Events in translate5

This document the state "draft" !!!

If you need an event in translate5, that is not listed here, please contact support@translate5.net.

Automatic events

Automatic events are defined in some global (basic ??) tranlate5 classes. For the events-names variables or automatisms are used. On that way a lot of internal trigger-points are generated on the fly.

Example:

```
        public function preDispatch()
        {
            $eventName = "before".ucfirst($this->_request->getActionName())."Action";
            $this->events->trigger($eventName, $this);
        }
```

will define an "beforeControllerAction"-event for each an every controller. For the default IndexController of the Editor-Modul this will lead to an event *Editor_IndexController#beforeIndexAction*

Controller-Events

defined in */library/ZfExtended/Controllers/Action.php* will trigger an event on each and every controller. The different controllers are named in the following list as *Controllername* which are Index, Login etc.

- **before*Actionname*Action** with parameter 'controller' containing the controller on Zend preDispatch
- **after*Actionname*Action** with parameter $this->view on Zend postDispatch

RestController-Events

- **before*Actionname*Action** with the following parameters:
  - entity: $this->entity
  - params: all parameters
  - controller: the controller instance
- **afterSetDataInEntity** with the following parameters:
  - entity: $this->entity
- **beforeSetDataInEntity** with the following parameters:
  - entity: $this->entity
  - data: $this->data
- **after*Actionname*Action** with the following parameters:
  - entity: $this->entity
  - view: $this->view

ZfExtended_Models_Entity_Abstract

- **beforeSave** with parameter array('entity' => $this), on every save as first function. !!! be careful on overwritten methods to call parent::save() in first place or to take care of events !!!

editor_Workflow_Abstract

Task

- **doReopen**
- **doEnd**
- **doConfirm** (getting status "open" coming from status "unconfirmed")

TaskUserAssoc

- **beforeFinish**
- **doUnfinish**
- **beforeOpen**
- **doOpen**
- **beforeView**
- **doView**
- **beforeEdit**
- **doEdit**
- **beforeFinish**

- **doFinish**
- **beforeWait**
- **doWait**

Editor_SegmentController

- **beforePutSave** with parameter array('entity' => $this->entity), used in function putAction() after normal processing before saving the entity (= the segment)

Editor_TaskController

- **afterTaskOpen** with parameter array('task' => $this->entity, 'view' => $this->view, 'openState' => the opened state), used after the task was registered in the session
- **afterTaskClose** with parameter array('task' => $this->entity, 'view' => $this->view), used after the task was unregistered / removed from the session

editor_LanguageresourcetaskassocController

- afterPost#(Language Resources service name)
    - fired after language resource to task association is created (post action). The event name contains the language resources service name (ex: when TermCollection is assigned as language resource to the task the event name will be 'afterPost#TermCollection')
- afterDelete#(Language Resources service name)
    - fired after language resource to task association is removed (delete action). Example of full event name when the language resoure is OpenTM2 afterDelete#OpenTM2

editor_Models_Import

- **importWorkerQueued** is fired after the Import Worker is queued (but not started yet). Parameter: 'task' => editor_Models_Task, 'workerId' => integer
- **afterImport** is fired after parsing the data and storing the segments in DB. Parameter: 'task' => editor_Models_Task, 'parentWorkerId' => integer, 'importConfig' => editor_Models_Import_Configuration
- **importCompleted** is fired after all import plugins were run, defines the end of import. Parameter: 'task' => editor_Models_Task

editor_Models_Import_SegmentProcessor_Review

- process: is fired in processing the segment before it is first saved to the DB. Parameter: 'segment' => editor_Models_Segment, 'segmentAttributes' => editor_Models_Import_FileParser_SegmentAttributes, 'importConfig' => editor_Models_Import_Configuration

All other SegmentProcessors may follow with a process event.

editor_Models_Import_Worker_FileTree

- **beforeDirectoryParsing** is fired before directory parsing of workfiles file. Parameter: 'importFolder' => string, 'task' => editor_Models_Task, 'workerParentId' => parent worker id of the filetree worker
- **afterDirectoryParsing** is fired after directory parsing of workfiles files but before further processing of the files. Parameters: 'importFolder' => string, 'task' => editor_Models_Task, 'filelist' => array (fileIds to file paths), 'workerParentId' => parent worker id of the filetree worker

editor_Models_Export

- **afterExport** is fired after exporting the data to a folder on the disk, also on ZIP export. Parameter: 'task' => editor_Models_Task, 'parentWorkerId' => integer
  triggered only for export in the original imported format, is not triggered for xliff2 export.

editor_Models_Export_ExportedWorker

- **exportCompleted** is fired after all export steps (inclusive all export workers) are finished. Parameter: 'task' => editor_Models_Task, 'parameters' => array, the initial parameters given to the worker

ZfExtended_Mail

- **afterMailViewInit** is fired after the view for rendering the mail templates is initialized. Parameter: 'view' => Zend_View

Handmade events

Handmade events are spezial events wich are defined direct in the code. No automatic definition is used while trigger.

At the moment there are no handmade events.

Recomendend Best-Practice for using events in classes

Because Zend has a special class for static (global) events, best-pratice to use events in a class is:

Event-Trigger

- use a protected variable $events to hold the event(-trigger)-object
- initialize $this->events in the class-constructor
- parameter should be send in an named-array 'name' => $value

Example:

```
/**
 * @var ZfExtended_EventManager
 */
protected $events = false;

public function __construct() {
    $this->events = ZfExtended_Factory::get('ZfExtended_EventManager', array(get_class($this)));
}

public function doSomething(){
    $this->events->trigger("eventName", $this, array('model' => $this, 'moreParam' => $moreParams));
}
```

Event-Listener

- use a protected variable $staticEvents to hold the event(-listener)-Object.
- initialize $this->staticEvents in the class-constructor
- define all event-listeners in the class-constructor
- if handler use a event-parameter which is an object make a var-definition-comment so the IDE autocomplete can work correct

Example:

```
/**
 * @var Zend_EventManager_StaticEventManager
 */
protected $staticEvents = false;

public function __construct(){
    $this->staticEvents = Zend_EventManager_StaticEventManager::getInstance();
    $this->staticEvents->attach('classNameTriggerClass', 'eventName', array($this, 'handleEvent'));
}

public function handleEvent(Zend_EventManager_Event $event) {
    // do something on event "eventName" with parameters send within event-trigger
    $model = $event->getParam('model');
    /* @var $model nameOfTheModelClass */ // to trigger IDE
    $moreParams = $event->getParam('moreParams');
}
```

Trigger and Listen in one class

If you use two different class-variables $events and $staticEvents you can combine event-triggering and event-listening in one class without problems.