JobBox Documentation

Release 0.1

Zygmunt Krynicki

March 04, 2013

CONTENTS

JobBox is a collection of tests that aim to cover all aspects of modern computer hardware. It is intended to be used alongside with CheckBox and PlainBox projects as a source of actual **data**.

The name *JobBox* is a wordplay that matches the naming scheme of the CheckBox project group. The *job* part refers to CheckBox *jobs* which represent the smallest piece of testing that can be performed. In essence, JobBox is just a box with jobs.

CHAPTER

ONE

INSTALLATION

JobBox can be installed from a PPA (Personal Package Archive) (recommended) or PYPI (python package index) on Ubuntu Precise (12.04) or newer.

```
$ sudo add-apt-repository ppa:checkbox-dev/ppa && sudo apt-get update && sudo apt-get install jobbox
```

CHAPTER

TWO

USING JOBBOX

JobBox is a dependency of PlainBox and CheckBox. It will be used automatically whenever those tools are used.

CHAPTER

THREE

TABLE OF CONTENTS

3.1 Job Specification

3.1.1 CheckBox Job Definition

3.1.2 JobBox Job Definition

JobBox jobs are a proposed evolution of CheckBox jobs. The intent is to retain compatibility with existing jobs, provide simple migration path and to address the shortcomings of existing CheckBox job definition system.

JobBox jobs are defined as a structure with key (field) - value pairs. The following fields are defined by this document. They are typically stored just as CheckBox jobs, in RFC-822 records.

name

Unique job name, same as in CheckBox.

This field is used to identify and refer to each job. The name has to be unique. This can be simplified by using vendor name spaces.

vendor-namespace

Implicit prefix of the job name.

This field has no corresponding representation in CheckBox.

The intent is to allow for an ecosystem of job provides to freely collaborate in one environment without the fear of name collisions.

This mechanism allows each job vendor to ensure that the key requirement of job *name* to be globally unique is easy to enforce.

Implicitly all jobs inherit a vendor namespace from the document that defines them. This allows the cumbersome vendor string to be mostly forgotten about in normal development or usage.

The syntax is of the vendor namespace string **MUST** follow the pattern `year:reverse-domain-name:namespace` where `year` is a non-abbreviated year, `reverse-domain-name` is a name of a domain controlled by the vendor written down in the reverse domain name notation while `namespace` is an arbitrary string assigned by the vendor. Vendors **MUST** control the DNS domain name at the time the vendor namespace is being defined. The `year` component allows the system to meaningfully refer to the `reverse-domain-name` while ensuring that the eternal ownership of a DNS domain name is not a prerequisite.

Note: For the Canonical Hardware Certification Team, the vendor namespace is `2013:com.canonical:pes/hardware-certification`

classifiers

List of classifiers. See *classifiers*.

This allows scenario editors and testers to pick tests applicable to a particular class of software or hardware. Classifiers allow job developers to associate a set of hierarchical labels to each job.

This is a replacement of the implicit category association informally defined by the job `name` (which typically follows a `category/name` pattern) and the equally informal abuse of *local plugin* jobs to use `__category__` jobs to put generated jobs into some category.

The improvement over the base idea is that a job may naturally belong to multiple categories, such as being a *hardware:monitor*, *functionality:suspend* test that is typically performed on *target-environment:desktop* and *target-environment:laptop* as well as *target-environment:smart-screen* and belonging to the general *category:power-management*. This was impossible to express in the previous system and while no concrete usage patterns are recommended it believed that introduction of this capability will result in definition of official classifiers and usage patterns in which they are applied.

summary

Human-readable, short, one line summary of the job.

This field is intended for *scenario* developers that wish to assemble a scenario out of the library of available test. Keeping the summary short and one-line will allow efficient representation of tests as a list of items. The summary may be more descriptive than the raw job name might otherwise be.

description

Human-readable description of arbitrary size.

This field is always displayed to the test operator. It may contain instructions vital for the proper verification of the outcome of the test.

Note: Perhaps we want to split the description (which is something we could display on websites and scenario editors) from actual instructions for the test operator.

startup

Keyword identifying how a test job is started.

The default value is `immediate`.

The allowed values are:

`immediate` The job can be started immediately.

`triggered` The job needs to be started explicitly by the test operator. This is intended for things that may be timing-sensitive or may require the tester to understand the necessary manipulations that he or she may have to perform ahead of time.

The test operator may select to skip certain tests, in that case the *outcome* is `skip`.

This is a replacement for a collection of CheckBox plugin types, including `manual`, `shell`, `user-verify` and `user-interact`.

verification

Keyword identifying how a test job is qualified as passing or failing.

The default value is `automatic`.

The allowed values are:

`automatic` The outcome is automatically verified based on the return status of the command or script embedded into the job. Depending on the return code the *outcome* of a job is either `pass` or `fail`

`manual` The outcome is manually verified by asking a question to the user. Typically this question is a form of yes-or-no question. Depending on the input from the test operator the *outcome* of a job is either `pass` or `fail`.

script

A **bash** script to execute.

There is no default value.

The script can be a multi-line command that is executed as a part of this job on *job startup*. The script can be as long as desired but it is suggested to keep it reasonably short and transform overly complicated scripts to actual standalone programs so that they can be treated as every other piece of software.

The output and return code of the script may affect the rest of the system. See the `script-output` and `verification` fields.

script-output

A keyword identifying what to do with the output produced by the script.

The default value is `hide`.

The allowed values are:

`hide` Both stdout and stderr are hidden from the tester.

The test operator may choose to reveal the output and inspect it if needed. The output is not stored after all testing is finished.

This is a replacement for CheckBox *shell plugin*.

The improvement over the base idea is that not all commands produce interesting output that should be immediately displayed. Except for jobs that require the test operator to carefully read the output this provides a good default value without compromising on the ability to access this data in all cases, if required.

`reveal` Both stdout and stderr are displayed to the tester.

The test operator may choose to hide the output. The output is not stored after all testing is finished.

This is a replacement for CheckBox *shell plugin*.

The improvement over the base idea is that it allows to identify jobs that depend on the test operator being able to see their output. Such scripts may be subject to extra scrutiny or localization requirements to ensure testers can comprehend the output if that is required by the test.

attach-text The stdout is converted to a text attachment.

All of the bytes produced on stdout must form a valid Unicode string encoded with UTF-8. Any characters that cannot be interpreted as UTF-8 are replaced with supplementary characters.

The stderr is discarded.

This is a replacement for CheckBox *attachment plugin*.

The improvement over the base idea is that we clearly differentiate text and binary attachments and there is a well-defined strategy for handling corrupted output.

attach-binary The stdout is converted to a binary attachment.

The stderr is discarded.

When using `attach-binary` you **MUST** also set the `attachment-mime-type` field.

This is a replacement for CheckBox *attachment plugin*.

The improvement over the base idea is that we clearly differentiate text and binary attachments and there is a way to specify MIME type which may aid test reviewers and downstream storage systems. For example a web-based test result browser may offer to download or display attachments in a way optimized to their content.

parse-resource The stdout is converted to text as described in `attach-text` and parsed as list of RFC-822 records separated by an empty line. The result is interpreted as a list of *resource definitions*.

The stderr is discarded.

This is a replacement for CheckBox *resource plugin*.

parse-job The stdout is converted to text as described in `attach-text` and parsed as a list of RFC-822 records separated by an empty line. The result is interpreted as a list of *job definitions*.

The stderr is discarded.

When using `parse-job` you **MUST** define `parse-job-pattern` to indicate the naming pattern of jobs that **MAY** be generated by the script. Jobs that are parsed but do not match that pattern are discarded.

This is a replacement for CheckBox *local plugin*.

The improvement over the base idea is that it allows PlainBox to build a full graph of all jobs and automatically discover job dependencies without executing any code.

parse-job-pattern

The pattern of jobs that may be defined by this job.

There is no default value.

This field describes the pattern of jobs names that may be defined by a job using `script-output` equal to `parse-job`.

The pattern **MUST** be a valid job name and **MUST NOT** have a `vendor-namespace` (in that it can only generate jobs in the same vendor namespace as the job definition that embeds the `script`). This ensures that no cross-vendor job generation is possible and in turn that each vendor can enforce and control their namespace.

The pattern **SHOULD** include at least one `wildcard`. The syntax of the wildcard is `{NAME}` where `NAME` is the name of the wildcard.

Note: A job that generates arbitrary jobs using a match-everything pattern such as `{}` will be rejected in practice. For details see the rules on pattern job collisions. The rule states that if two jobs contend to generate the same job then the

shortest pattern (not including the name of each wildcard) is discarded. This allows to resolve conflicts by allowing most-specialized pattern to win.

---

attachment-mime-type

The *MIME* type of attachment generated by this job.

There is no default value.

This field is mandatory to jobs that have `script-output` equal to `attach-binary`.

user

Name of the system user the script should be executed as.

There is no default value.

This is typically used to run certain scripts as `root`.

depends

A list of job names that describe test-level dependencies of the job.

There is no default value.

The list of dependencies must refer to existing jobs from the same vendor or fully qualified jobs from any vendor. The job is *ready* when **ALL** the term:*outcome* of all referenced jobs is `pass`.

The list **MAY** refer to a job generated by a job using `script-output` equal to `parse-job`. Jobs with unknown dependencies (not defined anywhere in the system) are removed from consideration. Jobs that have circular dependencies are also removed from consideration.

When the dependency is not met the *outcome* of a job is `fail`

requires

A list of *requirement programs* that describe system-level dependencies of the job.

There is no default value.

The list of dependencies is evaluated against all *resources*. The job is *ready* when **ALL** resource programs evaluate to true.

When the dependency is not met the *outcome* of a job is `not-supported`

3.2 Job Coverage

This document summarizes job / test coverage as provided by JobBox

3.2.1 Hardware Jobs

Category Name	Hardware in scope	Description
audio	Sound card, headphone jacks, etc	basic functionality tests
benchmarks	CPUs, GPUs, Disks, Network	assorted benchmarks
bluetooth	Bluetooth hardware	basic functionality tests
camera	video input devices, webcams	basic functionality tests
cpu	CPU	basic cpu features
disk	Hard drives and SSD	basic functionality and capacity tests
esata	eSATA-connected disks	
expresscard	Express cards	
fingerprint	Fingerprint scanners	
firewire	Firewire-connected disks	
floppy	Floppy disks	
graphics	Integrated and discrete GPUSs	
input	Mouse, touchpad and touchscreen	
keys	Special hardware keys	
led	Indicator LEDs	
mediacard	Memory card readers (SD, uSD, xD)	
memory	RAM	
monitor	CRT and LCD monitors	
networking	Ethernet, WiFI and modem	
optical	optical drives (CDs, DVDs)	
peripheral	external printer and modem tests	
stress	entire machine	extended stress testing

3.2.2 Software Jobs

Category Name	Software in scope
codecs	software audio codecs
daemons	essential system daemons
install	apt-get and oem-config
panel_clock_test	date and time display and control
panel_reboot	reboot control
piglit	various piglit tests (graphics)
rendercheck	various rendercheck tests (graphics)
server-services	typical server services

3.2.3 Power management jobs

Category Name	Description
hibernate	whole-system suspend-to-disk
suspend	whole-system suspend-to-ram
power-management	fine-grained ACPI tests

3.2.4 Misc jobs

Category Name	Description
info	hardware information logs
local	local jobs (checkbox legacy)
miscellanea	other assorted jobs
resource	software and hardware probes that enable specific tests
smoke	smoke tests for checkbox job management

3.3 Glossary

3.4 ChangeLog

Note: This changelog contains only a summary of changes. For a more accurate accounting of development history please inspect the source history directly.

3.4.1 PlainBox 0.1 (unreleased)

- Initial release

CHAPTER

FOUR

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*