# Server Deployment Package (SDP) for Perforce Helix

## *Workflow Enforcement Triggers*

Perforce Professional Services

Version v2019.3, 2020-08-05

# Table of Contents

# Introduction

This document describes various workflows that can be enforced by the use of Helix Triggers and other related scripts in a configurable manner.

Examples include:

- Creating default Swarm review descriptions according to a template
- Requiring a job to be associated with a changelist either when shelving (for review) or when attempting to submit
- Making most job fields read-only – to enforce good practice when linking with exernal defect trackers, such as JIRA
- Automatically submitting reviewed changelists if a Jenkins job succeeds

Such triggers enhance traceability, ensuring the users associate every changelist in defined projects with a JIRA issue.

The triggers are provided in the `sdp/Unsupported/Samples/triggers` directory of the SDP. They are intended as examples and are Community Supported rather than officially by Perforce Support.

# Chapter 1. Outline Design

The triggers run with a common configuration file (YAML format), which includes things such as messages to be displayed, fields which can be modified in jobs, and the definition of particular projects to which workflow values apply.

## 1.1. Workflow.yaml

This is a well-commented configuration file that is read by all the workflow triggers to control their actions.

For SDP installs, it should be located in `/p4/common/config/Workflow.yaml`

### 1.1.1. Defining Projects and Project Specific Configurables

The file contains an array of project entries, and standard flag entries for those projects.

Projects are read from top to bottom, and the first project that matches a changelist file is the one which is used.

Projects - an array of project entries

Each project expected to have entries for specific fields.

depot_paths: an array of values for which to process other fields

Controlling here allows you to not have to update the trigger table entry every time you wish to adjust. Also, it is possible to have this file be updated by ordinary users and not just superusers. You can even auto-sync this file if required via trigger.

Use quotes if spaces in the path, or if you have an exclude mapping ("-//some/path/...")

```
projects:
- name: ProjectA
  flag_A: y
  flag_B: y
  depot_paths:
  - //depot/inside/...
  - "-//depot/inside/....c"

- name: ProjectB
  depot_paths:
  - //depot/B/...
  - //depot/C/...
```

# Chapter 2. Trigger Details

The triggers inherit from `P4Triggers.py` and `WorkflowTriggers.py` (in some cases).

## 2.1. CheckFixes.py

This trigger validates that when users create or delete a fix (which is a link between a changelist and a job), that the job is in an appropriate state.

For example, it will prevent users from associating a changelist with a job where the JiraStatus field has a value of `Closed`.

### 2.1.1. Configuration values

The following global configurables are read by this trigger:

```
# ----------------------------------------
# For CheckFixes.py
# Note the use of fix_allowed_paths per project.

# fix_state_field: the name of the field in the Perforce job spec
fix_state_field: "JiraStatus"

# link_allowed_states: An array of values for fix_state_field in which links are
allowed to be created
# or deleted between jobs and changelists
fix_allowed_states:
- "Accepted"
- "In Work"

# msg_cant_link_jobs: An array of lines for the message
# For legibility it is good to have the first line blank
msg_cant_link_jobs:
- ""
- "You are not allowed to link changes to or unlink changes from these jobs"
- "because of the state of their associated JIRA issues."
- "Please change the state first in JIRA and try again."
```

Projects are allowed to specify `fix_allowed_paths`, which is an array like `depot_paths` of Perforce wildcards.

This ensures that the trigger is strict when changes are made to a _dev codeline, but that copying up of changes to an _int codeline is permitted, even when JiraStatus=Closed or similar.

 

```
projects:
- name: ProjectA
depot_paths:
- //depot/inside/...
- "-//depot/inside/....c"
fix_allowed_paths:
- //depot/inside/rel/...
```

### 2.1.2. Trigger Entry

This is:

```
check-fixes fix-add //... "python /p4/common/bin/triggers/CheckFixes.py %change%
%client% %jobs%"
```

```
check-fixes fix-delete //... "python /p4/common/bin/triggers/CheckFixes.py --delete
%change% %client% %jobs%"
```

Note the extra parameter –delete for fix-delete.

### 2.1.3. Notes

The deletion of a fix is allowed if the change is pending.

## 2.2. CheckJobEditTrigger.py

This trigger only allows the P4DTG user to make changes to most job fields. It has a configurable list of fields which ordinary users are allowed to change – by default:

- JobStatus
- Fixes

It also does not allow ordinary users to create jobs – instead they are instructed to set the value to do this in JIRA.

All other fields must be changed by editing the JIRA issues and allowing P4DTG replication to propagate those changes into Perforce.

### 2.2.1. Configuration Values

These are hard-coded in the trigger script.

 

```
# The error messages we give to the user
MSG_CANT_CREATE_JOBS = """

You are not allowed to create new jobs!
"""
MSG_CANT_CHANGE_FIELDS = """

You have changed one or more read-only job fields:
"""

# The list of writeable fields that users can change.
# Changes to any other fields are rejected.
# Please validate this against your jobspec.
WRITEABLE_FIELDS = ["Status", "Date"]

# Replicator user - this user is allowed to change fields
REPLICATOR_USER = "p4dtg"
JIRA_USER = "jira"
```

### 2.2.2. Trigger Entry

This is:

```
job_save_check form-in job "python /p4/common/bin/triggers/CheckJobEditTrigger.py
%user% %formfile% "
```

### 2.2.3. Notes

The trigger calculates which read-only fields have been updated and adds that to the error message reported to the user.

## 2.3. CheckSubmitHasReview.py

This trigger is a workflow trigger. When configured to fire, it validates that any attempted change-submit or shelve-submit has a valid Swarm review associated with the change.

### 2.3.1. Configuration Values

```
# For CheckSubmitHasReview.py

# Allow certain users to submit directly without review being required
# An array of user IDs
submit_without_review_users:
- jenkins

# msg_submit_requires_review: An array of lines for the message
# For legibility it is good to have the first line blank
msg_submit_requires_review:
- ""
- "You are not allowed to submit a change without a review."
- "Please shelve the change for Swarm review first."
- "Jenkins will build the change and if successful submit on your behalf."
```

This allows users such as the Jenkins user to submit changes, which might be useful after a successful build.

Project configuration entries – the flag `pre_submit_required_review` must be set to `y` for trigger to fire.

```
projects:
- name: ProjectA
  pre_submit_require_review: y
  depot_paths:
  - //depot/inside/...
  - "-//depot/inside/....c"
```

### 2.3.2. Trigger Entry

```
create_swarm_review change-submit //... "python
/p4/common/bin/triggers/CheckSubmitHasReview.py -c /p4/common/config/Workflow.yaml
%change% "
```

```
create_swarm_review2 shelve-submit //... "python
/p4/common/bin/triggers/CheckSubmitHasReview.py -c /p4/common/config/Workflow.yaml
%change% "
```

There are two entries possible, which allows shelve-submit triggers to also be validated.

### 2.3.3. Notes

Uses Swarm API to search for reviews based on the change id.

# 2.4. CreateSwarmReview.py

This trigger automatically creates a Swarm review for submitted change lists.

It is part of the post-submit build/post-submit review workflow which has been deprecated.

### 2.4.1. Configuration Values

Project configuration entries – the flag `post_submit_create_review` must be set to `y` for trigger to fire. The flag `update_review` controls whether the trigger will add the changelist to any existing review (via shared jobs), or just create a new review for every changelist.

```
projects:
- name: ProjectA
  post_submit_create_review: y
  update_review: y
  depot_paths:
  - //depot/inside/...
  - "-//depot/inside/....c"
```

The global entry:

```
# rev__iew_description: The default review description.
# Can be a single quoted string (with embedded '\n' for newlines, or
# an array of quoted strings.
# The values $jobDescription and $changeDescription if found will be
# expanded as appropriate.
review_description:
- "$jobDescription"
- ""
- "<Please edit these fields as desired>"
- "Review type: Delta/Full"
```

### 2.4.2. Trigger Entry

```
create_swarm_review change-commit //... "python
/p4/common/bin/triggers/CreateSwarmReview.py -c Workflow.yaml %change% "
```

### 2.4.3. Notes

None

# 2.5. SwarmReviewTemplate.py

This trigger uses the same template value as CreateSwarmReview.py to format review text.

 

It can be run both as a shelve-commit trigger (when Swarm user copies the user's shelved changes into its own shadow shelf), and as a form-in trigger.

In both cases, the trigger uses Swarm API to search for existing reviews for the specified change, and to decide whether to update the review description or not. (The trigger does not directly update the Perforce change description that Swarm uses to store the review description).

### 2.5.1. Configuration Values

Project configuration entries – the flag `pre_submit_require_review` must be set to `y` for trigger to fire.

```
projects:
- name: ProjectA
  pre_submit_require_review: y
  update_review: y
  depot_paths:
  - //depot/inside/...
  - "-//depot/inside/....c"
```

As above `review_description`. Also reads `swarm_user` value and exits if the user is not the specified swarm user.

### 2.5.2. Trigger Entry

```
swarm_template_review shelve-commit //... "python
/p4/common/bin/triggers/SwarmReviewTemplate.py -c Workflow.yaml %change% "
```

```
swarm_template_review2 form-save change "python
/p4/common/bin/triggers/SwarmReviewTemplate.py -c Workflow.yaml --user %user%
--formfile %formfile% "
```

### 2.5.3. Notes

Some complexity to allow it to be called as both shelve-commit and form-in trigger.

# Chapter 3. Test Frameworks

It is important that all triggers have a comprehensive test framework so that changes and enhancements can be made with relatively little risk.

All test harnesses are in the tests sub-directory.

## 3.1. Standalone Unit tests

Some tests can directly import the triggers modules and exercise individual classes.

## 3.2. Testing via P4D Triggers table

Some tests are installed in test environment where p4d is run using the rsh hack (also known as inetd mode) – this is similar to how P4DVCS works because the server process has no requirement to be run with a port (can be dangerous on machines with live or important Perforce servers).

The test harness directly creates trigger table entries so that the p4d server will execute the trigger as required.

The disadvantage of this test method is that you can't debug easily across the process boundary, because it is p4d which is directly executing the trigger script, and not the test harness.

## 3.3. Mocking the Swarm API Interface

This is done using the Mock framework (installed standalone for Python 2.x, standard with 3.x)

It ensures that the expected Swarm API calls are made, and is used to direct test code paths.

Without this, it would be necessary to have an installation of Swarm.

See TestCheckSubmitHasReview.py for an example.

 