Ammeon Workflow Engine User Guide

Version 1.3.0

© Ammeon 2015

All rights reserved. No part of this document may be reproduced in any form without the written permission of the copyright owner.

The contents of this document are subject to revision without notice due to continued progress in methodology, design and manufacturing. Ammeon shall have no liability for any error or damage of any kind resulting from the use of this document.

Table of Contents

Ammeon Workflow Engine User Guide

Software Version

The content in this document is applicable to Ammeon Workflow Engine software version 1.3.0.

Scope

This document is the Ammeon Workflow Engine (AWE) User Guide. It is intended for system administrators and any other users that use the AWE to manage their deployments.

Document Purpose

This document describes the features of the AWE and provides detailed information on how to use this tool.

Copyright

© 2015 Ammeon Ltd. All rights reserved.

Assumptions

It is assumed that users have the appropriate access rights and the background technical knowledge to use the Ammeon Workflow Engine.

Typographical Conventions

Convention	Description	Example
User Input and Command Variables	Used for commands that must be entered in a Command Line Interface (CLI) exactly as written. User input is in `Courier New` font.	`cd $HOME`
System Elements	Command and parameter names, program names, path names, URLs, and directory names. Directory paths are shown in `Courier New` font.	The files are located in `/etc/ntp.conf`
Output Information	Text displayed by the system.	Status OK

Code Examples and Command Output Examples	Code examples and command output examples.	The output displayed from a user command is shown in the format below:
GUI Objects	GUI objects, such as menus, menu items, fields, and buttons. The Bold font highlights a GUI object.	On the **File** menu, click **Exit**.
Special Notices	The following notices are used for items that need to be highlighted or require the reader's attention: • Tip • Note • Caution • Warning	This is a tip. A tip suggests alternative methods that may not be obvious and helps users understand the benefits and capabilities of the product. A tip is not essential to the basic understanding of the text. **Note:** This is a note. A note indicates neutral or positive information that emphasises or supplements important points of the main text. It provides information that is essential to the completion of a task. This is a caution. A caution is a type of note that advises users that failure to take or avoid a specified action could result in loss of data or damage to a program or system.

		This is a warning. A warning is a type of note that advises users that failure to take or avoid a specific action could result in physical harm to the user or the hardware.

Introduction

The Ammeon Worflow Engine (AWE) User Guide is intended for system administrators and any other users that use the AWE to manage their deployments.

The document covers the following topics:

- Overview of the Ammeon Workflow Engine
- Running the Ammeon Worflow Engine
- Version plugins
- Troubleshooting the Ammeon Workflow Engine

Overview of the Ammeon Workflow Engine

Introduction

The Ammeon Workflow Engine (AWE) is a collection of services designed to address the complexity of end-to-end deployment supported activities and reduce the cost and effort of these supported activities by removing complexity and manual interaction. For more information on workflow activities, see Supported Activities in a Workflow. AWE requires no knowledge of what is being run by the modules it is executing as long as the modules return a success or a fail in the appropriate manner.

The AWE uses a workflow, a file containing a series of tasks that are grouped into categories (or phases), to define and execute the steps needed to perform the supported activities on the servers in a deployment. This workflow file uses a hosts file, which identifies the hosts used in a particular physical/virtual deployment and includes parameters such as machine names and IP addresses. The AWE uses the workflow file and the hosts file to create a master status file, which is used to save the execution state in the event that the workflow is paused or encounters an error. It then runs the steps required (on remote servers or locally) to perform the required tasks.

AWE Architecture

The following diagram shows the architecture of the AWE:

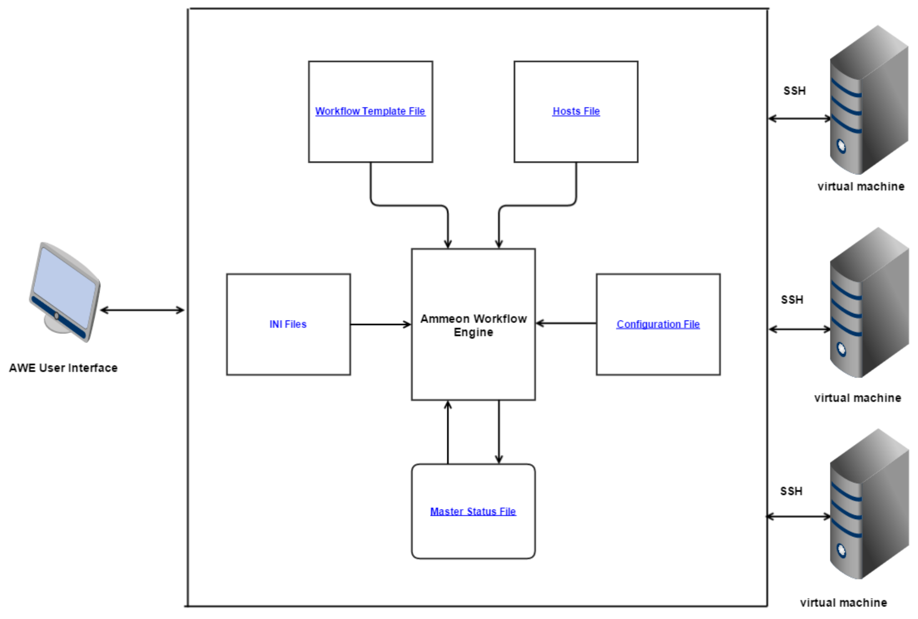

Figure: Ammeon Workflow Engine File Interaction Workflow

Supported Activities in a Workflow

The following describes the supported activities that can be handled by the engine:

- Tasks - may include command or scripts execution.
- Escapes - enables you to exit a running workflow if something unexpected occurs. For more information, see Pausing or Stopping a Workflow.
- Notices - enables you to display a notice to the user on screen before continuing the workflow. For more information, see Notice Element
- Pauses - enables you to set up the workflow so that it stops once a particular tasks executes and then waits for you to restart the workflow. For more information, see Pausing or Stopping a Workflow.
- Adding dynamic pause or escape for a particular deployment. These are added to the master status file instead of the workflow template file as they relate to a single deployment, see Adding or removing deployment specific pause or escape elements.

User Requirements

Domain knowledge is the primary skill needed. The goal for creating the input scripts is to capture any manual upgrade steps and convert them into something scriptable.

Knowledge of a scripting language such as Bash and a programming language such as Python is advantageous, as is an understanding of XML document formats.

> Execution scripts can be Bash or Python. AWE can use your existing Bash-based configuration without the need to re-write in Python.

Supported Platforms

AWE is run on Scientific Linux v6.4, but it is has also been tested, approved and packaged for installation on Solaris 10.

Workflow Engine Phase Overview

The Ammeon Workflow Engine (AWE) uses four distinct phases, which are defined in the workflow, to prepare, execute and review the supported activities performed on a deployment. The phase to be run can be specified using the non-interactive and interactive menus, and the AWE runs all phases up to and including the one specified. For more information on the user interfaces, see User Interfaces.

The AWE can also be configured to prompt the user at the end of each stage on whether they would like to continue.

The four phases are:

1. **Display phase**
   The display phase runs commands or scripts to confirm connectivity to the deployment and to find out information on the servers involved in the deployment, for example the software version that exists and whether the node is a master or slave. All tasks are run, including those that passed in a previous run of the display phase. For more information on this phase, see Running the Display Phase.

2. **Pre-check phase**
   The pre-check phase runs commands or scripts to check that the servers in the deployments are ready to undergo the planned work, as specified in the workflow. It also runs all pre-check tasks, including those that previously passed, and any display phase tasks that have not previously passed. For more information on this phase, see Running the Pre-Check Phase

3. **Execute phase**
   The execute phase runs the commands or scripts required to perform the required operations on the servers in the deployment. It also runs any display, pre-check and execute tasks that have not previously passed. For more information on this phase, see Running the Execute Phase

4. **Post-check phase**
   The post-check phase runs the commands or scripts needed to verify that the operation was completed successfully. It also runs any tasks from the previous phases that had not passed. For more information on this phase, see Running the Post-Check Phase

There is continuous health checking and reporting at each phase and during the execution phase. All AWE activities are logged and can be viewed on the console.

User Interfaces

This topic describes the non-interactive and interactive user interfaces of the Ammeon Workflow Engine.

Non-Interactive Command Line

The following example shows the command syntax used by the engine. Note that `wfeng` is an alias that must be set up in a user's profile.

```
wfeng --hostfile <hosts file> --workfile <workflow file> --phase precheck --output 2
```

Command Line Help

Use the following command to view help associated with the workflow engine command line:

```
wfeng -h
```

An example of the output is shown below:

```
usage: workfloweng.py [-h] [-w WORKFILE] [-H HOSTFILE] [-i INIFILE]
                      [-T TIMEOUT] [-p PHASE] [-s SERVERTYPE] [-n SERVERNAME]
                      [-g TAG] [-e EXCLUDE] [-t TASK] [-F] [-R] [-v]
                      [-o OUTPUT] [-y] [-a] [-d]
Workflow engine
optional arguments:
  -h, --help            show this help message and exit
  -w WORKFILE, --workfile WORKFILE
                        Full path to workflow XML file
  -H HOSTFILE, --hostfile HOSTFILE
                        Full path to hosts XML file
  -i INIFILE, --inifile INIFILE
                        Full path to INI file
  -T TIMEOUT, --timeout TIMEOUT
                        Network connection timeout in seconds
  -p PHASE, --phase PHASE
                        Phase to run: display, precheck, execute, postcheck
  -s SERVERTYPE, --servertype SERVERTYPE
                        Comma separated types of server to run on
  -n SERVERNAME, --servername SERVERNAME
                        Comma separated names of server to run on
  -g TAG, --tag TAG     Only run tasks in this tagged set
  -e EXCLUDE, --exclude EXCLUDE
                        Comma separated names of server not to run on
  -t TASK, --task TASK  Comma separated list of IDs of tasks to run
  -F, --fix             Mark matching tasks as fixed
  -R, --reset           Mark matching tasks as reset
  -v, --version         Display version information
  -o OUTPUT, --output OUTPUT
                        Output Level, 0 - no tags, 1 - error tags, 2 - error
                        and info tags
  -y, --yes             Silently answer yes to all questions except pauses
  -a, --automate        Silently answer yes to all questions including pauses
  -d, --alter_pause_escape
                        Invoke interactive add/delete of dynamic pause and
                        dynamic escape into master status file
```

Advanced Command Line Options

The following table lists the advanced command line options that are available and describes their use:

Hidden Option	Description
`--force`, `-f`	If this option is used, the workflow continues even when a phase has a failed task. If a version is specified in a task command but the version information was not extracted during the display phase and if the `--force` option is used, the task still runs and the user is not asked for the version. This option can also be used with `--reset` option, it is then used to reset a task whose dependants have already completed.
`--allow-multiple`, `-m`	By default only one instance of the Ammeon Workflow Engine can be run on a particular server. However, if the Ammeon Workflow Engine runs are initiated with the `--allow-multiple` option, multiple instances can be run, as long as they are executing with different workflow/host files.
`--nospinner`, `-N`	By default, a spinner is shown on screen to indicate that the task is in progress. When the AWE is run by external tools that have their own spinner, such as Jenkins, it can be desirable to remove this spinner. Use the `--nospinner` option to stop the Ammeon Workflow Engine from displaying its own spinner.

Table: Advanced Command Line Options

Interactive Menu

If the `-p (--phase)`, `-l (--list)`, `-t (--task), -R (--reset)`, or `-d (--alter_pause_escape)` options are not used in a command, the **Workflow Engine** menu is displayed. This allows you to specify the supported activities you want to perform by selecting the relevant option from the list. The top level menu is as follows:

```
                                   WORKFLOW ENGINE
                                  ----------------

 [1] Run display deployment
 [2] Run pre-checks
 [3] Run execute workflow
 [4] Run post-checks
 [5] Run named tasks
 [6] Run tagged set of tasks
 [7] Edit Workflow Engine options


Please select your option or q to quit:
```

You can drill down to sub-menus to determine which servers to run a task or phase on.

Files Used by the Workflow Engine

This topic describes the different files used by the Ammeon Workflow Engine (AWE).

- Template workflow file
- Configuration file
- Hosts file
- INI file
- Master Status File

You cannot edit the workflow, hosts and master status files once the workflow has started.

Template Workflow File

The workflow file is an XML file that defines the set of tasks and the dependencies between tasks required to perform an operation. The tasks for a specific phase are defined with the relevant element containers (display, pre-check, execute and post-check). The same workflow file can be used with different deployments (defined by the hosts file). The following shows an example of an empty workflow file (or template workflow file):

```
<workflow name="Sample solution deployment">
 <display>
 </display>
 <pre-check>
 </pre-check>
 <execute>
 </execute>
 <post-check>
 </post-check>
</workflow>
```

The most common element of the workflow file is a task element, which allows you to specify a command to run remotely or locally. The following example shows a task added to the display phase that runs `display.sh` remotely on each server.

```
<display>
<task cmd="/opt/ammeon/mysql/bin/display.bsh $SERVERIP" id="disp" hosts="*" server="*"
optional="false" continueOnFail="false" runLocal="true"/>
</display>
```

Configuration File

The `wfeng.cfg` configuration file resides in the `cfg` subdirectory of the AWE base directory. This file can be configured to pull out tags from the standard output, which can then be displayed on screen. On the command line, you can define whether these are output to the screen through the use of the `-o`, `--output` option.
The supported values are:

- 0 - no lines are output to the screen
- 1 - only error lines are output to the screen
- 2 - info and error lines are output to the screen

For example, if the engine is executing the pre-check phase and the `--option` parameter is set to `1`, the workflow engine will listen to the log file for the text "PRE_UPGRADE_VERIFICATION_NOTICE:" or "PRE_UPGRADE_VERIFICATION_ERROR" if this has been configured in the `wfeng.cfg` file. If this text is matched, the engine displays the whole line of text received on the screen.

The workflow engine has a single configuration file that specifies the values of configuration parameters in the format `<parametername>=<value>`. The supported parameters are shown below:

Parameter	Description
`DISPLAY_INFO`	A comma-separated list of tags used to identify INFO lines on modules used in the display phase.
`DISPLAY_ERR`	A comma-separated list of tags used to identify ERROR lines on modules used in the display phase.
`PRECHECK_INFO`	A comma-separated list of tags used to identify INFO lines on modules used in the pre-check phase.
`PRECHECK_ERR`	A comma-separated list of tags used to identify ERROR lines on modules used in the pre-check phase.
`EXECUTE_INFO`	A comma-separated list of tags used to identify INFO lines on modules used in the execute phase.
`EXECUTE_ERR`	A comma-separated list of tags used to identify ERROR lines on modules used in the execute phase.
`POSTCHECK_INFO`	A comma-separated list of tags used to identify INFO lines on modules used in the post-check phase.
`POSTCHECK_ERR`	A comma-separated list of tags used to identify ERROR lines on modules used in the post-check phase.
`SWVER`	The parameter within the `DISPLAY_INFO` tag that shows the software version. For example: DISPLAY_INFO: server_sw_version=123.
`OSVER`	The parameter within the `DISPLAY_INFO` tag that shows the O/S version. For example: DISPLAY_INFO: solaris_version=unknown .
`TYPE`	The parameter within the `DISPLAY_INFO` tag that shows the server type.
`UNKNOWN`	This text indicates that a version is unknown.
`KEEPALIVE`	Defines the SSH keepalive period to use, default 0 - np keepalive. (optional)

`SWVERSIONPLUGIN`	Name of a module that defines a custom_version function that is used to determine if the installed software version is >= than that installed. By default no plugin is used, and exact match checking is performed. AWE provides a sequenceverplugin that can be used to compare numeric . separated versions, such as 1.2.0 to 1.3.0 etc.
`OSVERSIONPLUGIN`	Name of a module that defines a custom_version function that is used to determine if the installed O/S version is >= than that installed. By default no plugin is used, and exact match checking is performed. AWE provides a sequenceverplugin that can be used to compare numeric . separated versions, such as 1.2.0 to 1.3.0 etc.
`EXTRA_LOGPARAM`	By default the engine writes to a log file $AWE_HOME/log/<workflow filename>_<hosts filename>_<date>_<time>.log. EXTRA_LOGPARAM can specify additional parameters to go into the log filename before the date. It can take a comma separated list from the values: task, tag, phase, servername, servertype, exclude, reset, list, fix. If the corresponding argument is then given when the engine is invoked then it will form part of the logfilename. For example if EXTRA_LOGPARAM=task,tag, and the engine is invoked with --tag t1, then the filename would become <workflow filename>_<hosts filename>_t1_<date>_<time>.log

Table: Configuration File Parameters

The following is an example workflow engine configuration file:

```
DISPLAY_ERR = DISPLAY_ERROR:
DISPLAY_INFO = DISPLAY_INFO:
PRECHECK_INFO = PRECHECK_INFO:,CHECK_INFO:
PRECHECK_ERR = PRECHECK_ERROR:,CHECK_ERROR:
POSTCHECK_ERR = POSTCHECK_ERROR:,CHECK_ERROR:
POSTCHECK_INFO = POSTCHECK_INFO:,CHECK_INFO:
EXECUTE_ERR = UPGRADE_ERROR:,ROLLBACK_ERROR:,CHECK_ERROR:
EXECUTE_INFO = UPGRADE_INFO:,ROLLBACK_INFO:,CHECK_INFO:
SWVER = SWVER
OSVER = OSVERSION
UNKNOWN = UNKNOWN
SWVERSIONPLUGIN =
OSVERSIONPLUGIN =
EXTRA_LOGPARAM =
```

Hosts File

The hosts file is an XML file that defines the hosts in a particular deployment. An example hosts file is shown below:

```
<hosts>
    <host name="wflowmaster" server="MASTER" ip="10.10.10.130" username="root"/>
    <host name="wflowslave1" server="SLAVE" ip="10.10.10.132" username="root"/>
    <host name="wflowslave2" server="SLAVE" ip="10.10.10.134" username="root"/>
</hosts>
```

The `server` indicates the type of server (for example, slave server) and `username` identifies the user allowed to connect to the remote box. The workflow engine expects there to be passwordless SSH setup between the server that the workflow engine runs on and the remote boxes.

The engine allows a command to use the following parameters, which are then replaced with the appropriate details from the hosts file:

- $SERVERIP - replaced with the IP address of the server associated with the host
- $SERVERNAME - replaced with the name of the server associated with the host
- $SERVERTYPE - replaced with the type of the server associated with the host

INI Files

Commands can take parameters from hard coded values, the hosts file or from INI file. The INI file format uses lines of `<name>=<value>`. The engine has a default INI file, but a custom file can also be created if further variables need to be defined. If a variable exists in the default and custom INI file, the value in the custom file overrides the default file.

An example INI file is shown below:

```
DUMP_DIR=/tmp
# Dump size needed in gigabytes
DUMP_SIZE=2
UPGRADE_DUMP=upgrade55_56.db
DOWNGRADE_DUMP=downgrade56_55.db
DATA_DUMP=TRUE
```

Master Status File

When the engine is run, it looks to see if a master status file for the workflow/hosts file combination exists. If no master status file exists, one is created. Each time the engine runs, it saves the details of what tasks passed, failed or were skipped in the master status file. Using this information, a stopped engine can resume the workflow and run only those tasks that previously failed or were skipped.

The following example shows a master status file with the tasks to be run on the deployment:

```
<workflow name="MySQL upgrade">
<display>
<parallel id="display_parallel">
<task cmd="/opt/ammeon/mysql/bin/display.bsh $SERVERIP" id="disp" hosts="*" server="*"
optional="false" continueOnFail="false" runLocal="true"/>
</parallel>
</display>
<pre-check>
<task cmd="/opt/ammeon/mysql/bin/prepare_wfmgr_sql.bsh $SERVERIP 5.6" id="prepare"
hosts="*" server="*" optional="false" continueOnFail="false" runLocal="true"
dependency="disp" depsinglehost="true"/>
<task cmd="/opt/ammeon/mysql/bin/check_replication_status.bsh"
id="prepare_check_replication" hosts="*" server="SLAVE" optional="false"
continueOnFail="true" dependency="prepare" depsinglehost="true"/>
<task cmd="/opt/ammeon/mysql/bin/prepare_upgrade.bsh $DUMP_DIR $DUMP_SIZE
$UPGRADE_DUMP $DATA_DUMP" id="prepare_slave" hosts="*" server="SLAVE" optional="false"
continueOnFail="false" dependency="prepare_check_replication" swversion="5.6.20"
depsinglehost="true"/>
<task cmd="/opt/ammeon/mysql/bin/prepare_upgrade.bsh $DUMP_DIR $DUMP_SIZE
$UPGRADE_DUMP $DATA_DUMP" id="prepare_master" hosts="*" server="MASTER"
optional="false" continueOnFail="false" dependency="prepare_slave"
swversion="5.6.20"/>
</pre-check>
<execute>
<group id="upgrade_slaves">
<task cmd="/opt/ammeon/mysql/bin/shutdown.bsh $SERVERTYPE" id="shutdown_slave"
hosts="*" server="SLAVE" optional="true" continueOnFail="true"
dependency="prepare_slave" swversion="5.6.20" depsinglehost="true"/>
<task cmd="/opt/ammeon/mysql/bin/upgrade.bsh $SERVERTYPE $DUMP_DIR $UPGRADE_DUMP"
id="upgrade_slave" hosts="*" server="SLAVE" optional="true" continueOnFail="true"
dependency="shutdown_slave" swversion="5.6.20" depsinglehost="true"/>
<task cmd="/opt/ammeon/mysql/bin/upgrade_repair.bsh $SERVERTYPE $DUMP_DIR
$UPGRADE_DUMP" id="upgrade_repair_slave" hosts="*" server="SLAVE" optional="true"
continueOnFail="true" dependency="upgrade_slave" swversion="5.6.20"
depsinglehost="true"/>
<task cmd="/opt/ammeon/mysql/bin/restart_replication.bsh" id="replication_slave"
hosts="*" server="SLAVE" optional="true" continueOnFail="true"
dependency="upgrade_repair_slave" swversion="5.6.20" depsinglehost="true"/>
```

```
<task cmd="/opt/ammeon/mysql/bin/check_replication_status.bsh" id="check_replication"
hosts="*" server="SLAVE" optional="false" continueOnFail="true"
dependency="replication_slave" depsinglehost="true"/>
</group>
<task cmd="/opt/ammeon/mysql/bin/shutdown.bsh $SERVERTYPE" id="shutdown_master"
hosts="*" server="MASTER" optional="false" continueOnFail="false"
dependency="check_replication,prepare_master" swversion="5.6.20"/>
<task cmd="/opt/ammeon/mysql/bin/upgrade.bsh $SERVERTYPE $DUMP_DIR $UPGRADE_DUMP"
id="upgrade_master" hosts="*" server="MASTER" optional="false" continueOnFail="false"
dependency="shutdown_master" swversion="5.6.20"/>
<task cmd="/opt/ammeon/mysql/bin/stop_replication.bsh" id="stop_replication" hosts="*"
server="SLAVE" optional="false" continueOnFail="true" dependency="upgrade_master"/>
<task cmd="/opt/ammeon/mysql/bin/upgrade_repair.bsh $SERVERTYPE $DUMP_DIR
$UPGRADE_DUMP" id="upgrade_repair_master" hosts="*" server="MASTER" optional="false"
continueOnFail="false" dependency="stop_replication" swversion="5.6.20"/>
<task cmd="/opt/ammeon/mysql/bin/restart_replication.bsh" id="restart_replication"
hosts="*" server="SLAVE" optional="true" continueOnFail="true"
dependency="upgrade_repair_master"/>
<task cmd="/opt/ammeon/mysql/bin/alter_table.bsh" id="alter_table" hosts="*"
server="MASTER" optional="true" continueOnFail="true"
dependency="restart_replication"/>
</execute>
<post-check>
<parallel id="check_slaves">
<task cmd="/opt/ammeon/mysql/bin/check_replication_status.bsh"
id="post_check_replication" hosts="*" server="SLAVE" optional="false"
continueOnFail="true" dependency="alter_table"/>
</parallel>
</post-check>
</workflow>
```

The following shows the same master status file when all the tasks have run:

```
<workflowsys name="MySQL upgrade">
  <displaysys>
    <parallelstatus id="display_parallel">
      <sequencestatus id="display_parallel:SLAVE:*:0" host="wflowslave1"
server="SLAVE">
        <taskstatus cmd="/opt/ammeon/mysql/bin/display.bsh $SERVERIP"
host="wflowslave1" server="SLAVE" continueOnFail="true" optional="false"
depsinglehost="false" runLocal="true" params="TYPE=SLAVE,SWVER=5.6.20" id="disp"
status="SUCCESS" actualDur="0:00:00"/>
      </sequencestatus>
      <sequencestatus id="display_parallel:SLAVE:*:0" host="wflowslave2"
server="SLAVE">
        <taskstatus cmd="/opt/ammeon/mysql/bin/display.bsh $SERVERIP"
host="wflowslave2" server="SLAVE" continueOnFail="true" optional="false"
depsinglehost="false" runLocal="true" params="TYPE=SLAVE,SWVER=5.6.20" id="disp"
status="SUCCESS" actualDur="0:00:00"/>
      </sequencestatus>
      <sequencestatus id="display_parallel:MASTER:*:0" host="wflowmaster"
server="MASTER">
        <taskstatus cmd="/opt/ammeon/mysql/bin/display.bsh $SERVERIP"
host="wflowmaster" server="MASTER" continueOnFail="true" optional="false"
depsinglehost="false" runLocal="true" params="TYPE=MASTER,SWVER=5.6.20" id="disp"
status="SUCCESS" actualDur="0:00:00"/>
      </sequencestatus>
    </parallelstatus>
  </displaysys>
  <pre-checksys>
    <taskstatus cmd="/opt/ammeon/mysql/bin/prepare_wfmgr_sql.bsh $SERVERIP 5.6"
host="wflowslave1" server="SLAVE" continueOnFail="true" optional="false"
depsinglehost="true" runLocal="true" id="prepare" status="SUCCESS" dependency="disp"
actualDur="0:00:05"/>
    <taskstatus cmd="/opt/ammeon/mysql/bin/prepare_wfmgr_sql.bsh $SERVERIP 5.6"
host="wflowslave2" server="SLAVE" continueOnFail="true" optional="false"
depsinglehost="true" runLocal="true" id="prepare" status="SUCCESS" dependency="disp"
actualDur="0:00:05"/>
    <taskstatus cmd="/opt/ammeon/mysql/bin/prepare_wfmgr_sql.bsh $SERVERIP 5.6"
host="wflowmaster" server="MASTER" continueOnFail="true" optional="false"
depsinglehost="true" runLocal="true" id="prepare" status="SUCCESS" dependency="disp"
actualDur="0:00:05"/>
    <taskstatus cmd="/opt/ammeon/mysql/bin/check_replication_status.bsh"
host="wflowslave1" server="SLAVE" continueOnFail="true" optional="false"
depsinglehost="true" runLocal="false" id="prepare_check_replication" status="SUCCESS"
dependency="prepare" actualDur="0:00:00"/>
    <taskstatus cmd="/opt/ammeon/mysql/bin/check_replication_status.bsh"
host="wflowslave2" server="SLAVE" continueOnFail="true" optional="false"
depsinglehost="true" runLocal="false" id="prepare_check_replication" status="SUCCESS"
dependency="prepare" actualDur="0:00:00"/>
    <taskstatus cmd="/opt/ammeon/mysql/bin/prepare_upgrade.bsh $DUMP_DIR $DUMP_SIZE
$UPGRADE_DUMP $DATA_DUMP" host="wflowslave1" server="SLAVE" continueOnFail="true"
optional="false" depsinglehost="true" runLocal="false" id="prepare_slave"
status="REACHED_VERSION" dependency="prepare_check_replication" swversion="5.6.20"/>
    <taskstatus cmd="/opt/ammeon/mysql/bin/prepare_upgrade.bsh $DUMP_DIR $DUMP_SIZE
$UPGRADE_DUMP $DATA_DUMP" host="wflowslave2" server="SLAVE" continueOnFail="true"
optional="false" depsinglehost="true" runLocal="false" id="prepare_slave"
status="REACHED_VERSION" dependency="prepare_check_replication" swversion="5.6.20"/>
    <taskstatus cmd="/opt/ammeon/mysql/bin/prepare_upgrade.bsh $DUMP_DIR $DUMP_SIZE
$UPGRADE_DUMP $DATA_DUMP" host="wflowmaster" server="MASTER" continueOnFail="true"
optional="false" depsinglehost="false" runLocal="false" id="prepare_master"
status="REACHED_VERSION" dependency="prepare_slave" swversion="5.6.20"/>
  </pre-checksys>
  <executesys>
    <taskstatus cmd="/opt/ammeon/mysql/bin/shutdown.bsh $SERVERTYPE"
host="wflowslave1" server="SLAVE" continueOnFail="true" optional="true"
depsinglehost="true" runLocal="false" id="shutdown_slave" status="REACHED_VERSION"
dependency="prepare_slave" gid="upgrade_slaves" swversion="5.6.20"/>
    <taskstatus cmd="/opt/ammeon/mysql/bin/upgrade.bsh $SERVERTYPE $DUMP_DIR
$UPGRADE_DUMP" host="wflowslave1" server="SLAVE" continueOnFail="true" optional="true"
depsinglehost="true" runLocal="false" id="upgrade_slave" status="REACHED_VERSION"
dependency="shutdown_slave" gid="upgrade_slaves" swversion="5.6.20"/>
    <taskstatus cmd="/opt/ammeon/mysql/bin/upgrade_repair.bsh $SERVERTYPE $DUMP_DIR
$UPGRADE_DUMP" host="wflowslave1" server="SLAVE" continueOnFail="true" optional="true"
depsinglehost="true" runLocal="false" id="upgrade_repair_slave"
status="REACHED_VERSION" dependency="upgrade_slave" gid="upgrade_slaves"
swversion="5.6.20"/>
    <taskstatus cmd="/opt/ammeon/mysql/bin/restart_replication.bsh" host="wflowslave1"
server="SLAVE" continueOnFail="true" optional="true" depsinglehost="true"
runLocal="false" id="replication_slave" status="REACHED_VERSION"
dependency="upgrade_repair_slave" gid="upgrade_slaves" swversion="5.6.20"/>
    <taskstatus cmd="/opt/ammeon/mysql/bin/check_replication_status.bsh"
host="wflowslave1" server="SLAVE" continueOnFail="true" optional="false"
depsinglehost="true" runLocal="false" id="check_replication" status="SUCCESS"
dependency="replication_slave" actualDur="0:00:00" gid="upgrade_slaves"/>
    <taskstatus cmd="/opt/ammeon/mysql/bin/shutdown.bsh $SERVERTYPE"
host="wflowslave2" server="SLAVE" continueOnFail="true" optional="true"
```

```
depsinglehost="true" runLocal="false" id="shutdown_slave" status="REACHED_VERSION"
dependency="prepare_slave" gid="upgrade_slaves" swversion="5.6.20"/>
    <taskstatus cmd="/opt/ammeon/mysql/bin/upgrade.bsh $SERVERTYPE $DUMP_DIR
$UPGRADE_DUMP" host="wflowslave2" server="SLAVE" continueOnFail="true" optional="true"
depsinglehost="true" runLocal="false" id="upgrade_slave" status="REACHED_VERSION"
dependency="shutdown_slave" gid="upgrade_slaves" swversion="5.6.20"/>
    <taskstatus cmd="/opt/ammeon/mysql/bin/upgrade_repair.bsh $SERVERTYPE $DUMP_DIR
$UPGRADE_DUMP" host="wflowslave2" server="SLAVE" continueOnFail="true" optional="true"
depsinglehost="true" runLocal="false" id="upgrade_repair_slave"
status="REACHED_VERSION" dependency="upgrade_slave" gid="upgrade_slaves"
swversion="5.6.20"/>
    <taskstatus cmd="/opt/ammeon/mysql/bin/restart_replication.bsh" host="wflowslave2"
server="SLAVE" continueOnFail="true" optional="true" depsinglehost="true"
runLocal="false" id="replication_slave" status="REACHED_VERSION"
dependency="upgrade_repair_slave" gid="upgrade_slaves" swversion="5.6.20"/>
    <taskstatus cmd="/opt/ammeon/mysql/bin/check_replication_status.bsh"
host="wflowslave2" server="SLAVE" continueOnFail="true" optional="false"
depsinglehost="true" runLocal="false" id="check_replication" status="SUCCESS"
dependency="replication_slave" actualDur="0:00:00" gid="upgrade_slaves"/>
    <taskstatus cmd="/opt/ammeon/mysql/bin/shutdown.bsh $SERVERTYPE"
host="wflowmaster" server="MASTER" continueOnFail="false" optional="false"
depsinglehost="false" runLocal="false" id="shutdown_master" status="REACHED_VERSION"
dependency="check_replication,prepare_master" swversion="5.6.20"/>
    <taskstatus cmd="/opt/ammeon/mysql/bin/upgrade.bsh $SERVERTYPE $DUMP_DIR
$UPGRADE_DUMP" host="wflowmaster" server="MASTER" continueOnFail="false"
optional="false" depsinglehost="false" runLocal="false" id="upgrade_master"
status="REACHED_VERSION" dependency="shutdown_master" swversion="5.6.20"/>
    <taskstatus cmd="/opt/ammeon/mysql/bin/stop_replication.bsh" host="wflowslave1"
server="SLAVE" continueOnFail="true" optional="false" depsinglehost="false"
runLocal="false" id="stop_replication" status="SUCCESS" dependency="upgrade_master"
actualDur="0:00:00"/>
    <taskstatus cmd="/opt/ammeon/mysql/bin/stop_replication.bsh" host="wflowslave2"
server="SLAVE" continueOnFail="true" optional="false" depsinglehost="false"
runLocal="false" id="stop_replication" status="SUCCESS" dependency="upgrade_master"
actualDur="0:00:01"/>
    <taskstatus cmd="/opt/ammeon/mysql/bin/upgrade_repair.bsh $SERVERTYPE $DUMP_DIR
$UPGRADE_DUMP" host="wflowmaster" server="MASTER" continueOnFail="false"
optional="false" depsinglehost="false" runLocal="false" id="upgrade_repair_master"
status="REACHED_VERSION" dependency="stop_replication" swversion="5.6.20"/>
    <taskstatus cmd="/opt/ammeon/mysql/bin/restart_replication.bsh" host="wflowslave1"
server="SLAVE" continueOnFail="true" optional="true" depsinglehost="false"
runLocal="false" id="restart_replication" status="SUCCESS"
dependency="upgrade_repair_master" actualDur="0:00:00"/>
    <taskstatus cmd="/opt/ammeon/mysql/bin/restart_replication.bsh" host="wflowslave2"
server="SLAVE" continueOnFail="true" optional="true" depsinglehost="false"
runLocal="false" id="restart_replication" status="SUCCESS"
dependency="upgrade_repair_master" actualDur="0:00:00"/>
    <taskstatus cmd="/opt/ammeon/mysql/bin/alter_table.bsh" host="wflowmaster"
server="MASTER" continueOnFail="true" optional="true" depsinglehost="false"
runLocal="false" id="alter_table" status="SUCCESS" dependency="restart_replication"
actualDur="0:00:00"/>
  </executesys>
  <post-checksys>
    <parallelstatus id="check_slaves">
      <sequencestatus id="check_slaves:SLAVE:*:0" host="wflowslave1" server="SLAVE">
        <taskstatus cmd="/opt/ammeon/mysql/bin/check_replication_status.bsh"
host="wflowslave1" server="SLAVE" continueOnFail="true" optional="false"
depsinglehost="false" runLocal="false" id="post_check_replication" status="SUCCESS"
dependency="alter_table" actualDur="0:00:01"/>
```

```
      </sequencestatus>
      <sequencestatus id="check_slaves:SLAVE:*:0" host="wflowslave2" server="SLAVE">
        <taskstatus cmd="/opt/ammeon/mysql/bin/check_replication_status.bsh"
host="wflowslave2" server="SLAVE" continueOnFail="true" optional="false"
depsinglehost="false" runLocal="false" id="post_check_replication" status="SUCCESS"
dependency="alter_table" actualDur="0:00:01"/>
      </sequencestatus>
    </parallelstatus>
  </post-checksys>
</workflowsys>
```

Running the Ammeon Workflow Engine

This section provides the information needed to run the Ammeon Workflow Engine and covers the following topics:

- Creating Workflow Files
- Running the Display Phase
- Running the Pre-Check Phase
- Running the Execute Phase
- Running the Post-Check Phase
- Running Named Workflow Task or Tasks

Creating Workflow Files

The Ammeon Workflow Engine must have a completed workflow file before it can run a phase. A completed workflow file is one that has all the necessary scripts and tasks needed to perform the supported activities on a deployment.

To create a workflow file, complete the following:

1. Write the necessary script files
2. Add tasks to the workflow

Step 1: Write the necessary scripts files

Write the necessary script files to extract out any information required by the workflow. These script files use the values defined in the configuration file. For example, a script designed to run in the display phase might use the following configuration parameters:

- `DISPLAY_INFO`
- `OSVER`
- `SWVER`
- `TYPE`
- `DISPLAY_ERR`

The following examples show the information that could be displayed by the display script for the servers Master1 and slave1:

Master1:

```
Checking OS version
Checking SW version
Checking server type
DISPLAY_INFO: osversion=3.2, swversion=4.3 PATCH 1, servertype=MASTER
```

slave1:

```
Checking OS version
Checking SW version
Checking server type
DISPLAY_INFO: osversion=3.2, swversion=4.3 PATCH 1, servertype=SLAVE
```

Step 2: Add tasks to the workflow

Add in the tasks needed to run the supported activities remotely on each server. The following example shows a task designed to run the display script `display.sh` added to the display section of the workflow XML file:

```
<display>
 <task cmd="/opt/cust/sol1/display.sh" id="display" server="*" hosts="*"
optional="false" continueOnFail="true"/>
</display>
```

Running the Display Phase

This topic describes the display phase of the Ammeon Workflow Engine (AWE). This phase runs commands or scripts to discover information on the servers in the deployment, such as the software version and server type.

Prerequisites

The necessary script files must be written and all necessary tasks added to the `workflow.xml` file. For more information, see Creating Workflow Files.

Run the Display Phase

Use the following command syntax to run the display phase on all servers and display INFO and ERROR lines on the screen::

```
wfeng --hostfile <hosts.xml file> --workfile <workflow.xml file> -i <filename.ini>
--phase display --output 2
```

The following example shows an example task that can be run in the display phase

```
<display>
  <parallel id="display_parallel">
    <task cmd="/opt/ammeon/mysql/bin/display.bsh $SERVERIP" id="disp" hosts="*"
server="*" optional="false" continueOnFail="false" runLocal="true"/>
  </parallel>
</display>
```

When the display phase is run using the configuration described above, the following output is returned:

```


                                WORKFLOW ENGINE
                               -----------------



Options chosen:
workflow: ./workflow_55_56.xml
hosts: ./hosts.xml
timeout: 10
phase: display
server types: *
server name: *
excluded servers: None
task id: None
tag id: None
output level: 2
previous run:  ./workflow_55_56_hosts_status.xml

                              RUNNING PHASE: display
                            -----------------------

PARALLEL_TASK display_parallel: START
TASK disp_wflowslave1: Running /opt/ammeon/mysql/bin/display.bsh 10.30.1.132 on LOCAL
...
STATUS disp_wflowslave1: RUNNING, START: 11:10:05
```

```
TASK disp_wflowslave2: Running /opt/ammeon/mysql/bin/display.bsh 10.30.1.134 on LOCAL
...
STATUS disp_wflowslave2: RUNNING, START: 11:10:05

TASK disp_wflowmaster: Running /opt/ammeon/mysql/bin/display.bsh 10.30.1.130 on LOCAL
...
STATUS disp_wflowmaster: RUNNING, START: 11:10:05

  -->disp_wflowslave1: DISPLAY_INFO: SWVER=5.5.39,TYPE=SLAVE
STATUS disp_wflowslave1: COMPLETED, START: 11:10:05, END: 11:10:05

RESULT disp_wflowslave1: SUCCESS, DURATION: 0:00:00
  -->disp_wflowslave2: DISPLAY_INFO: SWVER=5.5.39,TYPE=SLAVE
STATUS disp_wflowslave2: COMPLETED, START: 11:10:05, END: 11:10:05

RESULT disp_wflowslave2: SUCCESS, DURATION: 0:00:00
  -->disp_wflowmaster: DISPLAY_INFO: SWVER=5.5.39,TYPE=MASTER
STATUS disp_wflowmaster: COMPLETED, START: 11:10:05, END: 11:10:05

RESULT disp_wflowmaster: SUCCESS, DURATION: 0:00:00
PARALLEL_TASK display_parallel: END
SUMMARY: Success: 3 Failed: 0 Skipped: 0
Workflow completed
```

The following example shows the workflow status file when the display phase is run:

```
<displaysys>
    <parallelstatus id="display_parallel">
      <sequencestatus id="display_parallel:SLAVE:*:0" host="wflowslave1"
server="SLAVE">
        <taskstatus cmd="/opt/ammeon/mysql/bin/display.bsh $SERVERIP"
host="wflowslave1" server="SLAVE" continueOnFail="true" optional="false"
depsinglehost="false" runLocal="true" params="TYPE=SLAVE,SWVER=5.5.39" id="disp"
status="SUCCESS" actualDur="0:00:00"/>
      </sequencestatus>
      <sequencestatus id="display_parallel:SLAVE:*:0" host="wflowslave2"
server="SLAVE">
        <taskstatus cmd="/opt/ammeon/mysql/bin/display.bsh $SERVERIP"
host="wflowslave2" server="SLAVE" continueOnFail="true" optional="false"
depsinglehost="false" runLocal="true" params="TYPE=SLAVE,SWVER=5.5.39" id="disp"
status="SUCCESS" actualDur="0:00:00"/>
      </sequencestatus>
      <sequencestatus id="display_parallel:MASTER:*:0" host="wflowmaster"
server="MASTER">
        <taskstatus cmd="/opt/ammeon/mysql/bin/display.bsh $SERVERIP"
host="wflowmaster" server="MASTER" continueOnFail="true" optional="false"
depsinglehost="false" runLocal="true" params="TYPE=MASTER,SWVER=5.5.39" id="disp"
status="SUCCESS" actualDur="0:00:00"/>
      </sequencestatus>
    </parallelstatus>
  </displaysys>
```

Run Time Options

You can select which servers or server types can be run in the workflow using the interactive menu or the following commands:

Option	Description
`servertype`	This is a comma-separated list of server types to run on . Workflow only runs those tasks on servers that are defined in the hosts XML file with matching SERVER parameter.
`servername`	This is a comma-separated list of server names to run on, as defined by the name value in the hosts XML file.
`exclude`	This is a comma-separated list of servers NOT to run on. Can be used with or without the `servertype` parameter.

Table: Display Phase Run Time Options

To run tasks only on servers of type SLAVE, enter the following command:

```
wfeng --hostfile <hostfile> --workfile <workfile> --servertype SLAVE --phase display
```

To run tasks only on slave1 server, enter the following command:

```
wfeng --hostfile <hostfile> --workfile <workfile> --servername slave1 --phase display
```

To run tasks on all servers except slave2, enter the following command:

```
wfeng --hostfile <hostfile> --workfile <workfile> --exclude slave2 --phase display
```

Running the Pre-Check Phase

This topic describes the pre-check phase of the Ammeon Workflow Engine (AWE).

The pre-check phase runs commands or scripts to confirm connectivity to the deployment. To run the pre-check phase, complete the following:

Prerequisites

The necessary script files must be written and all necessary tasks added to the `workflow.xml` file. For more information, see Creating Workflow Files.

Run the pre-check phase

Use the following command syntax to run the pre-check phase on all servers and display INFO and ERROR lines on screen:

```
wfeng --hostfile <hosts file> --workfile <workflow file> -i <filename.ini> --phase
precheck --output 2
```

The following example shows a sample set of tasks that can be run in the pre-check phase

```
<pre-check>
  <task cmd="/opt/ammeon/mysql/bin/prepare_wfmgr_sql.bsh $SERVERIP 5.6" id="prepare"
hosts="*" server="*" optional="false" continueOnFail="false" runLocal="true"
dependency="disp" depsinglehost="true"/>
  <task cmd="/opt/ammeon/mysql/bin/check_replication_status.bsh"
id="prepare_check_replication" hosts="*" server="SLAVE" optional="false"
continueOnFail="true" dependency="prepare" depsinglehost="true"/>
  <task cmd="/opt/ammeon/mysql/bin/prepare_upgrade.bsh $DUMP_DIR $DUMP_SIZE
$UPGRADE_DUMP $DATA_DUMP" id="prepare_slave" hosts="*" server="SLAVE" optional="false"
continueOnFail="false" dependency="prepare_check_replication" swversion="5.6.20"
depsinglehost="true"/>
  <task cmd="/opt/ammeon/mysql/bin/prepare_upgrade.bsh $DUMP_DIR $DUMP_SIZE
$UPGRADE_DUMP $DATA_DUMP" id="prepare_master" hosts="*" server="MASTER"
optional="false" continueOnFail="false" dependency="prepare_slave"
swversion="5.6.20"/>
</pre-check>
```

When the pre-check phase is run using the configuration described above, the following output is returned:

```

                                WORKFLOW ENGINE
                               -----------------



Options chosen:
workflow: ./workflow_55_56.xml
    hosts: ./hosts.xml
    timeout: 10
    phase: precheck
    server types: *
    server name: *
    excluded servers: None
    task id: None
    tag id: None
    output level: 2
    previous run: ./workflow_55_56_hosts_status.xml


                               RUNNING PHASE: pre-check
                              ------------------------
TASK prepare_wflowslave1: Running /opt/ammeon/mysql/bin/prepare_wfmgr_sql.bsh
10.30.1.132 5.6 on LOCAL ...
STATUS prepare_wflowslave1: RUNNING, START: 11:12:45
  -->prepare_wflowslave1: PRECHECK_INFO: Successfully created directory structure on
10.30.1.132 [OK]
  -->prepare_wflowslave1: PRECHECK_INFO: Successfully copied scripts to remote server
[OK]
  -->prepare_wflowslave1: PRECHECK_INFO: Successfully copied RPMs to remote server
[OK]
STATUS prepare_wflowslave1: COMPLETED, START: 11:12:45, END: 11:12:51
RESULT prepare_wflowslave1: SUCCESS, DURATION: 0:00:06
TASK prepare_wflowslave2: Running /opt/ammeon/mysql/bin/prepare_wfmgr_sql.bsh
10.30.1.134 5.6 on LOCAL ...
STATUS prepare_wflowslave2: RUNNING, START: 11:12:51
  -->prepare_wflowslave2: PRECHECK_INFO: Successfully created directory structure on
10.30.1.134 [OK]
  -->prepare_wflowslave2: PRECHECK_INFO: Successfully copied scripts to remote server
[OK]
  -->prepare_wflowslave2: PRECHECK_INFO: Successfully copied RPMs to remote server
[OK]
STATUS prepare_wflowslave2: COMPLETED, START: 11:12:51, END: 11:12:57
RESULT prepare_wflowslave2: SUCCESS, DURATION: 0:00:05
TASK prepare_wflowmaster: Running /opt/ammeon/mysql/bin/prepare_wfmgr_sql.bsh
10.30.1.130 5.6 on LOCAL ...
STATUS prepare_wflowmaster: RUNNING, START: 11:12:57
  -->prepare_wflowmaster: PRECHECK_INFO: Successfully created directory structure on
10.30.1.130 [OK]
  -->prepare_wflowmaster: PRECHECK_INFO: Successfully copied scripts to remote server
[OK]
  -->prepare_wflowmaster: PRECHECK_INFO: Successfully copied RPMs to remote server
[OK]
...
STATUS prepare_slave_wflowslave2: COMPLETED, START: 11:13:17, END: 11:13:25
RESULT prepare_slave_wflowslave2: SUCCESS, DURATION: 0:00:08
TASK prepare_master_wflowmaster: Running /opt/ammeon/mysql/bin/prepare_upgrade.bsh
/tmp 2 upgrade55_56.db TRUE on 10.30.1.130 ...
STATUS prepare_master_wflowmaster: RUNNING, START: 11:13:26
  -->prepare_master_wflowmaster: PRECHECK_INFO: Renamed existing dump to
/tmp/upgrade55_56.db.2015-05-28_11:13:35 [OK]
  -->prepare_master_wflowmaster: PRECHECK_INFO: Passed checks for size of database
dump location [OK]
  -->prepare_master_wflowmaster: PRECHECK_INFO: Dumped database successfully to
/tmp/upgrade55_56.db [OK]
  -->prepare_master_wflowmaster: PRECHECK_INFO: Stopped mysql successfully [OK]
  -->prepare_master_wflowmaster: PRECHECK_INFO: Successfully dumped /var/lib/mysql to
/tmp/upgrade55_56.db.tar [OK]
  -->prepare_master_wflowmaster: PRECHECK_INFO: Successfully saved /etc/my.cnf to /tmp
[OK]
  -->prepare_master_wflowmaster: PRECHECK_INFO: Started mysql successfully [OK]
STATUS prepare_master_wflowmaster: COMPLETED, START: 11:13:26, END: 11:13:49
```

```
RESULT prepare_master_wflowmaster: SUCCESS, DURATION: 0:00:23
SUMMARY: Success: 8 Failed: 0 Skipped: 0
Workflow completed
```

The following example shows the workflow status file when the pre-check phase is run:

```
<pre-checksys>
    <taskstatus cmd="/opt/ammeon/mysql/bin/prepare_wfmgr_sql.bsh $SERVERIP 5.6"
host="wflowslave1" server="SLAVE" continueOnFail="true" optional="false"
depsinglehost="true" runLocal="true" id="prepare" status="SUCCESS" dependency="disp"
actualDur="0:00:06"/>
    <taskstatus cmd="/opt/ammeon/mysql/bin/prepare_wfmgr_sql.bsh $SERVERIP 5.6"
host="wflowslave2" server="SLAVE" continueOnFail="true" optional="false"
depsinglehost="true" runLocal="true" id="prepare" status="SUCCESS" dependency="disp"
actualDur="0:00:05"/>
    <taskstatus cmd="/opt/ammeon/mysql/bin/prepare_wfmgr_sql.bsh $SERVERIP 5.6"
host="wflowmaster" server="MASTER" continueOnFail="true" optional="false"
depsinglehost="true" runLocal="true" id="prepare" status="SUCCESS" dependency="disp"
actualDur="0:00:05"/>
    <taskstatus cmd="/opt/ammeon/mysql/bin/check_replication_status.bsh"
host="wflowslave1" server="SLAVE" continueOnFail="true" optional="false"
depsinglehost="true" runLocal="false" id="prepare_check_replication" status="SUCCESS"
dependency="prepare" actualDur="0:00:00"/>
    <taskstatus cmd="/opt/ammeon/mysql/bin/check_replication_status.bsh"
host="wflowslave2" server="SLAVE" continueOnFail="true" optional="false"
depsinglehost="true" runLocal="false" id="prepare_check_replication" status="SUCCESS"
dependency="prepare" actualDur="0:00:00"/>
    <taskstatus cmd="/opt/ammeon/mysql/bin/prepare_upgrade.bsh $DUMP_DIR $DUMP_SIZE
$UPGRADE_DUMP $DATA_DUMP" host="wflowslave1" server="SLAVE" continueOnFail="true"
optional="false" depsinglehost="true" runLocal="false" id="prepare_slave"
status="SUCCESS" dependency="prepare_check_replication" actualDur="0:00:12"
swversion="5.6.20"/>
    <taskstatus cmd="/opt/ammeon/mysql/bin/prepare_upgrade.bsh $DUMP_DIR $DUMP_SIZE
$UPGRADE_DUMP $DATA_DUMP" host="wflowslave2" server="SLAVE" continueOnFail="true"
optional="false" depsinglehost="true" runLocal="false" id="prepare_slave"
status="SUCCESS" dependency="prepare_check_replication" actualDur="0:00:08"
swversion="5.6.20"/>
    <taskstatus cmd="/opt/ammeon/mysql/bin/prepare_upgrade.bsh $DUMP_DIR $DUMP_SIZE
$UPGRADE_DUMP $DATA_DUMP" host="wflowmaster" server="MASTER" continueOnFail="true"
optional="false" depsinglehost="false" runLocal="false" id="prepare_master"
status="SUCCESS" dependency="prepare_slave" actualDur="0:00:23" swversion="5.6.20"/>
  </pre-checksys>
```

Run Time Options

You can select which servers or server types can be run in the workflow using the interactive menu or the following commands:

Option	Description
`servertype`	This is a comma-separated list of server types to run on . Workflow only runs those tasks on servers that are defined in the hosts XML file with matching SERVER parameter.
`servername`	This is a comma-separated list of server names to run on, as defined by the name value in the hosts XML file.
`exclude`	This is a comma-separated list of servers NOT to run on. Can be used with or without the `servertype` parameter.

Table: Pre-Check Phase Run Time Options

To run tasks only on servers of type SLAVE, enter the following command:

```
wfeng --hostfile <hostfile> --workfile <workfile> --servertype SLAVE --phase pre-check
```

To run tasks only on slave1 server, enter the following command:

```
wfeng --hostfile <hostfile> --workfile <workfile> --servername slave1 --phase
pre-check
```

To run tasks on all servers except slave2, enter the following command:

```
wfeng --hostfile <hostfile> --workfile <workfile> --exclude slave2 --phase pre-check
```

Running the Execute Phase

This topic describes the execute phase of the Ammeon Workflow Engine (AWE).

The execute phase runs the tasks needed to perform the supported activities required on the servers in a deployment. This phase can run tasks sequentially or in parallel, and can also run sub-sets of tasks. For more information on these options, see the following:

- Executing Tasks in Sequential Order
- Executing Tasks in Parallel
- Executing a Subset of Tasks
- Running a Group of Tasks

To run the execute phase, complete the following:

Prerequisites

The necessary script files must be written and all necessary tasks added to the `workflow.xml` file. For more information, see Creating Workflow Files.

Run the Execute Phase

Use the following command syntax to run the execute phase on all servers and display INFO and ERROR lines on screen:

```
wfeng --hostfile <hosts file> --workfile <workflow file> -i <filename.ini> --phase
execute --output 2
```

The execute phase performs the following actions:

1. Runs any display tasks that have not previously passed.
2. Runs any pre-check tasks that have not previously passed.
3. If this phase was run previously, executes any tasks that have not previously passed.

In the execute phase the option `continueOnFail` is usually set to `false` (`continueOnFail="false"`) so that if a tasks fails, the workflow engine stops and exits. It is necessary to rectify the cause of this failure before retrying to run the execute phase. For more information on troubleshooting, see Troubleshooting Information.

The following example shows an example set of tasks that can be run in the execute phase

```
<execute>
  <group id="upgrade_slaves">
    <task cmd="/opt/ammeon/mysql/bin/shutdown.bsh $SERVERTYPE" id="shutdown_slave"
hosts="*" server="SLAVE" optional="true" continueOnFail="true"
dependency="prepare_slave" swversion="5.6.20" depsinglehost="true"/>
    <task cmd="/opt/ammeon/mysql/bin/upgrade.bsh $SERVERTYPE $DUMP_DIR $UPGRADE_DUMP"
id="upgrade_slave" hosts="*" server="SLAVE" optional="true" continueOnFail="true"
dependency="shutdown_slave" swversion="5.6.20" depsinglehost="true"/>
  </group>
...
  <task cmd="/opt/ammeon/mysql/bin/alter_table.bsh" id="alter_table" hosts="*"
```

```
server="MASTER" optional="true" continueOnFail="true"
dependency="restart_replication"/>
</execute>
```

When the execute phase is run using the configuration described above, the following output is returned:

```

                                 WORKFLOW ENGINE
                                -----------------

Options chosen:
 workflow: ./workflow_55_56.xml
    hosts: ./hosts.xml
    timeout: 10
    phase: execute
    server types: *
    server name: *
    excluded servers: None
    task id: None
    tag id: None
    output level: 2
    yes: True
    previous run: ./workflow_55_56_hosts_status.xml

                               RUNNING PHASE: execute
                              ------------------------
TASK shutdown_slave_wflowslave1: Running /opt/ammeon/mysql/bin/shutdown.bsh SLAVE on
10.30.1.132 ...
STATUS shutdown_slave_wflowslave1: RUNNING, START: 11:14:46
  -->shutdown_slave_wflowslave1: UPGRADE_INFO: Stopped slave replication [OK]
  -->shutdown_slave_wflowslave1: UPGRADE_INFO: Flushed tables [OK]
  -->shutdown_slave_wflowslave1: UPGRADE_INFO: MySQL stopped successfully [OK]
STATUS shutdown_slave_wflowslave1: COMPLETED, START: 11:14:46, END: 11:14:49
RESULT shutdown_slave_wflowslave1: SUCCESS, DURATION: 0:00:02
TASK upgrade_slave_wflowslave1: Running /opt/ammeon/mysql/bin/upgrade.bsh SLAVE /tmp
upgrade55_56.db on 10.30.1.132 ...
STATUS upgrade_slave_wflowslave1: RUNNING, START: 11:14:49
  -->upgrade_slave_wflowslave1: UPGRADE_INFO: Saved my.cnf [OK]
  -->upgrade_slave_wflowslave1: UPGRADE_INFO: Successfully upgraded MySQL shared
compatabilities [OK]
  -->upgrade_slave_wflowslave1: UPGRADE_INFO: Successfully removed old MySQL client
[OK]
  -->upgrade_slave_wflowslave1: UPGRADE_INFO: Successfully removed old MySQL server
[OK]
  -->upgrade_slave_wflowslave1: UPGRADE_INFO: Successfully installed new MySQL server
[OK]
  -->upgrade_slave_wflowslave1: UPGRADE_INFO: Successfully installed MySQL client [OK]
  -->upgrade_slave_wflowslave1: UPGRADE_INFO: Restored my.cnf [OK]
...
TASK alter_table_wflowmaster: Running /opt/ammeon/mysql/bin/alter_table.bsh on
10.30.1.130 ...
STATUS alter_table_wflowmaster: RUNNING, START: 11:20:02
  -->alter_table_wflowmaster: UPGRADE_INFO: Updated DATETIME columns successfully [OK]
STATUS alter_table_wflowmaster: COMPLETED, START: 11:20:02, END: 11:20:04
RESULT alter_table_wflowmaster: SUCCESS, DURATION: 0:00:01
SUMMARY: Success: 18 Failed: 0 Skipped: 0
Workflow completed
```

The following example shows the workflow status file when the execute phase is run:

```
<executesys>
    <taskstatus cmd="/opt/ammeon/mysql/bin/shutdown.bsh $SERVERTYPE"
host="wflowslave1" server="SLAVE" continueOnFail="true" optional="true"
depsinglehost="true" runLocal="false" id="shutdown_slave" status="SUCCESS"
dependency="prepare_slave" actualDur="0:00:02" gid="upgrade_slaves"
swversion="5.6.20"/>
    <taskstatus cmd="/opt/ammeon/mysql/bin/upgrade.bsh $SERVERTYPE $DUMP_DIR
$UPGRADE_DUMP" host="wflowslave1" server="SLAVE" continueOnFail="true" optional="true"
depsinglehost="true" runLocal="false" id="upgrade_slave" status="SUCCESS"
dependency="shutdown_slave" actualDur="0:01:43" gid="upgrade_slaves"
swversion="5.6.20"/>
    <taskstatus cmd="/opt/ammeon/mysql/bin/upgrade_repair.bsh $SERVERTYPE $DUMP_DIR
$UPGRADE_DUMP" host="wflowslave1" server="SLAVE" continueOnFail="true" optional="true"
depsinglehost="true" runLocal="false" id="upgrade_repair_slave" status="SUCCESS"
dependency="upgrade_slave" actualDur="0:00:25" gid="upgrade_slaves"
swversion="5.6.20"/>
    <taskstatus cmd="/opt/ammeon/mysql/bin/restart_replication.bsh" host="wflowslave1"
server="SLAVE" continueOnFail="true" optional="true" depsinglehost="true"
runLocal="false" id="replication_slave" status="SUCCESS"
dependency="upgrade_repair_slave" actualDur="0:00:01" gid="upgrade_slaves"
swversion="5.6.20"/>
    <taskstatus cmd="/opt/ammeon/mysql/bin/check_replication_status.bsh"
host="wflowslave1" server="SLAVE" continueOnFail="true" optional="false"
depsinglehost="true" runLocal="false" id="check_replication" status="SUCCESS"
dependency="replication_slave" actualDur="0:00:00" gid="upgrade_slaves"/>
    <taskstatus cmd="/opt/ammeon/mysql/bin/shutdown.bsh $SERVERTYPE"
host="wflowslave2" server="SLAVE" continueOnFail="true" optional="true"
depsinglehost="true" runLocal="false" id="shutdown_slave" status="SUCCESS"
dependency="prepare_slave" actualDur="0:00:02" gid="upgrade_slaves"
swversion="5.6.20"/>
...
    <taskstatus cmd="/opt/ammeon/mysql/bin/upgrade_repair.bsh $SERVERTYPE $DUMP_DIR
$UPGRADE_DUMP" host="wflowmaster" server="MASTER" continueOnFail="false"
optional="false" depsinglehost="false" runLocal="false" id="upgrade_repair_master"
status="SUCCESS" dependency="stop_replication" actualDur="0:00:16"
swversion="5.6.20"/>
    <taskstatus cmd="/opt/ammeon/mysql/bin/restart_replication.bsh" host="wflowslave1"
server="SLAVE" continueOnFail="true" optional="true" depsinglehost="false"
runLocal="false" id="restart_replication" status="SUCCESS"
dependency="upgrade_repair_master" actualDur="0:00:00"/>
    <taskstatus cmd="/opt/ammeon/mysql/bin/restart_replication.bsh" host="wflowslave2"
server="SLAVE" continueOnFail="true" optional="true" depsinglehost="false"
runLocal="false" id="restart_replication" status="SUCCESS"
dependency="upgrade_repair_master" actualDur="0:00:00"/>
    <taskstatus cmd="/opt/ammeon/mysql/bin/alter_table.bsh" host="wflowmaster"
server="MASTER" continueOnFail="true" optional="true" depsinglehost="false"
runLocal="false" id="alter_table" status="SUCCESS" dependency="restart_replication"
actualDur="0:00:01"/>
  </executesys>
```

Run Time Options

Before running each execute task, the engine asks if you want to run or skip the task. The following attributes can be used to define how the engine reacts to this prompt:

Option	Description
`-y, --yes`	This attribute silently answers yes to all questions, except pauses. **Note:** This option can only be used in the execute phase.
`-a, --automate`	This attribute silently answer yes to all questions, including pauses. **Note:** This option can only be used in the execute phase.

Table: Running or Skipping Tasks

You can select which servers or server types can be run in the workflow using the interactive menu or the following commands:

Option	Description
`servertype`	This is a comma-separated list of server types to run on . Workflow only runs those tasks on servers that are defined in the hosts XML file with matching SERVER parameter.
`servername`	This is a comma-separated list of server names to run on, as defined by the name value in the hosts XML file.
`exclude`	This is a comma-separated list of servers NOT to run on. Can be used with or without the `servertype` parameter.

Table: Choosing Servers or Server Types to Run in a Workflow

To run tasks only on servers of type SLAVE, enter the following command:

```
wfeng --hostfile <hostfile> --workfile <workfile> --servertype SLAVE --phase execute
```

To run tasks only on slave1 server, enter the following command:

```
wfeng --hostfile <hostfile> --workfile <workfile> --servername slave1 --phase execute
```

To run tasks on all servers except slave2, enter the following command:

```
wfeng --hostfile <hostfile> --workfile <workfile> --exclude slave2 --phase execute
```

Executing Tasks in Sequential Order

This topic describes how tasks can be set to run sequentially during the execute phase.

For example, if you have two tasks (task_1 and task_2) to be performed on two servers (master_server and slave_server), you can perform these as described in the scenarios below:

- **Scenario A**: task_1 on master_server, task_2 on master_server, task_1 on slave_server and task_2 on slave_server
- **Scenario B**: task_1 on master_server, task_1 on slave_server, task_2 on master_server and task_2 on slave_server.

The scenarios described below also work when multiple servers of the same type are used.

Example Workflow File for Scenario A

```
<execute>
  <task cmd="/ammeon/deployment1/bin/task_1.bsh $MEDIA $SERVERIP" id="dep1_task_1"
hosts="*" server="master_server" optional="true" continueOnFail="false"
dependency="disp_server_1" runLocal="true" />
  <task cmd="/usr/local/bin/sudo /var/tmp/oam_utils/task_2.bsh $SERVERTYPE $_MEDIA"
id="dep1_ops" hosts="*" server="master_server" optional="true" continueOnFail="false"
dependency="dep1_task_1" />
  <task cmd="/ammeon/deployment1/bin/task_1.bsh $MEDIA $SERVERIP" id="dep2_task_1"
hosts="*" server="slave_server" optional="true" continueOnFail="false"
dependency="disp_server_2" runLocal="true"/>
  <task cmd="/usr/local/bin/sudo /var/tmp/oam_utils/task_2.bsh $SERVERTYPE $SERVER_IP
$MEDIA" id="dep2_ops" hosts="*" server="slave_server" optional="true"
continueOnFail="false" dependency="dep2_task_1"/>
</execute>
```

Example Workflow file for Scenario B

The main difference between this scenario and Scenario A is that the workflow is set to run on all hosts for all server types (hosts="*" and server="*"). In the previous example, the workflow specified that the tasks should run on servers "master_server" and "slave_server".

```
<display>
  <task cmd="/ammeon/deployment1/bin/orc_display.bsh $SERVERTYPE $SERVERIP" id="disp"
hosts="*" server="*" optional="true" continueOnFail="true" runLocal="true"/>
</display>

<execute>
  <task cmd="/ammeon/deployment1/bin/task_1.bsh $MEDIA $SERVERIP" id="dep1_task_1"
hosts="*" server="*" optional="true" continueOnFail="false" dependency="disp"
runLocal="true" depsinglehost="true"/>
  <task cmd="/usr/local/bin/sudo /var/tmp/oam_utils/task_2.bsh $SERVERTYPE $SERVER_IP
$MEDIA" id="dep1_task_2" hosts="*" server="*" optional="true" continueOnFail="false"
dependency="dep1_task_1" depsinglehost="true"/>
</execute>
```

Executing Tasks in Parallel

This topic describes how to run tasks in parallel, thereby reducing the overall time to complete the planned updates to a deployment.

The `parallel` element (`<parallel>`) is used to make a single task, or a set of tasks, run in parallel across multiple servers. The `parallel` element has a single attribute ID, which is a unique identifier to identify the task. Include the tasks that you want to run in parallel within the `parallel` element. The `parallel` element is supported in all phases of the workflow.

The following example runs the `display.sh` task across all there severs in the deployment:

```
<parallel id="display_parallel">
  <task cmd="/opt/cust/sol1/display.sh" id="display" server="*" hosts="*"
optional="false" continueOnFail="true"/>
</parallel>
```

The Ammeon Workflow Engine creates a separate process for each set of parallel tasks and uses the dependencies in the workflow to determine the number of parallel processes to create. Within a parallel workflow, if task B is dependant on task A, and both tasks are in the same parallel set, the dependency is treated as if the parameter `depsinglehost` is set to `true` (because task B is only dependant on another task running on the same server).

For example, the following XML file creates three separate processes (for master1, slave1 and slave2). Each process runs `stage_one.sh`, and if that succeeds, runs `stage_two.sh` on the server.

```
<parallel id="execute_parallel">
  <task cmd="/opt/cust/sol1/stage_one.sh" id="stg1" server="*" hosts="*"
optional="false" continueOnFail="true"/>
  <task cmd="/opt/cust/sol1/stage_two.sh" id="stg2" server="*" hosts="*"
optional="false" continueOnFail="true" dependency="stg1"/>
</parallel>
```

Executing a Subset of Tasks

This topic describes the use of tags to execute a subset of tasks rather than all the tasks contained within a complete phase.

The start of a subset of tasks is defined with a `<start-tag>` element and ends with an `<end-tag>` element. Both `start-tag` and `end-tag` have a name attribute that defines the tagged subset, as shown in the following example:

```
<execute>
...
<start-tag name="USING_TAGS/>
<task cmd="/ammeon/deployment1/bin/task_1.bsh $MEDIA $SERVERIP" id="dep1_task_1"
hosts="*" server="server_1" optional="true" continueOnFail="false"
dependency="disp_server_1" runLocal="true" />
<task cmd="/usr/local/bin/sudo /var/tmp/oam_utils/task_2.bsh $SERVERTYPE $_MEDIA"
id="dep1_ops" hosts="*" server="server_1" optional="true" continueOnFail="false"
dependency="dep1_task_1" />
<task cmd="/ammeon/deployment1/bin/task_1.bsh $MEDIA $SERVERIP" id="dep2_task_1"
hosts="*" server="server_2" optional="true" continueOnFail="false"
dependency="disp_server_2" runLocal="true"/>
<task cmd="/usr/local/bin/sudo /var/tmp/oam_utils/task_2.bsh $SERVERTYPE $SERVER_IP
$MEDIA" id="dep2_ops" hosts="*" server="server_2" optional="true"
continueOnFail="false" dependency="dep2_task_1"/>
<end-tag name="USING_TAGS"/>
...
</execute>
```

The tagged subset can be invoked by using the `--tag` or `-g` options on the command line, as shown below:

```
wfeng -w <workflow> -H <hosts> -i <params> -p execute -o 2 -y --tag USING_TAGS
```

The tagged subset can also be invoked by selecting option **6** on the interactive menu, as shown below:

```

                               WORKFLOW ENGINE
                               ---------------

 [1] Run display deployment
 [2] Run pre-checks
 [3] Run execute workflow
 [4] Run post-checks
 [5] Run single workflow task
 [6] Run tagged set of tasks
 [7] Edit Workflow Engine options
Please select your option or q to quit:
6
```

Running a Group of Tasks

By default, the Ammeon Workflow Engine runs a task on all servers before moving on to the next task. However, it is sometimes desirable to run a series of tasks on one server first before moving on the next server. This can be accomplished using the `<group>` element. This is illustrated in the following example.

If a workflow has the following tasks, the `shutdown.sh` task is run on all slaves before the upgrade task is started:

```
<task cmd="/opt/cust/sol1/shutdown.sh" id="shutdown" hosts="*" server="SLAVE"/>
<task cmd="/opt/cust/sol1/upgrade.sh" id="upgrade" hosts="*" server="SLAVE"/>
<task cmd="/opt/cust/sol1/start.sh" id="start" hosts="*" server="SLAVE"/>
```

However, if the `<group>` element is used, as shown below, the `shutdown.sh` task is run on first slave, followed by the `upgrade.sh` task and then the `start.sh` task. Once these three tasks have completed on the first slave, the workflow moves to the second slave.

```
<group id=”upgrade_slave”>
<task cmd="/opt/cust/sol1/shutdown.sh" id="shutdown" hosts="*" server="SLAVE"/>
<task cmd="/opt/cust/sol1/upgrade.sh" id="upgrade" hosts="*" server="SLAVE"/>
<task cmd="/opt/cust/sol1/start.sh" id="start" hosts="*" server="SLAVE"/>
</group>
```

Running the Post-Check Phase

This topic describes the post-check phase of the Ammeon Workflow Engine (AWE). The post-check phase runs the tasks needed to verify that the operations performed on the deployment have completed successfully. Additionally, this phase runs any tasks that have not passed in this phase or the previous phases.

Prerequisites

The necessary script files must be written and all necessary tasks added to the `workflow.xml` file. For more information, see Creating Workflow Files.

Run the Post-check Phase

Use the following command syntax to run the post-check phase on all servers and display INFO and ERROR lines on screen:

```
wfeng --hostfile <hosts.xml file> --workfile <workflow.xml file> -i <filename.ini>
--phase postcheck --output 2
```

When the post-check phase is run using the configuration described above, the following output is returned:

```

                                  WORKFLOW ENGINE
                                 -----------------

Options chosen:
 workflow: ./workflow_55_56.xml
    hosts: ./hosts.xml
    timeout: 10
    phase: postcheck
    server types: *
    server name: *
    excluded servers: None
    task id: None
    tag id: None
    output level: 2
    yes: True
    previous run: ./workflow_55_56_hosts_status.xml

                                RUNNING PHASE: post-check
                               ------------------------

PARALLEL_TASK check_slaves: START
TASK post_check_replication_wflowslave1: Running
/opt/ammeon/mysql/bin/check_replication_status.bsh on 10.30.1.132 ...
STATUS post_check_replication_wflowslave1: RUNNING, START: 11:21:07

TASK post_check_replication_wflowslave2: Running
/opt/ammeon/mysql/bin/check_replication_status.bsh on 10.30.1.134 ...
STATUS post_check_replication_wflowslave2: RUNNING, START: 11:21:07
  -->post_check_replication_wflowslave1: CHECK_INFO: Slave_IO_State was 'Waiting for
master to send event' [OK]
  -->post_check_replication_wflowslave1: CHECK_INFO: Slave_IO_Running was 'Yes' [OK]
  -->post_check_replication_wflowslave1: CHECK_INFO: Slave_SQL_Running was 'Yes' [OK]
  -->post_check_replication_wflowslave1: CHECK_INFO: Last_Error was '' [OK]
STATUS post_check_replication_wflowslave1: COMPLETED, START: 11:21:07, END: 11:21:08
```

```
RESULT post_check_replication_wflowslave1: SUCCESS, DURATION: 0:00:01
  -->post_check_replication_wflowslave2: CHECK_INFO: Slave_IO_State was 'Waiting for
master to send event' [OK]
  -->post_check_replication_wflowslave2: CHECK_INFO: Slave_IO_Running was 'Yes' [OK]
  -->post_check_replication_wflowslave2: CHECK_INFO: Slave_SQL_Running was 'Yes' [OK]
  -->post_check_replication_wflowslave2: CHECK_INFO: Last_Error was '' [OK]
STATUS post_check_replication_wflowslave2: COMPLETED, START: 11:21:07, END: 11:21:08

RESULT post_check_replication_wflowslave2: SUCCESS, DURATION: 0:00:01
PARALLEL_TASK check_slaves: END
SUMMARY: Success: 2 Failed: 0 Skipped: 0
Workflow completed
```

The following example shows the workflow status file when the post-check phase is run:

```
<post-checksys>
    <parallelstatus id="check_slaves">
      <sequencestatus id="check_slaves:SLAVE:*:0" host="wflowslave1" server="SLAVE">
        <taskstatus cmd="/opt/ammeon/mysql/bin/check_replication_status.bsh"
host="wflowslave1" server="SLAVE" continueOnFail="true" optional="false"
depsinglehost="false" runLocal="false" id="post_check_replication" status="SUCCESS"
dependency="alter_table" actualDur="0:00:01"/>
      </sequencestatus>
      <sequencestatus id="check_slaves:SLAVE:*:0" host="wflowslave2" server="SLAVE">
        <taskstatus cmd="/opt/ammeon/mysql/bin/check_replication_status.bsh"
host="wflowslave2" server="SLAVE" continueOnFail="true" optional="false"
depsinglehost="false" runLocal="false" id="post_check_replication" status="SUCCESS"
dependency="alter_table" actualDur="0:00:01"/>
      </sequencestatus>
    </parallelstatus>
  </post-checksys>
```

Run Time Options

You can select which servers or server types can be run in the workflow using the interactive menu or the following commands:

Option	Description
`servertype`	This is a comma-separated list of server types to run on . Workflow only runs those tasks on servers that are defined in the hosts XML file with matching SERVER parameter.
`servername`	This is a comma-separated list of server names to run on, as defined by the name value in the hosts XML file.
`exclude`	This is a comma-separated list of servers NOT to run on. Can be used with or without the `servertype` parameter.

Table: Post-Check Phase Run Time Options

To run tasks only on servers of type SLAVE, enter the following command:

```
wfeng --hostfile <hostfile> --workfile <workfile> --servertype SLAVE --phase postcheck
```

To run tasks only on slave1 server, enter the following command:

```
wfeng --hostfile <hostfile> --workfile <workfile> --servername slave1 --phase
postcheck
```

To run tasks on all servers except slave2, enter the following command:

```
wfeng --hostfile <hostfile> --workfile <workfile> --exclude slave2 --phase postcheck
```

Running Named Workflow Task or Tasks

This topic describes how to run one or more named workflow tasks.

In addition to running phases, the Ammeon Workflow Engine enables users to run individual tasks through the interactive menu or with the `--task` option on the command line. The --task option takes a comma separated list of tasks. For more information on the command line and interactive menu, see User Interfaces. A named workflow task or tasks can be set to run on a single server, a set of servers or all servers. Dependencies are still checked, so if a selected task has dependencies that have failed then that task does not run

The following example shows the command to run a task on the server `slave1` with the `--task` option set to `precheck_slave`:

```
wfeng --hostfile <hostfile> --workfile <workfile> --servername=slave1
--task=precheck_slave
```

An example of the output from this command is shown below:

```
                               WORKFLOW ENGINE
                              ----------------
Options chosen:
      workflow: ./workflow.xml
      hosts: ./hosts.xml
      timeout: 10
      phase: postcheck
      server types: *
      server name: slave1
      excluded servers: None
      task id: precheck_slave
      tag id: None
      output level: 0
      previous run: ./workflow_hosts_status.xml

                            RUNNING PHASE: display
                        ----------------------
SKIP_SUCCESS TASK display_slave1: /opt/cust/sol1/display.sh on 1.1.1.2 DURATION:
0:00:00
...
SUMMARY: Success: 3 Failed: 0 Skipped: 0
Do you want to continue to next phase (y/n)?: y

                         RUNNING PHASE: pre-check
                         -----------------------
TASK precheck_slave_slave1: Running /opt/cust/sol1/precheck.sh on 1.1.1.2 ...
STATUS precheck_slave_slave1: RUNNING, START: 12:07:26
STATUS precheck_slave_slave1: COMPLETED, START: 12:07:26, END: 12:07:26
RESULT precheck_slave_slave1: SUCCESS, DURATION: 0:00:00
SKIPPED TASK precheck_slave_slave2: /opt/cust/sol1/precheck.sh on 1.1.1.3
SKIPPED TASK precheck_master_master1: /opt/cust/sol1/precheck.sh on 1.1.1.1
SUMMARY: Success: 1 Failed: 0 Skipped: 2
Workflow completed
```

The following example shows the command to run two tasks on all servers:

```
wfeng --hostfile <hostfile> --workfile <workfile> --servername=slave1
--task=display_slave,precheck_slave
```

Version plugins

This topic describes how to configure AWE to use custom plugins for comparing version strings.

When the display phase runs AWE's tasks can return version strings for software version and O/S version. These can then be used to compare against the swversion and osversion attributes in elements in the workflow file. Tasks will be skipped if the node is already at the version defined by the task attributes.

By default AWE just provides an exact-match comparison, and will therefore skip a task with a swversion attribute if the display phase returned the same string for the swversion as that configured in the workflow, similarly for O/S versions.

However AWE can be configured to use either a sequence version checking plugin or any custom plugin that the user generates. AWE supports using the same or different plugins per AWE instance for software and O/S version checking.

Sequence version plugin

AWE comes with a sequenceverplugin that can be used to compare numeric version strings that can be . separated, e.g. 1.2.1, 1.3.0, 12 etc. To use this plugin then the wfeng.cfg (located in $AWE_HOME/cfg) needs to be configured to pick up this module, by specifying it as the plugin to use for software version (SWVERSIONPLUGIN) and/or O/S version (OSVERSIONPLUGIN), e.g.:

```
SWVERSIONPLUGIN = wfeng.plugins.sequenceverplugin
OSVERSIONPLUGIN = wfeng.plugins.sequenceverplugin
```

When the sequenceverplugin is used then a task will be skipped if the version returned from the display tasks is greater or equal to that configured in the workflow swversion/osversion attributes.

Creating a custom version plugin

A user can write their own custom version plugin by writing a module that provides a compare_version function with the following signature:

```
def compare_version(version_req, current_version):
    """ Performs version comparison to report if host is already at required
        version
        Args:
            version_req: Version that want to be at
            current_version: Version that host is currently at
        Returns:
            True if current_version >= version_req
            False otherwise
        Raises:
            ValueError: if version is in unsupported format
    """
```

To use the plugin then:

- the module needs to be placed in a directory on the pythonpath. It is recommended to place it in $AWE_HOME/lib/wfeng/plugins)
- wfeng.cfg (located in $AWE_HOME/cfg) needs to be configured to pick up this module for either the software version and/or the O/S version checking, and therefore the SWVERSIONPLUGIN and OSVERSIONPLUGIN

should reflect the full modulename, e.g.

```
SWVERSIONPLUGIN = wfeng.plugins.<name of module>
OSVERSIONPLUGIN = wfeng.plugins.<name of module>
```

Troubleshooting the Ammeon Workflow Engine

This section provides useful information on understanding the Ammeon Worflow Engine output and troubleshooting issues. The following topics are covered:

- Troubleshooting Information
- Pausing or Stopping a Workflow
- Adding or removing deployment specific pause or escape elements
- Using List Options
- Fixing Tasks
- Resetting Tasks

Troubleshooting Information

This topic describes the task status information returned by the Ammeon Workflow Engine (AWE) and some of the options available for troubleshooting.

The topic covers the following:

- Logging
- Task status descriptions
- Notice Element
- Interrupting the engine

Logging

The log files used by the AWE are stored in `<AWE_HOME>/log` directory. Filenames by default take the format:

```
<workflow name>_<host filename>_ddmmyyyy_hhMMss.log
```

The workflow engine log includes information about that run of the engine, such as:

- All the information sent to the screen during normal operation
- The standard output from each of the supported activities run by the engine

The filename can also include some of the command line arguments if the EXTRA_LOGPARAM configuration value is configured in the wfeng.cfg. For more information on configuring EXTRA_LOGPARAM, see Configuration file.

Task Status Descriptions

The following table describes the task statuses used by the workflow engine:

Status	Description
INITIAL	This is the initial status when that task has not yet run.
SUCCESS	The task has run successfully.
REACHED_VERSION	The task was not run as the host is already at the appropriate software/OS version. The task was only to run if the host was not at the version specified.
MANUAL_FIX	The task is marked as fixed using the `--fix` option. For more information, see Fixing Tasks.
PARAM_NOTMATCH	The task was not run as the `checkparams` for the task did not match that of the host. Therefore, the task was not required.
FAILED	The task ran but failed.
SKIPPED	The task was skipped. For example, the task only needed to run on a subset of servers.
MANUAL_RESET	The task is marked as reset using the `--reset` option. For more information, see Resetting Tasks

Notice Element

Use the `notice` element when you want to display a notice to the user on the screen during the workflow. The following shows an example of a notice element that could be included in a workflow file:

```
<notice id="not1" msg="Follow the instructions in Chapter 2 of the User Guide"
hosts="*" server="*"/>
```

This would display the following to the screen:

```
********************************************************************************
TASK not1_node3: Notice

Follow the instructions in Chapter 2 of the User Guide

RESULT not1_node3: Complete
********************************************************************************
```

It is possible to have the `msg` attribute set to something like `"$MSG1"`, where `MSG1` is defined in the INI file. This allows the user to insert long messages without cluttering the workflow file. Multi-line values are supported by using quotes. For example:

```
MSG1="At this stage, the workflow engine will perform the following:
1. Check if backup is required
2. Perform backup which may take several hours"
```

Interrupting the engine

If you want to interrupt the engine as it runs then you can Ctrl-C the engine from the terminal. The engine will NOT stop immediately, instead it will wait until the tasks it is currently running have completed. If the engine is running a parallel set of tasks then it will wait until each of the modules currently being run have finished, but it will not go onto the next tasks in that parallel sequence. The engine will then gracefully shutdown recording the status of those tasks that it waited to complete.

For example:

```
TASK display_slave1: Running /opt/cust/sol1/display.sh on 1.1.1.2 ...
STATUS display_slave1: RUNNING, START: 15:44:38
^CINTERRUPT CAUGHT - Engine will stop when running tasks have finished
STATUS display_slave1: COMPLETED, START: 15:44:38, END: 15:45:39
RESULT display_slave1: SUCCESS, DURATION: 0:01:00

SUMMARY: Success: 1 Failed: 0 Skipped: 0
STOPPING due to interrupt
```

If you want to signal to the engine to stop after its finished running all of the parallel set then instead you can send a SIGINT signal to the main AWE python process, using kill -SIGINT <AWE pid>

Pausing or Stopping a Workflow

This topic describes how to pause or stop a workflow using the `pause` and `escape` elements.

Using the pause Element

Use the `pause` element when you want to check something on the system before continuing with the workflow. The message defined in the `msg` attribute of the `pause` element is displayed on the screen. The following example shows how the `pause` element can be used in a workflow:

```
<pause id="admin1" msg="Do you wish to continue (y) or pause(n)?" hosts="*"
server="SLAVE"/>
```

The following shows an example output when this task is run:

```
TASK admin1: Do you wish to continue (y) or pause(n)? (y/n)
```

If the user answers y, the following is displayed:

```
TASK admin1: Do you wish to continue (y) or pause(n)? (y/n) y

RESULT admin1: Continuing
********************************************************************************
```

If the user answers n, the the workflow stops and a message similar to the following is returned:

```
TASK admin1: Do you wish to continue (y) or pause(n)? (y/n) n

SUMMARY: Success: 3 Failed: 1 Skipped: 0
STOPPING as requested
```

If you decide to pause the workflow, resume the workflow again later by running the `execute` command again. The engine resumes the workflow at the `pause` element.

This `pause` element includes the following attributes:

Attribute	Description
`id`	Mandatory: The unique ID identifying the element.
`msg`	Mandatory: The message displayed on the screen asking the user if they want to pause the workflow.
`server`	Mandatory: The server type to run on. This should either be * or the server type as defined by the server attribute in the hosts XML file.
`hosts`	Mandatory: Defines which host to run on.
`dependency`	Optional: The task ID it is dependent on.

`depsinglehost`	Optional: Identifies whether a dependent task passed on the same host or all hosts.
`optional`	Optional: If this is set to `false` and the `servertype` is not in the hosts XML file, the workflow fails.
`osversion`	Optional: If this attribute is specified, the command is only run if the host is at the specified OS version.
`swversion`	Optional: If this attribute is specified, the command is only run if the host is at the specified software version.
`checkparams`	Optional: This is a comma-separated list of NAME=VALUE pairs. A task is only run if the host has parameter `NAME` set to `VALUE`.

Table: <pause> Element Attributes

Using the escape Element

Use the `escape` element to stop running tasks and exit at a particular point in the workflow.The message defined in the `msg` attribute of the `escape` element is displayed on the screen. The following example shows how the `escape` element can be used in a workflow:

```
<escape id="escape_" msg="Exiting the workflow." server="SLAVE" hosts="*" />
```

If you decide to escape the workflow, you can resume the workflow again later by running the `execute` command again. The engine resumes the workflow with the first task after the `<escape>` element.

This `<escape>` element includes the following attributes:

Attribute	Description
`id`	Mandatory: The unique ID identifying the element.
`msg`	Mandatory: The message displayed on the screen before exiting.
`server`	Mandatory: The server type to run on. This should either be `*` or the server type as defined by the server attribute in the hosts XML file.
`hosts`	Mandatory: Defines which host to run on.
`dependency`	Optional: The task ID it is dependent on.
`depsinglehost`	Optional: Identifies whether a dependent task passed on the same host or all hosts.
`optional`	Optional: If this is set to `false` and the `servertype` is not in the hosts XML file, the workflow fails.
`osversion`	Optional: If this attribute is specified, the command is only run if the host is at the specified OS version.

`swversion`	Optional: If this attribute is specified, the command is only run if the host is at the specified software version.
`checkparams`	Optional: This is a comma-separated list of NAME=VALUE pairs. A task is only run if the host has parameter `NAME` set to `VALUE`.

Table: <escape> Element Attributes

Using List Options

The Ammeon Workflow Engine has a list option (`--list`) that enables it to output only the tasks that are run with the options you have chosen. This can be useful when you want to check which tasks are run. For example, to show the tasks involved in running the display phase on the server slave1, run the following command:

```
wfeng --workfile <workfile> --hostfile <hostfile> --list --servername slave1 --phase
display
```

An example output from this command is shown below:

```
                              WORKFLOW ENGINE
                             ----------------

Options chosen:
      workflow: <workflow file>
      hosts: <hosts file>
      timeout: 10
      phase: precheck
      server types: *
      server name: slave1
      excluded servers: None
      task id: None
      tag id: None
      output level: 0
      list: True
      previous run: N/A


                             LISTING PHASE: display
                             ----------------------

RUN TASK display_slave1: /opt/cust/sol1/display.sh on 1.1.1.2
SKIP TASK display_slave2: /opt/cust/sol1/display.sh on 1.1.1.2
SKIP TASK display_master1: /opt/cust/sol1/display.sh on 1.1.1.1

Workflow completed
```

Fixing Tasks

This topic describes how to manually set a task to a status of passed (MANUAL_FIX).

From the command line, you can mark an individual task or task(s) on specified servers as passed. This can be used when a task has failed through the workflow engine, but the failure has been manually resolved. Using the `--fix`, `--task` and `--servername` options, you can mark a task as passed so that any dependant tasks can then be run.

If instead you want to mark a task as NOT run, so that a passed task can be re-run, then you can use the --reset option instead, which is described in Resetting Tasks.

For example, to mark the `precheck_slave` task as passed on slave1, run the following command:

```
wfeng --hostfile <hostfile> --workfile <workfile> --fix --servername=slave1
--task=precheck_slave
```

An example output from this command is shown below:

```

                              WORKFLOW ENGINE
                             ----------------
Options chosen:
      workflow: ./workflow.xml
      hosts: ./hosts.xml
      ini: None
      timeout: 10
      phase: postcheck
      server types: *
      server name: slave1
      excluded servers: None
      task id: precheck_slave
      tag id: None
      output level: 0
      fix: True
      previous run: ./workflow_hosts_status.xml

                             RUNNING PHASE: display
                             ----------------------

SKIP_SUCCESS TASK display_slave1: /opt/cust/sol1/display.sh on 1.1.1.2
DURATION: 0:00:00
...
SUMMARY: Success: 3
Failed: 0 Skipped: 0

Do you want to continue to next phase (y/n)?: y

                            RUNNING PHASE: pre-check
                            ------------------------

MANUAL_FIX TASK precheck_slave_slave1: /opt/cust/sol1/precheck.sh on 1.1.1.2
SKIPPED TASK precheck_slave_slave2: /opt/cust/sol1/precheck.sh on 1.1.1.3
SKIPPED TASK precheck_master_master1: /opt/cust/sol1/precheck.sh on 1.1.1.1
SUMMARY: Success: 1 Failed: 0 Skipped: 2
Workflow completed
```

Multiple tasks can be fixed by specifying a comma separated list of task ids. For example to mark the `display_slave` and `precheck_slave` tasks as passed on slave1, run the following command:

```
wfeng --hostfile <hostfile> --workfile <workfile> --fix --servername=slave1
--task=display_slave,precheck_slave
```

Adding or removing deployment specific pause or escape elements

This topic describes how to add/remove pause or escape elements that are to be used only by a particular deployment. The previous topic described how pause and escape elements can be added to the workflow to add pause and stop points, these changes apply to all deployments that use that template workflow. In some circumstances a deployment that uses one of these template workflows may require additional pause or escape elements, or may want to add extra stop points after a workflow has been started

The engine menu allows the engineer to add those extra elements without editing any files. The --alter_pause_escape option allows the user to add these extra pause and escape points to the master status file.

Whenever a dynamic pause or escape is added the engine will create a backup of the previous master status file into the dynamic_alteration_backups sub-directory (relative to the current master status file).

Adding a dynamic pause Element

Add a dynamic `pause` element when you want to check something on the system before continuing with the workflow. When adding a dynamic pause element the user must decide which task the element should be added in relation to, and they can choose to enter it before or after that element. The user must therefore read the workflow.xml to determine which task they want to refer to. Use the following command syntax to invoke the dynamic alterations menu:

```
wfeng --hostfile <hosts.xml file> --workfile <workflow.xml file> -i <filename.ini>
--alter_pause_escape
```

The dynamic alterations menu is shown below:

```
WORKFLOW ENGINE DYNAMIC ALTERATIONS
                      -----------------------------------
 [1] Enter new dynamic ESCAPE task
 [2] Enter new dynamic PAUSE task
 [3] Delete existing dynamic ESCAPE task
 [4] Delete existing dynamic PAUSE task
Please select your option or q to quit:
```

If the user answers 2 the following is displayed:

```
Please select your option or q to quit:
2
              WORKFLOW ENGINE DYNAMIC ALTERATIONS - add PAUSE task
              ----------------------------------------------------
Enter task details, or type q into any field to quit back to menu
Enter PAUSE task id:
```

The following shows an example of adding a task pau1 after the task upgrade_repair_master, which is dependant on upgrade_repair_master and is performed on all hosts of type MASTER where the software version is not already at 5.6.20:

```
Enter PAUSE task id: pau1
Enter reference id (task before/after which new task will appear):
upgrade_repair_master
```

```
Enter [before] to insert before ref task, or [after] to insert after: after
msg: Now the master has been repaired, do you wish to continue?
Enter hosts and server types fields:
hosts: *
server: MASTER
dependency: upgrade_repair_master
depsingle:
swversion: 5.6.20
osversion:
checkparams:
PAUSE: New task will be inserted after existing task upgrade_repair_master
id:[pau1] msg:[Now the master has been repaired, do you wish to continue?] hosts:[*]
server:[MASTER] dep:[upgrade_repair_master] depsingle:[None] swver:[5.6.20]
osver:[None] checkparams:[None]
Please select y to insert PAUSE task, a to amend entries, or q to quit back to the
main menu without inserting the task:
```

After adding in all the entries the user has the choice to insert the task, amend it or quit. The following output is shown if they choose to insert the task, and then quit the menu:

```
Please select y to insert PAUSE task, a to amend entries, or q to quit back to the
main menu without inserting the task:
y
New master status file ./workflow_55_56_hosts_status.xml created with dynamic entry
pau1 added
Press enter to continue
                      WORKFLOW ENGINE DYNAMIC ALTERATIONS
                      -----------------------------------
 [1] Enter new dynamic ESCAPE task
 [2] Enter new dynamic PAUSE task
 [3] Delete existing dynamic ESCAPE task
 [4] Delete existing dynamic PAUSE task
Please select your option or q to quit:
q
Quitting dynamic alterations
```

The following shows alternative output if the user decides that they want to change the msg for example:

```
Please select y to insert PAUSE task, a to amend entries, or q to quit back to the
main menu without inserting the task:
a
Existing [id] entry: [pau1]  Accept? y/n: y
Existing [refid] entry: [upgrade_repair_master]  Accept? y/n: y
Existing [pos] entry: [after]  Accept? y/n: y
Existing [msg] entry: [Now the master has been repaired, do you wish to continue?]
Accept? y/n: n
msg: About to restart replication, do you wish to continue?
Enter hosts and server types fields:
Existing [hosts] entry: [*]  Accept? y/n: y
Existing [server] entry: [MASTER]  Accept? y/n: y
Existing [dep] entry: [upgrade_repair_master]  Accept? y/n: y
Existing [depsingle] entry: [None]  Accept? y/n: y
Existing [swver] entry: [5.6.20]  Accept? y/n: y
Existing [osver] entry: [None]  Accept? y/n: y
Existing [checkparams] entry: [None]  Accept? y/n: y
PAUSE: New task will be inserted after existing task upgrade_repair_master
```

```
id:[pau1] msg:[About to restart replication, do you wish to continue?] hosts:[*]
server:[MASTER] dep:[upgrade_repair_master] depsingle:[None] swver:[5.6.20]
osver:[None] checkparams:[None]
Please select y to insert PAUSE task, a to amend entries, or q to quit back to the
main menu without inserting the task:
y
New master status file ./workflow_55_56_hosts_status.xml created with dynamic entry
pau1 added
Press enter to continue
                      WORKFLOW ENGINE DYNAMIC ALTERATIONS
                      -----------------------------------
 [1] Enter new dynamic ESCAPE task
 [2] Enter new dynamic PAUSE task
 [3] Delete existing dynamic ESCAPE task
 [4] Delete existing dynamic PAUSE task
Please select your option or q to quit:
q
Quitting dynamic alterations
```

The dynamic `pause` element includes the following attributes:

Attribute	Description
`id`	Mandatory: The unique ID identifying the element.
`msg`	Mandatory: The message displayed on the screen asking the user if they want to pause the workflow.
`server`	Mandatory: The server type to run on. This should either be * or the server type as defined by the server attribute in the hosts XML file.
`hosts`	Mandatory: Defines which host to run on.
`dependency`	Optional: The task ID it is dependent on.
`depsinglehost`	Optional: Identifies whether a dependent task passed on the same host or all hosts.
`optional`	Optional: If this is set to `false` and the `servertype` is not in the hosts XML file, the workflow fails.
`osversion`	Optional: If this attribute is specified, the command is only run if the host is at the specified OS version.
`swversion`	Optional: If this attribute is specified, the command is only run if the host is at the specified software version.
`checkparams`	Optional: This is a comma-separated list of NAME=VALUE pairs. A task is only run if the host has parameter `NAME` set to `VALUE`.

Table: dynamic <pause> Element Attributes

Removing a dynamic pause Element

The dynamic alterations menu can also be used to remove any pause elements added via the dynamic alterations menu. To invoke the dynamic alterations menu the following command is used:

```
wfeng --hostfile <hosts.xml file> --workfile <workflow.xml file> -i <filename.ini>
--alter_pause_escape
```

The dynamic alterations menu is shown below:

```
WORKFLOW ENGINE DYNAMIC ALTERATIONS
                      -----------------------------------
 [1] Enter new dynamic ESCAPE task
 [2] Enter new dynamic PAUSE task
 [3] Delete existing dynamic ESCAPE task
 [4] Delete existing dynamic PAUSE task
Please select your option or q to quit:
```

If the user answers 4 the following is displayed:

```
Please select your option or q to quit:
4
            WORKFLOW ENGINE DYNAMIC ALTERATIONS - remove PAUSE task
            -------------------------------------------------------
Enter task details, or type q into any field to quit back to menu
Enter PAUSE task id:
```

The following shows an example of removing the dynamic pause task pau1:

```
WORKFLOW ENGINE DYNAMIC ALTERATIONS - remove PAUSE task
            -------------------------------------------------------
Enter task details, or type q into any field to quit back to menu
Enter PAUSE task id: pau1
Please select y to remove PAUSE task, a to amend entry, or q to quit back to the main
menu without removing the task:
y
New master status file ./workflow_55_56_hosts_status.xml created with dynamic entry
pau1 removed
Press enter to continue
                      WORKFLOW ENGINE DYNAMIC ALTERATIONS
                      -----------------------------------
 [1] Enter new dynamic ESCAPE task
 [2] Enter new dynamic PAUSE task
 [3] Delete existing dynamic ESCAPE task
 [4] Delete existing dynamic PAUSE task
Please select your option or q to quit:
q
Quitting dynamic alterations
```

Adding a dynamic escape Element

Add a dynamic element element when you want to stop running tasks and exit at a particular point in the workflow for a particular deployment. When adding a dynamic escape element the user must decide which task the element should be added in relation to, and they can choose to enter it before or after that element. The user must therefore read the workflow.xml to determine which task they want to refer to. Use the following command syntax to invoke the dynamic alterations menu:

```
wfeng --hostfile <hosts.xml file> --workfile <workflow.xml file> -i <filename.ini>
--alter_pause_escape
```

The dynamic alterations menu is shown below:

```
WORKFLOW ENGINE DYNAMIC ALTERATIONS
                      -----------------------------------
 [1] Enter new dynamic ESCAPE task
 [2] Enter new dynamic PAUSE task
 [3] Delete existing dynamic ESCAPE task
 [4] Delete existing dynamic PAUSE task
Please select your option or q to quit:
```

If the user answers 1 the following is displayed:

```
Please select your option or q to quit:
1
             WORKFLOW ENGINE DYNAMIC ALTERATIONS - add ESCAPE task
             -----------------------------------------------------
Enter task details, or type q into any field to quit back to menu
Enter ESCAPE task id:
```

The following shows an example of adding a task esc1 before the task upgrade_repair_master, which is dependant on stop_replication and is performed on all hosts of type MASTER where the software version is not already at 5.6.20:

```
Enter ESCAPE task id: esc1
Enter reference id (task before/after which new task will appear):
upgrade_repair_master
Enter [before] to insert before ref task, or [after] to insert after: before
msg: Exiting before master is repaired
Enter hosts and server types fields:
hosts: *
server: MASTER
dependency: stop_replication
depsingle:
swversion: 5.6.20
osversion:
checkparams:
ESCAPE: New task will be inserted before existing task upgrade_repair_master
id:[esc1] msg:[Exiting before master is repaired] hosts:[*] server:[MASTER]
dep:[stop_replication] depsingle:[None] swver:[5.6.20] osver:[None] checkparams:[None]
Please select y to insert ESCAPE task, a to amend entries, or q to quit back to the
main menu without inserting the task:
y
New master status file ./workflow_55_56_hosts_status.xml created with dynamic entry
esc1 added
Press enter to continue
                      WORKFLOW ENGINE DYNAMIC ALTERATIONS
                      -----------------------------------
```

```
 [1] Enter new dynamic ESCAPE task
 [2] Enter new dynamic PAUSE task
 [3] Delete existing dynamic ESCAPE task
 [4] Delete existing dynamic PAUSE task
Please select your option or q to quit:
q
Quitting dynamic alterations
```

The dyamic escape element includes the following attributes:

Attribute	Description
`id`	Mandatory: The unique ID identifying the element.
`msg`	Mandatory: The message displayed on the screen before exiting.
`server`	Mandatory: The server type to run on. This should either be * or the server type as defined by the server attribute in the hosts XML file.
`hosts`	Mandatory: Defines which host to run on.
`dependency`	Optional: The task ID it is dependent on.
`depsinglehost`	Optional: Identifies whether a dependent task passed on the same host or all hosts.
`optional`	Optional: If this is set to `false` and the `servertype` is not in the hosts XML file, the workflow fails.
`osversion`	Optional: If this attribute is specified, the command is only run if the host is at the specified OS version.
`swversion`	Optional: If this attribute is specified, the command is only run if the host is at the specified software version.
`checkparams`	Optional: This is a comma-separated list of NAME=VALUE pairs. A task is only run if the host has parameter NAME set to VALUE.

Table: dynamic <escape> Element Attributes

Removing a dynamic escape Element

The dynamic alterations menu can also be used to remove any escape elements added via the dynamic alterations menu. To invoke the dynamic alterations menu the following command is used:

```
wfeng --hostfile <hosts.xml file> --workfile <workflow.xml file> -i <filename.ini>
--alter_pause_escape
```

The dynamic alterations menu is shown below:

```
WORKFLOW ENGINE DYNAMIC ALTERATIONS
```

```
                      -----------------------------------
 [1] Enter new dynamic ESCAPE task
 [2] Enter new dynamic PAUSE task
 [3] Delete existing dynamic ESCAPE task
 [4] Delete existing dynamic PAUSE task
Please select your option or q to quit:
```

If the user answers 3 the following is displayed:

```
Please select your option or q to quit:
3
            WORKFLOW ENGINE DYNAMIC ALTERATIONS - remove ESCAPE task
            --------------------------------------------------------
Enter task details, or type q into any field to quit back to menu
Enter ESCAPE task id:
```

The following shows an example of removing the dynamic escape task esc1:

```
WORKFLOW ENGINE DYNAMIC ALTERATIONS - remove ESCAPE task
            --------------------------------------------------------
Enter task details, or type q into any field to quit back to menu
Enter ESCAPE task id: esc1
Please select y to remove ESCAPE task, a to amend entry, or q to quit back to the main
menu without removing the task:
y
New master status file ./workflow_55_56_hosts_status.xml created with dynamic entry
esc1 removed
Press enter to continue
                      WORKFLOW ENGINE DYNAMIC ALTERATIONS
                      -----------------------------------
 [1] Enter new dynamic ESCAPE task
 [2] Enter new dynamic PAUSE task
 [3] Delete existing dynamic ESCAPE task
 [4] Delete existing dynamic PAUSE task
Please select your option or q to quit:
q
Quitting dynamic alterations
```

Resetting Tasks

This topic describes how to reset a task or tasks so that it can be re-run. If a task is reset then it will be treated the same as if the task had never been run, and its status will be set to MANUAL_RESET.

This can be used if the effects of a task have been rolled back, and the user needs to reset the status so that the task is re-run on the next (or future) invocations of the engine.

From the command line, you can reset a group of tasks in the execute phase by using the `--task` and `--reset` options. The tasks can be reset for all servers, or restricted to certain servers or servertypes using the `--servername`, `--servertype` or `--exclude` parameters. You can also reset all tasks in a tagged set for all or restricted servers using the `--tag` and `--reset` options.

NB. It is currently only possible to reset tasks that are in the execute phase.

Whenever the engine performs a reset it will create a backup of the previous master status file into the manual_reset_backups sub-directory (relative to the current master status file).

For example, to reset the `upgrade_slave` and `shutdown_slave` tasks on `slave1`, run the following command:

```
wfeng --hostfile <hostfile> --workfile <workfile> --reset --servername=slave1
--task=upgrade_slave,shutdown_slave
```

An example output from this command is shown below:

```
                               WORKFLOW ENGINE
                              ----------------
Options chosen:
      workflow: ./workflow.xml
      hosts: ./hosts.xml
      ini: None
      timeout: 10
      phase: postcheck
      server types: *
      server name: slave1
      excluded servers: None
      task id: upgrade_slave,shutdown_slave
      tag id: None
      output level: 0
      reset: True
      previous run: ./workflow_hosts_status.xml

Processing reset...



Resetting task upgrade_slave on slave1 will change status from SUCCESS to MANUAL_RESET
Resetting task shutdown_slave on slave1 will change status from SUCCESS to
MANUAL_RESET

The following tasks are eligible for reset:

Task id: shutdown_slave on server slave1
Task id: upgrade_slave on server slave1
```

```
Manual reset completed successfully
```

If attempting to reset a task which another task is dependant on, and that later task has completed then by default the engine will NOT allow the task to be reset.

In the example if both `shutdown_slave` and `upgrade_slave` had been run, and `upgrade_slave` was dependant on `shutdown_slave`, the user couldn't reset just `shutdown_slave`. The normal expected behaviour would be to reset both tasks. If the user wanted to reset just the `shutdown_slave` task (even though `upgrade_slave` had completed), then the `--force` option would also need to supplied. With `--force` the user is warned of any completed dependants and has the option to continue or cancel the reset.

Appendix 1: Workflow File Definition

This appendix provides detailed information on the elements of the workflow file and their attributes.

Workflow XML Elements

The following table describes the elements of the Workflow XML:

Element	Description
`workflow`	A top-level element that contains the child elements `display`, `pre-check`, `execute`, and `post-check`. • `display`: A container for the tasks to run at display stage. This element has the optional child elements `task` and `parallel.` • `pre-check`: A container for the tasks to run at pre-check stage. This element has the optional child elements `task` and `parallel.` • `execute`: A container for the tasks to run at execute stage. This element has the optional child elements `task`, `group`, `parallel`, `start-tag`, `end-tag`, `escape` and `pause`. • `post-check`: A container for the tasks to run at post-check stage. This element has the optional child elements `task` and `parallel`.
`task`	Defines the command to run on a local or target server. This is a child element of the `display`, `pre-check`, `execute`, `post-check`, `group` and `parallel` elements.
`parallel`	Defines that its child tasks should be run in parallel (by default tasks are run sequentially). This is a child of the `display`, `execute`, `pre-check` and `post-check` elements.
`group`	Defines that its child tasks should be run as a group. For example, all child tasks are run on the first host and then all child tasks on the second host. This is a child element of the `execute` element.
`pause`	Pauses the upgrade. The engineer is asked whether they want to continue with the upgrade or not. This is a child element of the `execute` and `group` elements.
`escape`	Stops the upgrade at a specified point. This is a child element of the `execute` and `group` elements.

`notice`	Use the `<notice>` element when you want to display a notice to the user on screen during the workflow. This is a child element of the `execute` and `group` elements.
`start-tag`	Enables a set of tasks to be tagged so that they can be invoked with the `--tag` option. This indicates the start of the tagged set. This is a child element of the `execute` element.
`end-tag`	Enables a set of tasks to be tagged so that they can be invoked with the `--tag` option. This indicates the end of the tagged set. This is a child element of the `execute` element.

Table: Elements of the Workflow XML File

Task Element Attributes

The following table describes the attributes associated with the `task` element of the workflow file:

Attributes	Description	Required/ Optional
`id`	The unique ID to identify this task. This is used when displaying status of tasks and also for defining dependencies. Must not contain ,	Required
`cmd`	The command to run. Can contain parameters. The command is run on remote or local servers (depending on the value of `runLocal`).	Required
`server`	The server type to run on. The following values are possible: • * - runs on all server types defined in the hosts XML file • one of the server types defined in the server attribute in the hosts XML file • LOCAL - runs locally on the server and just once per deployment, and is not associated with any of the servers (see the `runLocal` attribute for running tasks locally but are associated with a server)	Required

`hosts`	Defines which hosts to run on. The following values are possible: • `*` - runs on all hosts of specified server type • *n* - where *n* is a decimal number. Runs on the nth host as defined in the hosts XML file of the specified type • `$` - runs on the last host as defined in the hosts XML file of the specified server type • !n - runs on all hosts except the nth as defined in the hosts XML file of the specified server type • !$ - runs on all hosts except the last host as defined in the hosts XML file of the specified server type	Required
`optional`	Indicates whether something is optional. If optional = `false` and the specified server is not found in the hosts XML file, the workflow is not run.	Required
`continueOnFail`	If set to `true` and the command fails, the workflow moves onto the next task in the phase. If set to `false`, the workflow is stopped immediately.	Required
`runLocal`	By default this is set to `false`. If set to `true`, the command is not run on the remote server but on the server running the workflow engine. It is also associated with the server specified in the `hosts/server` parameter. This means that: • it allows the use of the `$SERVERNAME`, `$SERVERIP` and `$SERVERTYPE` parameters to be passed to the command, which is expanded with the details of the remote server that the hosts/server apply to. • if the task applies to multiple servers, it is run once per server. • if the workflow is not run on all servers, it is only run if the task it applies to is one of those chosen.	Optional

`swversion`	If this attribute is specified, the command is only run if the host is at the specified software version. By default, this performs an exact match comparison, but by configuration of the SWVERSIONPLUGIN then any custom version comparison can be used.	Optional
`osversion`	If this attribute is specified, the command is only run if the host is at the specified OS version. By default, this performs an exact match comparison, but by configuration of the OSVERSIONPLUGIN then any custom version comparison can be used. **Notes**: • If both `osversion` and `swversion` are specified, the task is run only if the server matches both the OS and software versions. • If either `osversion` or `swversion` is listed in a task, but it was not extracted at the display stage, the workflow engine prompts the user to ask for the version unless `--force` is used. If `--force` is used and the version is not found, the task is run.	Optional
`estimatedDur`	If this attribute is specified, the output shows the estimated duration of a task in the `RUNNING STATUS` line that is displayed on the screen.	Optional
`dependency`	This is a comma-separated list of task IDs that this task is dependent on. If a dependency task fails, this task is skipped and not run.	Optional

`depsinglehost`	By default, if a dependency is stated, the dependent task must pass on all hosts. However, if the task is only dependent on a single host, you can specify this by setting `depsinglehost` to `true`.	Optional
`checkparams`	This is a comma-separated list of NAME=VALUE pairs. A task is only run if the host has parameter `NAME` set to `VALUE`. The `NAME` parameter comes from the parameters set by the standard output in the display info tag when run on a host. You can specify a combination of values through the | symbol. For example, `NAME=A|B` runs if parameter `NAME` has value `A` or `B`.	Optional

Table: Task Element Attributes

Escape, Pause and Notice Element Attributes

The following table describes the attributes associated with the `escape, pause and notice` elements of the workflow file:

Attributes	Description	Required/Optional
`id`	The unique ID to identify this task. This is used when displaying status of tasks and also for defining dependencies.	Required
`msg`	Defines the message to be displayed on screen.	Required

`hosts`	Defines which hosts to run on. The following values are possible: • `*` - runs on all hosts of the specified server type • *n* - where *n* is a decimal number. Runs on the nth host as defined in the hosts XML file of the specified type • `$` - runs on the last host as defined in the hosts XML file of the specified server type • !n - runs on all hosts except the nth as defined in the hosts XML file of the specified server type • `$` - runs on all hosts except the last host as defined in the hosts XML file of the specified server type	Required
`server`	The server type to run on. The following values are possible: • `*` - runs on all server types defined in the hosts XML file • one of the server types defined in the server attribute in the hosts XML file • LOCAL - runs locally on the server and just once per deployment, and is not associated with any of the servers (see the `runLocal` attribute for running tasks locally but are associated with a server)	Required
`dependency`	This is a comma-separated list of task IDs that this task is dependent on. If a dependency task fails, this task is skipped and not run.	Optional
`optional`	Indicates whether something is optional. If optional = `false` and the specified server is not found in the hosts XML file, the workflow is not run.	Required
`depsinglehost`	By default, if a dependency is stated, the dependent task must pass on all hosts. However, if the task is only dependent on a single host, you can specify this by setting `depsinglehost` to `true`.	Optional

`swversion`	If this attribute is specified, the command is only run if the host is at the specified software version. The software version has been extracted from the `DISPLAY_INFO` lines in the output from a task run on the server during the display stage. By default, this performs an exact match comparison.	Optional
`osversion`	If this attribute is specified, the command is only run if the host is at the specified OS version. The OS version has been extracted from the `DISPLAY_INFO` lines in the output from a task run on the server during the display stage. By default, this performs an exact match comparison. **Notes**: • If both `osversion` and `swversion` are specified, the task is run only if the server matches both the OS and software versions. • If either `osversion` or `swversion` is listed in a task, but it was not extracted at the display stage, the workflow engine prompts the user to ask for the version unless `--force` is used. If `--force` is used and the version was not found, the task is run.	Optional
`checkparams`	This is a comma-separated list of NAME=VALUE pairs. A task is only run if the host has parameter `NAME` set to `VALUE`. The `NAME` parameter comes from the parameters set by the standard output in the DISPLAY_INFO lines when run on a host. You can specify a combination of values through the | symbol. For example, `NAME=A|B` runs if parameter `NAME` has value `A` or `B`.	Optional

Table: Escape, Pause and Notice Element Attributes

The `start-tag`, `end-tag`, `group` and `parallel` elements each have one attribute `id`. The `workflow` element also has one attribute called `name`.