Status.out and Debugging SWAMP Failures

James A. Kupsch

Version 1.0.2, 2017-01-17 14:14:02 CST

Table of Contents

1. Introduction

This document describes the status.out file produced when running a build, assessment, and/or parsing of results in a SWAMP VM, and using it to debug failures. The status.out file is a summary of the tasks performed in a SWAMP VM. The tasks appear in chronological order of completion. From the status.out file, you can quickly determine if the assessment succeeded, how long it took to run, where the time was spent, and in the case of failure, what step caused the failure. An example status.out file is shown below.

Example status.out file from assessing lighttpd with cppcheck

```
NOTE: begin
PASS: install-os-dependencies                                    18.683063s
PASS: install-strace (/opt/strace-4.10/bin/strace)                0.090283s
PASS: tool-unarchive                                              0.300803s
PASS: package-unarchive                                           0.243168s
PASS: configure                                                  14.911602s
PASS: build                                                      64.587349s
PASS: build-trace-decode                                         21.838286s
PASS: assess (pass: 104, fail: 0)                                83.294354s
PASS: buildbug                                                  171.294004s
PASS: build-archive                                               1.455068s
PASS: results-archive                                             0.029203s
PASS: resultparser-unarchive                                      0.016161s
PASS: parse-results (weaknesses: 214)                             4.561078s
PASS: parsed_results-archive                                      0.006893s
PASS: all                                                       211.602487s
NOTE: end
```

Three main types of tasks occur in status.out files produced in the SWAMP:

1. **informational** — provide information such as the *begin* and *end* tasks, and do not correspond to an activity (all of these have a status of NOTE),
2. **activity** — correspond to tasks or events that occurred as part of the build, assessment or result parsing, such as the *build* or *assess* tasks, and
3. **aggregate** — tasks that aggregate a sequence of preceding subtasks such as the *all* or *buildbug* tasks; the status (PASS or FAIL) depends on the status of the subtasks, and the duration is the total time from the first task starting to the end of the last task in the sequence.

The rest of this document describes the status.out file format, the definitions of a valid status.out and successful run, the current tasks, and debugging task failures.

2. File Format

A status.out file consists of a chronologically ordered sequence of records (tasks) that convey the status of a single task. Each record consists of two to six components. The task name and its status are required components. The duration and duration unit of the task, a short single line message,

and a long multi-line message are optional. The remainder of this section shows the physical format, and then explains each component.

An example status.out task without a long message, and with parts labeled is shown below:

Single status.out task with parts labeled and no long message part.

```
PASS: task-name (short-message)                                    89.123456s
|     |          |                                                 |        |
|     task       shortMsg                                        dur      |
status                                                              durUnit
```

Each task may optionally have a multi-line message. An example of such a task is shown below:

Single status.out task with a two line long message.

```
PASS: task-name (short-message)                                    89.123456s
  ----------
  long message line 1
  long message line 2
  ----------
```

The individual parts of a task record are described next.

2.1. Task

The task is a string that is a sequence of letters, numbers, the hyphen character, and the underscore characters. The task is separated from the status by a colon. The task is required in each record, and each task in the file should be unique.

The currently defined task along with their description is found in the *Task Definitions* Section below.

2.2. Status

The status of a task is a string that is a sequence of letters, numbers, the hyphen character, and the underscore characters. The status is required in each record.

The currently defined status values are:

PASS
: indicates a successful task.

FAIL
: indicates an unsuccessful task.

SKIP
: indicates a task that was not run.

NOTE

indicates a task that is informational. Two tasks with this status are the *begin* and *end* tasks that by convention are the first and last tasks.

A successful run is one where none of the tasks have a status of FAIL.

Current practice is for status values to be four uppercase characters.

2.3. Duration

The duration of a task is how long it took to complete. The duration is optional, and is not included for tasks that indicate a point in time. It consists of a decimal number (consisting of numbers and single optional period (decimal point) to separate the fraction part of the value), and a unit (consisting of alphabetic characters). Current duration units are as follows:

Table 1. Duration Units

Unit	**Definition**
s	seconds
(*none*)	seconds
ms	milliseconds
ns	microseconds

Current practice is to use a unit of *s* with a resolution of microseconds, have at least one digit to the left of the decimal point, all six digits after the decimal point, and the last character of unit appearing in column 80 if possible.

2.4. Short Message

The short message is used to provide additional information about the task, and is optional. If present, it is enclosed in parenthesis and appears immediately after the task. It consists of sequence of characters that does not include the new line character or a left parenthesis.

The main function of the short message is to provide counts for a few tasks, or to convey the reason a task failed or was skipped.

2.5. Long Message

The long message is used to provide additional multi-line information about the task, and is optional. The main function of the long message is to provide additional information about failed tasks.

The long message consists of multiple lines. A long message for a task starts is indicated by the line following the task being a delimiter consisting of a sequence of one or more spaces and then a sequence of one or more hyphen characters. The long message is placed in subsequent lines each prefixed with the number of spaces in the delimiter. Finally a new line is added and the delimiter is

repeated to indicate the end of the long message. Processors must remove the prepended spaces of each line and final new line character to form the original data.

Current practice is to use two spaces and ten hyphens for the delimiter.

3. Valid *status.out* and Successful Run Definitions

A **valid *status.out*** file is syntactically correct, and in addition meets the following requirements:

1. status.out file exists in the *out* directory
2. no duplicate tasks
3. *begin* is the first task and has a status of NOTE
4. *all* task exists
5. *end* is the last task and has a status of NOTE

All *status.out* files produced during SWAMP build, assessment and/or parsing should be valid. If not, the assessment or host it is running failed in an unexpected way.

A **successful run** produces a valid status.out, and also meets the additional requirements:

6. *all* task has a status of PASS
7. no task has a status of FAIL

4. Task Definitions

all

is the aggregate task that covers all other subtasks. Its status indicates the overall status of the entire run and its duration is the duration of the entire run.

android-update

is the task of updating an Android package to work with the Android SDK installed on the machine or VM.

assess

is the task of running the assessment tools. Additional information is provide in the short message indicating the number of invocations of the assessment tool and their success, failure or number of assessments skipped (due to missing files or other valid reasons not to run the assessment tool on a source file). Each assessment processes one or files. An example short message for *assess* is

```
    (pass: 93, fail: 0, skip: 3)
```

The status is FAIL if any of the assessments failed, SKIP if no files were found to assess, or PASS

otherwise.

assess-acquire-license

is the task of acquiring the license to operate a tool that uses a license manager. If the status is FAIL, acquiring the license failed and the long message may contain more details about the reason for failure.

begin

is an informational task. This record is written upon start up.

build

is the task of building the software package with build monitoring. The build duration includes the added time to monitor the build process.

In the case of failure the long message will have information on the non-zero exit status or signal that caused the failure. Additional information may be available in the long message. See the *build_stdout.out* and *build_stderr.out* files found in the build archive for more information about the cause of the failure.

build-archive

is the task of archiving the build directory.

build-trace-decode

is the task of decoding the build monitoring for C and C++ builds.

build-unarchive

is the task of unarchiving the build directory.

buildbug

is the aggregate task covering all the build and assess tasks including *build*, *build-trace-decode*, and *assess*. If any of those fail buildbug will also be failed.

chdir-build-dir

is the task of changing to the build directory after changing to the package directory. Only present on failure with the long message containing the name of the directory and the reason for the failure.

chdir-config-dir

is the task of changing to the config directory after changing to the package directory. Only present on failure with the long message containing the name of the directory and the reason for the failure.

chdir-package-dir

is the task of changing to the package directory. Only present on failure with the long message containing the name of the directory and the reason for the failure.

configure

is the task of configuring the software package.

In the case of failure the long message will have information on the non-zero exit status or signal that caused the failure. Additional information may be available in the long message. See the *config_stdout.out* and *config_stderr.out* files found in the build archive for more information about the cause of the failure.

end

is an informational task. This record is written as the framework is ending. If it is not present, the framework or VM likely crashed.

fetch-pkg-dependencies

is the task to fetch the dependencies using a non operating system package manager. This task may be a task that is always included in the task list such as with pip packages, or done as part of the build task and is only included if the build fails due to the pacakge manager's failure to fetch dependencies. The short message contains the package manager (currently `gradle`, `maven`, `ant+ivy` or `pip`). The long message contains details about the failure.

gem-install

is the task of installing the user's Ruby gem file (user's package) along with the gem dependencies downloaded from the network into the local Ruby user install directory

gem-unpack

is the task that unpacks the user's Ruby gem file to make the source files accessible from the file system (similar to the package-unarchive but with the command: `gem unpack <gem-file-path> —target <package-dir>`).

install-os-dependencies

is the task of installing OS dependencies for the run using the OS's package manager such as `yum` or `apt-get`.

A status of PASS and FAIL are based on the success or failure of the OS's package management system. A status of SKIP can occur from two sources indicated by the task's short message:

none

no OS packages to install

(internet inaccessible)

assessment run is configured to indicate that the VM is in an environment where the internet is inaccessible (*run.conf* file contains `internet-inaccessible = true`)

install-pip-dependencies

is the task to install PIP dependencies for Python packages.

install-strace

is the task to install the strace command packaged with c-assess. If an strace was packaged c-assess the short message is the location of strace binary. If no strace was packaged, the status is SKIP, the short message gives the reason, and the system strace binary found in the PATH is used.

network

is the task that that indicates a properly functioning network in the VM. It is only present on failure. The short message indicates the type of failure (the only currently defined value is *no-ipv4-address*).

As this is a temporary failure, a *retry* task is also included indicating the VM should be run again.

no-build-setup

is the task to find and validate source files to assess for C, C++ or Java packages with a build system of *no-build*. The number of source files found with valid extensions and the number of files that actually compile are specified in the short message in the format shown in the following example:

```
(source-files: 1, compilable: 1)
```

The status is FAIL if no source files are found or none are compilable. The short message can be used to distinguish the specific type of failure.

package-checksum

is the task of validating the package's checksum against the value found in package.conf.

package-unarchive

is the task of unarchiving the package.

parse-results

is the task of running the result parser on the assessment results to produce parsed results in SCARF (SWAMP Common Assessment Results Format). The short message contains the number of weaknesses found as in the following example:

```
(weaknesses: 214)
```

parsed-results-archive

is the task of archiving the parsed results directory.

python2-install

is the task of installing the Python 2.x language.

python3-install

is the task of installing the Python 3.x language.

read-gem-spec

is the task of reading a Ruby package's gem file to determine what files to assess

resultparser-unarchive

is the task of unarchiving the result parser.

results-archive

is the task of archiving the results directory.

results-unarchive

is the task of unarchiving the results.

retry

is an informational task that indicates the VM should be recreated and restarted as a temporary condition was detected that caused the run to fail. The first failed task is the reason for the retry.

swamp-maven-plugin-install

is the task to setup the Java framework's plug-in to monitor Maven builds.

tool-install

is the task of installing the tool.

tool-package-compatibility

is the task to check compatibility of the tool for use with the software package being assessed. For instance there are tools that require Java 1.6 or above and fail if they are used on Java 1.4 or 1.5 packages.

The short message contains the type of error. The long message contains more details about the incompatibility. Currently defined short messages and their meanings are as follows:

(Java language version)

The Java source language used during the build is incompatible with the tool.

(gcc version)

The version of `gcc` used during the build is incompatible with the tool.

(android+maven)

The android-lint tool is not configured properly for a Maven project.

(known tool bug)

The tool failed assessing the source code due to a known bug with the tool.

tool-runtime-compatibility

is the task to check compatibility of the tool for use in the runtime environment.

The short message contains the type of error The long message contains more information about the incompatibility. Currently defined short messages and their meanings are as follows:

(JVM version)

The version of the JVM used is incompatible with the tool.

tool-unarchive

is the task of unarchiving the results.

validate-package

is the task to validate that the Java Bytecode files specified to assess are valid in the package.

5. Debugging Task Failures

If the run is successful (see Valid *status.out* and Successful Run Definitions), there is nothing to debug, but there may be useful information to report to the user as described in the *Potential Issues In Successful Runs* Section. The rest of this Section describes how to debug status.out files that are invalid, valid but unsuccessful, and finally issues that should be brought to the attention for runs that are successful.

5.1. Missing or Invalid status.out File

Both the *all* task and at least one other task having a status of FAIL indicates normal operation of the framework code, but a failure occurred during the run. Debugging these types of errors are explained in the *Debugging Specific Task Failures* section.

Other types failures indicate a missing or invalid status.out file. Explanation for these failures is

status.out missing

Internal problem starting the VM, such as /mnt/in or /mnt/out not being mounted.

duplicate tasks

The assessment framework was run multiple times, or a bug in the framework. Possibly VM was started multiple times.

first task not begin

VM or framework crashed or exited before the framework was run or early in the framework code.

all task missing

VM exited or crashed before framework completed. A previous failure in status.out, the *run.out* file from the VM's /mnt/out directory may contain additional information.

last task not end

VM exited or crashed before framework completed. A previous failure in status.out, the *run.out* file from the VM's /mnt/out directory may contain additional information. Data from the run could be OK if the *all* task has a status of PASS, but should be treated with suspicion.

5.2. Debugging Specific Task Failures

If the status.out file is valid, but the *all* task has a status of FAIL, then the run has failed. The first step to debug a failed, but valid status.out is to create a list of all the tasks that have a status of FAIL, and to determine if there is a *retry* task.

If there is a *retry* task, the VM should be recreated and rerun as this indicates a temporary failure. Retrying should be attempted a small fixed number of times, such as 3, until *retry* is no longer a task or the count is reached. If all attempts contain a *retry* task, there is a persistent problem with the infrastructure and the diagnosis should proceed with the next step.

The next step is find the first task in the failed task list that is in the list below of tasks and remedies. If a failed task is not in the list below report it to the developers and continue with subsequent elements until a failed task is in the list (there should be at least one since the *all* task must be in the list). Subsequent failed tasks should be ignored as they are likely caused by the first failure.

In the remedies below, if the task is marked as an *internal error*, this is is a problem with frameworks, VM hypervisors, or the environment. An internal error is not generally something that a user can fix by making changes to their package and how it is configured. Internal errors should be reported to the operator SWAMP or developers of the SWAMP software.

The next step is use the remedy in the list below:

all

Internal error. As an aggregate task some other task should have failed earlier in the list.

android-update

This failure indicates that the user's Android package does not conform to the convention Google uses to configure Android packages. This can be caused by copying an Android configuration file from an older Android SDK and modifying it instead of configure the package correctly to Google's convention.

To fix this the user can try to modify the package's SWAMP build options to enable the *Android Redo Build* option. This will erase the existing Android build file and the Android SDK will recreate the file from the metadata found in the package. This may not work if the user made changes that package depends on to the configuration file they copied as it is replaced.

The other option a developer has is to update their package to follow Google's build conventions using the latest Android SDK.

assess

Internal error. This is a failure of the assessment tool to run successfully. This may be caused by the tool not properly interpreting the input files due to deficiencies in the tool, a bug in the tool, or a misconfiguration of the tool.

To find more information about a failure of this type, first inspect the *assessment_summary.xml* file in the results archive. This file will has information about each assessment (tool run) including its success, and its output and error output. These output files should contain the actual error that occurred.

assess-acquire-license

Internal error. This may be due to networking issues between the VM running the tool, or an issue with the license server. The long message may contain more details on the failure.

(concurrency limit)

If the number of VMs running concurrently trying to use the licensed tool exceeds the maximum number of concurrent licenses allowed, recreating and rerunning the VM may solve the problem. This should not occur if the concurrency limits are properly enforced.

(network connection)

The host must be able to connect with the license server on the port it is listening for requests. It may not be able to due to the VM or network firewall blocking the connection, the license server host (or VM) is not up, or the license server is not running on the license server host.

(license expired)

Licenses have limited lifetimes and need to renewed periodically.

begin

Internal error. This task is informational and should always have a status of NOTE.

build

The build for the user's package failed. This can be caused by the user selecting the wrong build system, missing libraries, the package not being compatible with the environment of the selected platform (operating system), or the package containing code that is not syntactically valid.

Some packages download resources from the internet as part of the *build* task such as the use of a build system with an integrated package manager, or the tools like `wget` or `curl` during the build. For these packages, a *build* task status of FAIL can be caused by the remote resource being moved or removed, an incorrect specification for the remote resource, an off-line or malfunctioning remote server, the system is run in an environment that is isolated from the internet, or a network issue such as an incorrectly configured firewall or router. The user can remove the need for the remote resource, or correct the problem and try again at a later time. If the failure is due to the network the SWAMP administrator will need to correct the problem, or the user can try again at a later time as the problem could be transient.

The long message of the task contains the exact command that failed to run successfully, and how it failed. Debugging this failure is accomplished by inspecting the output of the build. The output and error output for the build step is found in the *build_stdout.out* and *build_stderr.out* files found in the build archive. The exact paths to these files is found by using the data in the *build.conf* file (found in the /mnt/out directory) to locate the build_summary.xml file, that in turn points to the output files.

The long message and the two output files should be provided for users to inspect if the *build* task fails, so they can determine what failed, and how to update the package archive or build parameters.

Users should be directed to try building the package on their own computer.

build-archive

Internal error. Error creating the build archive. A possible cause is that /mnt/out is full. The *run.out* file from the VM's /mnt/out directory may contain additional information.

build-trace-decode

Internal error. This indicates that the format of strace data produced by the build monitoring of C and C++ packages was not formatted as expected.

build-unarchive

Internal error. The build archive is corrupt or missing, or the file system is full. The *run.out* file from the VM's /mnt/out directory may contain additional information.

buildbug

Internal error. As an aggregate task some other task should have failed earlier in the list.

chdir-build-dir

The *build directory* (relative to the *package directory*) specified by the user is invalid for the user's package; the directory may not exist or may have invalid permissions. The long message contains more details about the directory and type of problem. The user should be directed to fix the build directory setting or package archive to add the missing directory. This directory must exist in the user's package archive, if there is not a *configure command*. If there is a configure command, it may create the directory.

Users should be directed to try building the package on their own computer.

chdir-config-dir

The *config directory* (relative to the *package directory*) specified by the user is invalid for the user's package; the directory may not exist or may have invalid permissions. The long message contains more details about the directory and type of problem. The user should be directed to fix the *configure directory* setting or package archive to add the missing directory. This directory must exist in the user's package archive if there is a *configure command*.

Users should be directed to try building the package on their own computer.

chdir-package-dir

The *package directory* specified by the user is invalid for the user's package; the directory may not exist or may have invalid permissions. The long message contains more details about the directory and type of problem. The user should be directed to fix the build directory setting or package archive to add the missing directory. This directory must exist in the user's package archive, if there is a config command.

Users should be directed to try building the package on their own computer.

configure

The configuration step for the user's package failed. This can be caused by the user selecting the wrong build system, missing libraries, the package not being compatible with the environment of the selected platform (operating system), or configuration command not existing.

Some packages download resources from the internet as part of the *configure* task such as the use of tools like `wget` or `curl` during the configuration. For these packages, a *configuration* task status of FAIL can be caused by the remote resource being moved or removed, an incorrect specification for the remote resource, an off-line or malfunctioning remote server, the system is run in an environment that is isolated from the internet, or a network issue such as an incorrectly configured firewall or router. The user can remove the need for the remote resource, or correct the problem and try again at a later time. If the failure is due to the network the SWAMP administrator will need to correct the problem, or the user can try again at a later time as the problem could be transient.

The long message of the task contains the exact command that failed to run successfully, and how it failed. Debugging this task can be accomplished by inspecting the output of the configuration task. The output and error output of the configuration step is found in the *config_stdout.out* and *config_stderr.out* files found in the build archive.

The long message, and the two files should be provided for users to inspect if the *build* task fails, so they can determine what failed, and how to update the package archive or build parameters.

Users should be directed to try building the package on their own computer.

end

Internal error. This task is informational and should always have a status of NOTE.

fetch-pkg-dependencies

Fetching the dependencies of the package using the build system's mechanism to fetch dependencies from the internet failed. This could be caused by the dependencies in the package's build configuration file no longer existing (or being incorrect), a networking issue such as an incorrectly configured firewall, or the remote package manager server not being up or functioning incorrectly. The short message contains the name of the build system being used, and current is either `gradle`, `maven`, `ant+ivy` or `pip`. The long message contains more information about the dependencies that failed to download. Users should be shown the long message to communicate what failed.

If it is due to the dependencies in the package's configuration no longer existing or being incorrect, the user will need to fix the package's dependencies, and upload an updated package archive. It is not atypical for old versions of dependencies to be removed from repositories.

If the failure is due to the network the SWAMP administrator will need to correct the problem, or the user can try again at a later time.

Occasionally a repository does not function correctly. The only recourse is to try again at a later time.

gem-install

Installing the user's gem package file failed. The output and error output of the gem install task is in gem_install.out and gem_install.err files found in the build archive. These files should be inspected for more information.

gem-unpack

Unarchiving the user's gem package file failed. The output and error output of the gem unpack task is in gem_unpack.out and gem_unpack.err files found in the build archive. These files should be inspected for more information.

install-os-dependencies

This is a failure to install OS dependencies using the OS package manager. The *run.out* file from the VM's /mnt/out directory may contain additional information. This could be caused by the package name specified by the user being incorrect for the selected platform, a networking issue such as an incorrectly configured firewall, or the remote OS package manager server not being up or functioning incorrectly.

Users should be allowed to inspect *run.out* to determine the cause of the failure. If it is due to an incorrect OS package name, the user can correct this. If the failure is due to remote OS package manager server not functioning correctly, the run may succeed at a later time. If the failure is due to a networking issue the administrator of the SWAMP will need to correct the issue.

install-pip-dependencies

Installation of the PIP dependencies specified by the user failed. The output and error output of the PIP install task is in pip_install.out and pip_install.err files found in the build archive. These files should be inspected for more information.

install-strace

Internal error. The strace archive in c-assess is corrupt or missing, or the file system is full. The *run.out* file from the VM's /mnt/out directory may contain additional information.

network

The VM's network is not functioning correctly. This is usually caused by an temporary issue on the hypervisor, and will work if restarted (a retry *task* is included to indicate this). If this issue persists, other parts of the infrastructure are having issues that the system administrator will need to correct.

no-build-setup

A build system type of *no build* was selected for C, C++ or Java source code. This fails for three reasons. The first is that no source files of the selected type (C/C++ or Java) were found in the *build directory* selected by the user. The second is that none of the source files found compiled without errors. The third is an internal error.

The short message of the task contains data to determine the type of error. If missing, an internal error occurred. If no files of the appropriate type were discovered in the package directory, the compilable and source-files will both be 0. If compilable is 0, source files were found, but none compiled without error.

The *source-compiles.xml* file from the VM's /mnt/out directory contains the complete list of source files discovered, if they compiled successfully, how they were compiled, and the output from the compiler. The contents of this file should be parsed and presented to the user to show the user the actual files discovered and if they were compilable.

package-checksum

The checksum of the package archive in the package.conf does not match the checksum of the package archive file. The package archive or package.conf is corrupt, or the checksum is invalid.

package-unarchive

The user's package archive is corrupt, the archive is missing, the archive is of an unknown format, or the file system is full. The long message of the task contains more information about command that was run, and why the failure occurred. The *run.out* file from the VM's /mnt/out directory may contain additional information.

The user should be shown the long message of task, and access to *run.out* to determine the problem with the archive. The user should then replace the archive with a valid version.

Users should be directed to try unarchiving the package archive on their own computer.

parse-results

Internal error. Error creating the parsed results archive. The result parser failed to convert output to SCARF. This can be caused by an unexpected format of result file, the result file being corrupted, or a bug in the result parser.

parsed-results-archive

Internal error. A possible cause is that /mnt/out is full. The *run.out* file from the VM's /mnt/out directory may contain additional information.

python2-install

Internal error. This task is indicates that the Python 2.x installation failed. Either python-assess is configured incorrectly, it is corrupt, or the disk is full.

python3-install

Internal error. This task is indicates that the Python 3.x installation failed. Either python-assess is configured incorrectly, it is corrupt, or the disk is full.

read-gem-spec

Failed to read the gem specification file from the user's gem package file. The output and error output of this task is in *gem-name*-err.spec file files found in the build archive. This file should be inspected for more information.

resultparser-unarchive

Internal error. The result parser archive is corrupt or missing, or the file system is full. The *run.out* file from the VM's /mnt/out directory may contain additional information.

results-archive

Internal error. Error creating the results archive. A possible cause is that /mnt/out is full. The *run.out* file from the VM's /mnt/out directory may contain additional information.

results-unarchive

Internal error. The results archive is corrupt or missing, or the file system is full. The *run.out* file from the VM's /mnt/out directory may contain additional information.

retry

Internal error. This task is informational and should always have a status of NOTE. If the status is NOTE, the VM should be recreated and rerun.

swamp-maven-plugin-install

Installation of SWAMP maven plugin failed. The main reason this task fails is due to networking issues or Maven Central unavailability. The long message of the task contains more information about the failure. Rerunning this in the future may succeed if the issue is a temporary problem with the network or Maven Central.

tool-install

Internal error. This task is indicates that the tool installation failed. Either the tool is configured

incorrectly, the tool is corrupt, or the disk is full.

tool-package-compatibility

The tool is incompatible with the version of the language of package's source code. The short message is the type of error, and the long message contains additional details about the incompatibility. If possible, the user interface should not allow such an assessment to run, but in practice it may be difficult to determine the package's language version. The user can select a different tool or version of the tool that is compatible. The short and long messages should be displayed to the user. Currently defined short messages and how to resolve them are as follows:

(Java language version)

The Java source language used during the build is incompatible with the tool (*error-prone* at this time). The user can try to modify the package to build using a newer version of Java, or they can use a different version of the tool (*error-prone* version 1.x).

(gcc version)

The version of `gcc` used during the build is incompatible with the tool. This occurs when using the *Parasoft C/C++test* tool (versions 9.5.4, 9.5.6 and 9.6.1) and a platform with a version of `gcc` of 5.0 or newer. Platforms with a `gcc` version of 5.0 or newer include *Fedora 22* and newer, and *Ubuntu 16.04* and newer. The user can use another platform with an older version of `gcc` or try to install, and use an older version of `gcc`.

(android+maven)

The android-lint tool is usually not configured properly for a Maven project. The user should update their project to use Maven or Gradle as a build system.

(known tool bug)

A known bug with the tool was triggered by the package. Until the tool is updated to fix the bug, users will not be able to get results using this tool with their package.

tool-runtime-compatibility

The tool is incompatible with the platform's runtime environment such as the JVM or operating system. The short message is the type of error, and the long message contains more information about the incompatibility. The user interface should not allow such an assessment to run. The user can select a different tool, version of tool, or platform that is compatible. The short and long messages should be displayed to the user.

tool-unarchive

Internal error. The tool archive is corrupt or missing, or the file system is full. The *run.out* file from the VM's /mnt/out directory may contain additional information.

validate-package

The *package classpath* specified for the Java Bytecode package is not valid. Jar files or directories listed in the classpath are not present in the package. The long message contains more information about the failure. Correct the *package classpath* option, or use a new archive that contains the missing files or directories.

5.3. Potential Issues In Successful Runs

This section describes conditions that are not failures in the task list that should be checked for and reported to the user:

Status of assess task is SKIP

If the status of the assess task is SKIP, and the short message is *no files*, this indicates that no files were found during the *build* task.

This may be intentional as the user submitted the packages as a web package and ran all web tools, but the package did not contain all files types such as CSS. In this case, the use of tool such as csslint would have a status of SKIP for the *assess* task.

If the package is a C/C++, Java, Ruby, Android, or Python package, the user's package archive or configuration of the package is almost certainly incorrect. Possible causes include:

1. The user built the software on their computer, and then archived the built package, so when the build is run as part of the build task, no files are compiled. Since the SWAMP assesses the files compiled during the build task, no files are assessed. The user should upload a freshly checked out version of their software, (or run their build system's *clean* target), archive the unbuilt package, and upload it again.
2. The package contains no source files.
3. The package's build or configuration step settings are incorrect.

no-build-setup task exists

The actual files assessed should be shown to the user as it may not contain all the files they expect, as it only assesses files in the *build directory*, and only those that compile. See no-build-setup in the debugging section for details.

5.4. Known Issues with Tools

This section documents known issues with tools. The issues are mainly do to tools being incompatible with the package's source language or the runtime environment used to run the tool.

error-prone

Version 2.0.0 and later of error-prone only requires the use of a Java source language 1.6 (Java 6) or later. When a package that uses an incompatible version of the Java language is used, the *tool-package-compatibility* task FAILs with a short message of *Java language version*.

Parasoft C/C++test

Versions 9.5.x and 9.6.x of Parasoft C/C++test do not recognize `gcc` compiler versions 5.0 and newer, and fails on platforms where the build uses such a version. Platforms with a `gcc` version of 5.0 or newer include *Fedora 22* and newer, and *Ubuntu 16.04* and newer. When a package that uses an incompatible version of the `gcc` is used, the *tool-package-compatibility* task FAILs with a short message of *gcc version*.

PHPMD

PHPMD has a bug that causes it to crash when analyzing certain types of PHP code. When this known failure occurs, the *tool-package-compatibility* task FAILs with a short message of *known tool bug*.

Android Lint

Most Android Java packages that use the Maven build system do not properly setup the environment for Android Lint to run correctly, causing the tool to fail. When an Android package is not configured correctly to use the android-lint tool, the *tool-package-compatibility* task FAILs with a short message of *android+maven*.