Events API and Logging

- The new Events API was developed in Moodle 2.6 and required in Moodle 2.7, extended the previous 'legacy' events and logging APIs.
- Moodle 2.7 and later versions still have support for the 'legacy' systems, but this will eventually be deprecated.

Observers

- Components that capture events define *observers* for the particular event in its ***db/events.php*** file in the `$observers` array.
- Observers must never cause a fatal error, but can raise exceptions which are caught and sent to the logs.
- View all the available core and plugin events by navigating to '*Site administration->Reports->Events list*' (***/report/eventlist/index.php***)

Observers (cont)

- Observers are defined in the `$observers` array in the **db/events.php** file and each array member has the following keys:
  - **eventname** – fully qualified event class name, e.g. `\core\event\user_created` or * for all events
  - **callback** – function, method, etc.; preferably an autoloaded class method, but see include file
  - **includefile** – (optional) file to be included before calling the observer to allow the callback to be available
  - **priority** – (optional) defaults to 0; observers with higher priority are notified first
  - **internal** – (optional) defaults to true; non-internal observers are not called during database transactions, but, instead, after a successful commit of the transaction

db/events.php

```php
$observers = array(
    array(
        'eventname'   => '\core\event\sample_executed',
        'callback'    => 'core_event_sample_observer::observe_one',
    ),
    array(
        'eventname'   => '\core\event\sample_executed',
        'callback'    => 'core_event_sample_observer::external_observer',
        'priority'    => 200,
        'internal'    => false,
    ),
    array(
        'eventname'   => '*',
        'callback'    => 'core_event_sample_observer::observe_all',
        'includefile' => null,
        'internal'    => true,
        'priority'    => 9999,
    ),
);
```

Dispatchers

- All events are defined by dispatchers.
- All events are logged – no need for separate logging.
- They define the events in classes defined in **namespaced** files defined in the ***classes/event*** folder.
- In naming the class, the convention is `some_object_action`, where `action` is a verb such as *created*, *added*, *deleted* or *attempted*. A list of recommended verbs is defined on the 'Events API' page

  https://docs.moodle.org/dev/Events_API#Verb_list

Dispatchers (cont)

- All event definitions are classes extending the `\core\event\base` class, and are triggered by creating a new instance of the class event and then executing the `trigger()` method.
- Each event is expected to be defined in a separate class file and is a unique identifier for each event e.g. `\core\event\user_created` is defined in the ***/lib/classes/event/user_created.php*** file.
- The event class has numerous properties, many of which are automatically computed when the event is triggered.

Event Properties (cont)

Some properties are statically declared in the event class' ***init()*** method.

Property name	Comment
objecttable (optional)	Database table name if relevant
crud (optional)	One of [crud] letters - indicating 'c'reate, 'r'ead, 'u'pdate or 'd'elete operation.
edulevel (optional)	Level of educational value of the event. One of • Teaching - self::LEVEL_TEACHING • Participating - self::LEVEL_PARTICIPATING • Other - self::LEVEL_OTHER

Event Properties (cont)

Property name	Comment
objectid (optional – required when objecttable is defined.)	Id of the object record from objecttable.
contextid	Computed for the context passed in.
userid (required)	Defaults to current userid, or 0 when not logged in, or -1 when other (System, CLI, Cron, ...)
relateduserid (optional)	Is this action related to some user? This could be used for some personal timeline view.
anonymous (optional)	Is this action anonymous?
other	Any other data for the event - json_encoded scalars or arrays.

Dispatchers (cont)

- The optional property `objectid` relates to the affected record's id in the `objecttable` property which relates to the relevant table - making it easy to identify the record
- In the case of deletions, the `add_record_snapshot()` and `get_record_snapshot()` methods store a copy of the relevant record BUT the snapshot is not saved; it is solely for the use of observers during the event.
- One other useful optional property is `other`, which can be a JSON encoded scalar or array data variable.