	Code Coverage						
	Classes and Traits		Functions and Methods			Lines	
Total	100.00%	1 / 1	100.00%	8 / 8	CRAP	100.00%	132 / 132
phpunit_dataset	100.00%	1 / 1	100.00%	8 / 8	62	100.00%	132 / 132
from_files			100.00%	1 / 1	3	100.00%	4 / 4
from_file			100.00%	1 / 1	5	100.00%	9 / 9
from_string			100.00%	1 / 1	5	100.00%	11 / 11
from_array			100.00%	1 / 1	8	100.00%	24 / 24
to_database			100.00%	1 / 1	12	100.00%	18 / 18
get_rows			100.00%	1 / 1	7	100.00%	9 / 9
load_csv			100.00%	1 / 1	4	100.00%	17 / 17
load_xml			100.00%	1 / 1	18	100.00%	40 / 40

```php
<?php
// This file is part of Moodle - http://moodle.org/
//
// Moodle is free software: you can redistribute it and/or modify
// it under the terms of the GNU General Public License as published by
// the Free Software Foundation, either version 3 of the License, or
// (at your option) any later version.
//
// Moodle is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
// GNU General Public License for more details.
//
// You should have received a copy of the GNU General Public License
// along with Moodle.  If not, see <http://www.gnu.org/licenses/>.

/**
 * Handle simple PHP/CSV/XML datasets to be use with ease by unit tests.
 *
 * This is a very minimal class, able to load data from PHP arrays and
 * CSV/XML files, optionally uploading them to database.
 *
 * This doesn't aim to be a complex or complete solution, but just a
 * utility class to replace old phpunit/dbunit uses, because that package
 * is not longer maintained. Note that, ideally, generators should provide
 * the needed utilities to proceed with this loading of information to
 * database and, if there is any future that should be it.
 *
 * @package    core
 * @category   test
 * @copyright  2020 onwards Eloy Lafuente (stronk7) {@link http://stronk7.com}
 * @license    http://www.gnu.org/copyleft/gpl.html GNU GPL v3 or later
 */

declare(strict_types=1);

/**
 * Lightweight dataset class for phpunit, supports XML, CSV and array datasets.
 *
 * This is a simple replacement class for the old old phpunit/dbunit, now
 * archived. It allows to load CSV, XML and array structures to database.
 */
class phpunit_dataset {

    /** @var array tables being handled by the dataset */
    protected $tables = [];
    /** @var array columns belonging to every table (keys) handled by the dataset */
    protected $columns = [];
    /** @var array rows belonging to every table (keys) handled by the dataset */
    protected $rows = [];

    /**
     * Load information from multiple files (XML, CSV) to the dataset.
     *
     * This method accepts an array of full paths to CSV or XML files to be loaded
     * into the dataset. For CSV files, the name of the table which the file belongs
     * to needs to be specified. Example:
     *
     *   $fullpaths = [
     *       '/path/to/users.xml',
     *       'course' => '/path/to/courses.csv',
     *   ];
     *
     * @param array $fullpaths full paths to CSV or XML files to load.
     */
    public function from_files(array $fullpaths): void {
        foreach ($fullpaths as $table => $fullpath) {
            $table = is_int($table) ? null : $table; // Only a table when it's an associative array.
            $this->from_file($fullpath, $table);
        }
    }

    /**
     * Load information from one file (XML, CSV) to the dataset.
     *
     * @param string $fullpath full path to CSV or XML file to load.
     * @param string $table name of the table which the file belongs to (only for CSV files).
     */
    public function from_file(string $fullpath, ?string $table = null): void {
        if (!file_exists($fullpath)) {
            throw new coding_exception('from_file, file not found: ' . $fullpath);
        }

        if (!is_readable($fullpath)) {
            throw new coding_exception('from_file, file not readable: ' . $fullpath);
        }

        $extension = strtolower(pathinfo($fullpath, PATHINFO_EXTENSION));
        if (!in_array($extension, ['csv', 'xml'])) {
            throw new coding_exception('from_file, cannot handle files with extension: ' . $extension);
        }

        $this->from_string(file_get_contents($fullpath), $extension, $table);
    }

    /**
     * Load information from a string (XML, CSV) to the dataset.
     *
     * @param string $content contents (CSV or XML) to load.
     * @param string $type format of the content to be loaded (csv or xml).
     * @param string $table name of the table which the file belongs to (only for CSV files).
     */
    public function from_string(string $content, string $type, ?string $table = null): void {
        switch ($type) {
            case 'xml':
                $this->load_xml($content);
                break;
            case 'csv':
                if (empty($table)) {
                    throw new coding_exception('from_string, contents of type "cvs" require a $table to be passed, none found');
                }
                $this->load_csv($content, $table);
                break;
            default:
                throw new coding_exception('from_string, cannot handle contents of type: ' . $type);
        }
    }

    /**
     * Load information from a PHP array to the dataset.
     *
     * The general structure of the PHP array must be
     *   [table name] => [array of rows, each one being an array of values or column => values.
     * The format of the array must be one of the following:
     * - non-associative array, with column names in the first row (pretty much like CSV files are):
     *     $structure = [
     *         'table 1' => [
     *             ['column name 1', 'column name 2'],
     *             ['row 1 column 1 value', 'row 1 column 2 value'*,
     *             ['row 2 column 1 value', 'row 2 column 2 value'*,
     *         ],
     *         'table 2' => ...
     *     ];
     * - associative array, with column names being keys in the array.
     *     $structure = [
     *         'table 1' => [
     *             ['column name 1' => 'row 1 column 1 value', 'column name 2' => 'row 1 column 2 value'],
     *             ['column name 1' => 'row 2 column 1 value', 'column name 2' => 'row 2 column 2 value'],
     *         ],
     *         'table 2' => ...
     *     ];
     * @param array $structure php array with a valid structure to be loaded to the dataset.
     */
    public function from_array(array $structure): void {
        foreach ($structure as $tablename => $rows) {
            if (in_array($tablename, $this->tables)) {
                throw new coding_exception('from_array, table already added to dataset: ' . $tablename);
            }

            $this->tables[] = $tablename;
            $this->columns[$tablename] = [];
            $this->rows[$tablename] = [];

            $isassociative = false;
            $firstrow = reset($rows);

            if (array_key_exists(0, $firstrow)) {
                // Columns are the first row (csv-like).
                $this->columns[$tablename] = $firstrow;
                array_shift($rows);
            } else {
                // Columns are the keys on every record, first one must have all.
                $this->columns[$tablename] = array_keys($firstrow);
                $isassociative = true;
            }

            $countcols = count($this->columns[$tablename]);
            foreach ($rows as $row) {
                $countvalues = count($row);
                if ($countcols != $countvalues) {
                    throw new coding_exception('from_array, number of columns must match number of values, found: ' .
                        $countcols . ' vs ' . $countvalues);
                }
                if ($isassociative && $this->columns[$tablename] != array_keys($row)) {
                    throw new coding_exception('from_array, columns in all elements must match first one, found: ' .
                        implode(', ', array_keys($row)));
                }
                $this->rows[$tablename][] = array_combine($this->columns[$tablename], array_values($row));
            }
        }
    }

    /**
     * Send all the information to the dataset to the database.
     *
     * This method gets all the information loaded in the dataset, using the from_xxx() methods
     * and sends it to the database; table and column names must match.
     *
     * Note that, if the information to be sent to database contains sequence columns (usually 'id')
     * then those values will be preserved (performing an import and adjusting sequences later). Else
     * normal inserts will happen and sequence (auto-increment) columns will be fed automatically.
     *
     * @param ?string[] $filter Tables to be sent to database. If not specified, all tables are processed.
     */
    public function to_database(?array $filter = []): void {
        global $DB;

        // Verify all filter elements are correct.
        foreach ($filter as $table) {
            if (!in_array($table, $this->tables)) {
                throw new coding_exception('dataset_to_database, table is not in the dataset: ' . $table);
            }
        }

        $structure = phpunit_util::get_tablestructure();

        foreach ($this->tables as $table) {
            // Apply filter.
            if (!empty($filter) && !in_array($table, $filter)) {
                continue;
            }

            $doimport = false;

            if (isset($structure[$table]['id']) and $structure[$table]['id']->auto_increment) {
                $doimport = in_array('id', $this->columns[$table]);
            }

            foreach ($this->rows[$table] as $row) {
                if ($doimport) {
                    $DB->import_record($table, $row);
                } else {
                    $DB->insert_record($table, $row);
                }
            }

            if ($doimport) {
                $DB->get_manager()->reset_sequence(new xmldb_table($table));
            }
        }
    }

    /**
     * Returns the rows, for a given table, that the dataset holds.
     *
     * @param ?string[] $filter Tables to return rows. If not specified, all tables are processed.
     * @return array tables as keys with rows on each as sub array.
     */
    public function get_rows(?array $filter = []): array {
        // Verify all filter elements are correct.
        foreach ($filter as $table) {
            if (!in_array($table, $this->tables)) {
                throw new coding_exception('dataset_get_rows, table is not in the dataset: ' . $table);
            }
        }

        $result = [];
        foreach ($this->tables as $table) {
            // Apply filter.
            if (!empty($filter) && !in_array($table, $filter)) {
                continue;
            }
            $result[$table] = $this->rows[$table];
        }
        return $result;
    }

    /**
     * Given a CSV content, process and load it as a table into the dataset.
     *
     * @param string $content CSV content to be loaded (only one table).
     * @param string $tablename Name of the table the content belongs to.
     */
    protected function load_csv(string $content, string $tablename): void {
        if (in_array($tablename, $this->tables)) {
            throw new coding_exception('csv_dataset_format, table already added to dataset: ' . $tablename);
        }

        $this->tables[] = $tablename;
        $this->columns[$tablename] = [];
        $this->rows[$tablename] = [];

        // Normalise newlines.
        $content = preg_replace('#\r\n?#', '\n', $content);

        // Function str_getcsv() is not good for new lines within the data, so lets use temp file and fgetcsv() instead.
        $tempfile = tempnam(make_temp_directory('phpunit'), 'csv');
        $fh = fopen($tempfile, 'w+b');
        fwrite($fh, $content);

        // And let's read it using fgetcsv().
        rewind($fh);

        // We just accept default, delimiter = comma, enclosure = double quote.
        while ( ($row = fgetcsv($fh) ) !== false ) {
            if (empty($this->columns[$tablename])) {
                $this->columns[$tablename] = $row;
            } else {
                $this->rows[$tablename][] = array_combine($this->columns[$tablename], $row);
            }
        }
        fclose($fh);
        unlink($tempfile);
    }

    /**
     * Given a XML content, process and load it as tables into the dataset.
     *
     * @param string $content XML content to be loaded (can be multi-table).
     */
    protected function load_xml(string $content): void {
        $xml = new SimpleXMLElement($content);
        // Main element must be dataset.
        if ($xml->getName() !== 'dataset') {
            throw new coding_exception('xml_dataset_format, main xml element must be "dataset", found: ' . $xml->getName());
        }

        foreach ($xml->children() as $table) {
            // Only table elements allowed.
            if ($table->getName() !== 'table') {
                throw new coding_exception('xml_dataset_format, only "table" elements allowed, found: ' . $table->getName());
            }
            // Only allowed attribute of table is "name".
            if (!isset($table['name'])) {
                throw new coding_exception('xml_dataset_format, "table" element only allows "name" attribute.');
            }

            $tablename = (string)$table['name'];
            if (in_array($tablename, $this->tables)) {
                throw new coding_exception('xml_dataset_format, table already added to dataset: ' . $tablename);
            }

            $this->tables[] = $tablename;
            $this->columns[$tablename] = [];
            $this->rows[$tablename] = [];

            $countcols = 0;
            foreach ($table->children() as $colrow) {
                // Only column and row allowed.
                if ($colrow->getName() !== 'column' && $colrow->getName() !== 'row') {
                    throw new coding_exception('xml_dataset_format, only "column or "row" elements allowed, found: ' .
                        $colrow->getName());
                }
                // Column always before row.
                if ($colrow->getName() == 'column' && !empty($this->rows[$tablename])) {
                    throw new coding_exception('xml_dataset_format, "column" elements always must be before "row" ones');
                }
                // Row always after column.
                if ($colrow->getName() == 'row' && empty($this->columns[$tablename])) {
                    throw new coding_exception('xml_dataset_format, "row" elements always must be after "column" ones');
                }

                // Process column.
                if ($colrow->getName() == 'column') {
                    $this->columns[$tablename][] = (string)$colrow;
                    $countcols++;
                }

                // Process row.
                if ($colrow->getName() == 'row') {
                    $countvalues = 0;
                    $row = [];
                    foreach ($colrow->children() as $value) {
                        // Only value allowed under row.
                        if ($value->getName() !== 'value') {
                            throw new coding_exception('xml_dataset_format, only "value" elements allowed, found: ' .
                                $value->getName());
                        }
                        $row[$this->columns[$tablename][$countvalues]] = (string)$value;
                        $countvalues++;
                    }
                    if ($countcols !== $countvalues) {
                        throw new coding_exception('xml_dataset_format, number of columns must match number of values, found: ' .
                            $countcols . ' vs ' . $countvalues);
                    }
                    $this->rows[$tablename][] = $row;
                }
            }
        }
    }
}
```

Legend

Executed | Not Executed | Dead Code