Moodle module backup and restore

Moodle module backup and restore

By default, a module is ignored by the backup and restore processes, i.e. its database entries are not backed up or restored. To support backup and restore, the module needs to have the files backuplib.php and restorelib.php in its directory, and the files should implement several functions and in a certain way. Unfortunately, there is very little documentation out there about the implementation. Moodle Doc only has one stub page on backup, and none on restore.

In addition to the GUI option for course backup Moodle provides api for backing up a course via a script by passing the user options and modules.

Backup

The course backup process

Step One (backup_form.html)

The user starts by choosing which of the instances of modules to backup and whether to back up user data. There are other settings such as whether to back up users at the bottom.

Step Two (backup_check.html)

This page shows the user what will be backed up, and gives the user choices of proceeding or backing out.

Steep Three (backup_execute.html)

The actual backup is done in this step. Modules take turn to write to the backup XML file while their names are listed. If there is any error, this page also indicates that the backup failed for or skipped that particular module. Finally, the files are compressed and the ZIP file is stored in the files area.

Functions

backup_execute() gets called from the step three above and initiates the process in backuplib.php

backup.lib

backup.lib is responsible for including backuplib.php for each chosen module. This raises the condition that none of the module names are alike and common files across modules can be included exactly once.

backup/lib.php

Function backup_course_silently – takes user options from another function and process backup

Function backup_generate_preferences_artificially – generates user preferences to instruct the backup process what modules, instances, user_data to be backed up.

mod/MODULENAME/backuplib.php

MODULENAME_check_backup_mods($course,$user_data=false,$backup_unique_code,$instances=null) — required

$course – the course to be backed up.

$user_data – array of objects. Each of these objects represents a module instance in the course, and basically contains information from a record in mdl_MODULENAME. For example,

```
Array (
0 => stdClass Object (
    [id] => 3
    [course] => 3
    [name] => Test Label One
    …
```

```
  )
1 => stdClass Object (
    [id] => 4
    [course] => 3
    [name] => Test Label Two
    …
  )
)
```

$backup_unique_code – a code unique to this backup action, apparently to avoid two backup action being run twice.

$instances – array of objects. Each of these objects represents a module instance selected for backup and has three properties: “name” (module instance name), “userdata” (1 if backing up user data; 0 if not), “id” (module instance ID). For example, if the user chooses to back up Test Label One (label #3) with user data and Test Label Two (label #4) without user data, $instance will look like this:

```
Array (
3 => stdClass Object (
    [name] => Test Label One
    [userdata] => 1
    [id] => 3
  )
4 => stdClass Object (
    [name] => Test Label Two
    [userdata] => 0
    [id] => 4
  )
)
```

Return value – information about instances of a module be backed up in Step Two. Its format is that used by $table→data given to the print_table() function. That is, if the function return this array:

```
Array (  [key1] => Array (    [key1a] => row 1 column 1    [key1b] => row 1 column 2  )  [key2]
```

Then the following table is printed in Step Two under the module’s heading:

row 1 column 1 row 1 column 2 row 2 column 2 row 2 column 2
Note that the content can contain HTML code, and that’s how the assignment module achieves the effect of subheadings.

MODULENAME_encode_content_links($content,$preferences) — optional

Replace domain-dependent links to pages within a module with some domain-independent short form, so that the backup can be restored to another server and work. It looks like the format of the short form is up to the developer of the module, as long as it is understood by function

MODULENAME_decode_content_links (which reverses the process). The modules I checked replaced something like “http://www.yourname.com/yourmoodlesite/mod/MODULENAME /somepage.php?param1=123&param2=456” with something like “$`SOMETHINGYOUMAKEUP*$2*$3`$”.

$content – the original content with links to be encoded.

$perferences – don’t know what this is. It is not used in any of the modules I looked at.

Return value – the processed content with encoded links.

Restore

Backup IDs

When restoring, you cannot just get the data and insert records without modifying, because any foreign key you store is going to be invalid. For example, if you restore forum posts, the user IDs associated with the posts will be different in the restored course from the original course.

Moodle solves this problem with the mdl_backup_ids table, which maintains a map between values of IDs in the original course and the new course. Presumably, Moodle takes care of user, groups and roles before the functions in your restorelib.php are called. So your functions will use the following workflow:

1. When you prepare a record for insertion, if a field is an ID that changes in the backup/restore process, use the backup_getid() function to get its value.
2. After inserting a record, note the insert ID (i.e. the id of the new record) and use the backup_putid() function to store it in the mdl_backup_ids table.
3. Process the next item.

Revision #4
Created 2010-07-21 19:04:50 UTC by Chan, Wing Kai
Updated 2010-07-23 18:36:50 UTC by Chan, Wing Kai