Configure event observers and scheduled jobs

Observers

Events are dispatched by modules when certain actions are triggered. In addition to its own events, Magento allows you to create your own events that can be dispatched in your code. When an event is dispatched, it can pass data to any observers configured to watch that event.

Best practices:

- Make your observer efficient
- Do not include business logic
- Declare observer in the appropriate scope
- Avoid cyclical event loops
- Do not rely on invocation order

> `14. Events`
>
> 14.1. All values (including objects) passed to an event MUST NOT be modified in the event observer. Instead, plugins SHOULD BE used for modifying the input or output of a function.
>
> – Magento DevDocs - Technical guidelines

Demonstrate how to configure observers

- can define observer in global area, then disable in specific area

Observers can be configured in the `events.xml` file.

Properties:

- name (required) - Name of the observer for the event definition
- instance (required) - Class name of the observer
- disabled - Is observer active or not (Default: false)
- shared - Class lifestyle (Default: false)

Example:

```
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:framework:Event/etc/events.xsd">
    <event name="cms_block_save_before">
        <observer name="myObserverName"
instance="MyCompany\MyModule\Observer\MyObserver" />
    </event>
    <event name="cms_block_save_commit_after">
        <observer name="myObserverName2"
instance="MyCompany\MyModule\Observer\AnotherObserver" disabled="true"
shared="true"/>
    </event>
</config>
```

Observer class should be placed in the/Observer directory and implement Magento\Framework\Event\ObserverInterface interface.

```
<?php

class MyObserver implements \Magento\Framework\Event\ObserverInterface
{
    public function execute(\Magento\Framework\Event\Observer $observer)
    {
        $order = $observer->getEvent()->getData('order'); // $observer-
>getEvent()->getOrder();
        // observer code...
    }
}
```

Dispatching events

Events dispatch by Magento\Framework\Event\Manager class that implement Magento\Framework\Event\ManagerInterface interface:

> dispatch($eventName, array $data = []);

Example:

> $this->eventManager->dispatch('cms_page_prepare_save', ['page' => $model, 'request' => $this->getRequest()]);

Links

- Magento DevDocs - Events and observers
- Magento DevDocs - Observers Best Practices

Scheduled jobs

Demonstrate how to configure a scheduled job

Cron groups

A cron group is a logical group that enables you to easily run cron for more than one process at a time. Most Magento modules use the default cron group.

To declare new group and specify settings, create `<module>/etc/cron_groups.xml` file (store view scope):

```
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:module:Magento_Cron:etc/cron_groups.xsd">
    <group id="default">
        <schedule_generate_every>15</schedule_generate_every>
        <schedule_ahead_for>20</schedule_ahead_for>
        <schedule_lifetime>15</schedule_lifetime>
        <history_cleanup_every>10</history_cleanup_every>
        <history_success_lifetime>10080</history_success_lifetime>
        <history_failure_lifetime>10080</history_failure_lifetime>
        <use_separate_process>0</use_separate_process>
    </group>
</config>
```

Existing groups:

- default (no separate process)
- index - mview, targetrule
- catalog_event - catalog_event_status_checker - mark event open/closed
- consumers - consumers_runner if configured to run by cron.

`bin/magento queue:consumers:start`. PID file var/{$consumer}.pid

- staging - staging_apply_version, staging_remove_updates, staging_synchronize_entities_period
- ddg_automation (dotmailer)

crontab.xml

`crontab.xml` file is used to execute an action on schedule. This file always located in the/etc/ folder (not in /etc/{area}/)!

```
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:module:Magento_Cron:etc/crontab.xsd">
    <group id="GROUP_NAME">
        <job name="CRON_JOB_UNIQUE_ID" instance="CLASS" method="METHOD">
            <config_path>some/config/path</config_path>
        </job>
        <job name="indexer_clean_all_changelogs"
instance="Magento\Indexer\Cron\ClearChangelog" method="execute">
            <schedule>0 * * * *</config_path>
        </job>
    </group>
</config>
```

run:

- magento cron:run [–group=””]
- pub/cron.php?[group=] in a web browser, protect with basic auth

Stack trace:

```
\Magento\Cron\Console\Command\CronCommand::execute
\Magento\Framework\App\Cron::launch
`default` event
\Magento\Cron\Observer\ProcessCronQueueObserver
check for specific group
cleanup
generate
check for standalone process
```

\Magento\Cron\Model\Config\Data extends \Magento\Framework\Config\Data

- merges \Magento\Cron\Model\Config\Reader\Db::get from Database

  Sample DB structure:

```
default/crontab/GROUP/jobs/JOB/schedule/cron_expr = '* * * * *'
default/crontab/GROUP/jobs/JOB/schedule/config_path = 'some/config/path' -- try to
read schedule from this config, store view scope
default/crontab/GROUP/jobs/JOB/run/model = 'class::method'
```

DB config usage example: ProductAlert, system.xml, ProductAlert, crontab.xml, backend model: Magento\Cron\Model\Config\Backend\Product\Alert

`bin/magento cron:install` example:

```
#~ MAGENTO START 4d557a63fe1eac8a2827a4eca020c6bb
* * * * * /usr/bin/php7.0 /var/www/m22ee/bin/magento cron:run 2>&1 | grep -v "Ran
jobs by schedule" >> /var/www/m22ee/var/log/magento.cron.log
* * * * * /usr/bin/php7.0 /var/www/m22ee/update/cron.php >>
/var/www/m22ee/var/log/update.cron.log
* * * * * /usr/bin/php7.0 /var/www/m22ee/bin/magento setup:cron:run >>
/var/www/m22ee/var/log/setup.cron.log
#~ MAGENTO END 4d557a63fe1eac8a2827a4eca020c6bb
```

how is separate process ran?
`bin/magento cron:run --group=NAME --bootstrap=standaloneProcessStarted=1`

what is update/cron.php? TODO: find out

what is setup:cron:run? TODO: find out

Links

- Magento DevDocs - Configure a custom cron job and cron group (tutorial)
- Magento DevDocs - Custom cron job and cron group reference

Identify the function and proper use of automatically available events

Model events \Magento\Framework\Model\AbstractModel:

- `model_load_before`, `{$_eventPrefix}_load_before`
- `model_load_after`, `{$_eventPrefix}_load_after`
- `model_save_commit_after`, `{$_eventPrefix}_save_commit_after`
- `model_save_before`, `{$_eventPrefix}_save_before`
- `model_save_after`, `{$_eventPrefix}_save_after`
- `model_delete_before`, `{$_eventPrefix}_delete_before`
- `model_delete_after`, `{$_eventPrefix}_delete_after`
- `model_delete_commit_after`, `{$_eventPrefix}_delete_commit_after`

Flat collection events \Magento\Framework\Model\ResourceModel\Db\Collection\AbstractCollection:

- `core_collection_abstract_load_before`, `{_eventPrefix}_load_before`
- `core_collection_abstract_load_after`, `{_eventPrefix}_load_after`

only if `_eventPrefix` and `_eventObject` defined: `{prefix}_load_before`, `{prefix}_load_after`

EAV collection events \Magento\Eav\Model\Entity\Collection\AbstractCollection:

- eav_collection_abstract_load_before

\Magento\Framework\Model\AbstractModel:

- _eventObject = 'object'
- _eventPrefix = 'core_abstract', e.g. 'catalog_category'
- _getEventData() - 'data_object' + $_eventObject
- `model_load_before` (object, field=null, value=ID)
- `{_eventPrefix}_load_before`, e.g. `catalog_category_load_before` (object, field, value, data_object, category)

Links

- Cyrill Schumacher's Blog - List of all dispatched events