Batch API <https://drupal.org/node/180528>

**Functions allowing forms processing to be spread out over several page requests, thus ensuring that the processing does not get interrupted because of a PHP timeout, while allowing the user to receive feedback on the progress of the ongoing operations.**

The API is primarily designed to integrate nicely with the Form API workflow, but can also be used by non-Form API scripts (like update.php) or even simple page callbacks (which should probably be used sparingly).

Example:

$batch = array(

'title' => t('Exporting'),

**'operations'** => array(

array('my\_function\_1', array($account->uid, 'story')),

array('my\_function\_2', array()),

),

**'finished'** => 'my\_finished\_callback',

**'file'** => 'path\_to\_file\_containing\_myfunctions',

);

**batch\_set($batch);**

// Only needed if not inside a form \_submit handler.

// Setting redirect in batch\_process.

**batch\_process('node/1');**

Note: if the batch 'title', 'init\_message', 'progress\_message', or 'error\_message' could contain any user input, **it is the responsibility of the code calling batch\_set() to sanitize them first with a function like check\_plain() or filter\_xss().** Furthermore, if the batch operation returns any user input in the 'results' or 'message' keys of $context, it must also sanitize them first.

**batch\_get Retrieves the current batch.**

**batch\_process Processes the batch.**

**batch\_set Adds a new batch.**

**hook\_batch\_alter Alter batch information before a batch is processed.**

**\_batch\_populate\_queue Populates a job queue with the operations of a batch set.**

**\_batch\_queue Returns a queue object for a batch set.**

/\*\*

\* The $batch can include the following values. Only 'operations'

\* and 'finished' are required, all others will be set to default values.

\*

\* @param operations

\* An array of callbacks and arguments for the callbacks.

\* There can be one callback called one time, one callback

\* called repeatedly with different arguments, different

\* callbacks with the same arguments, one callback with no

\* arguments, etc. (Use an empty array if you want to pass

\* no arguments.)

\*

\* @param finished

\* A callback to be used when the batch finishes.

\*

\* @param title

\* A title to be displayed to the end user when the batch starts. The default is 'Processing'.

\*

\* @param init\_message

\* An initial message to be displayed to the end user when the batch starts.

\*

\* @param progress\_message

\* A progress message for the end user. Placeholders are available.

\* Placeholders note the progression by operation, i.e. if there are

\* 2 operations, the message will look like:

\* 'Processed 1 out of 2.'

\* 'Processed 2 out of 2.'

\* Placeholders include:

\* @current, @remaining, @total and @percentage

\*

\* @param error\_message

\* The error message that will be displayed to the end user if the batch

\* fails.

\*

\* @param file

\* Path to file containing the callbacks declared above. Always needed when

\* the callbacks are not in a .module file.

\*

\*/

function batch\_example($options1, $options2, $options3, $options4) {

$batch = array(

**'operations' => array(**

**array('batch\_example\_process', array($options1, $options2)),**

**array('batch\_example\_process', array($options3, $options4)),**

**),**

**'finished' => 'batch\_example\_finished',**

**'title' => t('Processing Example Batch'),**

**'init\_message' => t('Example Batch is starting.'),**

**'progress\_message' => t('Processed @current out of @total.'),**

**'error\_message' => t('Example Batch has encountered an error.'),**

**'file' => drupal\_get\_path('module', 'batch\_example') . '/batch\_example.inc',**

);

**batch\_set($batch);**

// If this function was called from a form submit handler, stop here,

// FAPI will handle calling batch\_process().

// If not called from a submit handler, add the following,

// noting the url the user should be sent to once the batch

// is finished.

// IMPORTANT:

// If you set a blank parameter, the batch\_process() will cause an infinite loop

**batch\_process('node/1');**

}

/\*\*

\* Batch Operation Callback

\*

**\* Each batch operation callback will iterate over and over until**

**\* $context['finished'] is set to 1. After each pass, batch.inc will**

**\* check its timer and see if it is time for a new http request,**

**\* i.e. when more than 1 minute has elapsed since the last request.**

**\* Note that $context['finished'] is set to 1 on entry - a single pass**

**\* operation is assumed by default.**

\*

\* An entire batch that processes very quickly might only need a single

\* http request even if it iterates through the callback several times,

\* while slower processes might initiate a new http request on every

\* iteration of the callback.

\*

\* This means you should set your processing up to do in each iteration

\* only as much as you can do without a php timeout, then let batch.inc

\* decide if it needs to make a fresh http request.

\*

\* @param options1, options2

\* If any arguments were sent to the operations callback, they

\* will be the first arguments available to the callback.

\*

**\* @param context**

**\* $context is an array that will contain information about the**

**\* status of the batch.** The values in $context will retain their

\* values as the batch progresses.

\*

**\* @param $context['sandbox']**

**\* Use the $context['sandbox'] rather than $\_SESSION to store the**

**\* information needed to track information between successive calls to**

**\* the current operation. If you need to pass values to the next operation**

**\* use $context['results'].**

\*

**\* The values in the sandbox will be stored and updated in the database**

**\* between http requests until the batch finishes processing. This will**

**\* avoid problems if the user navigates away from the page before the**

**\* batch finishes.**

\*

**\* @param $context['results']**

\* The array of results gathered so far by the batch processing. This

\* array is highly useful for passing data between operations. After all

\* operations have finished, these results may be referenced to display

\* information to the end-user, such as how many total items were

\* processed.

\*

**\* @param $context['message']**

\* A text message displayed in the progress page.

\*

**\* @param $context['finished']**

\* A float number between 0 and 1 informing the processing engine

\* of the completion level for the operation.

\*

\* 1 (or no value explicitly set) means the operation is finished

\* and the batch processing can continue to the next operation.

\*

\* Batch API resets this to 1 each time the operation callback is called.

\*/

function batch\_example\_process($options1, $options2, &$context) {

if (!isset($context['sandbox']['progress'])) {

$context['sandbox']['progress'] = 0;

$context['sandbox']['current\_node'] = 0;

$context['sandbox']['max'] = db\_query('SELECT COUNT(DISTINCT nid) FROM {node}')->fetchField();

}

// For this example, we decide that we can safely process

// 5 nodes at a time without a timeout.

$limit = 5;

// With each pass through the callback, retrieve the next group of nids.

$result = db\_query\_range("SELECT nid FROM {node} WHERE nid > %d ORDER BY nid ASC", $context['sandbox']['current\_node'], 0, $limit);

while ($row = db\_fetch\_array($result)) {

// Here we actually perform our processing on the current node.

$node = node\_load($row['nid'], NULL, TRUE);

$node->value1 = $options1;

$node->value2 = $options2;

node\_save($node);

// Store some result for post-processing in the finished callback.

$context['results'][] = check\_plain($node->title);

// Update our progress information.

$context['sandbox']['progress']++;

$context['sandbox']['current\_node'] = $node->nid;

$context['message'] = t('Now processing %node', array('%node' => $node->title));

}

// Inform the batch engine that we are not finished,

// and provide an estimation of the completion level we reached.

if ($context['sandbox']['progress'] != $context['sandbox']['max']) {

$context['finished'] = $context['sandbox']['progress'] / $context['sandbox']['max'];

}

}

/\*\*

\* Batch 'finished' callback

\*/

function batch\_example\_finished($success, $results, $operations) {

if ($success) {

// Here we do something meaningful with the results.

$message = t('@count items successfully processed:', array('@count' => count($results)));

$message .= theme('item\_list', array('items' => $results));

drupal\_set\_message($message);

}

else {

// An error occurred.

// $operations contains the operations that remained unprocessed.

$error\_operation = reset($operations);

$message = t('An error occurred while processing %error\_operation with arguments: @arguments', array('%error\_operation' => $error\_operation[0], '@arguments' => print\_r($error\_operation[1], TRUE)));

drupal\_set\_message($message, 'error');

}

}