Linaro Dashboard Bundle Documentation

Release 1.7

Zygmunt Krynicki

October 21, 2011

CONTENTS

See Also:

To see 1.3 format documentation see *Raw Schema (1.3)*

CHAPTER

ONE

INSTALLATION

1.1 Prerequisites

This package has the following prerequisites:

- linaro-json
- simplejson
- versiontools

To run the test suite you will also need:

- testtools
- testscenarios

To build the documentation from source you will also need:

- sphinx

1.2 Installation Options

There are several installation options available:

1.2.1 Using Ubuntu PPAs

For Ubuntu 10.04 onward there is a PPA (personal package archive):

- ppa:linaro-validation/ppa

This PPA has only stable releases. To add it to an Ubuntu system use the add-apt-repository command:

```
sudo add-apt-repository ppa:linaro-validation/ppa
```

After you add the PPA you need to update your package cache:

```
sudo apt-get update
```

Finally you can install the package, it is called *python-linaro-dashboard-bundle*:

```
sudo apt-get install python-linaro-dashboard-bundle
```

1.2.2 Using Python Package Index

This package is being actively maintained and published in the Python Package Index. You can install it if you have pip tool using just one line:

```
pip install linaro-dashboard-bundle
```

1.2.3 Using source tarball

To install from source you must first obtain a source tarball from either pypi project page or from Launchpad project page. To install the package unpack the tarball and run:

```
python setup.py install
```

You can pass –user if you prefer to do a local (non system-wide) installation.

Note: To install from source you will need distutils (replacement of setuptools) They are typically installed on any Linux system with python but on Windows you may need to install that separately.

CHAPTER

TWO

GLOSSARY

LAVA uses various terminology that is sometimes confusing as many of the words are commonly used and have many meanings. This glossary should help to understand the rest of the documentation better.

Test A test is a piece of code that can be invoked to check something. LAVA assumes that each test has a unique identifier. A test is also a container of test cases.

Test Case A test case is a sub-structure of a Test. Typically individual test cases are separate functions or classes compiled into one test executable. Test cases have identifiers, similar to each test, however the test case identifier needs to be unique only among the test it is a part of.

Test Run A test run is an act of running a test on some hardware in some software (typically operating system kernel, set of installed and configured packages).

Bundle, results bundle A document format that can describe test results. The document is defined as a JSON document matching any of the version of the bundle schema *Schema Description (1.3)*

Software Context A non-exhaustive description of the software configuration of the device that is performing a test run.

Hardware Context A non-exhaustive description of the hardware configuration of the device that is performing a test run. One important aspect is that currently hardware context is limited to one device only.

CHAPTER

THREE

DASHBOARD BUNDLE SCHEMA

Dashboard bundle is a JSON document. As such it can be transmitted,

manipulated and processed with various different tools, languages and toolkits.

This specification intends to formalize the schema of supported documents as well as give additional insight into how each field was designed to be used.

Throughout this document we will be using path references. That is, starting with the root object a sequence of traversals, either object access (dictionary access in Python) or array access (list access in Python).

For array indices we will also use wild-card character $\star$ to indicate that all indices of the specified array are considered equal.

3.1 Schema Description (1.3)

This document specifies the semantics of the 1.3 format.

3.1.1 Root object

The root object contains only two properties.

1. The format specified (`format`)
2. The array of test runs (`test_runs`).

The format needs to be a fixed value, namely the string `"Dashboard Bundle Format 1.3"`. The test array must have zero or more test run objects.

3.1.2 Test run objects

This is the most fundamental piece of actual data. Test runs represent an act of running a test (a collection of logical test cases wrapped in a testing program) in a specific hardware context (currently constrained to a single machine) and software context (currently constrained to installed software packages and abstract source references, described later)

Each test run has multiple mandatory and a few more optional properties. The mandatory properties are as follows:

1. The unique identifier assigned by the log analyzer (`analyzer_assigned_uuid`). Historically each test run was created by analyzing the text output of a test program. The analyzer was the only entity that we could control so it had to assign identifiers to test run instances. This identifier must be an UUID (that is, a string of 36 hexadecimal characters in the following format

`[0-9a-f]{8}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{12}`. While upper and lower case letters are not distinguished lowercase values are preferred.

2. Analyzer time-stamp (`analyzer_assigned_date`). This one of the few time stamps stored in a bundle. This one is authoritatively assigned by the log analyzer. It does not indicate when a test was actually started but rather when it was read and processed by the analyzer.
3. Time check performed by the log analyzer (`time_check_performed`). Since LAVA uses analyzers that often run on the target device the value of the `analyzer_assigned_date` field was inherently unreliable. To differentiate between a device that has somewhat trusted time and date settings (such as a device running NTP service and having battery-powered real-time clock) from others we have introduced this boolean field. The interpretation remains open to the user but whenever one creates test run object it should be set to false unless the time can be trusted enough.
4. A free-form collection of attributes (`attributes`). This is a simple object with arbitrary properties. It can be used as a simple extension point to append data that test runs cannot natively represent. To make this field database friendly it has been limited to a string-only values. That is, each property can be named freely but must point to a string value. In particular nested objects are not allowed.
5. An array of tags (`tags`). Each tag is a string in the following format: `[a-z0-9-]+`. There should be no duplicates in the list although this is not enforced at format level yet.
6. Test identifier (`test_id`). Test identifier is an unique string that designates a test. Tests are simply a logical container of test cases. The identifier must be a string matching the following regular expression `[a-z0-9.-]+`. It is recommended that reverse domain name scheme is used for test ID. Following this pattern would allow one to construct logical containers rooted at the top-level-domain owned by test owner.
6. An array of test results (`test_results`). Test results are described in a dedicated section below.
7. An array of attachments (`attachments`). Attachments are also described in a dedicated section below. It is worth mentioning that the format for storing attachments has changed and was different in 1.1 format and earlier. When processing unknown documents make sure to validate and evolve the format to the one that is recognized by your program.
8. The hardware context in which the test was invoked (`hardware_context`).
9. The software context in which the test was invoked (`software_context`).

3.1.3 Test results objects

Test result is an object with three essential and many less used properties. The most important properties are:

1. `test_case_id` that identifies the test case (the test case identifier is a string, unique to the test it belongs to, the test is recored, as `test_id` in a test run object)
2. `result` that encodes the outcome of the test. Currently only four values are allowed here, they are 'pass', 'fail', 'skip' and 'unknown'. The first three are rather obvious, the last one has a special purpose. It is the recommended value of a benchmark. Essentially since benchmarks are not pass-fail tests (the measurements can be dissatisfying from a certain point of view but this is relative) and the result code is mandatory we have decided to use it by default for all benchmarks.
3. `measurement` (optional) that encodes the benchmark result. This is a single decimal number. The system handles this without precision loss so don't be afraid to use it for things that would be impractical with simple floating point numbers.

Since there is only one measurement allowed per test case our recommendation is to store each measurement (each number that comes out of some test code) as a separate result with an unique test case identifier. The only exception, perhaps, would be a case where the test case is really the same and subsequent results sample the same value periodically. We have not explored this area yet and it is likely that for profiling we'll introduce a dedicated schema.

3.1.4 Attachments objects

Attachments are simple objects that store the `pathname` (or just name), `mime_type` and either the raw `content` (here stored as base64-encoded string) or a reference to a copy in a `public_url` field.

We don't recommend storing very large attachments directly in the bundle. While it will work the performance of handling such bundles will pale in comparison to the same bundle with attachments stored externally.

3.1.5 Software context objects

Software context is a object that describes the "software" part of the environment in which a test was running. Currently this is limited to three pieces of information:

1. A list of packages that was installed (`packages`). Each package is a simple object with `name` and `version`. So far we used this system for Debian packages but it should map fine for RedHat and Android as well.
2. A list of software sources. A source is a loose association (it does not tell you exactly how the source was used/present on the device) between the test and some precise source code. The source code is identified by a particular commit (revision, check-in, change set, recored as `branch_revision`) in a particular version control system (`branch_vcs`) that is found at a particular, version control system specific, url (`branch_url`). Since many of our users use Launchpad we also decided to store the name of the project. Thus we associated the `project_name` property with a launchpad project name. There is one extra attribute, that is optional, which records the date and time of the commit (`commit_timestamp`). While we realise that time stamps do not form an ancestor-descendant chain (definitely not in distributed version control systems) they non the less provide useful context.
3. A name of the system `image`. There is very little rationale behind this field apart from end-user usefulness (what was the user running?). We recommend to store the output of `lsb_release --description --short` if appropriate. Note that this does not allow one to uniquely identify images (at the very least this was not the indented usage of this field)

3.1.6 Hardware context objects

Hardware context is an object that holds an array of devices in its sole property (`devices`). Each device object has a type (`device_type`), a human-readable description (`device_description`) and an arbitrary set of attributes (contained in `attributes`). Each attribute may be a string or an integer.

There are some device types currently used by LAVA. The convention is "device, dot, device type", for example we currently have `device.usb`, `device.cpu` and `device.memory`.

In practice devices are modeled ad-hoc, as the need arises. The attributes can store enough information to be looked up later that we did not try to standardize how all actual devices should be described (there is no strict schema for, say, PCI cards). We hope to see a set of mini-standards developing around this concept where a device of a particular class has a standardized set of attributes that everyone agrees on.

3.2 Raw Schema (1.3)

For those versed in JSON-Schema we'd like to recommend reading the full schema directly. Note that this is the latest format schema, the full catalogue is included in the source distribution.

```
1  {
2      "description": "DashboadBundle object",
3      "type": "object",
4      "additionalProperties": false,
```

```
    "properties": {
        "format": {
            "description": "Document format identifier.",
            "type": "string",
            "enum": [
                "Dashboard Bundle Format 1.3"
            ]
        },
        "test_runs": {
            "description": "Array of TestRun objects",
            "type": "array",
            "optional": true,
            "items": {
                "description": "TestRun object",
                "type": "object",
                "additionalProperties": false,
                "properties": {
                    "analyzer_assigned_uuid": {
                        "description": "UUID that was assigned by the log analyzer during processing'
                        "type": "string"
                    },
                    "analyzer_assigned_date": {
                        "description": "Time stamp in ISO 8601 format that was assigned by the log ar
                        "type": "string",
                        "format": "date-time"
                    },
                    "time_check_performed": {
                        "description": "Indicator on whether the log analyzer had accurate time infor
                        "type": "boolean"
                    },
                    "attributes": {
                        "description": "Container for additional attributes defined by the test and t
                        "type": "object",
                        "optional": true,
                        "additionalProperties": {
                            "description": "Arbitrary properties that are defined by the test",
                            "type": "string"
                        }
                    },
                    "tags": {
                        "description": "An optional array of tags that are associated with this test
                        "type": "array",
                        "optional": true,
                        "items": {
                            "description": "Tag name",
                            "type": "string"
                        }
                    },
                    "test_id": {
                        "description": "Test identifier. Must be a well-defined (in scope of the dash
                        "type": "string"
                    },
                    "test_results": {
                        "description": "Array of TestResult objects",
                        "type": "array",
                        "items": {
                            "description": "TestResult object",
                            "type": "object",
```

```
                "additionalProperties": false,
                "properties": {
                    "test_case_id": {
                        "description": "Identifier of the TestCase this test result came
                        "type": "string",
                        "optional": true
                    },
                    "result": {
                        "description": "Status code of this test result",
                        "type": "string",
                        "enum": ["pass", "fail", "skip", "unknown"]
                    },
                    "message": {
                        "description": "Message scrubbed from the log file",
                        "type": "string",
                        "optional": true
                    },
                    "measurement": {
                        "description": "Numerical measurement associated with the test re
                        "type": "number",
                        "optional": true,
                        "requires": "test_case_id"
                    },
                    "units": {
                        "description": "Units for measurement",
                        "type": "string",
                        "optional": true,
                        "requires": "measurement"
                    },
                    "timestamp": {
                        "description": "Date and time when the test was performed",
                        "type": "string",
                        "optional": true,
                        "format": "date-time"
                    },
                    "duration": {
                        "description": "Duration of the test case. Duration is stored in
                        "type": "string",
                        "optional": true
                    },
                    "log_filename": {
                        "description": "Filename of the log file which this test result w
                        "type": "string",
                        "optional": true
                    },
                    "log_lineno": {
                        "description": "Precise location in the log file (line number)",
                        "type": "integer",
                        "optional": true,
                        "requires": "log_filename"
                    },
                    "attributes": {
                        "description": "Container for additional attributes defined by te
                        "type": "object",
                        "optional": true,
                        "additionalProperties": {
                            "description": "Arbitrary properties that are defined by the
                            "type": "string"
```

```
                                }
                            }
                        }
                    }
                },
                "attachments": {
                    "description": "Array of attachments",
                    "optional": true,
                    "type": "array",
                    "items": {
                        "type": "object",
                        "additionalProperties": false,
                        "properties": {
                            "pathname": {
                                "description": "Attachment pathname",
                                "type": "string"
                            },
                            "mime_type": {
                                 "description": "Attachment MIME type",
                                 "type": "string"
                            },
                            "content": {
                                "description": "Attachment content encoded as base64 string with
                                "type": "string",
                                "optional": true
                            },
                            "public_url": {
                                "description": "Public URL of this attachment",
                                "type": "string",
                                "optional": true
                            }
                        }
                    }
                },
                "hardware_context": {
                    "description": "Description of the hardware context in which this test was ru
                    "type": "object",
                    "optional": true,
                    "additionalProperties": false,
                    "properties": {
                        "devices": {
                            "description": "Array of HardwareDevice objects",
                            "type": "array",
                            "items": {
                                "description": "HardwareDevice object",
                                "type": "object",
                                "properties": {
                                    "device_type": {
                                        "type": "string",
                                        "description": "Device type"
                                    },
                                    "description": {
                                        "type": "string",
                                        "description": "Human readable description of the device'
                                    },
                                    "attributes": {
                                        "description": "Container for additional attributes defi
                                        "type": "object",
```

```
                                    "optional": true,
                                    "additionalProperties": {
                                        "description": "Arbitrary properties that are defined
                                        "type": ["number", "string"]
                                    }
                                }
                            },
                            "additionalProperties": false
                        }
                    }
                }
            },
            "software_context": {
                "description": "Description of the software context in which this test was ru
                "type": "object",
                "additionalProperties": false,
                "optional": true,
                "properties": {
                    "image": {
                        "description": "SoftwareImage object",
                        "type": "object",
                        "optional": true,
                        "additionalProperties": false,
                        "properties": {
                            "name": {
                                "description": "Name of the operating system image",
                                "type": "string"
                            }
                        }
                    },
                    "sources": {
                        "type": "array",
                        "optional": true,
                        "items": {
                            "type": "object",
                            "additionalProperties": false,
                            "properties": {
                                "project_name": {
                                    "type": "string"
                                },
                                "branch_vcs": {
                                    "type": "string",
                                    "enum": ["bzr", "git", "svn"]
                                },
                                "branch_url": {
                                    "type": "string"
                                },
                                "branch_revision": {
                                    "type": ["string", "integer"]
                                },
                                "commit_timestamp": {
                                    "type": "string",
                                    "format": "date-time",
                                    "optional": true
                                }
                            }
                        }
                    },
```

```
                                "packages": {
                                    "description": "Array of SoftwarePackage objects",
                                    "type": "array",
                                    "optional": true,
                                    "items": {
                                        "description": "SoftwarePackage object",
                                        "type": "object",
                                        "additionalProperties": false,
                                        "properties": {
                                            "name": {
                                                "description": "Package name",
                                                "type": "string"
                                            },
                                            "version": {
                                                "description": "Package version",
                                                "type": "string"
                                            }
                                        }
                                    }
                                }
                            }
                        }
                    }
                }
            }
        }
    }
}
```

3.3 Example Bundles

This document contains a few example bundles. More bundles are included in the source distribution.

3.3.1 Smallest bundle

Actually all that is needed is the format. The rest is entirely optional so this bundle is just fine.

```
{
  "format": "Dashboard Bundle Format 1.0"
}
```

3.3.2 Everything (1.3)

This bundle has everything possible at once. It is also using the most recent format (currently 1.3)

```
{
  "format": "Dashboard Bundle Format 1.3",
  "test_runs": [
    {
      "analyzer_assigned_date": "2010-11-14T13:42:31Z",
      "analyzer_assigned_uuid": "01234567-0123-0123-0123-01234567890A",
      "test_id": "example test id",
      "time_check_performed": false,
      "attributes": {
```

```
      "attr1": "value1",
      "attr2": "value2"
    },
    "tags": [
      "example-content",
      "example-content-format-1.3"
    ],
    "attachments": [
      {
        "pathname": "attachment1.txt",
        "mime_type": "text/plain",
        "content": "bGluZTEKbGluZTIKbGluZTMK"
      },
      {
        "pathname": "attachment2.txt",
        "mime_type": "text/plain",
        "content": "b3RoZXIgbGluZTEK",
        "public_url": "http://www.example.org/attachment2.txt"
      }
    ],
    "test_results": [
      {
        "test_case_id": "example test case id",
        "result": "pass",
        "message": "example message",
        "measurement": 3.5,
        "units": "s",
        "timestamp": "2010-11-14T13:49:56Z",
        "duration": "0d 1s 134us",
        "log_filename": "attachment1.txt",
        "log_lineno": 2,
        "attributes": {
          "test result attribute": "value"
        }
      }
    ],
    "hardware_context": {
      "devices": [
        {
          "device_type": "example device type",
          "description": "example device",
          "attributes": {
            "hw attr1": "value1",
            "hw attr2": "value2"
          }
        }
      ]
    },
    "software_context": {
      "packages": [
        {
          "name": "pkg1",
          "version": "version1"
        },
        {
          "name": "pkg2",
          "version": "version2"
        }
```

```
68          ],
69          "image": {
70            "name": "example os image"
71          },
72          "sources": [
73            {
74              "branch_revision": 93556,
75              "branch_url": "lp:gcc-linaro/4.4",
76              "branch_vcs": "bzr",
77              "commit_timestamp": "2010-09-07T14:49:43Z",
78              "project_name": "linaro-gcc"
79            }
80          ]
81        }
82      }
83    ]
84  }
```

CHAPTER

FOUR

USAGE

4.1 Loading a document

There are two functions that perform this task. For streams there is `DocumentIO.load()`, for simple strings there is `DocumentIO.loads()`.

Each function does three things:

- read the document text
- deserialize it properly (using propert configuration for parsing floating point numbers).
- check the document by inspecting the format and validating it against a built-in schema

If you use this API there is no need to double-check the contents of the document you've got. It's going to match the description of the schema.

Example of reading something from a file:

```
with open("bundle.json", "rt") as stream:
    format, bundle = DocumentIO.load(stream)
    print "Loaded document type: %s" % format
```

The error path is a little more complex. Loading can fail at the following levels:

1. You can get an IOError while reading from the stream
2. You can get a ValueError or JsonDecodeError depending on which version of simplejson you have while processing the text
3. You can get a DocumentFormatError when the format string is missing or has an unknown value
4. You can get a ValidationError when the document does not match the format

4.2 Saving a document

There is just one function for saving a document `DocumentIO.dump()`. It will always validate the document before saving it so there is little chance of producing invalid files this way.

Currently this function uses a hardcoded human-readable profile. If you care about representation efficiency use a compressed storage such as `gzip.GzipFile`.

Example of writing a bundle to a file:

```
with open("bundle.json", "wt") as stream:
    bundle = {"format": "Dashboard Bundle Format 1.0"}
    DocumentIO.dump(stream, bundle)
```

4.3 Checking document for errors

To validate document for correctness you can use `DocumentIO.check()`.

Note: Most of the time you don't need to validate a document explicitly. It is automatically validated when loading and saving.

4.4 Converting older documents to current format

To convert a document to the most recent format you can use `DocumentEvolution.evolve_document()`. It is safe to call this method on any valid document. If you just need to check if the document is using the most recent format call `DocumentEvolution.is_latest()`.

CHAPTER

FIVE

CODE DOCUMENTATION

5.1 Document Manipulation APIs

class `linaro_dashboard_bundle.io.`**`DocumentIO`**

Document IO encapsulates various (current and past) file formats and provides a single entry point for analyzing a document, determining its format and validating the contents.

classmethod `check`(*doc*)

Check document format and validate the contents against a schema.

Discussion The document is validated against a set of known versions and their schemas.

Return value String identifying document format

Exceptions

linaro_json.ValidationError When the document does not match the appropriate schema.

linaro_dashboard_bundle.errors.DocumentFormatError When the document format is not in the known set of formats.

classmethod `dump`(*stream*, *doc*, *human_readable=True*, *sort_keys=False*)

Check and save a JSON document to the specified stream

Discussion The document is validated against a set of known formats and schemas and saved to the specified stream.

If human_readable is True the serialized stream is meant to be read by humans, it will have newlines, proper indentation and spaces after commas and colons. This option is enabled by default.

If sort_keys is True then resulting JSON object will have sorted keys in all objects. This is useful for predictable format but is not recommended if you want to load-modify-save an existing document without altering it's general structure. This option is not enabled by default.

Return value None

Exceptions

Other exceptions This method can also raise exceptions raised by DocumentIO.check()

classmethod `dumps`(*doc*, *human_readable=True*, *sort_keys=False*)

Check and save a JSON document as string

Discussion The document is validated against a set of known formats and schemas and saved to a string.

If human_readable is True the serialized value is meant to be read by humans, it will have newlines, proper indentation and spaces after commas and colons. This option is enabled by default.

If sort_keys is True then resulting JSON object will have sorted keys in all objects. This is useful for predictable format but is not recommended if you want to load-modify-save an existing document without altering it's general structure. This option is not enabled by default.

Return value JSON document as string

Exceptions

Other exceptions This method can also raise exceptions raised by DocumentIO.check()

classmethod **load**(*stream*, *retain_order=True*)
Load and check a JSON document from the specified stream

Discussion The document is read from the stream and parsed as JSON text. It is then validated against a set of known formats and their schemas.

Return value Tuple (format, document) where format is the string identifying document format and document is a JSON document loaded from the passed text. If retain_order is True then the resulting objects are composed of ordered dictionaries. This mode is slightly slower and consumes more memory.

Exceptions

ValueError When the text does not represent a correct JSON document.

Other exceptions This method can also raise exceptions raised by DocumentIO.check()

classmethod **loads**(*text*, *retain_order=True*)
Same as load() but reads data from a string

exception linaro_dashboard_bundle.errors.**DocumentFormatError**(*format*)
Exception raised when document format is not in the set of known values.

You can access the :format: property to inspect the format that was found in the document

class linaro_dashboard_bundle.evolution.**DocumentEvolution**
Document Evolution encapsulates format changes between subsequent document format versions. This is useful when your code is designed to handle single, for example the most recent, format of the document but would like to interact with any previous format transparently.

classmethod **evolve_document**(*doc*, *one_step=False*)
Evolve document to the latest known version.

Runs an in-place evolution of the document *doc* converting it to more recent versions. The conversion process is lossless.

Parameters

- **doc** (*JSON document, usually python dictionary*) – document (changed in place)
- **one_step** – if true then just one step of the evolution path is taken before exiting.

Return type None

classmethod **is_latest**(*doc*)
Check if the document is at the latest known version

CHAPTER

SIX

VERSION HISTORY

6.1 Version 1.7

- Document most of the *Raw Schema (1.3)* schema and *Schema Description (1.3)* (recommended)
- Provide some example documents *Example Bundles*

6.2 Version 1.6

- Add support for 1.3 format. This format makes it possible to tag test runs with arbitrary tag names. Tagging allows one to organize results more flexibly.

6.3 Version 1.5

- Add support for 1.2 format. This format makes attachments more flexible by allowing one to omit the contents of the attachment and store a public URL instead
- Allow 'svn' version control systems in source references (schema.properties.test_runs.items.properties.software_context.properties
- Move everything away from __init__.py so that we can safely import it in setup.py

6.4 Version 1.4

- Add support for DocumentIO.loads() and load() retain_order keyword argument. It defaults to True (preserving old behavior) and allows for either safe load-modify-save cycles that minimise differences or more efficient processing as plain python dictionaries.
- Add support for DocumentIO.dumps() and dump() human_readable keyword argument. It defaults to True (preserving old behavior) and allows to control the desired format of the output document. For machine processing or storage the compact option will save a few bytes.
- Add support for DocumentIO.dumps() and dump() sort_keys keyword argument. It defaults to False (preserving old behavior) and allows to create predictable documents from plain python dictionaries that would otherwise result in random ordering depending on python implementation details.

6.5 Version 1.3

- Add mime_type support to attachments in 1.1 format. Seal the 1.1 format.
- Add support for document evolution for between 1.0.1 and 1.1 formats.
- More unit tests (evolution from 1.0.1 to 1.1, lossless IO fro 1.1 format)

6.6 Version 1.2

- New document format with support for binary attachments and precise source information (extended software context)
- Refresh installation instructions to point to the new PPA, provide links to lp.net project page and pypi project page.

6.7 Version 1.1.1

- Sign source package
- Fix installation problem with pip due to versiontools not being available when parsing initial setup.py

6.8 Version 1.1

- Project renamed to linaro-dashboard-bundle
- Started using pypi for hosting releases and documentation
- Add initial parts of class-based JSON schema

6.9 Version 1.0

- First public release

CHAPTER

SEVEN

INDICES AND TABLES

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

PYTHON MODULE INDEX

l