How-to trigger a workflow using the Public API

The ContentService implemented in eZ Publish 5.x has no knowledge of the Legacy workflow engine. This means that all features in the 4.x series that used workflows need a workaround. One of the most common use-cases is pre-publishing approval, for instance of user generated content.

Legacy kernel callbacks

Fortunately, there is a quite simple way to trigger workflows from eZ Publish 5 code: by using a Legacy Kernel Callback. The overall process is described in the Legacy code and features documentation chapter.

Create with the Public API, publish with the Legacy Kernel

Thanks to this, it is possible to create a draft using native eZ Publish 5 code, and get this draft pre-approved using the legacy workflow & collaboration systems, including email notifications.

Legacy workflow configuration

Let's take an approval workflow as an example. It needs to be configured the old way, using the backoffice. Create a workflow, add an Approve event to it, and connect it to the content/publish/before trigger.

Create a draft using the public API

This is done the usual way, as extensively covered in the Public API Cookbook. The main different is that we will NOT call `ContentService::publishVersion()`.

Create content using the Public API

```
$contentType = $contentTypeService->loadContentTypeByIdentifier(
$contentTypeIdentifier );

$contentCreateStruct = $contentService->newContentCreateStruct( $contentType, 'eng-GB'
);
$contentCreateStruct->setField( 'title', $title );
$contentCreateStruct->setField( 'intro', $intro );
$contentCreateStruct->setField( 'body', $body );

$locationCreateStruct = $locationService->newLocationCreateStruct( $parentLocationId
);


// create a draft using the content and location create struct and publish it
$draft = $contentService->createContent( $contentCreateStruct, array(
$locationCreateStruct ) );
```

Publish a Public API draft using the Legacy Kernel

You need to use the object to run a Legacy Kernel Callback (e.g. code fragment). This legacy code will execute the publish operation with the given content id and version number. It will then check the operation result, and return true or false, depending if the operation went all the way through, or if it was interrupted (approval workflow event, for instance). This handling could and should *of course* be enhanced, with error handling

to begin with, but it doesn't change the core principle.

This example will work in any controller that extends `eZ\Publish\EzPublishCoreBundle\Controller`:

```
/**
 * @param $contentId
 * @param $versionId
 * @return bool true if the operation was completed, false if it was interrupted
 */
protected function runLegacyPublish( $contentId, $versionId, $runAsUserId = null )
{
    $legacyKernel = $this->getLegacyKernel();
    return $legacyKernel->runCallback(
        function () use ( $contentId, $versionId, $runAsUserId )
        {

            $db = \eZDB::instance();
            $transactionCounter = $db->transactionCounter();
            if ( $runAsUserId )
            {
                \eZUser::setCurrentlyLoggedInUser( \eZUser::fetch( $runAsUserId ),
$runAsUserId );
            }
            $operationResult = \eZOperationHandler::execute(
                'content', 'publish', array( 'object_id' => $contentId, 'version' =>
$versionId )
            );
            if ( ( array_key_exists( 'status', $operationResult ) &&
$operationResult['status'] != \eZModuleOperationInfo::STATUS_CONTINUE ) )
            {
                // Required by https://jira.ez.no/browse/EZP-20558
                for ( $i = 0, $counter = ( $db->transactionCounter() -
$transactionCounter ); $i < $counter; ++$i )
                {
                    $db->commit();
                }
                return false;
            }
            return true;
        }
    );
}
```

eZ Script context
If you are running the callback in a command line script, you need to add these calls before executing any legacy code:

```
$script = \eZScript::instance( array( 'use-modules' => true ) );
$script->initialize();
```

Any Legacy code can be executed this way. While it is a *backward compatibility* feature, it is considered good practice to work around missing features. It is however not recommended to rely heavily on Legacy features for performances reasons.

Pay attention to the user
Public API code is always executed as a user, by default anonymous. Since workflows are often user dependent (Approval is), it is

important that this code is executed as an editor, or as a user who will trigger the workflow.

Logging in a user on the Public API repository

```
$repository->setCurrentUser( $repository->getUserService()->loadUser( $userId )
);
```

Of course, this isn't required when using Public API code in controllers, since those will use the currently-logged in user.

Legacy code also requires a user

By default, code in `LegacyKernel::runCallback()` calls is executed as anonymous. You'll have to log the executing user manually *on legacy as well*.

Logging in a user with the Legacy Kernel

```
\eZUser::setCurrentlyLoggedInUser( \eZUser::fetch( $runAsUserId ), $runAsUserId
);
```

Data compatibility and workflow collaboration

Content created using the Public API does appear as expected in the dashboard, and the draft can be checked from here. Collaboration options from the backoffice are available, and the object's lifecycle can be managed the way it used to with eZ Publish 4.x.

Test script

The code below is a working Command script example that demonstrates the workaround. Just drop it in a bundle and update the namespace: https://gist.github.com/bdunogier/e05ed564770fa6a51e51