productmd Documentation

Release 1.0

Daniel Mach <dmach@redhat.com>

Nov 23, 2018

Contents

ProductMD is a Python library providing parsers for metadata related to composes and installation media.

Contents:

CHAPTER 1

Terminology

1.1 Release

(Product) *Release* is a collection of software with unique identity and life cycle.

Notes

- It is advised to separate updates from minor releases for better content tracking.
- Release should be immutable - once a release is finished, it's content or definition must never change.
- When designing metadata or data schemas, *Releases* should carry all information even if there is certain duplicity. *Product Versions* and *Products* should not carry any data. These are only for grouping and organizing *Releases*.

Examples

- Fedora 21 (f-21)
- Fedora 21 Updates (f-21-updates)
- Red Hat Enterprise Linux 7.0 (rhel-7.0)
- Red Hat Enterprise Linux 7.1 (rhel-7.1)
- Satellite 5.6.0 for Red Hat Enterprise Linux 7 (satellite-5.6.0-rhel7)

1.2 Product Version

Product Version is a group of product *Releases* with the same name and major version.

Examples

- Fedora 21 (f-21)
- Red Hat Enterprise Linux 7 (rhel-7)
- Satellite 5.6 (satellite-5.6)

1.3 Product

Product is a group of *Product Versions* with the same name.

Examples

- Fedora (f)
- Red Hat Enterprise Linux (rhel)
- Satellite (satellite)

1.4 Base Product

Base Product usually indicates operating system a *Release* runs on. In reality it often matches with *Product Version* of that OS.

Examples

- Fedora 21
- Red Hat Enteprise Linux 7

1.5 Compose

Compose is a *Release* snapshot with an unique ID derived from *Release* and compose date.

Notes

- *Compose* should consist of well defined building blocks, ideally described by metadata (.treeinfo, repodata, . . .)

Examples

- RHEL-7.0-YYYYMMDD.0
- Satellite-5.6.0-RHEL-7-YYYYMMDD.0

1.6 Variant

Both *Composes* and *Releases* are divide into *Variants*. These contain different *Release* content subsets targeting different users (Server, Workstation).

Examples

- RHEL-7.0-YYYYMMDD.0 / Server
- RHEL-7.0-YYYYMMDD.0 / Workstation

1.7 Tree

Tree is a *Variant* for specified architecture.

Examples

- RHEL-7.0-YYYYMMDD.0 / Server / x86_64
- RHEL-7.0-YYYYMMDD.0 / Server / ppc64
- RHEL-7.0-YYYYMMDD.0 / Workstation / x86_64

CHAPTER 2

Versioning

Versioning should be as simple as possible. Recommended schema is dot separated numbers:

- **X** – major version / product version
- **X.Y** – minor version / update version / release version
- **X.Y.Z** – bugfix version / hotfix version

Note: It is technically possible to use arbitrary string as a version, but this is highly discouraged as it does not allow sorting. If you need it, just start your version with any non-digit character.

CHAPTER 3

Milestones

Milestones are just labels on a *Release*. They shouldn't affect how a *Release* is versioned (e.g. no *Release* version change on Beta).

Milestone Labels

- <milestone_name>-<version>.<respin>
- <milestone_name>-<version> stands for planned milestone
- <respin> is internal-only numbering

Milestone Names

- DevelPhaseExit – DEV finished major features, QE performs acceptance testing for entering the Testing phase
- InternalAlpha – Internal Alpha, usually tweaking compose for the first public release
- Alpha – Public Alpha
- InternalSnapshot – Snapshots between Alpha and Beta, usually only for QE purposes
- Beta – Public Beta
- Snapshot – Snapshots between Beta and RC
- RC – Release Candidates
- Update – post-GA updates

Examples

- (rhel-7.0) Alpha-1.0
- (rhel-7.0) Beta-1.0
- (rhel-7.0) Beta-1.1

Python modules:

CHAPTER 4

common – Base classes and common functions

This module provides base classes and common functions used in other productmd modules.

4.1 Constants

productmd.common.**RELEASE_TYPES = ['fast', 'ga', 'updates', 'updates-testing', 'eus', 'aus',**
list() -> new empty list list(iterable) -> new list initialized from iterable's items

productmd.common.**RELEASE_SHORT_RE = <_sre.SRE_Pattern object>**
Validation regex for release short name: [a-z] followed by [a-z0-9] separated with dashes.

productmd.common.**RELEASE_VERSION_RE = <_sre.SRE_Pattern object>**
Validation regex for release version: any string or [0-9] separated with dots.

4.2 Functions

productmd.common.**parse_nvra**(*nvra*)
Parse RPM N-E:V-R.A string to a dict.

Parameters **nvra** (*str*) – N-E:V-R.A string. This can be a file name or a file path including the '.rpm' suffix.

Return type dict, with "name", "epoch", "version", "release", and "arch" elements.

productmd.common.**create_release_id**(*short*, *version*, *type*, *bp_short=None*, *bp_version=None*, *bp_type=None*)
Create release_id from given parts.

Parameters

- **short** (*str*) – Release short name
- **version** (*str*) – Release version
- **version** – Release type

- **bp_short** (*str*) – Base Product short name
- **bp_version** (*str*) – Base Product version
- **bp_type** – Base Product type

Return type str

productmd.common.**parse_release_id**(*release_id*)
Parse release_id to parts: {short, version, type} or {short, version, type, bp_short, bp_version, bp_type}

Parameters **release_id** (*str*) – Release ID string

Return type dict

productmd.common.**is_valid_release_short**(*short*)
Determine if given release short name is valid.

Parameters **short** (*str*) – Release short name

Return type bool

productmd.common.**is_valid_release_version**(*version*)
Determine if given release version is valid.

Parameters **version** (*str*) – Release version

Return type bool

productmd.common.**split_version**(*version*)
Split version to a list of integers that can be easily compared.

Parameters **version** (*str*) – Release version

Return type [int] or [string]

productmd.common.**get_major_version**(*version*, *remove=None*)
Return major version of a provided version string.

Parameters **version** (*str*) – Version string

Return type str

productmd.common.**get_minor_version**(*version*, *remove=1*)
Return minor version of a provided version string.

Parameters

- **version** (*str*) – Version string
- **remove** (*int*) – Number of version parts to remove; defaults to 1

Return type str

4.3 Classes

class productmd.common.**MetadataBase**

validate()
Validate attributes by running all self._validate_*() methods.

Raises

- **TypeError** – if an attribute has invalid type

- **ValueError** – if an attribute contains invalid value

load(*f*)
Load data from a file.

Parameters **f** (*file or str*) – file-like object or path to file

loads(*s*)
Load data from a string.

Parameters **s** (*str*) – input data

dump(*f*)
Dump data to a file.

Parameters **f** (*file or str*) – file-like object or path to file

dumps()
Dump data to a string.

Return type str

class productmd.common.**Header**(*parent*, *metadata_type*)
This class represents the header used in serialized metadata files.

It consists of a type and a version. The type is meant purely for consumers of the file to know what they are dealing with without having to check filename. The version is used by productmd when parsing the file.

CHAPTER 5

compose – Easy access to compose metadata

This module provides Compose class that provides easy access to ComposeInfo, Rpms, Modules and Images in compose metadata.

Example:

```
import productmd.compose
compose = productmd.compose.Compose("/path/to/compose")

# then you can access compose metadata via following properties:
compose.info
compose.images
compose.rpms
compose.modules
```

5.1 Classes

class productmd.compose.**Compose**(*compose_path*)

This class provides easy access to compose metadata.

Parameters **compose_path** (*str*) – Path to a compose. HTTP(s) URL is also accepted.

info

(*productmd.composeinfo.ComposeInfo*) – Compose metadata

images

(*productmd.images.Images*) – Compose images metadata

rpms

(*productmd.rpms.Rpms*) – Compose RPMs metadata

modules

(productmd.modules.Modules) – Compose Modules metadata

CHAPTER 6

composeinfo – Compose metadata

This module provides classes for manipulating composeinfo.json files. composeinfo.json files provide details about composes which includes product information, variants, architectures and paths.

Example:

```
import productmd.compose
compose = productmd.compose.Compose("/path/to/compose")
print(compose.info.compose.id)  # prints "Fedora-Rawhide-20180616.n.0"
```

6.1 Constants

productmd.composeinfo.**COMPOSE_TYPES = ['test', 'ci', 'nightly', 'production']**
supported compose types

productmd.composeinfo.**LABEL_NAMES = ['EA', 'DevelPhaseExit', 'InternalAlpha', 'Alpha', 'Int**
supported milestone label names

productmd.composeinfo.**VARIANT_TYPES = ['variant', 'optional', 'addon', 'layered-product']**
supported variant types

6.2 Classes

class productmd.composeinfo.**ComposeInfo**
This class only encapsulates other classes with actual data.

header = None
(*Header*) – Metadata header

compose = None
(*Compose*) – Compose details

release = None
(*Release*) – Release details

base_product = None
(*BaseProduct*) – Base product details (optional)

variants = None
(*Variants*) – release variants

dump(*f*)
Dump data to a file.

Parameters **f** (*file or str*) – file-like object or path to file

dumps()
Dump data to a string.

Return type str

load(*f*)
Load data from a file.

Parameters **f** (*file or str*) – file-like object or path to file

loads(*s*)
Load data from a string.

Parameters **s** (*str*) – input data

validate()
Validate attributes by running all self._validate_*() methods.

Raises

- **TypeError** – if an attribute has invalid type
- **ValueError** – if an attribute contains invalid value

class productmd.composeinfo.**Compose**(*metadata*)
This class represents the top level of metadata for a compose.

It provides access to general information about the compose (ID, type, date, etc.) and structures with RPMs and images.

label_major_version
Return major version for a label.

Examples: Beta-1.2 -> Beta-1, GA -> GA

class productmd.composeinfo.**Release**(*metadata*)
This class represents a product release.

name = None
(*str*) – Release name, for example: “Fedora”, “Red Hat Enterprise Linux”

version = None
(*str*) – Release version (incl. minor version), for example: “20”, “7.0”

short = None
(*str*) – Release short name, for example: “f”, “rhel”

type = None
(*str*) – Release type, for example: “ga”, “updates”

is_layered = None
(*bool=False*) – Determines if release is a layered product

internal = None
(*bool=False*) – Determine if release is meant for public consumption

class productmd.composeinfo.**BaseProduct**(*metadata*)
This class represents a base product a release is based on. For example: Spacewalk 2.2 release requires Fedora 20 base product. Information from this class is used only if release.is_layered is set.

name = None
(*str*) – Product name, for example: "Fedora", "Red Hat Enterprise Linux"

version = None
(*str*) – Product version (typically major version), for example: "20", "7"

short = None
(*str*) – Product short name, for example: "f", "rhel"

type = None
(*str*) – Product type, for example: "ga", "eus"

type_suffix
This is used in compose ID.

class productmd.composeinfo.**Variants**(*metadata*)
This class is a container for compose variants.

class productmd.composeinfo.**Variant**(*metadata*)

id = None
(*str*) – variant ID, for example: "Client", "Server", "optional"

uid = None
(*str*) – variant unique ID: $PARENT_UID-$ID, for example: "Server-optional"

name = None
(*str*) – variant name (pretty text), for example: "Enterprise Server"

type = None
(*str*) – variant type, see VARIANT_TYPES for supported values

arches = None
(*set(<str>)*) – set of arches for a variant

variants = None
(*dict*) – child variants

parent = None
(*Variant* or *None*) – parent variant

paths = None
(*VariantPaths*) – path mappings for a variant

release = None
(*Release*) –

class productmd.composeinfo.**VariantPaths**(*variant*)
This class stores relative paths for a variant in a compose. Paths are represented as dictionaries mapping arches to actual paths. List of supported paths follows.

Binary

- **os_tree** – installable tree with binary RPMs, kickstart trees, readme etc.
- **packages** – directory with binary RPMs

- **repository** – YUM repository with binary RPMs
- **isos** – Binary ISOs
- **jigdos** – Jigdo files for binary ISOs

Source

- **source_tree** – tree with source RPMs
- **source_packages** – directory with source RPMs
- **source_repository** – YUM repository with source RPMs
- **source_isos** – Source ISOs
- **source_jigdos** – Jigdo files for source ISOs

Debug

- **debug_tree** – tree with debug RPMs
- **debug_packages** – directory with debug RPMs
- **debug_repository** – YUM repository with debug RPMs

Example:

```
self.os_tree = {
    "i386": "Server/i386/os",
    "x86_64": "Server/x86_64/os",
}
self.packages = {
    "i386": "Server/i386/os/Packages",
    "x86_64": "Server/x86_64/os/Packages",
}
```

6.3 Private Classes

class productmd.composeinfo.**VariantBase**(*metadata*)

__init__(*metadata*)
x.__init__(...) initializes x; see help(type(x)) for signature

__repr__() <==> *repr(x)*

get_variants(*arch=None*, *types=None*, *recursive=False*)
Return all variants of given arch and types.

Supported variant types: self - include the top-level ("self") variant as well addon variant optional

CHAPTER 7

discinfo – Installation media metadata

This module provides classes for manipulating .discinfo files. .discinfo files can be found on Fedora installation media and provide media information to Anaconda installer.

7.1 Classes

class productmd.discinfo.**DiscInfo**

This class manipulates .discinfo files used by Anaconda installer.

timestamp = None

Timestamp in float format

description = None

Release description, for example: Fedora 20

arch = None

Media architecture, for example: x86_64

disc_numbers = None

List with disc numbers or ["ALL"]

now()

Shortcut for setting timestamp to now().

dump(*f*)

Dump data to a file.

Parameters **f** (*file or str*) – file-like object or path to file

dumps()

Dump data to a string.

Return type str

load(*f*)

Load data from a file.

Parameters **f** (*file or str*) – file-like object or path to file

loads(*s*)
Load data from a string.

Parameters **s** (*str*) – input data

validate()
Validate attributes by running all self._validate_*() methods.

Raises

- **TypeError** – if an attribute has invalid type
- **ValueError** – if an attribute contains invalid value

CHAPTER 8

images – Image metadata

This module provides classes for manipulating images.json files. images.json files provide details about images included in composes.

productmd.images.**SUPPORTED_IMAGE_TYPES = ['boot', 'cd', 'docker', 'dvd', 'dvd-debuginfo',**
supported image types

productmd.images.**SUPPORTED_IMAGE_FORMATS = ['iso', 'qcow', 'qcow2', 'raw', 'raw.xz', 'rhevm**
supported image formats, they match with file suffix

productmd.images.**UNIQUE_IMAGE_ATTRIBUTES = ['subvariant', 'type', 'format', 'arch', 'disc_n**
combination of attributes which uniquely identifies an image across composes

class productmd.images.**UniqueImage**(*subvariant*, *type*, *format*, *arch*, *disc_number*, *unified*, *additional_variants*)
a namedtuple with unique attributes, use identify_image to create an instance

additional_variants
Alias for field number 6

arch
Alias for field number 3

disc_number
Alias for field number 4

format
Alias for field number 2

subvariant
Alias for field number 0

type
Alias for field number 1

unified
Alias for field number 5

8.1 Classes

class productmd.images.**Images**

add(*variant*, *arch*, *image*)
Assign an *Image* object to variant and arch.

Parameters

- **variant** (*str*) – compose variant UID
- **arch** (*str*) – compose architecture
- **image** (*Image*) – image

dump(*f*)
Dump data to a file.

Parameters **f** (*file or str*) – file-like object or path to file

dumps()
Dump data to a string.

Return type str

load(*f*)
Load data from a file.

Parameters **f** (*file or str*) – file-like object or path to file

loads(*s*)
Load data from a string.

Parameters **s** (*str*) – input data

validate()
Validate attributes by running all self._validate_*() methods.

Raises

- **TypeError** – if an attribute has invalid type
- **ValueError** – if an attribute contains invalid value

class productmd.images.**Image**(*parent*)

path = None
(*str*) – relative path to an image, for example: "Server/x86_64/iso/boot.iso"

mtime = None
(*int*) – image mtime

size = None
(*int*) – image size

volume_id = None
(*str*) –

type = None
(*str*) –

format = None
(*str*) – Release name, for example: "Fedora", "Red Hat Enterprise Linux"

arch = None
(*str*) – image architecture, for example: “x86_64”, “src”

disc_number = None
(*int*) – Release name, for example: “Fedora”, “Red Hat Enterprise Linux”

disc_count = None
(*int*) – Release name, for example: “Fedora”, “Red Hat Enterprise Linux”

checksums = None
(*str*) – Release name, for example: “Fedora”, “Red Hat Enterprise Linux”

implant_md5 = None
(*str* or *None*) – value of implanted md5

bootable = None
(*bool=False*) –

subvariant = None
(*str*) – image contents, may be same as variant or e.g. ‘KDE’, ‘LXDE’

unified = None
(*bool=False*) – indicates if the ISO contains content from multiple variants

additional_variants = None
(*[str]*) – indicates which variants are present on the ISO

dump(*f*)
Dump data to a file.

Parameters **f** (*file or str*) – file-like object or path to file

dumps()
Dump data to a string.

Return type str

load(*f*)
Load data from a file.

Parameters **f** (*file or str*) – file-like object or path to file

loads(*s*)
Load data from a string.

Parameters **s** (*str*) – input data

validate()
Validate attributes by running all self._validate_*() methods.

Raises

- **TypeError** – if an attribute has invalid type
- **ValueError** – if an attribute contains invalid value

CHAPTER 9

rpms – RPM metadata

This module provides classes for manipulating rpms.json files. rpms.json files provide details about RPMs included in composes.

Example:

```
import productmd.compose
compose = productmd.compose.Compose("/path/to/compose")

# Print the entire dict that maps all variants, arches, and RPMs for this
# compose:
print(compose.rpms.rpms)

# Find all the source RPMs in this compose:
srpms = set()

for variant in compose.rpms.rpms:
    for arch in compose.rpms.rpms[variant]:
        for srpm in compose.rpms.rpms[variant][arch]:
            srpms.add(srpm)

print(srpms)
# ... prints the set of SRPMs in all our variants:
# ['ceph-2:12.2.5-25.el7cp.src',
#  'ceph-ansible-0:3.1.0-0.1.rc9.el7cp.src',
#  'ceph-iscsi-cli-0:2.7-1.el7cp.src',
#  ...
# ]
```

9.1 Classes

class productmd.rpms.**Rpms**

add(*variant*, *arch*, *nevra*, *path*, *sigkey*, *category*, *srpm_nevra=None*)
Map RPM to to variant and arch.

Parameters

- **variant** (*str*) – compose variant UID
- **arch** (*str*) – compose architecture
- **nevra** (*str*) – name-epoch:version-release.arch
- **sigkey** (*str or None*) – sigkey hash
- **category** (*str*) – RPM category, one of binary, debug, source
- **srpm_nevra** (*str*) – name-epoch:version-release.arch of RPM's SRPM

dump(*f*)
Dump data to a file.

Parameters **f** (*file or str*) – file-like object or path to file

dumps()
Dump data to a string.

Return type str

load(*f*)
Load data from a file.

Parameters **f** (*file or str*) – file-like object or path to file

loads(*s*)
Load data from a string.

Parameters **s** (*str*) – input data

validate()
Validate attributes by running all self._validate_*() methods.

Raises

- **TypeError** – if an attribute has invalid type
- **ValueError** – if an attribute contains invalid value

CHAPTER 10

treeinfo – Instalable trees metadata

This module provides classes for manipulating .treeinfo files. Treeinfo files provide details about installable trees in Fedora composes and media.

10.1 Classes

class productmd.treeinfo.**TreeInfo**

header = None
(*productmd.common.Header*) – Metadata header

release = None
(*Release*) – Release details

base_product = None
(*BaseProduct*) – Base product details (optional)

tree = None
(Tree) – Tree details

variants = None
(*Variants*) – Release variants

checksums = None
(Checksums) – Checksums of images included in a tree

images = None
(*Images*) – Paths to images included in a tree

stage2 = None
(Stage2) – Stage 2 image path (for Anaconda installer)

media = None
(Media) – Media set information (optional)

dump(*f*)
Dump data to a file.

Parameters **f** (*file or str*) – file-like object or path to file

dumps()
Dump data to a string.

Return type str

load(*f*)
Load data from a file.

Parameters **f** (*file or str*) – file-like object or path to file

loads(*s*)
Load data from a string.

Parameters **s** (*str*) – input data

validate()
Validate attributes by running all self._validate_*() methods.

Raises

- **TypeError** – if an attribute has invalid type
- **ValueError** – if an attribute contains invalid value

class productmd.treeinfo.**Release**(*metadata*)

name = None
(*str*) – release name, for example: "Fedora", "Red Hat Enterprise Linux", "Spacewalk"

short = None
(*str*) – release short name, for example: "F", "RHEL", "Spacewalk"

version = None
(*str*) – release version, for example: "21", "7.0", "2.1"

is_layered = None
(*bool*) – typically False for an operating system, True otherwise

major_version
Version string without the last part. For example: version == 1.2.0 -> major_version == 1.2

minor_version
Last part of the version string. For example: version == 1.2.0 -> minor_version == 0

class productmd.treeinfo.**BaseProduct**(*metadata*)
BaseProduct provides information about operating system a *Release* runs on.

name = None
(*str*) – base product name, for example: "Fedora", "Red Hat Enterprise Linux"

short = None
(*str*) – base product short name, for example: "F", "RHEL"

version = None
(*str*) – base product *major* version, for example: "21", "7"

class productmd.treeinfo.**Variants**(*metadata*)

class productmd.treeinfo.**Variant**(*metadata*)

id = None
(*str*) – variant ID, for example “Server”, “optional”

uid = None
(*str*) – variant UID ($parent_UID.$ID), for example “Server”, “Server-optional”

name = None
(*str*) – variant name, for example “Server”

type = None
(*str*) – “variant”, “addon”, “optional”

parent = None
(*Variant* or *None*) – reference to parent *Variant*

variants = None
(*dict*) *Variant*

paths = None
(*VariantPaths*) – relative paths to repositories, packages, etc.

class productmd.treeinfo.**VariantPaths**(*variant*)
This class stores paths for a variant in a tree. All paths are relative to .treeinfo location.

Binary

- **packages** – directory with binary RPMs
- **repository** – YUM repository with binary RPMs

Source

- **source_packages** – directory with source RPMs
- **source_repository** – YUM repository with source RPMs

Debug

- **debug_packages** – directory with debug RPMs
- **debug_repository** – YUM repository with debug RPMs

Others

- **identity** – path to a pem file which identifies a product

Example:

```
variant = ...
variant.paths.packages = "Packages"
variant.paths.repository = "."
```

class productmd.treeinfo.**Images**(*metadata*)

platforms
Return all platforms with available images

File formats:

CHAPTER 11

Composeinfo file format 1.1

composeinfo.json files provide details about composes which includes product information, variants, architectures and paths.

11.1 Changes from 1.0

- Added 'type' field to 'header', "productmd.composeinfo" required
- Added 'type' field to 'release'
- Added 'type' field to 'base_product'

11.2 File Format

Composeinfo is stored as a JSON serialized dictionary. It's recommended to sort keys alphabetically and use 4 spaces for indentation in order to read and diff composeinfo.json files easily.

```
{
    "header": {
        "type": "productmd.composeinfo",            # metadata type; "productmd.
↪composeinfo" required; [new in 1.1]
        "version": "1.1"                            # metadata version; format: $major
↪<int>.$minor<int>
    },
    "payload": {
        "compose": {
            "id": <str>,
            "date": <str>,
            "respin": <int>,
            "type": <str>,
            "label": <str|unset>,
```

(continues on next page)

(continued from previous page)

```
            "final": <bool=false>                       # true if a compose is final for␣
↪a milestone (for example latest Beta-1.x)
        },
        "release": {
            "name": <str>,
            "version": <str>,
            "short": <str>,
            "type": <str>,                              # [new in 1.1]
            "is_layered": <bool=false>,
        },
        "base_product": {
            "name": <str>,
            "version": <str>,
            "short": <str>,
            "type": <str>,                              # [new in 1.1]
        },
        "variants": {
            variant_uid<str>: {
                "id": <str>,
                "uid": <str>,
                "name": <str>,
                "type": <str>,
                "arches": [<str>],
                "paths": {
                    path_category<str>: {
                        arch<str>: <str>,
                    },
                },
            },
        },
    },
}
```

CHAPTER 12

Discinfo file format 1.0

.discinfo files can be found on Fedora installation media and provide media information to Anaconda installer.

12.1 File Format

.discinfo is a plain-text file containing following fields, one value per line:

```
timestamp: float
release: str
architecture: str
disc_numbers: ALL or comma separated numbers
```

12.2 Examples

Fedora 21 Server.x86_64, disc_numbers: ALL:

```
1417653453.026288
Fedora Server 21
x86_64
ALL
```

Fedora 21 Server.x86_64, disc_numbers: [1, 2, 3]:

```
1417653453.026288
Fedora Server 21
x86_64
1,2,3
```

CHAPTER 13

Images file format 1.1

images.json files provide details about images included in composes.

13.1 Changes from 1.0

- Added 'type' field to 'header', "productmd.images" required
- Added 'subvariant' field to image

13.2 File Format

Compose images metadata is stored as a JSON serialized dictionary. It's recommended to sort keys alphabetically and use 4 spaces for indentation in order to read and diff images.json files easily.

13.3 Image Identity

It is required that the combination of subvariant, type, format, arch and disc_number attributes is unique to each image in the compose. This is to ensure these attributes can be used to identify 'the same' image across composes. This may require including the variant string in the subvariant.

```
{
    "header": {
        "type": "productmd.images",                 # metadata type; "productmd.images
↪" required; [new in 1.1]
        "version": "1.1"                            # metadata version; format: $major
↪<int>.$minor<int>
    },
    "payload": {
        "compose": {                                # see composeinfo for details
```

(continues on next page)

(continued from previous page)

```
            "date": <str>,
            "id": <str>,
            "respin": <int>,
            "type": <str>
        },
        "images": {
            variant_uid<str>: {                     # compose variant UID
                arch<str>: [                        # compose variant arch
                    {
                        "arch": <str>,              # image arch
                        "bootable": <bool>,         # can the image be booted?
                        "checksums": {
                            type<str>: <str>        #
                        },
                        "disc_count": <int>,        # number of discs in media set
                        "disc_number": <int>,       # disc number
                        "format": <str>,            # see productmd.images.SUPPORTED_
↪IMAGE_FORMATS
                        "implant_md5": <str|null>,  # md5 checksum implanted directly␣
↪on media (see implantisomd5 and checkisomd5 commands)
                        "mtime": <int>,             # mtime of the image stored as a␣
↪decimal unix timestamp
                        "path": <str>,              # relative path to the image
                        "subvariant": <str>,        # image content (e.g. 'Workstation
↪' or 'KDE'); [new in 1.1]
                        "size": <int>,              # file size of the image
                        "type": <str>,              # see productmd.images.SUPPORTED_
↪IMAGE_TYPES
                        "volume_id": <str|null>     # volume ID; null if not␣
↪available/applicable
                    }
                ]
            }
        }
    }
}
```

CHAPTER 14

RPMs file format 1.1

rpms.json files provide details about RPMs included in composes.

14.1 Changes from 1.0

- Added 'type' field to 'header', "productmd.rpms" required

14.2 File Format

Compose RPMs metadata is stored as a JSON serialized dictionary. It's recommended to sort keys alphabetically and use 4 spaces for indentation in order to read and diff rpms.json files easily.

```
{
    "header": {
        "type": "productmd.rpms",                    # metadata type; "productmd.rpms"␣
↪required; [new in 1.1]
        "version": "1.1"                             # metadata version; format: $major
↪<int>.$minor<int>
    },
    "payload": {
        "compose": {                                 # see composeinfo for details
            "date": <str>,
            "id": <str>,
            "respin": <int>,
            "type": <str>
        },
        "rpms": {
            variant_uid<str>: {                      # compose variant UID
                arch<str>: {                         # compose variant arch
                    srpm_nevra<str>: {               # %name-%epoch:%version-%release-
↪%arch of source RPM (koji build with epoch included)
```

(continues on next page)

(continued from previous page)

```
                        rpm_nevra<str>: {           # %name-%epoch:%version-%release-
↪%arch of RPM file
                            "path": <str>,          # relative path to RPM file
                            "sigkey": <str|null>,   # sigkey ID: hex string 8␣
↪characters long, lower case; null for unsigned RPMs
                            "category": <str>       # binary, debug, source
                        }
                    }
                }
            }
        }
    }
}
```

14.3 Examples

Bash in Fedora 21:

```
{
    "header": {
        "version": "1.0"
    },
    "payload": {
        "compose": {
            "date": "20141203",
            "id": "Fedora-21-20141203.0",
            "respin": 0,
            "type": "production"
        },
        "rpms": {
            "Server": {
                "armhfp": {
                    "bash-0:4.3.30-2.fc21.src": {
                        "bash-0:4.3.30-2.fc21.armv7hl": {
                            "path": "Server/armhfp/os/Packages/b/bash-4.3.30-2.fc21.
↪armv7hl.rpm",
                            "sigkey": "95a43f54",
                            "category": "binary"
                        },
                        "bash-0:4.3.30-2.fc21.src": {
                            "path": "Server/source/SRPMS/b/bash-4.3.30-2.fc21.src.rpm
↪",
                            "sigkey": "95a43f54",
                            "category": "binary"
                        }
                    }
                },
                "i386": {
                    "bash-0:4.3.30-2.fc21.src": {
                        "bash-0:4.3.30-2.fc21.i686": {
                            "path": "Server/i386/os/Packages/b/bash-4.3.30-2.fc21.
↪i686.rpm",
                            "sigkey": "95a43f54",
                            "category": "binary"
```

(continues on next page)

(continued from previous page)


```
                        },
                        "bash-0:4.3.30-2.fc21.src": {
                            "path": "Server/source/SRPMS/b/bash-4.3.30-2.fc21.src.rpm
↪",
                            "sigkey": "95a43f54",
                            "category": "binary"
                        }
                    }
                },
                "x86_64": {
                    "bash-0:4.3.30-2.fc21.src": {
                        "bash-0:4.3.30-2.fc21.x86_64": {
                            "path": "Server/x86_64/os/Packages/b/bash-4.3.30-2.fc21.
↪x86_64.rpm",
                            "sigkey": "95a43f54",
                            "category": "binary"
                        },
                        "bash-0:4.3.30-2.fc21.src": {
                            "path": "Server/source/SRPMS/b/bash-4.3.30-2.fc21.src.rpm
↪",
                            "sigkey": "95a43f54",
                            "category": "binary"
                        }
                    }
                }
            },
            "Workstation": {
                "armhfp": {
                    "bash-0:4.3.30-2.fc21.src": {
                        "bash-0:4.3.30-2.fc21.armv7hl": {
                            "path": "Workstation/armhfp/os/Packages/b/bash-4.3.30-2.
↪fc21.armv7hl.rpm",
                            "sigkey": "95a43f54",
                            "category": "binary"
                        },
                        "bash-0:4.3.30-2.fc21.src": {
                            "path": "Workstation/source/SRPMS/b/bash-4.3.30-2.fc21.
↪src.rpm",
                            "sigkey": "95a43f54",
                            "category": "binary"
                        }
                    }
                },
                "i386": {
                    "bash-0:4.3.30-2.fc21.src": {
                        "bash-0:4.3.30-2.fc21.i686": {
                            "path": "Workstation/i386/os/Packages/b/bash-4.3.30-2.
↪fc21.i686.rpm",
                            "sigkey": "95a43f54",
                            "category": "binary"
                        },
                        "bash-0:4.3.30-2.fc21.src": {
                            "path": "Workstation/source/SRPMS/b/bash-4.3.30-2.fc21.
↪src.rpm",
                            "sigkey": "95a43f54",
                            "category": "binary"
                        }
```


(continues on next page)

(continued from previous page)

```
                    }
                },
                "x86_64": {
                    "bash-0:4.3.30-2.fc21.src": {
                        "bash-0:4.3.30-2.fc21.x86_64": {
                            "path": "Workstation/x86_64/os/Packages/b/bash-4.3.30-2.
↪fc21.x86_64.rpm",
                            "sigkey": "95a43f54",
                            "category": "binary"
                        },
                        "bash-0:4.3.30-2.fc21.src": {
                            "path": "Workstation/source/SRPMS/b/bash-4.3.30-2.fc21.
↪src.rpm",
                            "sigkey": "95a43f54",
                            "category": "binary"
                        }
                    }
                }
            }
        }
    }
}
```

CHAPTER 15

Treeinfo file format 1.1

Treeinfo files provide details about installable trees in Fedora composes and media.

15.1 Changes from 1.0

- Added 'type' field to 'header', "productmd.treeinfo" required
- Added 'type' field to 'release'
- Added 'type' field to 'base_product'

15.2 File Format

Treeinfo is an INI file. It's recommended to sort sections and keys alphabetically in order to diff .treeinfo files easily.

```
[header]
type = <str>                            ; metadata type; "productmd.treeinfo" required;␣
↪[new in 1.1]
version = 1.1                           ; metadata version; format: $major<int>.$minor
↪<int>

[release]
name = <str>                            ; release name, for example: "Fedora", "Red Hat␣
↪Enterprise Linux", "Spacewalk"
short = <str>                           ; release short name, for example: "F", "RHEL",
↪"Spacewalk"
version = <str>                         ; release version, for example: "21", "7.0", "2.
↪1"
type = <str>                            ; release type, for example: "ga", "updates",
↪"eus"; [new in 1.1]
is_layered = <bool=False>               ; typically False for an operating system, True␣
↪otherwise
```

(continues on next page)

(continued from previous page)

```
[base_product]
name = <str>                            ; base product name, for example: "Fedora",
↪"Red Hat Enterprise Linux"
short = <str>                           ; base product short name, for example: "F",
↪"RHEL"
version = <str>                         ; base product *major* version, for example: "21
↪", "7"
type = <str>                            ; base product release type, for example: "ga",
↪"eus"; [new in 1.1]

[tree]
arch = <str>                            ; tree architecture, for example x86_64
build_timestamp = <int|float>           ; tree build time timestamp; format: unix time
platforms = <str>[, <str> ...]          ; supported platforms; for example x86_64,xen
variants = <str>[, <str> ...]           ; UIDs of available variants, for example
↪"Server,Workstation"

[checksums]
; checksums of selected files in a tree:
; * all repodata/repomd.xml
; * all images captured in [images-*] and [stage2] sections
$path = $checksum_type<str>:checksum_value<str>

[images-$platform<str>]
; images compatible with particular $platform
$file_name = $relative_path<str>

[stage2]
; optional section, available only on bootable media with Anaconda installer
instimage = <str>                       ; relative path to Anaconda instimage (obsolete)
mainimage = <str>                       ; relative path to Anaconda stage2 image

[media]
; optional section, available only on media
discnum = <int>                         ; disc number
totaldiscs = <int>                      ; number of discs in media set

[variant-$variant_uid]
id = <str>                              ; variant ID
uid = <str>                             ; variant UID ($parent_UID.$ID)
name = <str>                            ; variant name
type = <str>                            ; variant, optional
variants = <str>[,<str>...]             ; UIDs of child variants
addons = <str>[,<str>...]               ; UIDs of child addons

; variant paths
; all paths are relative to .treeinfo location
packages = <str>                        ; directory with binary RPMs
repository = <str>                      ; YUM repository with binary RPMs
source_packages = <str>                 ; directory with source RPMs
source_repository = <str>               ; YUM repository with source RPMs
debug_packages = <str>                  ; directory with debug RPMs
debug_repository = <str>                ; YUM repository with debug RPMs
identity = <str>                        ; path to a pem file that identifies a product

[addon-$addon_uid]
```

(continues on next page)

(continued from previous page)

```
id = <str>                              ; addon ID
uid = <str>                             ; addon UID ($parent_UID.$ID)
name = <str>                            ; addon name
type = addon

; addon paths
; see variant paths

[general]
; WARNING.0 = This section provides compatibility with pre-productmd treeinfos.
; WARNING.1 = Read productmd documentation for details about new format.
family = <str>                          ; equal to [release]/name
version = <str>                         ; equal to [release]/version
name = <str>                            ; equal to "$family $version"
arch = <str>                            ; equal to [tree]/arch
platforms = <str>[,<str>...]            ; equal to [tree]/platforms
packagedir = <str>                      ; equal to [variant-*]/packages
repository = <str>                      ; equal to [variant-*]/repository
timestamp = <int>                       ; equal to [tree]/build_timestamp
variant = <str>                         ; variant UID of first variant (sorted␣
↪alphabetically)
```

15.3 Examples

Fedora 21 Server.x86_64 .treinfo converted to 1.0 format:

```
[checksums]
images/boot.iso =␣
↪sha256:56af126a50c227d779a200b414f68ea7bcf58e21c8035500cd21ba164f85b9b4
images/efiboot.img =␣
↪sha256:de48c8b25f03861c00c355ccf78108159f1f2aa63d0d63f92815146c24f60164
images/macboot.img =␣
↪sha256:da76ff5490b4ae7e123f19b8f4b36efd6b7c435073551978d50c5181852a87f5
images/product.img =␣
↪sha256:ffce14a7a95be20b36f302cb0698be8c19fda798807d3d63a491d6f7c1b23b5b
images/pxeboot/initrd.img =␣
↪sha256:aadebd07c4c0f19304f0df7535a8f4218e5141602f95adec08ad1e22ff1e2d43
images/pxeboot/upgrade.img =␣
↪sha256:224d098fb3903583b491692c5e0e1d20ea840d51f4da671ced97d422402bbf1c
images/pxeboot/vmlinuz =␣
↪sha256:81c28a439f1d23786057d3b57db66e00b2b1a39b64d54de1a90cf2617e53c986
repodata/repomd.xml =␣
↪sha256:3af1609aa27949bf1e02e9204a7d4da7efee470063dadbc3ea0be3ef7f1f4d14

[general]
arch = x86_64
family = Fedora
name = Fedora 21
packagedir = Packages
platforms = x86_64,xen
repository = .
timestamp = 1417653911
variant = Server
version = 21
```

(continues on next page)

(continued from previous page)

```
[header]
version = 1.0

[images-x86_64]
boot.iso = images/boot.iso
initrd = images/pxeboot/initrd.img
kernel = images/pxeboot/vmlinuz
upgrade = images/pxeboot/upgrade.img

[images-xen]
initrd = images/pxeboot/initrd.img
kernel = images/pxeboot/vmlinuz
upgrade = images/pxeboot/upgrade.img

[release]
name = Fedora
short = Fedora
version = 21
type = ga

[stage2]
mainimage = LiveOS/squashfs.img

[tree]
arch = x86_64
build_timestamp = 1417653911
platforms = x86_64,xen
variants = Server

[variant-Server]
id = Server
name = Server
packages = Packages
repository = .
type = variant
uid = Server
```

Original Fedora 21 Server.x86_64 .treinfo file (before conversion):

```
[general]
name = Fedora-Server-21
family = Fedora-Server
timestamp = 1417653911.68
variant = Server
version = 21
packagedir =
arch = x86_64

[stage2]
mainimage = LiveOS/squashfs.img

[images-x86_64]
kernel = images/pxeboot/vmlinuz
initrd = images/pxeboot/initrd.img
upgrade = images/pxeboot/upgrade.img
boot.iso = images/boot.iso
```

(continues on next page)

(continued from previous page)

```
[images-xen]
kernel = images/pxeboot/vmlinuz
initrd = images/pxeboot/initrd.img
upgrade = images/pxeboot/upgrade.img

[checksums]
images/efiboot.img =␣
↪sha256:de48c8b25f03861c00c355ccf78108159f1f2aa63d0d63f92815146c24f60164
images/macboot.img =␣
↪sha256:da76ff5490b4ae7e123f19b8f4b36efd6b7c435073551978d50c5181852a87f5
images/product.img =␣
↪sha256:ffce14a7a95be20b36f302cb0698be8c19fda798807d3d63a491d6f7c1b23b5b
images/boot.iso =␣
↪sha256:56af126a50c227d779a200b414f68ea7bcf58e21c8035500cd21ba164f85b9b4
images/pxeboot/vmlinuz =␣
↪sha256:81c28a439f1d23786057d3b57db66e00b2b1a39b64d54de1a90cf2617e53c986
images/pxeboot/initrd.img =␣
↪sha256:aadebd07c4c0f19304f0df7535a8f4218e5141602f95adec08ad1e22ff1e2d43
images/pxeboot/upgrade.img =␣
↪sha256:224d098fb3903583b491692c5e0e1d20ea840d51f4da671ced97d422402bbf1c
repodata/repomd.xml =␣
↪sha256:3af1609aa27949bf1e02e9204a7d4da7efee470063dadbc3ea0be3ef7f1f4d14
```

Old file formats:

CHAPTER 16

Composeinfo file format 1.0

composeinfo.json files provide details about composes which includes product information, variants, architectures and paths.

16.1 File Format

Composeinfo is stored as a JSON serialized dictionary. It's recommended to sort keys alphabetically and use 4 spaces for indentation in order to read and diff composeinfo.json files easily.

```
{
    "header": {
        "version": "1.0"                                # metadata version; format: $major
↪<int>.$minor<int>
    },
    "payload": {
        "compose": {
            "id": <str>,
            "date": <str>,
            "respin": <int>,
            "type": <str>,
            "label": <str|unset>,
            "final": <bool=false>                       # true if a compose is final for␣
↪a milestone (for example latest Beta-1.x)
        },
        "release": {
            "name": <str>,
            "version": <str>,
            "short": <str>,
            "is_layered": <bool=false>,
        },
        "base_product": {
            "name": <str>,
            "version": <str>,
```

(continues on next page)

(continued from previous page)

```
            "short": <str>,
        },
        "variants": {
            variant_uid<str>: {
                "id": <str>,
                "uid": <str>,
                "name": <str>,
                "type": <str>,
                "arches": [<str>],
                "paths": {
                    path_category<str>: {
                        arch<str>: <str>,
                    },
                },
            },
        },
    },
}
```

CHAPTER 17

Images file format 1.0

images.json files provide details about images included in composes.

17.1 File Format

Compose images metadata is stored as a JSON serialized dictionary. It's recommended to sort keys alphabetically and use 4 spaces for indentation in order to read and diff images.json files easily.

```
{
    "header": {
        "version": "1.0"                                # metadata version; format: $major
↪<int>.$minor<int>
    },
    "payload": {
        "compose": {                                    # see composeinfo for details
            "date": <str>,
            "id": <str>,
            "respin": <int>,
            "type": <str>
        },
        "images": {
            variant_uid<str>: {                         # compose variant UID
                arch<str>: [                            # compose variant arch
                    {
                        "arch": <str>,                  # image arch
                        "bootable": <bool>,             # can the image be booted?
                        "checksums": {
                            type<str>: <str>            #
                        },
                        "disc_count": <int>,            # number of discs in media set
                        "disc_number": <int>,           # disc number
                        "format": <str>,                # see productmd.images.SUPPORTED_
↪IMAGE_FORMATS
```

(continues on next page)

(continued from previous page)

```
                        "implant_md5": <str|null>,  # md5 checksum implanted directly␣
↪on media (see implantisomd5 and checkisomd5 commands)
                        "mtime": <int>,             # mtime of the image stored as a␣
↪decimal unix timestamp
                        "path": <str>,              # relative path to the image
                        "size": <int>,              # file size of the image
                        "type": <str>,              # see productmd.images.SUPPORTED_
↪IMAGE_TYPES
                        "volume_id": <str|null>     # volume ID; null if not␣
↪available/applicable
                    }
                ]
            }
        }
    }
}
```

CHAPTER 18

RPMs file format 1.0

rpms.json files provide details about RPMs included in composes.

18.1 File Format

Compose RPMs metadata is stored as a JSON serialized dictionary. It's recommended to sort keys alphabetically and use 4 spaces for indentation in order to read and diff rpms.json files easily.

```
{
    "header": {
        "version": "1.0"                                # metadata version; format: $major
↪<int>.$minor<int>
    },
    "payload": {
        "compose": {                                    # see composeinfo for details
            "date": <str>,
            "id": <str>,
            "respin": <int>,
            "type": <str>
        },
        "rpms": {
            variant_uid<str>: {                         # compose variant UID
                arch<str>: {                            # compose variant arch
                    srpm_nevra<str>: {                  # %name-%epoch:%version-%release-
↪%arch of source RPM (koji build with epoch included)
                        rpm_nevra<str>: {               # %name-%epoch:%version-%release-
↪%arch of RPM file
                            "path": <str>,              # relative path to RPM file
                            "sigkey": <str|null>,       # sigkey ID: hex string 8␣
↪characters long, lower case; null for unsigned RPMs
                            "category": <str>           # binary, debug, source
                        }
                    }
```

(continues on next page)

(continued from previous page)

```
                }
            }
        }
    }
}
```

18.2 Examples

Bash in Fedora 21:

```
{
    "header": {
        "version": "1.0"
    },
    "payload": {
        "compose": {
            "date": "20141203",
            "id": "Fedora-21-20141203.0",
            "respin": 0,
            "type": "production"
        },
        "rpms": {
            "Server": {
                "armhfp": {
                    "bash-0:4.3.30-2.fc21.src": {
                        "bash-0:4.3.30-2.fc21.armv7hl": {
                            "path": "Server/armhfp/os/Packages/b/bash-4.3.30-2.fc21.
↪armv7hl.rpm",
                            "sigkey": "95a43f54",
                            "category": "binary"
                        },
                        "bash-0:4.3.30-2.fc21.src": {
                            "path": "Server/source/SRPMS/b/bash-4.3.30-2.fc21.src.rpm
↪",
                            "sigkey": "95a43f54",
                            "category": "binary"
                        }
                    }
                },
                "i386": {
                    "bash-0:4.3.30-2.fc21.src": {
                        "bash-0:4.3.30-2.fc21.i686": {
                            "path": "Server/i386/os/Packages/b/bash-4.3.30-2.fc21.
↪i686.rpm",
                            "sigkey": "95a43f54",
                            "category": "binary"
                        },
                        "bash-0:4.3.30-2.fc21.src": {
                            "path": "Server/source/SRPMS/b/bash-4.3.30-2.fc21.src.rpm
↪",
                            "sigkey": "95a43f54",
                            "category": "binary"
                        }
                    }
```

(continues on next page)

(continued from previous page)


```
                },
                "x86_64": {
                    "bash-0:4.3.30-2.fc21.src": {
                        "bash-0:4.3.30-2.fc21.x86_64": {
                            "path": "Server/x86_64/os/Packages/b/bash-4.3.30-2.fc21.
↪x86_64.rpm",
                            "sigkey": "95a43f54",
                            "category": "binary"
                        },
                        "bash-0:4.3.30-2.fc21.src": {
                            "path": "Server/source/SRPMS/b/bash-4.3.30-2.fc21.src.rpm
↪",
                            "sigkey": "95a43f54",
                            "category": "binary"
                        }
                    }
                }
            },
            "Workstation": {
                "armhfp": {
                    "bash-0:4.3.30-2.fc21.src": {
                        "bash-0:4.3.30-2.fc21.armv7hl": {
                            "path": "Workstation/armhfp/os/Packages/b/bash-4.3.30-2.
↪fc21.armv7hl.rpm",
                            "sigkey": "95a43f54",
                            "category": "binary"
                        },
                        "bash-0:4.3.30-2.fc21.src": {
                            "path": "Workstation/source/SRPMS/b/bash-4.3.30-2.fc21.
↪src.rpm",
                            "sigkey": "95a43f54",
                            "category": "binary"
                        }
                    }
                },
                "i386": {
                    "bash-0:4.3.30-2.fc21.src": {
                        "bash-0:4.3.30-2.fc21.i686": {
                            "path": "Workstation/i386/os/Packages/b/bash-4.3.30-2.
↪fc21.i686.rpm",
                            "sigkey": "95a43f54",
                            "category": "binary"
                        },
                        "bash-0:4.3.30-2.fc21.src": {
                            "path": "Workstation/source/SRPMS/b/bash-4.3.30-2.fc21.
↪src.rpm",
                            "sigkey": "95a43f54",
                            "category": "binary"
                        }
                    }
                },
                "x86_64": {
                    "bash-0:4.3.30-2.fc21.src": {
                        "bash-0:4.3.30-2.fc21.x86_64": {
                            "path": "Workstation/x86_64/os/Packages/b/bash-4.3.30-2.
↪fc21.x86_64.rpm",
                            "sigkey": "95a43f54",
```


(continues on next page)

(continued from previous page)

```
                            "category": "binary"
                        },
                        "bash-0:4.3.30-2.fc21.src": {
                            "path": "Workstation/source/SRPMS/b/bash-4.3.30-2.fc21.
↪src.rpm",
                            "sigkey": "95a43f54",
                            "category": "binary"
                        }
                    }
                }
            }
        }
    }
}
```

CHAPTER 19

Treeinfo file format 1.0

Treeinfo files provide details about installable trees in Fedora composes and media.

19.1 File Format

Treeinfo is an INI file. It's recommended to sort sections and keys alphabetically in order to diff .treeinfo files easily.

```
[header]
version = 1.0                           ; metadata version; format: $major<int>.$minor
↪<int>

[release]
name = <str>                            ; release name, for example: "Fedora", "Red Hat␣
↪Enterprise Linux", "Spacewalk"
short = <str>                           ; release short name, for example: "F", "RHEL",
↪"Spacewalk"
version = <str>                         ; release version, for example: "21", "7.0", "2.
↪1"
is_layered = <bool=False>               ; typically False for an operating system, True␣
↪otherwise

[base_product]
name = <str>                            ; base product name, for example: "Fedora",
↪"Red Hat Enterprise Linux"
short = <str>                           ; base product short name, for example: "F",
↪"RHEL"
version = <str>                         ; base product *major* version, for example: "21
↪", "7"

[tree]
arch = <str>                            ; tree architecture, for example x86_64
build_timestamp = <int|float>           ; tree build time timestamp; format: unix time
platforms = <str>[, <str> ...]          ; supported platforms; for example x86_64,xen
```

(continues on next page)

(continued from previous page)

```
variants = <str>[, <str> ...]         ; UIDs of available variants, for example
↪"Server,Workstation"

[checksums]
; checksums of selected files in a tree:
; * all repodata/repomd.xml
; * all images captured in [images-*] and [stage2] sections
$path = $checksum_type<str>:checksum_value<str>

[images-$platform<str>]
; images compatible with particular $platform
$file_name = $relative_path<str>

[stage2]
; optional section, available only on bootable media with Anaconda installer
instimage = <str>                     ; relative path to Anaconda instimage (obsolete)
mainimage = <str>                     ; relative path to Anaconda stage2 image

[media]
; optional section, available only on media
discnum = <int>                       ; disc number
totaldiscs = <int>                    ; number of discs in media set

[variant-$variant_uid]
id = <str>                            ; variant ID
uid = <str>                           ; variant UID ($parent_UID.$ID)
name = <str>                          ; variant name
type = <str>                          ; variant, optional
variants = <str>[,<str>...]           ; UIDs of child variants
addons = <str>[,<str>...]             ; UIDs of child addons

; variant paths
; all paths are relative to .treeinfo location
packages = <str>                      ; directory with binary RPMs
repository = <str>                    ; YUM repository with binary RPMs
source_packages = <str>               ; directory with source RPMs
source_repository = <str>             ; YUM repository with source RPMs
debug_packages = <str>                ; directory with debug RPMs
debug_repository = <str>              ; YUM repository with debug RPMs
identity = <str>                      ; path to a pem file that identifies a product

[addon-$addon_uid]
id = <str>                            ; addon ID
uid = <str>                           ; addon UID ($parent_UID.$ID)
name = <str>                          ; addon name
type = addon

; addon paths
; see variant paths

[general]
; WARNING.0 = This section provides compatibility with pre-productmd treeinfos.
; WARNING.1 = Read productmd documentation for details about new format.
family = <str>                        ; equal to [release]/name
version = <str>                       ; equal to [release]/version
name = <str>                          ; equal to "$family $version"
arch = <str>                          ; equal to [tree]/arch
```

(continues on next page)

(continued from previous page)

```
platforms = <str>[,<str>...]          ; equal to [tree]/platforms
packagedir = <str>                    ; equal to [variant-*]/packages
repository = <str>                    ; equal to [variant-*]/repository
timestamp = <int>                     ; equal to [tree]/build_timestamp
variant = <str>                       ; variant UID of first variant (sorted␣
↪alphabetically)
```

19.2 Examples

Fedora 21 Server.x86_64 .treinfo converted to 1.0 format:

```
[checksums]
images/boot.iso =␣
↪sha256:56af126a50c227d779a200b414f68ea7bcf58e21c8035500cd21ba164f85b9b4
images/efiboot.img =␣
↪sha256:de48c8b25f03861c00c355ccf78108159f1f2aa63d0d63f92815146c24f60164
images/macboot.img =␣
↪sha256:da76ff5490b4ae7e123f19b8f4b36efd6b7c435073551978d50c5181852a87f5
images/product.img =␣
↪sha256:ffce14a7a95be20b36f302cb0698be8c19fda798807d3d63a491d6f7c1b23b5b
images/pxeboot/initrd.img =␣
↪sha256:aadebd07c4c0f19304f0df7535a8f4218e5141602f95adec08ad1e22ff1e2d43
images/pxeboot/upgrade.img =␣
↪sha256:224d098fb3903583b491692c5e0e1d20ea840d51f4da671ced97d422402bbf1c
images/pxeboot/vmlinuz =␣
↪sha256:81c28a439f1d23786057d3b57db66e00b2b1a39b64d54de1a90cf2617e53c986
repodata/repomd.xml =␣
↪sha256:3af1609aa27949bf1e02e9204a7d4da7efee470063dadbc3ea0be3ef7f1f4d14

[general]
arch = x86_64
family = Fedora
name = Fedora 21
packagedir = Packages
platforms = x86_64,xen
repository = .
timestamp = 1417653911
variant = Server
version = 21

[header]
version = 1.0

[images-x86_64]
boot.iso = images/boot.iso
initrd = images/pxeboot/initrd.img
kernel = images/pxeboot/vmlinuz
upgrade = images/pxeboot/upgrade.img

[images-xen]
initrd = images/pxeboot/initrd.img
kernel = images/pxeboot/vmlinuz
upgrade = images/pxeboot/upgrade.img
```

(continues on next page)

(continued from previous page)

```
[release]
name = Fedora
short = Fedora
version = 21

[stage2]
mainimage = LiveOS/squashfs.img

[tree]
arch = x86_64
build_timestamp = 1417653911
platforms = x86_64,xen
variants = Server

[variant-Server]
id = Server
name = Server
packages = Packages
repository = .
type = variant
uid = Server
```

Original Fedora 21 Server.x86_64 .treinfo file (before conversion):

```
[general]
name = Fedora-Server-21
family = Fedora-Server
timestamp = 1417653911.68
variant = Server
version = 21
packagedir =
arch = x86_64

[stage2]
mainimage = LiveOS/squashfs.img

[images-x86_64]
kernel = images/pxeboot/vmlinuz
initrd = images/pxeboot/initrd.img
upgrade = images/pxeboot/upgrade.img
boot.iso = images/boot.iso

[images-xen]
kernel = images/pxeboot/vmlinuz
initrd = images/pxeboot/initrd.img
upgrade = images/pxeboot/upgrade.img

[checksums]
images/efiboot.img =␣
↪sha256:de48c8b25f03861c00c355ccf78108159f1f2aa63d0d63f92815146c24f60164
images/macboot.img =␣
↪sha256:da76ff5490b4ae7e123f19b8f4b36efd6b7c435073551978d50c5181852a87f5
images/product.img =␣
↪sha256:ffce14a7a95be20b36f302cb0698be8c19fda798807d3d63a491d6f7c1b23b5b
images/boot.iso =␣
↪sha256:56af126a50c227d779a200b414f68ea7bcf58e21c8035500cd21ba164f85b9b4
images/pxeboot/vmlinuz =␣
↪sha256:81c28a439f1d23786057d3b57db66e00b2b1a39b64d54de1a90cf2617e53c986
```

(continues on next page)

(continued from previous page)

```
images/pxeboot/initrd.img =␣
↪sha256:aadebd07c4c0f19304f0df7535a8f4218e5141602f95adec08ad1e22ff1e2d43
images/pxeboot/upgrade.img =␣
↪sha256:224d098fb3903583b491692c5e0e1d20ea840d51f4da671ced97d422402bbf1c
repodata/repomd.xml =␣
↪sha256:3af1609aa27949bf1e02e9204a7d4da7efee470063dadbc3ea0be3ef7f1f4d14
```

CHAPTER 20

Indices and tables

- genindex
- modindex
- search

Python Module Index

p

Index

Symbols

A

B

C

D

F

G

H

I

L

M

N

P

R

S

T

U

V