Plugin developer guide

Zabbix Report

This guide is aimed at moodle plugins developpers that want implement Zabbix Report compatibility so the plugin can be discovered by the Zabbix Report and have indicators emission handled by the sending tasks.

Note that only the Pro version of the Zabbix Report plugin with a valid license key will support this integration. Free releases of the report plugin **will not** provide this extension capability.

Plugin registration method

The plugin can be registrated within the Zabbix Report among two scenarios:

- The third-party plugin is installed AFTER the Zabbix Report has been installed.
- The third parry-plugin is installed BEFORE the Zabbix Report plugin.

Note : note that if you install a complete moodle preequiped with third-party plugins, then both cases might happen depending on the pluginlist order.

Registration function in lib.php

You will add to your main `lib.php` file the following function:

```
/**
 * This function is a "late" callback, when zabbix report
 * is installed AFTER this plugin has been installed. It is called
 * by the Zabbix Report installer to register all compatible plugins.
 */
function <plugin_frankenname>_zabbix_register() {
    global $CFG;

    if (is_dir($CFG->dirroot.'/report/zabbix')) {
        include_once($CFG->dirroot.'/report/zabbix/xlib.php');
        report_zabbix_register_plugin('<plugintype>', '<pluginname>');
    }
}
```

Handling post installation case

If the Zabbix report plugin is already installed, it will not scan for your third-party plugin, but the third-party plugin will use it's post install sequence to do so.

In your `db/install.php` post install file, add the following;

```
function xmldb_<plugin_frankenname>_after_install() {
```

```
    [...]
```

```
    // Register zabbix indicators if installed.
    // Note will only work with report_zabbix "pro" version.
    // This call is only a wrapper.
    if (is_dir($CFG->dirroot.'/report/zabbix')) {
        include_once($CFG->dirroot.'/<pluginpath>/lib.php'); // If not yet done.
        learningtimecheck_zabbix_register();
    }
    [...]
}
```

This function will reuse the thrird-party registering callback at install time.

Defining indicators

Create a `indicators` directory in your `classes` plugin's subdirectory.

Depending on how often you want to send data to Zabbix, you will adopt the following rules :

- Sending each cron run : create indicator classes directly in `classes/indicators` (whatever name they have).
- Sending hourly values : create a `hourly` directory in `indicators` and implement your indicator classes there.
- Sending daily values : same as above, in a `daily` directory.
- Sending weekly values : same as above creating a `weekly` directory.
- Sending monthly values : ... no worth explaining how, uh ?

Implementing indicator class

An indicator class is a class in the `report_zabbix\indicators` namespace, even if located in your plugin. It has a unique parent class:

```
namespace report_zabbix\indicators;

use moodle_exception; // If needed
use coding_exception; // If needed
use StdClass; // If needed, usually it is.

require_once($CFG->dirroot.'/report/zabbix/classes/indicator.class.php');

class daily_<indicatorset>_indicators extends zabbix_indicator {
```

```
    [...]
}
```

Note that the php file must:

- Have one single indicatorset class
- be named as the class i.e. : daily_<indicatorset>_indicators

The indicator class can send one **or several** values, called “submodes” of the main indcatorset. It will be usually the case, so we start immediatly with this use case.

Usually we name class as:

- `<indicatorset>_indicators` for full range values.
- `<range>_<indicatorqet>_indicators` for values collected on a <range> limited scope (f.e. daily, scannning data on 24 hours in the past).

Describing submodes

Now describe your submodes as a static constant:

```
  static $submodes = 'submode1,submode2,submode3';
```

Each submode is a single measurement and a single value sent to Zabbix.

Constructor

The constructor only defines the Zabbix value namespace that will be prepended to all submodes. Zabbix model will use the FQDN of the indicator values for defining items.

```
  public function __construct() {
      parent::__construct();
      $this->key = 'moodle.<indicatorset>';
  }
```

Acquiring measures

Here is the most important part of the implementation. For each submode listed in the `submodes` static attribute, You will provide an acquisition handler that must get a single value.

```
  /**
   * the function that contains the logic to acquire the indicator instant
value.
   * @param string $submode to target an aquisition to an explicit submode,
elsewhere
   */
  public function acquire_submode($submode) {
```

```
      global $DB;

      if(!isset($this->value)) {
          $this->value = new Stdclass;
      }

      if (is_null($submode)) {
          $submode = $this->submode;
      }

      switch ($submode) {

          case 'submode1': {

              $submodevalue = [...]; // Whatever gets a single value in
Moodle.

              $this->value->$submode = $submodevalue;
              break;
          }

          default: {
              if ($CFG->debug == DEBUG_DEVELOPER) {
                  throw new coding_exception("Indicator has a submode that
is not handled in acquire_submode().");
              }
          }
      }
```

Once each submode case is written, that's all done !

Back to Zabbix Report index - Back to plugin index - Back to catalog

From:
https://docsen.activeprolearn.com/ - **Documentation Moodle ActiveProLearn**

Permanent link:
https://docsen.activeprolearn.com/doku.php?id=report:zabbix:developerguide&rev=1770493248

Last update: **2026/02/07 20:40**