObject($student);
00392         foreach ($answers as $answer) {
00393             $shortFileName = $answer->getShortFileName();
00394             if (self::$refFileDao->isGradable($shortFileName)) {
00395                 $fileEmail = self::getEmail($answer->getFullText());
00396                 if ($email != $fileEmail) {
00397                     $what .=  "
00398                     <tr>
00399                     <td> $shortFileName</td>
00400                     <td> $email</td>
00401                     <td> $fileEmail</td>
00402                     </tr>";
00403                     $isValid = false;
00404                 }
00405             }
00406         }
00407         $what .= "</table>";
00408         if (!$isValid) {
00409             HT::error($what);
00410         }
00411     }
00412 
00416     protected static function load()
00417     {
00418         self::init();
00419         $exam = self::$exam;
00420         HT::head();
00421         foreach ($exam->getSections() as $section) {
00422             HT::success("<h3>Section: " . $section->getName() . "</h3>");
00423             foreach ($section->getStudents() as $student) {
00424                 if ($student->getStatus() != "Absent") {
00425                     self::listStudent($student);
00426                 }
00427             }
00428         }
00429         HT::foot();
00430     }
00431 
00438     private static function wait($iframe)
00439     { // iframe = right, or right.right1 etc.
00440         $js = "onclick='window.parent.$iframe.document.write(\"<h1>Please wait!</h1>\");'";
00441         return $js;
00442     }
00443 
00449     private static function listStudent($student)
00450     {
00451         $nStudents = self::$studentDao->getNStudents();
00452         $email = $student->getEmail();
00453         $iStudent = self::$studentDao->getIStudent($email);
```

```
00454         // $msg = "[Section: " . $student->getSection() . " Email: " .
00455         //     $student->getEmail() . "]";
00456         $msg = "[" . $student->getEmail() . "]";
00457         $wait = self::wait("right");
00458         $what = "<strong>{$student->getName()}</strong> <small>$msg 
00459             ($iStudent of " . $nStudents . ")</small>
00460             <table align='right'><tr>
00461             <td><form method='post' action='?do' target='right'>
00462             <input type='submit' value='Prepare' $wait>
00463             <input type='hidden' name='do' value='processOne'>
00464             <input type='hidden' name='email' value='$email'>
00465             </form></td>
00466             <td><form method='post' action='?do' target='right'>
00467             <input type='submit' value='Investigate'>
00468             <input type='hidden' name='do' value='investigate'>
00469             <input type='hidden' name='email' value='$email'>
00470             </form></td>
00471             <td><form method='post' action='?do' target='right'>
00472             <input type='submit' value='Summary'>
00473             <input type='hidden' name='do' value='showSummary'>
00474             <input type='hidden' name='email' value='$email'>
00475             </form></td></tr></table>";
00476         HT::warn($what);
00477     }
00478
00487     private static function putFBI($student, $answer, $similarity, $fileEmail)
00488     {
00489         $row = [];
00490         $row['student_email'] = $student->getEmail();
00491         $row['subquestion_name'] = $answer->getSubQuestionName();
00492         $row['filename'] = $answer->getFileName();
00493         $row['hash0'] = $answer->getHash0();
00494         $row['hash1'] = $answer->getHash1();
00495         $row['file_email'] = $fileEmail;
00496         $file = $answer->getFileName();
00497         $basename = basename($file);
00498         $resource = self::$refFileDao->getRefFile($basename, 'resources');
00499         $solution = self::$refFileDao->getRefFile($basename, 'solutions');
00500         $row['sim_solution'] = max(self::getSimilarity($answer, $solution));
00501         $row['sim_resource'] = max(self::getSimilarity($answer, $resource));
00502         $keys = array_keys($similarity);
00503         $bound = min(5, count($keys));
00504         for ($i = 0; $i < $bound; $i++) {
00505             $row["sim$i"] = $similarity[$keys[$i]];
00506             $row["sim$i" . "_email"] = $keys[$i];
00507         }
00508         $table = CFG::mkTableName('fbi');
00509         DB::$db->update($table, array_keys($row), $row);
00510     }
00511
00519     private static function getFBI($student, $answer)
00520     {
00521         $where = [];
00522         $where['student_email'] = $student->getEmail();
00523         $where['subquestion_name'] = $answer->getSubQuestionName();
00524         $where['filename'] = $answer->getFileName();
00525         $table = CFG::mkTableName('fbi');
00526         $row = DB::$db->get($table, "*", $where);
00527         return $row;
00528     }
00529
00536     private static function getSimFromDB($where)
00537     {
00538         $table = CFG::mkTableName('fbi_matrix');
00539         $row = DB::$db->get($table, "sim", $where);
00540         if (empty($row)) {
00541             extract($where);
00542             $where['email1'] = $email2;
00543             $where['email2'] = $email1;
00544             $row = DB::$db->get($table, "sim", $where);
00545         }
00546         if (empty($row)) {
00547             return false;
00548         }
00549         return $row[0]['sim'];
00550     }
00551
00557     private static function putSimToDB($toDB)
00558     {
00559         $table = CFG::mkTableName('fbi_matrix');
00560         DB::$db->update($table, array_keys($toDB), $toDB);
00561     }
00562
00566     protected static function start()
00567     {
00568         $fbiFile = self::$fbiFile;
00569         $fbi = HT::head(true, "Status Watcher");
```

```
00570         $wait = self::wait("right");
00571         $fbi .= HT::warn(
00572             "<p>Building the FBI table for the first time will take a long time.
00573             Processing by section will be faster because time is proportional
00574             to the square of the number of sections processed. But comparing
00575             the answers from all students may reveal organized cheating
00576             attempts.</p>
00577             <p>After launching the process, you can monitor the progress, which
00578             will run in the background as long as the right frame is open.</p>
00579             <p>You stop the process any time. Relaunching it will pick up from
00580             where you last left off.</p>
00581             <table align='center'><tr>
00582             <td><form target='right' method='post' action='report.php?do'>
00583             <input type='hidden' name='do' value='process'>
00584             <label>
00585             <input type='checkbox' name='sectioned' checked>Process By Section?
00586             </label> 
00587             <button type='submit' class='error' $wait>
00588             Proceed to Build FBI Table?
00589             </button></form></td>
00590             <td><form target='left' action=''>
00591              <button type='submit' class='success'>
00592             Monitor Status
00593             </button></form></td>
00594             </tr></table>",
00595             true
00596         );
00597         $fbi .= HT::foot(true);
00598         file_put_contents($fbiFile, $fbi);
00599         header("Location: $fbiFile");
00600     }
00601 
00605     protected static function stop()
00606     {
00607         $fbiFile = self::$fbiFile;
00608         $heystack = file_get_contents($fbiFile);
00609         $needle = "<body>";
00610         $start = stripos($heystack, $needle) + strlen($needle);
00611         $needle = "<table";
00612         $end = stripos($heystack, $needle);
00613         $body = substr($heystack, $start, $end - $start);
00614         $fbi = HT::head(true);
00615         $fbi .= HT::warn($body, true);
00616         $fbi .= HT::error("FBI Database Processing stopped.
00617         Run it again to finish preparing the FBI database.", true);
00618         $fbi .= HT::foot(true);
00619         file_put_contents($fbiFile, $fbi);
00620         echo $fbi;
00621         // unlink($fbiFile);
00622     }
00623 
00633     protected static function process()
00634     {
00635         function rutime($ru, $rus, $index)
00636         {
00637             return ($ru["ru_$index.tv_sec"] * 1000 +
00638                 intval($ru["ru_$index.tv_usec"] / 1000))
00639                 - ($rus["ru_$index.tv_sec"] * 1000 +
00640                     intval($rus["ru_$index.tv_usec"] / 1000));
00641         }
00642         self::init();
00643         set_time_limit((int) self::$maxExecutionTime);
00644         date_default_timezone_set('Asia/Singapore');
00645         $start = microtime(true);
00646         $cpuStart = getrusage();
00647         $date = date('m/d/Y h:i:s a');
00648         $fbi0 = "<h4>Process Monitor</h4>[$date] Processing Started<br>";
00649         $fbiFile = self::$fbiFile;
00650         $studentDao = self::$studentDao;
00651         $students = $studentDao->get();
00652         $nStudents = $studentDao->getNStudents();
00653         foreach ($students as $student) {
00654             $studentEmail = $student->getEmail();
00655             $iStudent = $studentDao->getIStudent($studentEmail);
00656             $fbi = HT::head(true, "Status Watcher", '5');
00657             $date = date('m/d/Y h:i:s a');
00658             $end = microtime(true);
00659             $elapsed = gmdate("H:i:s", round($end - $start));
00660             $cpuNow = getrusage();
00661             $usedCPU = rutime($cpuNow, $cpuStart, 'utime') / 1000;
00662             $usedCPU1 = gmdate("H:i:s", round($usedCPU));
00663             $fbi1 = "[$date] Processing Student $iStudent of $nStudents <br>
00664             [CPU/Elapsed Time: $usedCPU1/$elapsed]<br>";
00665             if ($iStudent % 10 == 1) {
00666                 $fbi0 .= $fbi1;
00667             }
00668             $fbi2 = "<table align='right'><tr>
```

```
00669             <td><form target='right' method='post' action='report.php?do'>
00670             <input type='hidden' name='do' value='stop'>
00671             <button type='submit' class='error'>
00672             Stop Processing
00673             </button></form></td>
00674             </tr></table><br>";
00675             $fbi .= HT::info("<code>$fbi0</code>" .
00676                 HT::warn("<code>$fbi1</code>", true), true);
00677             $fbi .= HT::foot(true);
00678             if (!file_exists($fbiFile)) { // Interrupted by self::stop()
00679                 break;
00680             } else {
00681                 file_put_contents($fbiFile, $fbi);
00682             }
00683             // Quit 15 secs before execution time limit
00684             if ($usedCPU > self::$maxExecutionTime - 15) {
00685                 break;
00686             }
00687             self::processOne($studentEmail, true);
00688         }
00689         $fbi = HT::head(true, "Status Watcher");
00690         if ($iStudent == $nStudents) {
00691             $fbi .= HT::success(
00692                 "<code>$fbi0 $fbi1<br>
00693                 Processed all $nStudents students. FBI Database is
00694                 ready!</code>",
00695                 true
00696             );
00697         } else {
00698             $fbi = HT::warn(
00699                 "<code>$fbi0 $fbi1<br>
00700                 Processed $iStudent of $nStudents students. Run it again to
00701                 finish preparing the FBI database.</code>",
00702                 true
00703             );
00704         }
00705         $fbi .= HT::foot(true);
00706         file_put_contents($fbiFile, $fbi);
00707         echo $fbi;
00708     }
00709
00716     private static function getEmail($fullText)
00717     {
00718         $lines = preg_split('/\n|\r/', $fullText, -1, PREG_SPLIT_NO_EMPTY);
00719         foreach ($lines as $line) {
00720             if (stripos($line, "email:") !== false) {
00721                 $email = trim(explode(":", $line)[1]);
00722                 $email = trim(strtolower(explode("@", $email)[0]));
00723                 if (empty($email)) {
00724                     $email = "Not specified!";
00725                 }
00726                 return $email;
00727             }
00728         }
00729     }
00730
00737     protected static function processOne($studentEmail = '', $toStr = true)
00738     {
00739         self::init();
00740         if (empty($studentEmail)) {
00741             $studentEmail = $_POST['email'];
00742         }
00743         if (ini_get('max_execution_time') < 60) {
00744             ini_set('max_execution_time', 60);
00745         }
00746         $studentDao = self::$studentDao;
00747         $student = $studentDao->getByEmail($studentEmail)[0];
00748         $submissionFolder = $student->getAnswerFolder();
00749         $where = [];
00750         if (isset($_POST['sectioned'])) {
00751             $where = ['section' => $student->getSection()];
00752         }
00753         $others = $studentDao->get($where);
00754         $exam = self::$exam;
00755         $answerDao = self::$answerDao;
00756         $str = HT::head($toStr);
00757         $str0 = "";
00758         $toDB = [];
00759         $toDB['email1'] = $studentEmail;
00760         foreach ($exam->getQuestions() as $question) {
00761             $str0 .= HT::warn("Student: " . $student->getName() .
00762                 ", Question " . $question->getNum(), $toStr);
00763             foreach ($question->getSubQuestions() as $subQuestion) {
00764                 $subQuestionName = $subQuestion->getName();
00765                 $toDB['subquestion_name'] = $subQuestionName;
00766                 $similarity = [];
00767                 $answers = $answerDao->getByObject($student, $subQuestion);
```

```
00768                 if (count($answers) == 0) {
00769                     continue;
00770                 }
00771                 foreach ($answers as $answer) {
00772                     $row = self::getFBI($student, $answer);
00773                     $shortFileName = $answer->getShortFileName();
00774                     if (!self::$refFileDao->isGradable($shortFileName)) {
00775                         continue;
00776                     }
00777                     $resource = self::$refFileDao->getRefFile($shortFileName, 'resources');
00778                     if ($resource->getSim() > self::$simLimit) {
00779                         continue;
00780                     }
00781                     if (count($row) > 0) {
00782                         $str0 .= HT::success(" &rarr; SubQuestion
00783                         $subQuestionName [$shortFileName] read from DB.", $toStr);
00784                         continue;
00785                     } else {
00786                         $str0 .= HT::error(
00787                             " &rarr; SubQuestion
00788                         $subQuestionName [$shortFileName] building: Will take a minute.",
00789                             $toStr
00790                         );
00791                     }
00792                     $toDB['short_filename'] = $shortFileName;
00793                     $fileEmail = self::getEmail($answer->getFullText());
00794                     foreach ($others as $other) {
00795                         $otherEmail = $other->getEmail();
00796                         if ($otherEmail == $studentEmail) {
00797                             continue;
00798                         }
00799                         $otherAnswer = $answerDao->getByObject(
00800                             $other,
00801                             $subQuestion,
00802                             $answer
00803                         );
00804                         $count = count($otherAnswer);
00805                         if ($count == 0) {
00806                             continue;
00807                         }
00808                         if ($count > 1) {
00809                             $str0 .= HT::error(
00810                                 " &rarr; Got too many answers ($count) from "
00811                                     . $other->getName()
00812                             );
00813                         }
00814                         $toDB['email2'] = $otherEmail;
00815                         $fromDB = self::getSimFromDB($toDB);
00816                         if ($fromDB === false) {
00817                             $otherAnswer = reset($otherAnswer);
00818                             $sim = max(self::getSimilarity($otherAnswer, $answer));
00819                             $similarity[$otherEmail] = $sim;
00820                             $toDB['sim'] = $sim;
00821                             self::putSimToDB($toDB);
00822                         } else {
00823                             $similarity[$otherEmail] = $fromDB;
00824                         }
00825                     }
00826                     arsort($similarity, SORT_NUMERIC);
00827                     self::putFBI($student, $answer, $similarity, $fileEmail);
00828                 }
00829             }
00830         }
00831         $str .= HT::info($str0);
00832         $str .= HT::foot($toStr);
00833         echo $str;
00834     }
00835
00843     private static function simToMsg($sim, $type)
00844     {
00845         $cutoffs = self::$cutoffs;
00846         foreach ($cutoffs as $cutoff => $details) {
00847             if ($sim >= $cutoff) {
00848                 break;
00849             }
00850         }
00851         $class = $details['class'];
00852         $msg = "All good";
00853         if (array_key_exists($type, $details['msg'])) {
00854             $msg = $details['msg'][$type];
00855         }
00856         return [$class, $msg];
00857     }
00858
00865     private static function investigateRow($row)
00866     {
00867         $r = $row[0];
```

```
00868         $tr = "<tr><th>Issue</th><th>Details</th><th>Comments</th></tr>";
00869         if (empty($r['file_email'])) {
00870             $tr .= "<tr class='warn'><td>File Email missing</td>
00871             <td>No email found in the file</td><td></td></tr>";
00872         } elseif (stripos($r['file_email'], $r['student_email']) === false) {
00873             $tr .= "<tr class='error'><td>Email mismatch</td>
00874             <td>Student: {$r['student_email']} != File: {$r['file_email']}</td>
00875             <td>Possible Cheating</td></tr>";
00876         } else {
00877             $tr .= "<tr class='success'><td>Emails match</td>
00878             <td> Student and file emails: {$r['student_email']}</td>
00879             <td>All Good</td></tr>";
00880         }
00881         $sim = $r['sim_solution'];
00882         [$class, $msg] = self::simToMsg($sim, 'solution');
00883         $tr .= "<tr class='$class'><td>Similarity to Solution</td>
00884         <td>Similarity = $sim</td>
00885         <td>$msg</td></tr>";
00886         $sim = $r['sim_resource'];
00887         [$class, $msg] = self::simToMsg($sim, 'resource');
00888         if ($sim > 99.999) {
00889             $tr .= "<tr class='$class'><td>Similarity to Resource</td>
00890             <td>Similarity = $sim</td>
00891             <td>$msg</td></tr>";
00892         } else {
00893             for ($i = 0; $i < 5; $i++) {
00894                 $key = "sim$i";
00895                 $sim = $r[$key];
00896                 [$class, $msg] = self::simToMsg($sim, 'other');
00897                 $other = $r[$key . "_email"];
00898                 $tr .= "<tr class='$class'><td>Similarity to  $other </td>
00899                 <td>Similarity = $sim</td>
00900                 <td>$msg</td></tr>";
00901             }
00902         }
00903         return $tr;
00904     }
00905 
00909     protected static function investigate()
00910     {
00911         self::init();
00912         $toStr = true;
00913         $studentEmail = $_POST['email'];
00914         $studentDao = self::$studentDao;
00915         $student = $studentDao->getByEmail($studentEmail)[0];
00916         $exam = self::$exam;
00917         $answerDao = self::$answerDao;
00918         $str = HT::head($toStr);
00919         $studentName = $student->getName();
00920         $str0 = "";
00921         $str1 = "<table class='grader plain' align='center'>
00922         <tr><th colspan='100%' style='font-size:110%;'>
00923         FBI Report: $studentName</th></tr>";
00924         foreach ($exam->getQuestions() as $question) {
00925             foreach ($question->getSubQuestions() as $subQuestion) {
00926                 $subQuestionName = $subQuestion->getName();
00927                 $answers = $answerDao->getByObject($student, $subQuestion);
00928                 if (count($answers) == 0) {
00929                     $str0 .= HT::error(" &rarr; SubQuestion
00930                     $subQuestionName: Got no answers.", $toStr);
00931                     continue;
00932                 }
00933                 foreach ($answers as $answer) {
00934                     $shortFileName = $answer->getShortFileName();
00935                     if (!self::$refFileDao->isGradable($shortFileName)) {
00936                         continue;
00937                     }
00938                     $resource = self::$refFileDao->getRefFile($shortFileName, 'resources');
00939                     if ($resource->getSim() > self::$simLimit) {
00940                         continue;
00941                     }
00942                     $basename = basename($shortFileName);
00943                     $str1 .= "<tr><th colspan='100%'>
00944                     SubQuestion: $subQuestionName,
00945                     File: <code>$basename</code></th></tr>";
00946                     $row = self::getFBI($student, $answer);
00947                     $count = count($row);
00948                     if ($count == 1) {
00949                         $str0 .= HT::success(" &rarr; SubQuestion
00950                         $subQuestionName [$basename] read from DB.", $toStr);
00951                     } else {
00952                         $str0 .= HT::error(" &rarr; SubQuestion
00953                         $subQuestionName [$basename]: Please process first
00954                         [$count rows].", $toStr);
00955                         continue;
00956                     }
00957                     $str1 .= self::investigateRow($row);
```

```
00958                 }
00959             }
00960         }
00961         $str1 .= "</table>";
00962         $str1 = HT::info($str1, true);
00963         // $str .= HT::warn($str0);
00964         $str .= HT::warn($str1);
00965         $str .= HT::foot($toStr);
00966         echo $str;
00967     }
00968 }
```

5.57 ui/report.php File Reference

5.58 ui/report.php

Go to the documentation of this file.

```
00001 <?php
00002 require_once("../model/autoload.php");
00003 require_once("../config.php");
00004
00005 Report::index();
```

5.59 setup.php File Reference

5.60 setup.php

Go to the documentation of this file.

```
00001 <?php
00002 require_once("../model/autoload.php");
00003 require_once("../config.php");
00004
00005 CFG::index();
```

5.61 stats.php File Reference

5.62 stats.php

Go to the documentation of this file.

```
00001 <?php
00002 require_once("../model/autoload.php");
00003 require_once("../config.php");
00004
00005 Stat::index();
```

5.63 test.php File Reference

Variables

- $sql0
- $sql = str_replace('{dbprefix}', CFG::$dbprefix, $sql0)

5.63.1 Variable Documentation

$sql

```
$sql = str_replace('{dbprefix}', CFG::$dbprefix, $sql0)
```

Definition at line 22 of file test.php.

Referenced by DB::colExists(), CFG::dropDB(), DB::get(), DB::getEnum(), DB::importSQL(), Stat::mistakes(), CFG::mkDropDB(), CFG::mkSql(), DB::mkSql(), Stat::mkSql(), DB::multiQuery(), DB::put(), DB::remove_inline(), DB::remove_remarks(), CFG::runSql(), DB::split_sql_file(), DB::tablesExist(), and DB::update().

$sql0

```
$sql0
```

Initial value:

```
= "SELECT
s.student_email AS 'email',
s.student_name AS 'name',
s.section AS 'section',
sq.subquestion_name AS 'subquestion',
sq.question_num AS 'question',
SUM(m.marks) AS 'marks'
FROM
`{dbprefix}students` AS s
INNER JOIN `{dbprefix}marks` AS m ON m.student_email = s.student_email
INNER JOIN `{dbprefix}subquestions` AS sq ON m.subquestion_name = sq.subquestion_name
GROUP BY
`email`, `name`, `section`, `subquestion`, `question`"
```

Definition at line 8 of file test.php.

Referenced by CFG::mkDropDB().

5.64 test.php

Go to the documentation of this file.

```
00001 <?php
00002 require_once("../model/autoload.php");
00003 require_once("../config.php");
00004
00005
00006
00007
00008 $sql0 = "SELECT
00009 s.student_email AS 'email',
00010 s.student_name AS 'name',
00011 s.section AS 'section',
00012 sq.subquestion_name AS 'subquestion',
00013 sq.question_num AS 'question',
00014 SUM(m.marks) AS 'marks'
00015 FROM
00016 `{dbprefix}students` AS s
00017 INNER JOIN `{dbprefix}marks` AS m ON m.student_email = s.student_email
00018 INNER JOIN `{dbprefix}subquestions` AS sq ON m.subquestion_name = sq.subquestion_name
00019 GROUP BY
00020 `email`, `name`, `section`, `subquestion`, `question`";
00021
00022 $sql = str_replace('{dbprefix}', CFG::$dbprefix, $sql0);
00023
00024 var_dump($sql);
```

Index