
Pyblish Documentation

Release 1.2.1

Marcus Ottosson

October 27, 2015

1	Functions	3
2	Configuration	5
3	Library	7
4	Exceptions	9
4.1	AbstractEntity	9
4.2	Context	10
4.3	Instance	11
4.4	Plugin	11
4.5	Selector	13
4.6	Validator	13
4.7	Extractor	14
4.8	Conformer	14
4.9	discover	14
4.10	plugin_paths	15
4.11	registered_paths	15
4.12	configured_paths	15
4.13	environment_paths	15
4.14	register_plugin_path	15
4.15	deregister_plugin_path	16
4.16	deregister_all	16
4.17	plugins_by_family	16
4.18	plugins_by_host	16
4.19	instances_by_plugin	16
4.20	Config	16
4.21	log	16
4.22	format_filename	16
4.23	PyblishError	17
4.24	SelectionError	17
4.25	ValidationError	17
4.26	ExtractionError	17
4.27	ConformError	17
	Python Module Index	19



API documentation for Pyblish v1.2.1.

Central objects used throughout Pyblish.

<code>AbstractEntity</code>	Superclass for <code>Context</code> and <code>Instance</code>
<code>Context</code>	Maintain a collection of <code>Instances</code>
<code>Instance</code>	An in-memory representation of one or more files
<code>Plugin</code>	Base-class for plugins
<code>Selector</code>	alias of <code>Collector</code>
<code>Validator</code>	Validate/check/test individual instance for correctness.
<code>Extractor</code>	Physically separate <code>Instance</code> from <code>Host</code> into corresponding resources
<code>Conformer</code>	alias of <code>Integrator</code>

Functions

Helper utilities.

<code>discover</code>	Find and return available plug-ins
<code>plugin_paths</code>	Collect paths from all sources.
<code>registered_paths</code>	Return paths added via registration
<code>configured_paths</code>	Return paths added via configuration
<code>environment_paths</code>	Return paths added via environment variable
<code>register_plugin_path</code>	Plug-ins are looked up at run-time from directories registered here
<code>deregister_plugin_path</code>	Remove a <code>pyblish._registered_paths</code> path
<code>deregister_all</code>	
<code>plugins_by_family</code>	
<code>plugins_by_host</code>	
<code>instances_by_plugin</code>	

Configuration

Config

Library

<code>log</code>	Decorator for attaching a logger to the class <i>cls</i>
<code>format_filename</code>	Convert arbitrary string to valid filename, django-style.

Exceptions

Exceptions raised that are specific to Pyblish.

<code>PyblishError</code>	Baseclass for all Pyblish exceptions
<code>SelectionError</code>	Baseclass for selection errors
<code>ValidationError</code>	Baseclass for validation errors
<code>ExtractionError</code>	Baseclass for extraction errors
<code>ConformError</code>	Baseclass for conforming errors

4.1 AbstractEntity

Superclass to Context and Instance, providing the data plug-in to plug-in API via the data member.

class `pyblish.plugin.AbstractEntity`

Superclass for Context and Instance

add (**args, **kwargs*)

DEPRECATED - USE `.append`

Add member to self

This is to mimic the interface of `set()`

has_data (*key*)

DEPRECATED - USE `.data` DICTIONARY DIRECTLY

Check if entity has key

Parameters *key* (*str*) – Key to check

Returns True if it exists, False otherwise

remove (**args, **kwargs*)

DEPRECATED - USE `.pop`

Remove member from self

This is to mimic the interface of `set()`

remove_data (*key*)

DEPRECATED - USE `.data` DICTIONARY DIRECTLY

Remove data from entity

Arguments; `key` (str): Name of data to remove

set_data (*key*, *value*)

DEPRECATED - USE `.data` DICTIONARY DIRECTLY

Modify/insert data into entity

Parameters

- **key** (*str*) – Name of data to add
- **value** (*object*) – Value of data to add

4.2 Context

The context is a container of one or more objects of type `Instance` along with metadata to describe them all; such as the current working directory or logged on user.

class `pyblish.plugin.Context`

Maintain a collection of Instances

__contains__ (*key*)

Support both Instance objects and *id* strings

Example

```
>>> context = Context()
>>> instance = context.create_instance("MyInstance")
>>> "MyInstance" in context
True
>>> instance in context
True
>>> "NotExists" in context
False
```

__getitem__ (*item*)

Enable support for dict-like getting of children by id

Example

```
>>> context = Context()
>>> instance = context.create_instance("MyInstance")
>>> assert context["MyInstance"].name == "MyInstance"
>>> assert context[0].name == "MyInstance"
>>> assert context.get("MyInstance").name == "MyInstance"
```

add (**args*, ***kwargs*)

create_asset (**args*, ***kwargs*)

create_instance (*name*, ***kwargs*)

Convenience method of the following.

```
>>> ctx = Context()
>>> inst = Instance("name", parent=ctx)
```

Example

```
>>> ctx = Context()
>>> inst = ctx.create_instance(name="Name")
```

get (*key*, *default=None*)

id

4.3 Instance

An instance describes one or more items in a working scene; you can think of it as the counter-part of a file on disk - once the file has been loaded, it's an *instance*.

class `pyblish.plugin.Instance` (*name*, *parent=None*)

An in-memory representation of one or more files

Examples include rigs, models.

Parameters

- **name** (*str*) – Name of instance, typically used during extraction as name of resulting files.
- **parent** (*AbstractEntity*) – Optional parent. This is supplied automatically when creating instances with `Context.create_instance()`.

id

str – Unique identifier of instance

name

str – Name of instance

parent

AbstractEntity – Optional parent of instance

context

Return top-level parent; the context

id

log = <logging.Logger object at 0x7f0820771210>

4.4 Plugin

As a plug-in driven framework, any action is implemented as a plug-in and this is the superclass from which all plug-ins are derived. The superclass defines behaviour common across all plug-ins, such as its internally executed method `Plugin.process()` or its virtual members `Plugin.process_instance()` and `Plugin.process_context()`.

Each plug-in MAY define one or more of the following attributes prior to being useful to Pyblish.

- `Plugin.hosts`
- `Plugin.optional`
- `Plugin.version`

Some of which are MANDATORY, others which are OPTIONAL. See each corresponding subclass for details.

- `Selector`
- `Validator`
- `Extractor`
- `Conformer`

class `pyblish.plugin.Plugin`

Base-class for plugins

hosts

Optionally limit a plug-in to one or more hosts

families

Optionally limit a plug-in to one or more families

label

Printed name of plug-in

active

Whether or not to use plug-in during processing

version

Optional version for forwards-compatibility. Pyblish is (currently not) using the version to allow for plug-ins incompatible with a particular running instance of Pyblish to co-exist alongside compatible versions.

order

Order in which this plug-in is processed. This is used internally to control which plug-ins are processed before another so as to allow plug-ins to communicate with each other. E.g. one plug-in may provide critical information to another and so must be allowed to be processed first.

optional

Whether or not plug-in can be skipped by the user.

requires

Which version of Pyblish is required by this plug-in. Plug-ins requiring a version newer than the current version will not be loaded. 1.0.8 was when `Plugin.requires` was first introduced.

actions

Actions associated to this plug-in

actions = []

active = True

families = ['*']

hosts = ['*']

id = 'Plugin'

label = None

log = <logging.Logger object at 0x7f0820764a50>

optional = False

order = -1

process ()

Primary processing method

This method is called whenever your plug-in is invoked and is injected with object relative to it's signature.

E.g. `process(self, context, instance)` will have the current context and instance injected into it at run-time.

Available objects:

- context
- instance
- user
- time

Raises Any error

repair ()

DEPRECATED

requires = 'pyblish>=1'

version = (0, 0, 0)

4.5 Selector

A selector finds instances within a working file.

Note: The following attributes must be present when implementing this plug-in.

- `Selector.hosts`
- `Selector.version`

`pyblish.plugin.Selector`
alias of `Collector`

4.6 Validator

A validator validates selected instances.

Note: The following attributes must be present when implementing this plug-in.

- `Plugin.hosts`
- `Plugin.version`
- `Validator.families`

```
class pyblish.plugin.Validator
    Validate/check/test individual instance for correctness.

    log = <logging.Logger object at 0x7f0820764c90>

    order = 1
```

4.7 Extractor

Extractors are responsible for serialising selected data into a format suited for persistence on disk. Keep in mind that although an extractor does place file on disk, it isn't responsible for the final destination of files. See [Conformer](#) for more information.

Note: The following attributes must be present when implementing this plug-in.

- `Plugin.hosts`
- `Plugin.version`
- `Extractor.families`

class `pyblish.plugin.Extractor`

Physically separate Instance from Host into corresponding resources

`log = <logging.Logger object at 0x7f0820764d10>`

`order = 2`

4.8 Conformer

The conformer, also known as *integrator*, integrates data produced by extraction.

Its responsibilities include:

1. Placing files into their final destination
2. To manage and increment versions, typically involving a third-party versioning library.
3. To notify artists of events
4. To provide hooks for out-of-band processes

Note: The following attributes must be present when implementing this plug-in.

- `Plugin.hosts`
- `Plugin.version`
- `Conformer.families`

`pyblish.plugin.Conformer`

alias of `Integrator`

4.9 discover

`pyblish.plugin.discover` (*type=None, regex=None, paths=None*)

Find and return available plug-ins

This function looks for files within paths registered via `register_plugin_path()` and those added to `PYBLISHPLUGINPATH`.

It determines *type* - `Selector`, `Validator`, `Extractor` or `Conform` - based on whether it matches it's corresponding regular expression; e.g. `"$validator_.*"` for plug-ins of type `Validator`.

Parameters

- **type** (*str, optional*) – !DEPRECATED! Only return plugins of specified type. E.g. validators, extractors. In None is specified, return all plugins. Available options are “selectors”, “validators”, “extractors”, “conformers”, “collectors” and “integrators”.
- **regex** (*str, optional*) – Limit results to those matching *regex*. Matching is done on classes, as opposed to filenames, due to a file possibly hosting multiple plugins.
- **paths** (*list, optional*) – Paths to discover plug-ins from. If no paths are provided, all paths are searched.

4.10 plugin_paths

`pyblish.plugin.plugin_paths()`

Collect paths from all sources.

This function looks at the three potential sources of paths and returns a list with all of them together.

The sources are:

- Registered paths using `register_plugin_path()`,
- Paths from the environment variable `PYBLISHPLUGINPATH`
- Paths from configuration

Returns list of paths in which plugins may be located

4.11 registered_paths

`pyblish.plugin.registered_paths()`

Return paths added via registration

..note:: This returns a copy of the registered paths and can therefore not be modified directly.

4.12 configured_paths

`pyblish.plugin.configured_paths()`

Return paths added via configuration

4.13 environment_paths

`pyblish.plugin.environment_paths()`

Return paths added via environment variable

4.14 register_plugin_path

`pyblish.plugin.register_plugin_path(path)`

Plug-ins are looked up at run-time from directories registered here

To register a new directory, run this command along with the absolute path to where you’re plug-ins are located.

Example

```
>>> import os
>>> my_plugins = "/server/plugins"
>>> register_plugin_path(my_plugins)
'/server/plugins'
```

Returns Actual path added, including any post-processing

4.15 deregister_plugin_path

`pyblish.plugin.deregister_plugin_path(path)`

Remove a `pyblish._registered_paths` path

Raises `KeyError` if `path` isn't registered

4.16 deregister_all

4.17 plugins_by_family

4.18 plugins_by_host

4.19 instances_by_plugin

4.20 Config

4.21 log

`pyblish.lib.log(cls)`

Decorator for attaching a logger to the class `cls`

Loggers inherit the syntax `{module}.{submodule}`

Example

```
>>> @log
... class MyClass(object):
...     pass
>>>
>>> myclass = MyClass()
>>> myclass.log.info('Hello World')
```

4.22 format_filename

`pyblish.lib.format_filename(filename)`

Convert arbitrary string to valid filename, django-style.

Modified from `django.utils.text.get_valid_filename()`

Returns the given string converted to a string that can be used for a clean filename. Specifically, leading and trailing spaces are removed; other spaces are converted to underscores; and anything that is not a unicode alphanumeric, dash, underscore, or dot, is removed.

Usage:

```
>>> format_filename("john's portrait in 2004.jpg")
'johns_portrait_in_2004.jpg'
>>> format_filename("something^_SD.dda.//fd/ad.exe")
'something_SD.dda.fdad.exe'
>>> format_filename("Napoleon_:namespaces_GRP|group1_GRP")
'Napoleon_namespaces_GRPgroup1_GRP'
```

4.23 PyblishError

`class pyblish.error.PyblishError`
Baseclass for all Pyblish exceptions

4.24 SelectionError

`class pyblish.error.SelectionError`
Baseclass for selection errors

4.25 ValidationError

`class pyblish.error.ValidationError`
Baseclass for validation errors

4.26 ExtractionError

`class pyblish.error.ExtractionError`
Baseclass for extraction errors

4.27 ConformError

`class pyblish.error.ConformError`
Baseclass for conforming errors

p

pyblish, 16
pyblish.error, 17
pyblish.lib, 16
pyblish.plugin, 9

Symbols

`__contains__()` (pyblish.plugin.Context method), 10
`__getitem__()` (pyblish.plugin.Context method), 10

A

AbstractEntity (class in pyblish.plugin), 9
actions (pyblish.plugin.Plugin attribute), 12
active (pyblish.plugin.Plugin attribute), 12
add() (pyblish.plugin.AbstractEntity method), 9
add() (pyblish.plugin.Context method), 10

C

configured_paths() (in module pyblish.plugin), 15
Conformer (in module pyblish.plugin), 14
ConformError (class in pyblish.error), 17
Context (class in pyblish.plugin), 10
context (pyblish.plugin.Instance attribute), 11
create_asset() (pyblish.plugin.Context method), 10
create_instance() (pyblish.plugin.Context method), 10

D

deregister_plugin_path() (in module pyblish.plugin), 16
discover() (in module pyblish.plugin), 14

E

environment_paths() (in module pyblish.plugin), 15
ExtractionError (class in pyblish.error), 17
Extractor (class in pyblish.plugin), 14

F

families (pyblish.plugin.Plugin attribute), 12
format_filename() (in module pyblish.lib), 16

G

get() (pyblish.plugin.Context method), 11

H

has_data() (pyblish.plugin.AbstractEntity method), 9
hosts (pyblish.plugin.Plugin attribute), 12

I

id (pyblish.plugin.Context attribute), 11
id (pyblish.plugin.Instance attribute), 11
id (pyblish.plugin.Plugin attribute), 12
Instance (class in pyblish.plugin), 11

L

label (pyblish.plugin.Plugin attribute), 12
log (pyblish.plugin.Extractor attribute), 14
log (pyblish.plugin.Instance attribute), 11
log (pyblish.plugin.Plugin attribute), 12
log (pyblish.plugin.Validator attribute), 13
log() (in module pyblish.lib), 16

N

name (pyblish.plugin.Instance attribute), 11

O

optional (pyblish.plugin.Plugin attribute), 12
order (pyblish.plugin.Extractor attribute), 14
order (pyblish.plugin.Plugin attribute), 12
order (pyblish.plugin.Validator attribute), 13

P

parent (pyblish.plugin.Instance attribute), 11
Plugin (class in pyblish.plugin), 12
plugin_paths() (in module pyblish.plugin), 15
process() (pyblish.plugin.Plugin method), 12
pyblish (module), 3, 16
pyblish.error (module), 7, 17
pyblish.lib (module), 5, 16
pyblish.plugin (module), 1, 9
PyblishError (class in pyblish.error), 17

R

register_plugin_path() (in module pyblish.plugin), 15
registered_paths() (in module pyblish.plugin), 15
remove() (pyblish.plugin.AbstractEntity method), 9
remove_data() (pyblish.plugin.AbstractEntity method), 9
repair() (pyblish.plugin.Plugin method), 13

requires (pyblish.plugin.Plugin attribute), 12, 13

S

SelectionError (class in pyblish.error), 17

Selector (in module pyblish.plugin), 13

set_data() (pyblish.plugin.AbstractEntity method), 10

V

ValidationError (class in pyblish.error), 17

Validator (class in pyblish.plugin), 13

version (pyblish.plugin.Plugin attribute), 12, 13