
Parzzley

Release 4.0.3003

Author name not set

Dec 09, 2025

CONTENTS

1	License	3
2	Up-To-Date?	5
3	Dependencies	7
4	Introduction	9
4.1	First Steps	9
4.2	Synchronization Model	9
5	Using Parzzley In Service Mode	11
6	Configuration	13
6.1	Synchronization Volumes	13
6.2	Loggers	13
7	Customizing Parzzley	15
7.1	Synchronization Workflow	15
7.2	Customizable Parts	21
8	Appendix: Installation	23
8.1	Source Code Archive	23
9	Command Line Interface Reference	25
9.1	Positional Arguments	25
9.2	Sub-commands	25
10	API Reference	27
10.1	parzzley package	27
	Python Module Index	139
	Index	141

Parzzley keeps a configured set of places in filesystems in sync.

Features:

- Keeps configured file system places in sync (local and ssh).
- Robust infrastructure with working retry and error handling.
- Customizable behavior with the availability to add additional program logic for various situations.
- Optional 'move to sink mode': always moves all files from the sources to a sink.
- Supports extended attributes.
- Can be used stand-alone, as a service, or embedded in other tools with a flexible and extensible API.

LICENSE

Parzzley is distributed under the terms of the AGPL 3 license. This also affects all included files without a license header (non-source files like images), unless they are explicitly mentioned as third-party content. Read the ‘Dependencies’ section for included third-party stuff.

UP-TO-DATE?

Are you currently reading from another source than the homepage? If you are in doubt whether your package is up-to-date, you should visit the project homepage and check that. You are currently reading the documentation for version 4.0.3003.

DEPENDENCIES

Parzzley makes use of some third-party parts.



Required: **Python 3.13**



Required: **Python package lxml ~= 6.0**



Required: **Python package watchdog ~= 6.0.0**



Recommended: **inotifywait** ((on remote machines for change detection; part of ‘inotify-tools’; optional))



Recommended: **ssh** ((for connections to remote machines))



Recommended: **GNU/Linux**



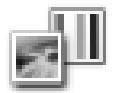
Included: **background image** (License: <https://creativecommons.org/licenses/by-sa/3.0/>; from [here](#))



Included: **font ‘Inconsolata’** (for websites; by Raph Levien; License: OFL; from [here](#))



Included: **font ‘Khula’** (for websites; by Erin McLaughlin; License: OFL; from [here](#))



Included: **font ‘Symbola’** (for logo symbol; License: free for use; from [here](#))

INTRODUCTION

Please read how to make Parzzley ready for the first steps in *Appendix: Installation*.

4.1 First Steps

TODO

4.2 Synchronization Model

TODO

USING PARZZLEY IN SERVICE MODE

TODO

CONFIGURATION

TODO

6.1 Synchronization Volumes

TODO

6.2 Loggers

TODO

CUSTOMIZING PARZZLEY

7.1 Synchronization Workflow

7.1.1 Overview

The following describes how the inner parts of Parzzley work together. This knowledge is very helpful for designing and implementing custom logic. It is not relevant for basic usage, though. Please note that this chapter will regularly apply some (sometimes heavy) simplifications in order to improve readability, and to not burden the reader with all the technical details. In order to find out what really happens to the last detail, continue reading to the end of this chapter, or finally study the Parzzley source code itself.

Each sync run is associated to a particular `parzzley.sync.Volume`. It usually comes from the user's configuration. Inside it, there are at least two sites (`parzzley.fs.Site`) defined that need to be synchronized. On each site, some aspects (`parzzley.sync.aspect.Aspect`) are defined (configuration-wise, aspects are often defined on volume-level; this is the same as defining them for each site). Aspects implement all the actual synchronization behavior. Without any aspect, the sync engine would do virtually nothing (it would not even touch any site at all). The `DefaultSync` aspect can be found in most typical Parzzley configurations. It is a bundle of various kinds of smaller aspects, each participating in the synchronization routine in a particular way.

During a sync run, the engine triggers a defined routine of events (details below). Each aspect implements its behavior by providing handlers to some of these events. Whenever an event occurs, all registered event handlers are executed. Each event handler execution takes place for one of your specified site (since aspects are defined per site). Each event handler typically does its work in that particular site.

There is a mechanism for ordering the execution of the event handlers within one event. Please read `parzzley.sync.aspect.event_handler()` for more details about the ordering and more about the internals.

7.1.2 High-Level Model

Important at first is to understand the model of a file synchronization task, independent of Parzzley design details.

In order to synchronize one particular file (more precisely: one path) between all connected sites, first we need to determine some information about that path on each site. This includes the item type, if it even exists, and a 'cookie' (of the main stream to be precise). The cookie is an opaque data structure that is determined by site and by path (see later: also by stream). It is guaranteed to change whenever the content of that file stream changes.

On top of this information, it can be determined which site(s) were updated meanwhile (assuming the cookies from the last run are known; see later). If no site has seen any update, then everything is still up-to-date and no further actions are needed. If more than one site has seen an update, then there is a conflict (unless the updates were identical). If exactly one site has seen an update, all other sites should later receive the main stream content from this one.

All that happened so far was related to the "main stream" of that file, i.e. its actual file content. There are other streams as well, although usually for smaller pieces of metadata. One of those other streams is responsible for some POSIX file attributes, e.g. it keeps the file modification times in sync. After the routine from above was executed for the main

stream, we do similar things for all these other streams as well. The precise way, e.g. how they detect changes, can be a bit different for them though, for various reasons.

After that took place, we know for each stream whether it needs an update at all, at from which site to take it. At that moment, conflict handling takes place (and, in the worst case, skips most of the further steps and leaves conflict resolution to the user). Once conflict handling finishes successfully, for each stream the content can be transferred from the source site to the destination sites, as determined in earlier steps.

Whenever possible, this should happen to an intermediate location at first, so the switch from the old version to the new version can happen quickly, but did not take place yet. Then, as soon as these transfers are completed, there can be a check for each stream, whether its source site remained stable or it has seen changes during the transfer. In the latter case, all transfers just get dropped if possible and the entire item sync gets retried with the same stream master sites. If everything remained stable during transfer, it can then be actually applied. Directly after that, all destination sites have to determine a new cookie for that file, which gets stored for comparisons in later runs.

Once that is done, this file is considered as successfully synchronized. Of course, there is more to say about various other cases, like directories or deletion of items. The next section will explain all these cases by a more detailed look into the Parzzley engine routine.

The following text in this section tries to collect pitfalls and potential dangers and how Parzzley deals with them if needed (all assuming a typical configuration, using the default aspects only; and nothing here is a guarantee in any legally relevant way, of course):

- How does it behave for the first synchronization, i.e. if no cookies from earlier runs are available?

If there are already items stored in some sites at the time of first synchronization, there is no state info about it yet. This is the same as when the user has created new files later on. If there is conflicting content on some sites, conflicts will come up, which need to be resolved by the user. Also, if there are already files, even if they are equal on all sites, it will take time to compare them. There are further questions below that look at some other situations that are similar to ‘first synchronization’.

- What happens if a site loses its connection during synchronization?

Whenever a site is used that has lost its connection, this will raise a critical error. See also the question about “a critical error during synchronization”.

- What happens if some aspect raises a critical error during synchronization?

In general, this will stop the entire sync run as soon as possible and only store the new state info for the items that were already passed successfully. There are various situations where exactly an error could occur (and e.g. various sites or aspects could even be in somewhat different situations). In the early steps of item synchronization, nothing gets touched, so errors will only abort the sync run, but a subsequent one can follow up seamlessly. If a critical error occurs during an in-place update of an item, this may lead to conflicts and further problems in theory (see also the question about “transfer should ideally take place to some intermediate location”). If a critical error occurs during an update via a temporary working item, but before applying it, it had no effect yet, so it will only abort the sync run. If a critical error occurs during the apply-update step of an update via a temporary working item, it will either leave the destination untouched or apply the update atomically. If a critical error occurs between the apply-update step and the end (or in the latter case of the last situation), e.g. during determining new cookies, the issues could be similar to the “in-place update” situation.

Although all these considerations from above are correct most of the times, there are nuances. E.g. in very specific cases, Parzzley needs to remove an existing item on a site, just in order to replace it with a different one later (this usually happens when a file turned into a directory or vice versa). Without further measurements, those situations could lead to data loss if Parzzley runs into problems between cleaning up and replacing the item. Parzzley is designed to avoid those situations as good as possible, but does it in a very safe way (keeping rollback-data) when it has to.

- What happens if Parzzley gets killed during synchronization (or the machine just loses its power supply)?

The effects will be similar to “a critical error during synchronization” (see also that question) would have, with

the additional fact that no state info updates from this sync run will be persisted. That does not lead to any issues in real situations (see also the question about “problems with persisting the state info”).

- Can my manual updates overlap in some way with an ongoing synchronization, leading to broken synchronization or even data loss?

Parzzley is explicitly designed to be careful with that, so overlapping manual updates will just lead to conflicts most of the time. However, there are corner cases where concurrent updates may lead to data loss. For example, if Parzzley detects changes on the source site during transfer, the entire item sync will be retried with the same stream master sites (which might overwrite your changes). It is also problematic if the user makes a change to some non-main stream on a site after this site just got updated, but before determining the new stream cookies (this change will never be detected later).

Those cases are rare, but in general it is not recommended to apply changes to the same item on multiple sites between sync runs.

- What happens if very often only some of the defined sites are available?

Parzzley will always skip an item if none of the sites that have synced it successfully last time are connected. This avoids any real problems, but cannot prevent conflicts appearing more often. It is recommended to have at least one site in each volume that can always connect (e.g. a local one).

- What happens if disk errors occur?

This can lead to various issues, also depending on the kind of error and whether it occurred on local Parzzley data or site data. Reading wrong data will lead to data corruption. Whenever IO operations run into problems, the direct effect will be like in the question about “a site loses its connection”. If the disk error leads to the loss of some data (either files on some site or some Parzzley state info), there are various other questions about more details.

- What happens if there are problems with persisting the state info, like the last seen stream cookies?

First, this would be a very uncommon event, either related to a bug in Parzzley or to local IO problems. The worst thing that can happen during persisting one variable of state info is that it gets interrupted, leaving it with its old value. Furthermore, all relevant state info are stored in one variable, so Parzzley will always get a consistent view on that. If there were problems during persistence, these are the information from the end of an earlier sync run, though. This should not lead to any data loss in real situations. It might lead to some conflicts detected in the next run, which should be resolved automatically (because item content is still equal). So, beyond degraded performance in the next run, there would be no danger in such an event (it should not happen regularly of course).

- Will conflict detection always prevent data loss?

Parzzley is designed to prevent data loss due to concurrent changes and let the user resolve the conflict instead. However, in some situations, with unfortunate timing, data loss can happen instead. See also the question about when “updates overlap in some way”.

- The text above says that stream content transfer should ideally take place to some intermediate location, so it can be applied quickly after some checks but has no visible effects before; in what cases is this not possible and what are the consequences?

Whenever a destination is up-to-date regarding its main stream, but needs to be refreshed regarding some non-main streams, there will be direct in-place updates. These updates could be interrupted at any moment (e.g. if the site loses its connection), maybe leaving it in a broken state. No matter what kind of interruption takes place, the internal state info for this item will stay untouched. So, a subsequent sync run will detect a conflict that maybe needs to be resolved by the user. In theory, the now broken destination could have been the source as well, though. It has then lost its old data. The overall risk is low enough to ignore this issue for practical purposes, though.

- What happens if a site was already synchronized by Parzzley earlier and now gets synchronized by a fresh Parzzley installation?

Parzzley will detect this case and automatically does the necessary cleanup on all sites. The initial synchronization will take longer, because Parzzley needs some state data from earlier runs in order to synchronize faster (but by nature, a fresh Parzzley installation had no chance to collect these data yet).

- What happens if a particular site loses its internal Parzzley site data (i.e. the `..parzzley.control` directory)?

The effects are mostly the same as for the question before. Parzzley will drop all state data for that site, so the next sync run will take longer in this case as well.

- What happens if i have one filesystem location used as a site by more than one volume (either of the same Parzzley instance or even different ones)?

The concurrent Parzzley instances will constantly try to take over control from the last one. It will trigger a warning in that case and fall back to a slow sync run (like in the question about “site was already synchronized by Parzzley earlier”). Even worse things might happen, including data loss. You should never sync one filesystem location by more than one Parzzley volume. This also includes the reuse of subdirectories (although this will not be detected and is also less dangerous).

- What happens if a particular site gets physically replaced by a completely empty one, so the next synchronization will see nothing there?

At first, the effects are mostly the same as for the questions before. Then, this site will receive all the content from the other sites. In the end, the files stored on the new disk are the same as before (assuming there were no conflicts before, nothing was excluded from synchronization, ...).

- What happens in a situation like described in the following?: At first, the user applies changes to a particular file, at first on site 1, then different changes on site 2. Then a Parzzley sync run gets started, where it was able to connect to site2 and site22 only. Then another sync run gets started, where it was able to connect to site1 and site11 only. Then a third sync run gets started, where it was able to connect to all sites.

At first, Parzzley will skip an item if none of the sites that have synced it successfully last time are connected. So, the given situation will end up in conflicts that need to be resolved by the user. In order to prevent conflicts in lots of such situations, it is recommended to have at least one site in each volume that can always connect (e.g. a local one). See also `test_parallel_site_updates`.

- What happens if a particular site has no support for a particular stream, e.g. the ‘xattrs’ stream for extended file attributes?

Currently, all sites support the same set of streams and sync runs would instantly abort otherwise.

- What happens if a file gets modified by the user on one site but loses all its extended attributes due to weaknesses of the editor software?

Parzzley will detect that case and restore the extended attributes from the other sites.

- What happens (with real machines) if either a remote site or the Parzzley machine itself loses power during a sync run?

Unfortunately, on real machines, data has a long way to travel. Some of them are decoupled from each other and Parzzley is not in control about all of them. In extreme cases, even shortly after a sync run, a machine could lose power and corrupt some files this way! Data could be still in an operating system cache only, and even after sent to the disk, the disk itself also has caches and buffers. Particularly consumer disks can be dangerous and even corrupt existing data or the filesystem itself.

Internal Parzzley state data structures are designed to be robust in such cases, so in typical cases, their corruption will only make the next sync run taking much longer. One potential consequence of a power loss on some (maybe remote) site, though, could be that a particular file is corrupted on its disk, but Parzzley considers it as correct (until it has to do a slow run for some reasons, where it actually compares file content), and maybe even updates other sites later with that corrupted copy.

So, the only safe way is to avoid regular power losses in general. If they happen while actual sync work was going on (i.e. there was some actual transfer happening), a good safety measure would be to force a slow sync

run (which actually compares file contents).

7.1.3 Event Routine

The following will explain the event routine that is triggered by the engine, with a short explanation for each one (there is further documentation for each event):

- At the beginning of the sync run, `parzzley.sync.aspect.events.sync_run.Prepare` is triggered. Aspects can listen to that event in order to do arbitrary initialization steps, e.g. for data structures used in later events.
- Then it triggers `parzzley.sync.aspect.events.sync_run.DetermineStreamSupport` in order to determine which streams are supported, and for which item types (e.g. some streams are only supported for ordinary files and directories or similar). There will be events for each of those streams in later stages. Beyond the main stream, typical streams are "posix" (POSIX file metadata like the modification time) and "xattrs" (extended file attributes). For each stream to be supported, there must be a dedicated aspect that explicitly checks and declares its availability. Directly afterwards, it triggers `parzzley.sync.aspect.events.sync_run.ValidateStreamSupport` in order to get validated global stream support info from the per-site info determined in the previous step. The typical implementation checks if all sites have determined the same supported streams (with the same item types). It will then take this as the global result or will fail otherwise.
- Then it triggers a series of events for the root directory of the sync volume. Due to a recursion, the same series of events will be triggered for each subdirectory and each file below that root (i.e. all your files). Each of your files, directories, etc. is called an "item"! These "per-item events" will be triggered:
 - At first, `parzzley.sync.aspect.events.item.DecideToSkip` is triggered in order to check things like path exclusions. If it does decide to skip this item, all further events for this item will be skipped and the next item gets processed. If not, these events will be triggered:
 - * First, `parzzley.sync.aspect.events.item.Prepare` is triggered in order to prepare item synchronization. Typical implementations use this event e.g. for handling (parts of) the item-retry logic.
 - * Then, for each supported stream, these "per-stream events" will be triggered (the first stream is always the main stream "", then all other available streams) in order to do various preparation steps like detecting changes, determining the master site and checking for conflicts:
 - Some information about the current stream get collected from the involved sites. It triggers `parzzley.sync.aspect.events.item.stream.DetermineComparator` for setting up stream-specific comparators. Then it triggers `parzzley.sync.aspect.events.item.stream.DetermineCookie` in order to determine the current cookie for the stream per site. This is used later for detection of changes (e.g. on a file) and more things. The typical implementation just queries the cookie from the site. Then, if we are dealing with the main stream, it also triggers `parzzley.sync.aspect.events.item.DetermineType` in order to determine the item type (is it a directory, a normal file, a symlink, ...?) from the cookie. The typical implementation just queries `item_type_by_cookie` from the site, using the cookie from the step before.
 - The following steps are about determining the master site for this stream. At first, in order to detect which sites have seen changes to this item's stream since the last sync run, it triggers `parzzley.sync.aspect.events.item.stream.DetectChanges`. This needs to be implemented individually for each stream. Implementations often involve checking whether the cookie has been changed compared to earlier moments). Then it triggers `parzzley.sync.aspect.events.item.stream.DetermineMasterSiteTable` in order to determine the master site for this stream (and the score table). This is usually the site that has the most recent version of the stream, which in turn is going to be transferred to all other sites (if they are not up-to-date already). The typical implementation determines that based on the information gathered in `item.stream.DetectChanges`. After that, it triggers `parzzley.sync.aspect.events.item.stream.MasterSiteDetermined` for arbitrary steps after the stream's master site is determined (there is no "typical implementation" for it).

- * Then, in order to collect source streamables, `parzzley.sync.aspect.events.item.stream.CollectSourceStreamables` gets triggered for each stream. Each site will provide its own source streamable here. For most purposes, only the source streamable of the master site will be used later; but that's not strictly true for all aspect implementations. The typical implementation directly gets the source streamable from the site. For some streams, there may be additional aspects that do further things, e.g. adding stream pipes for some arbitrary stream translations.
- * Then conflict handling takes place. For each stream, that triggers at first `parzzley.sync.aspect.events.item.stream.DetermineConflicts` in order to check whether there are conflicts between some sites for the current stream. The typical implementations do checks around the item type, but can also compare stream contents (i.e. the source streamables collected earlier). If there were conflicts, it triggers `parzzley.sync.aspect.events.item.stream.TryResolveConflicts` (one typical implementation allows to do that based on externally provided conflict resolution data) and `parzzley.sync.aspect.events.item.stream.ApplyConflictResolution` (the typical implementation applies the resolution from the step before) in order to try resolving them.
- * If there are any unresolvable conflicts for any stream of this file/directory, it triggers `parzzley.sync.aspect.events.item.SkipDueToConflicts` and skips most further per-item or even per-stream events. Otherwise the following happens:
 - It triggers `parzzley.sync.aspect.events.item.PrepareUpdating` in order to prepare further item processing in some way (there is no "typical implementation" for it) and then `parzzley.sync.aspect.events.item.SetUpWorkingItems` in order to optionally prepare working items. One major usage of working items is to prepare new items in a temporary, invisible location so it can be moved to its real destination atomically.
 - Then these (usually per-stream-) events are triggered for each available stream:
 - It triggers `parzzley.sync.aspect.events.item.stream.CollectDestinationStreamables` in order to collect destination streamables. For most purposes, only the destination streamables of the non-master sites will be used later; but that's not strictly true for all aspect implementations. The typical implementation takes the destination streamable either from the working item that was set up earlier in `item.SetUpWorkingItems` or directly from the site, but other aspects can add further handlers for particular streams.
 - It triggers `parzzley.sync.aspect.events.item.stream.TransferUpdate` in order to do the necessary data transfers for updating the stream. The typical implementation runs a loop that reads from the source streamable and writes to the destination streamables (for directories, it makes sure that it exists at least before). There are some aspects that do additional things for particular streams.
 - If we are dealing with the main stream and the current item is a directory (on master site), some "per-directory events" are triggered for updating a directory. At first it triggers `parzzley.sync.aspect.events.item.dir.Prepare` in order to prepare the directory traversal in some way. Then triggers `parzzley.sync.aspect.events.item.dir.List` in order to get a list of all children. The typical implementation retrieves that list directly from the site. Then it triggers the entire series of per-item (and per-stream) events for all children! This is the recursion that lets the engine visit all files in the tree below the root directory! After that, it triggers `parzzley.sync.aspect.events.item.dir.Iterated` in order to finish directory traversal in some way.
 - It then triggers `parzzley.sync.aspect.events.item.ApplyUpdate` for this item in order to finally commit the updates made in the earlier steps (ideally in an atomic way). One usual handler implementation would move the working item to its final actual location. During this event, each updated site must also store an "updated cookie" and some more information by means of `item.ApplyUpdate.update_cookies()`.

- * It triggers `parzzley.sync.aspect.events.item.RefreshItemsBook` in order to store some information about the item and its streams for the next sync run. The typical implementation primarily stores the “updated cookie” (determined in `item.ApplyUpdate`), so it can be used for decisions in the next sync run; e.g. in `item.stream.DetectChanges`.
- At the end of the sync run, if no critical error has occurred, `parzzley.sync.aspect.events.sync_run.Finish` is triggered (e.g. in order to store some status information).
- Finally, and even if a critical error has occurred, `parzzley.sync.aspect.events.sync_run.Close` is triggered (e.g. in order to store some status information locally, even if critical problems occurred in the sync run and some sites might have been disconnected).

7.2 Customizable Parts

If you want to override or enhance some parts of the default behavior, read the following parts:

- `parzzley.sync.aspect.Aspect` is the base class of your implementation if you want to develop a part of logical behavior. Inspect the sources in `parzzley.builtin.aspects` for lots of practical examples.
- `parzzley.fs` is the module about filesystems. Read this if you want to implement a custom kind of site.

APPENDIX: INSTALLATION

Install Parzzley via the installation package for your environment, if a suitable one exists for download. This also takes care of installing dependencies and doing preparation (unless mentioned otherwise in the installation procedure). After the installation, you can skip the rest of this section.

8.1 Source Code Archive

Use the source code archive as fallback. Extract it to a location that is convenient to you. Also take care of installing all requirements.

It is highly recommended to also establish a symlink or alias from `parzzley` to `src/parzzley/parzzley_cli.py`.

COMMAND LINE INTERFACE REFERENCE

```
usage: parzzley [-h] [command] ...
```

9.1 Positional Arguments

[command]	Possible choices: sync, sync_loop What to do?
------------------	--

9.2 Sub-commands

9.2.1 sync

Execute synchronization for once for a given volume.

```
parzzley sync [-h] config_dir [volume_names ...]
```

Positional Arguments

config_dir	Directory with the Parzzley sync configuration.
volume_names	Names of the volumes to sync. If none are specified, all volumes will be synced.

9.2.2 sync_loop

Execute synchronization loop. Note: This should be done by a background service. Depending on how Parzzley was set up, such a service already exists and is running!

```
parzzley sync_loop [-h] config_dir
```

Positional Arguments

config_dir	Directory with the Parzzley sync configuration.
-------------------	---

API REFERENCE

10.1 parzzley package

Parzzley.

10.1.1 Subpackages

parzzley.asset package

Auxiliary info about Parzzley.

Submodules

parzzley.asset.data module

Parzzley data files.

`parzzley.asset.data.readme_pdf(culture)`

Return the path to the README file for a given culture.

Parameters

culture (*str*) – The two-letter culture code.

Return type

Path

parzzley.asset.project_info module

parzzley.builtin package

Builtin implementations of some interfaces.

`parzzley.builtin.load_builtin_implementations()`

Load all builtin implementations.

Do not use it directly. It is already called implicitly.

Subpackages

parzzley.builtin.aspects package

Parzzley builtin aspects. See also `parzzley.sync.aspect`.

Submodules

parzzley.builtin.aspects.attach_sites module

Aspects for attaching sites to the current manager and detecting foreign sites.

class parzzley.builtin.aspects.attach_sites.**AttachSites**(*inner_aspects)

Bases: [Aspect](#)

Attaches sites and detects foreign sites.

Parameters

inner_aspects ([Aspect](#))

_LAST_SYNC_RUN_SN_FILE = <parzzley.fs._Item object>

_EARLY_LAST_SYNC_RUN_SN_FILE = <parzzley.fs._Item object>

_WORKING_LAST_SYNC_RUN_SN_FILE = <parzzley.fs._Item object>

_MANAGED_BY_FILE = <parzzley.fs._Item object>

_EXCEPTED_LAST_SYNC_RUN_SN_PER_SITE_KEY = 'expected_last_sync_run_sn_per_site'

async **attach_site**(event)

Attaches sites.

Parameters

event ([Prepare](#))

async **__check_whether_to_stay_attached**(event)

Parameters

event ([Prepare](#))

Return type

bool

async **__read_last_sync_run_sn**(site, from_items)

Parameters

- **site** ([Site](#))
- **from_items** ([Iterable](#)[[Item](#)])

Return type

int | None

async **__take_over_site_control**(event)

Parameters

event ([Prepare](#))

Return type

None

async **__store_current_sync_run**(event)

Parameters

event ([Prepare](#))

Return type

None


```
async __write_last_sync_run_sn(site, with_items, value)
```

Parameters

- **site** ([Site](#))
- **with_items** (*tuple*[[bytes](#) | [Item](#), [bytes](#) | [Item](#), [bytes](#) | [Item](#)])
- **value** (*int*)

Return type

[None](#)

parzzley.builtin.aspects.conflicts module

Aspects for conflict detection and handling.

```
class parzzley.builtin.aspects.conflicts.DetectItemTypeConflicts(*inner_aspects)
```

Bases: [Aspect](#)

Detect item type conflicts (e.g. file vs. directory, file vs. symlink, ...).

Parameters

inner_aspects ([Aspect](#))

```
async determine_conflicts(event)
```

Detect type conflicts.

Parameters

event ([DetermineConflicts](#))

```
class parzzley.builtin.aspects.conflicts.DetectContentConflicts(*inner_aspects)
```

Bases: [Aspect](#)

Detect content conflicts (i.e. the stream content differs).

Parameters

inner_aspects ([Aspect](#))

```
_DATA__CONTENT_EQUAL_TO_MASTER = <parzzley.sync.aspect.events.Data object>
```

```
async determine_conflicts(event)
```

Detect stream content conflicts.

Parameters

event ([DetermineConflicts](#))

```
async __are_streams_equal(streamable_1, streamable_2)
```

```
__source_streamable(event, site)
```

Parameters

- **event** ([DetermineConflicts](#))
- **site** ([Site](#))

Return type

[ReadStreamable](#)

```
class parzzley.builtin.aspects.conflicts.TrackConflicts(*inner_aspects)
```

Bases: [Aspect](#)

Stores conflict data for later resolution.

Parameters**inner_aspects** ([Aspect](#))**_DATA__CONFLICTS** = <parzzley.sync.aspect.events.Data object>**async store_conflict_data**(*event*)

Store conflict data for later resolution from outside (e.g. manually by the user).

Parameters**event** ([SkipDueToConflicts](#))**async cleanup_conflicts_storage_site**(*event*)

Cleanup of outdated stored conflict data.

Parameters**event** ([Finish](#))**class** parzzley.builtin.aspects.conflicts.**TryResolveConflictsByHint**(**inner_aspects*)Bases: [Aspect](#)

Resolves conflicts by external resolution hint files.

Parameters**inner_aspects** ([Aspect](#))**async resolve_conflicts_by_hint_file**(*event*)

Resolves conflicts by stored conflict resolution data.

Parameters**event** ([TryResolveConflicts](#))**class** parzzley.builtin.aspects.conflicts.**ApplyConflictResolution**(**inner_aspects*)Bases: [Aspect](#)Apply the conflict resolution determined in [parzzley.sync.aspect.events.item.stream.TryResolveConflicts](#).**Parameters****inner_aspects** ([Aspect](#))**async apply_conflict_resolution**(*event*)

Apply the conflict resolution.

Parameters**event** ([ApplyConflictResolution](#))**parzzley.builtin.aspects.conflicts._conflict_directory_item**(*item*)**Parameters****item** ([Item](#))**Return type**[Item](#)**async** parzzley.builtin.aspects.conflicts.**_conflicts_storage_site**(*event*)**class** parzzley.builtin.aspects.conflicts.**_Source**(*stream*: [parzzley.fs.stream.ReadStream](#), *buffer*: bytes = b'', *finished*: bool = False)

Bases: object

Parameters

- **stream** ([ReadStream](#))

- **buffer** (*bytes*)
- **finished** (*bool*)

stream: *ReadStream*

buffer: `bytes = b''`

finished: `bool = False`

async read_more()
Read some more data.

Return type
None

parzzley.builtin.aspects.default_sync module

Common default sync aspects; basically combining some of the other ones together for simpler usage.

class `parzzley.builtin.aspects.default_sync.DefaultBase`

Bases: *Aspect*

Collection of base aspects required in many setups, but not yet a complete setup. See e.g. *DefaultSync* and *parzzley.builtin.aspects.pull_and_purge*.

class `parzzley.builtin.aspects.default_sync.DefaultSync`

Bases: *Aspect*

Nearly complete sync setup; only missing a removal strategy (see *parzzley.builtin.aspects.remove*).

parzzley.builtin.aspects.directory module

Directory support.

class `parzzley.builtin.aspects.directory.ListDirectory(*inner_aspects)`

Bases: *Aspect*

Retrieves the list of all item children names.

Parameters

inner_aspects (*Aspect*)

async list_dir(event)

List all children of this directory.

Parameters

event (*List*)

class `parzzley.builtin.aspects.directory.MarkChangedIfItemIsDirectoryButWasNotBefore(*inner_aspects)`

Bases: *Aspect*

Marks directories as changed if they were not there (or not a directory) last time.

Parameters

inner_aspects (*Aspect*)

async detect_changes(event)

Detect changes.

Parameters

event (*DetectChanges*)

```
class parzzley.builtin.aspects.directory.DirectoryCreation(*inner_aspects)
```

Bases: [Aspect](#)

Creates new directories.

Parameters

inner_aspects ([Aspect](#))

```
async create_dir(event)
```

Create directory.

Parameters

event ([PrepareUpdating](#))

```
class parzzley.builtin.aspects.directory.RemoveForReplacement(*inner_aspects)
```

Bases: [Aspect](#)

Cleanup sites for replacements, often after resolved conflicts.

Parameters

inner_aspects ([Aspect](#))

```
_DATA__ROLLBACK_DIR = <parzzley.sync.aspect.events.Data object>
```

```
async prepare Updating(event)
```

Remove (with rollback data) the item if its item type is incompatible to the master site's one.

Parameters

event ([PrepareUpdating](#))

```
async remove_rollback_data_after_finished(event)
```

Remove rollback data after item sync is finished.

Parameters

event ([RefreshItemsBook](#))

```
class parzzley.builtin.aspects.directory.RollbackCrashedTransfers(*inner_aspects)
```

Bases: [Aspect](#)

Rollback crashed transfers.

Parameters

inner_aspects ([Aspect](#))

```
async prepare_sync_run(event)
```

Rollback crashed transfers.

Parameters

event ([Prepare](#))

parzzley.builtin.aspects.exclude_paths module

Exclude paths.

```
class parzzley.builtin.aspects.exclude_paths.ExcludePaths(* (Keyword-only parameters separator  
                                                             (PEP 3102)), pattern)
```

Bases: [Aspect](#)

Exclude paths by a regular expression pattern.

Parameters

pattern ([str](#))

async decide_to_skip(*event*)

Check whether to skip this item by applying the regexp pattern to its path.

Parameters

event ([DecideToSkip](#))

parzzley.builtin.aspects.item_type module

Item type detection.

class parzzley.builtin.aspects.item_type.**DetermineItemTypes**(**inner_aspects*)

Bases: [Aspect](#)

Determines the item type for this site by querying its [parzzley.fs.Site.item_type_by_cookie\(\)](#).

Parameters

inner_aspects ([Aspect](#))

async determine_item_type(*event*)

Determine the item type (by asking the site to interpret the main stream cookie).

Parameters

event ([DetermineType](#))

parzzley.builtin.aspects.main_stream module

Main stream support.

class parzzley.builtin.aspects.main_stream.**MainStreamSynchronization**(**inner_aspects*)

Bases: [Aspect](#)

Main stream specific behavior.

Parameters

inner_aspects ([Aspect](#))

_SUPPORTED_ITEM_TYPES = ([ItemType.FILE](#), [ItemType.SYMLINK](#))

async stream_support(*event*)

Declare stream support.

Parameters

event ([DetermineStreamSupport](#))

async detect_changes(*event*)

Detect changes. If a change was detected, and this site was not involved in the last successful sync run on this item, then mark one of these as changed as well (so we would never override other versions but raise a conflict instead later).

The check around the involved sites is sufficient to do on the main stream only, as the potential conflict would take the desired effect on the entire item.

Parameters

event ([DetectChanges](#))

async alien_detect_changes(*event*)

Always detect alien items as changed. They will either be mostly ignored later, or raise a type conflict.

Parameters

event ([DetectChanges](#))

parzzley.builtin.aspects.posix_attributes module

POSIX attributes support.

class parzzley.builtin.aspects.posix_attributes.**PosixAttributesSynchronization**(*inner_aspects)

Bases: [Aspect](#)

Synchronization of some POSIX file attributes.

Parameters

inner_aspects ([Aspect](#))

_SUPPORTED_ITEM_TYPES = ([ItemType.FILE](#), [ItemType.DIRECTORY](#), [ItemType.SYMLINK](#))

async stream_support(*event*)

Declare stream support.

Parameters

event ([DetermineStreamSupport](#))

async comparator(*event*)

Compare POSIX file attribute streams.

Parameters

event ([DetermineComparator](#))

async detect_changes(*event*)

Mark the main stream master as changed for the posix stream if it was marked as changed for the main stream.

Parameters

event ([DetectChanges](#))

async resolve_conflicts_on_directories(*event*)

Resolves conflicts on directories (that will come up regularly, but are not really relevant).

Parameters

event ([TryResolveConflicts](#))

async disable_change_detection(*event*)

Disable change detection for the posix stream.

Parameters

event ([CollectDestinationStreamables](#))

parzzley.builtin.aspects.pull_and_purge module

Pull&purge is a special way to configure a Parzzley volume. There are one or more sources and one sink. Files will be transferred from the sources to the sink and removed on the source site after that.

class parzzley.builtin.aspects.pull_and_purge.**PullAndPurgeSyncSink**

Bases: [Aspect](#)

Pull&purge sync sink. There must be exactly one in a pull&purge volume (and the other sites must be sources).

_MOVABLE_ITEM_TYPES = ([ItemType.FILE](#), [ItemType.SYMLINK](#))

async detect_changes(*event*)

Mark each directory as changed.

Parameters**event** ([DetectChanges](#))**async rename_already_existing**(*event*)

Rename an already existing entry to some new name.

Parameters**event** ([MasterSiteDetermined](#))**class** `parzzley.builtin.aspects.pull_and_purge.PullAndPurgeSyncSource`Bases: [Aspect](#)

Pull&purge sync source.

async remove(*event*)

Remove the file in the source site.

Parameters**event** ([ApplyUpdate](#))**parzzley.builtin.aspects.remove module**

Removal.

class `parzzley.builtin.aspects.remove.DirectRemove`Bases: [Aspect](#)Basic, non-trashing removal. For a trash bin, see [TrashRemove](#) instead.**_DATA__LOG_REMOVAL** = `<parzzley.sync.aspect.events.Data object>`**async remove_non_dir**(*event*)

Remove the (non-directory) item if this reflects the most recent state.

Parameters**event** ([ApplyUpdate](#))**async remove_dir**(*event*)

Remove directory if this reflects the most recent state.

Parameters**event** ([ApplyUpdate](#))**async log_removal**(*event*)

Log removals.

Parameters**event** ([RefreshItemsBook](#))**class** `parzzley.builtin.aspects.remove.TrashRemove`(**, trash_max_age='30d'*)Bases: [Aspect](#)Removal with a trash bin. For no trash bin, see [DirectRemove](#) instead.**Parameters****trash_max_age**(*str | float | timedelta*)**_DATA__LOG_REMOVAL** = `<parzzley.sync.aspect.events.Data object>`

async move_to_trash(event)

Move the (non-directory) item to the trash bin if this reflects the most recent state.

Parameters

event ([ApplyUpdate](#))

async log_removal(event)

Log removals.

Parameters

event ([RefreshItemsBook](#))

class parzzley.builtin.aspects.remove.DetectRemoval(*inner_aspects)

Bases: [Aspect](#)

Detect a change on the main stream when an item was removed on this site (and not modified on any other site).

Parameters

inner_aspects ([Aspect](#))

async detect_changes(event)

Detect removal.

Parameters

event ([DetectChanges](#))

class parzzley.builtin.aspects.remove.CleanupTrashBin(*, trash_max_age='30d')

Bases: [Aspect](#)

Removes aged stuff from the trash bin (only for the root directory).

Parameters

trash_max_age (*str | float | timedelta*)

async cleanup_trash_bin(event)

Clean up the trash bin.

Parameters

event ([Iterated](#))

parzzley.builtin.aspects.revision_tracking module

Revision tracking.

class parzzley.builtin.aspects.revision_tracking.RevisionTracking(number_unarchived_revisions=3, number_revisions_per_archive=20)

Bases: [Aspect](#)

Revision tracking.

Parameters

- **number_unarchived_revisions** (*int*) – Number of revisions per file to be stored directly, not inside an archive file.
- **number_revisions_per_archive** (*int*) – Number of revisions per file to be stored in one archive, before the next archive file begins.

_DATA__REVISIONS_SITE = <parzzley.sync.aspect.events.Data object>


```
_DATA__REVISION_ITEM = <parzzley.sync.aspect.events.Data object>
```

```
_SUPPORTED_ITEM_TYPES = (ItemType.FILE,)
```

```
async init(event)
```

Initialize.

Parameters

event ([Prepare](#))

```
async add_working_item_for_revision(event)
```

Set up the revision store as one additional destination.

Parameters

event ([SetUpWorkingItems](#))

```
async store_working_item(event)
```

Handle new files in the revision store.

Parameters

event ([RefreshItemsBook](#))

parzzley.builtin.aspects.streaming module

Streaming.

```
class parzzley.builtin.aspects.streaming.SkipItemIfNoUpToDateSitesAreConnected(*inner_aspects)
```

Bases: [Aspect](#)

Skip an item if no up-to-date sites are connected (in order to prevent data loss in some situations).

Parameters

inner_aspects ([Aspect](#))

```
_DATA__CHECK_DONE = <parzzley.sync.aspect.events.Data object>
```

```
async decide_to_skip(event)
```

Skip for now if no up-to-date sites are connected.

Parameters

event ([DecideToSkip](#))

```
class parzzley.builtin.aspects.streaming.GlobalStreamSupport(*inner_aspects)
```

Bases: [Aspect](#)

Derives global stream support information.

It checks if all sites have determined the same supported streams (with the same item types) in [parzzley.sync.aspect.events.sync_run.DetermineStreamSupport](#). It will then take this as the global result or will fail otherwise.

Parameters

inner_aspects ([Aspect](#))

```
_DATA__STREAM_SUPPORT_DICT = <parzzley.sync.aspect.events.Data object>
```

```
async validate_stream_support(event)
```

Validate the determined stream support.

Parameters

event ([ValidateStreamSupport](#))

class parzzley.builtin.aspects.streaming.SetItemsBookEntry(*inner_aspects)

Bases: [Aspect](#)

Primarily stores the “updated cookie”, so it can be used for decisions in the next sync run; e.g. in [parzzley.sync.aspect.events.item.stream.DetectChanges](#).

Parameters

inner_aspects ([Aspect](#))

async refresh_items_book(event)

Refresh items book.

Parameters

event ([RefreshItemsBook](#))

class parzzley.builtin.aspects.streaming.SourceStreamable(*inner_aspects)

Bases: [Aspect](#)

Sets the source streamable from the site.

Parameters

inner_aspects ([Aspect](#))

async source_streamable(event)

Collect the source streamable.

Parameters

event ([CollectSourceStreamables](#))

class parzzley.builtin.aspects.streaming.GetCookie(*inner_aspects)

Bases: [Aspect](#)

Queries the cookie from the site.

Parameters

inner_aspects ([Aspect](#))

async determine_cookie(event)

Determine the cookie.

Parameters

event ([DetermineCookie](#))

class parzzley.builtin.aspects.streaming.ComputeMasterSiteTable(*inner_aspects)

Bases: [Aspect](#)

Determines the master site for this stream (and the score table) based on the information gathered in [parzzley.sync.aspect.events.item.stream.DetectChanges](#).

Parameters

inner_aspects ([Aspect](#))

_DATA__MASTER_SITE_TUPLE = <parzzley.sync.aspect.events.Data object>

async master_site_table(event)

Determine the master site table.

Parameters

event ([DetermineMasterSiteTable](#))

```
class parzzley.builtin.aspects.streaming.UpdateTransfer(*inner_aspects)
```

Bases: [Aspect](#)

Runs a loop that reads from the source streamable and writes to the destination streamables. If the source has been changed during transfer, it marks this item to need full retry (and stores some data for it).

Parameters

inner_aspects ([Aspect](#))

_DATA__UPDATE_LOG_INFO = <parzzley.sync.aspect.events.Data object>

async transfer_update(event)

Transfer the update from the source stream to the destination streams.

Parameters

event ([TransferUpdate](#))

async log_update(event)

Log updates.

Parameters

event ([RefreshItemsBook](#))

```
class parzzley.builtin.aspects.streaming.StickToOldMasterSitesOnItemRetry(*inner_aspects)
```

Bases: [Aspect](#)

Stick to old master sites on item retry, so some residuals from our last attempt do not get preferred now.

This is the right things to do in general, but leads to a dangerous timing effect: If the user applies a change on a non-master site while the retry is happening, this change will get lost.

Parameters

inner_aspects ([Aspect](#))

_DATA__RETRY = <parzzley.sync.aspect.events.Data object>

async prepare_item(event)

Prepare item sync.

Parameters

event ([Prepare](#))

```
class parzzley.builtin.aspects.streaming.WorkingItem(*inner_aspects)
```

Bases: [Aspect](#)

Manage working items and use them as destination streamables.

Parameters

inner_aspects ([Aspect](#))

_DATA__TEMP_ITEM = <parzzley.sync.aspect.events.Data object>

async temp_working_item(event)

Create temporary working item if on this site the main stream needs an update from the master site.

Parameters

event ([SetUpWorkingItems](#))

async collect_destination(event)

Collect the destination streamable.

Parameters

event ([CollectDestinationStreamables](#))

async commit_update(event)

Commit the update.

Parameters

event ([ApplyUpdate](#))

async revert_content_equal_to_master_flag(event)

Revert 'content equal to master' flag during conflict resolution (since that might change the master!).

Parameters

event ([ApplyConflictResolution](#))

__master_site_is_more_recent(event)

Return type

bool

parzzley.builtin.aspects.sync_report module

Sync reports.

class parzzley.builtin.aspects.sync_report.**SyncReport**(*inner_aspects)

Bases: [Aspect](#)

Writes a report after each sync run for later problem diagnostics or performance analysis.

Parameters

inner_aspects ([Aspect](#))

_DATA__REPORT_DONE = <parzzley.sync.aspect.events.Data object>

async write_report(event)

Write the report.

Parameters

event ([Close](#))

parzzley.builtin.aspects.xattrs module

Extended attributes support.

class parzzley.builtin.aspects.xattrs.**XattrSynchronization**(*inner_aspects)

Bases: [Aspect](#)

Extended attributes support.

Parameters

inner_aspects ([Aspect](#))

_DATA__NEEDS_TAGGING = <parzzley.sync.aspect.events.Data object>

_SUPPORTED_ITEM_TYPES = (ItemType.DIRECTORY, ItemType.FILE)

_XATTR_TAG_KEY = b'user.__parzzley_f'

async stream_support(event)

Declare stream support.

Parameters

event ([DetermineStreamSupport](#))

async detect_changes(*event*)

Detect changes.

Parameters

event ([DetectChanges](#))

async collect_source(*event*)

Collect the source streamable.

Parameters

event ([CollectSourceStreamables](#))

async collect_destination(*event*)

Add a destination streamable for the master site as well if the item was marked this way.

Parameters

event ([CollectDestinationStreamables](#))

async disable_change_detection(*event*)

Disable change detection for the xattrs stream.

Parameters

event ([CollectDestinationStreamables](#))

class _PatchDictPipe(*args, **kwargs)

Bases: [FullContentPipe](#)

async pipe_content(*data*)

Receive and process the full content of a stream and return the processed content.

Parameters

data – The stream’s entire content.

_abc_impl = <_abc._abc_data object>

parzzley.builtin.fs package

Parzzley builtin filesystem implementations. See also [parzzley.fs](#).

class parzzley.builtin.fs.ShellBasedBackend

Bases: [Backend](#), ABC

Base class for Unix shell based filesystem backends.

async connect(*, *path*, **kwargs)

Connect and return a site backend for given arguments (or None if this is impossible for some reasons).

This method does not raise any exceptions.

Parameters

- **kwargs** – Additional arguments for the site backend (specific to its type).
- **path** (*str*)

async disconnect(*site_backend*)

Disconnect a given site backend (returned earlier by [connect\(\)](#)).

Must only be called once for each site backend.

This method does not raise any exceptions.

Parameters

site_backend ([_SiteBackend](#)) – The site backend to disconnect. This is what the last [connect\(\)](#) call returned.

abstractmethod [_shell](#)(**kwargs)

Return a shell for the given arguments.

Parameters

kwargs – The backend arguments.

Return type

[Shell](#)

class [_SiteBackend](#)(shell_factory, shell_kwargs, root_path)

Bases: [SiteBackend](#)

Parameters

- **shell_factory** ([Callable](#))
- **shell_kwargs** ([dict](#))
- **root_path** ([str](#))

class [_ShellPool](#)(shell_factory, shell_kwargs)

Bases: [object](#)

Parameters

- **shell_factory** ([Callable](#))
- **shell_kwargs** ([dict](#))

WORKER_COUNT = 3

disconnect()

Disconnect.

shell()

Return a context manager that reserves and returns a free shell for arbitrary usage.

Return type

[Generator](#)[[Shell](#), None, None]

_ShellPool__new_shell()

Return type

[Shell](#)

item_type_by_cookie(cookie)

Return the item type by the main stream cookie.

Parameters

cookie – The main stream cookie.

async move_item(item, to_item)

Move an item.

If the source is a normal file and the destination already exists and is also a normal file, the destination will be overwritten.

Parameters

- **item** – The source item location.
- **to_item** – The destination location.

async create_item(*item*, *item_type*)

Create a new item.

See `Site.create_item()` about what exception it must raise in which situations.

Parameters

- **item** – The item location.
- **item_type** – The new item type.

async cookie(*item*, *stream_name*)

Return the item cookie for an item. See also `Site.cookie()`.

Parameters

- **item** – The item location.
- **stream_name** – The stream name.

async remove_item(*item*, *recursive*)

Remove an item.

Parameters

- **item** – The item location.
- **recursive** – For a directory, whether to remove all content inside it as well if not empty.

async child_names(*item*)

Return the names of all children of an item.

Parameters

- item** – The item location.

async read_streamable(*item*, *stream_name*="")

Return a read-streamable for an item.

Parameters

- **item** – The item location.
- **stream_name** – The stream name.

async write_streamable(*item*, *stream_name*="")

Return a write-streamable for an item.

Parameters

- **item** – The item location.
- **stream_name** – The stream name.

async wait_for_changes(*on_changed*)

Until canceled, watch this site for any changes.

There will be on filtering, so it will also report changes e.g. on the control site directory.

Parameters

- on_changed** – The function to call whenever changes were observed.

async disconnect()

Disconnect.

class _FileMainStreamWriteStreamable(*shell_pool*, *path*)

Bases: [WriteStreamable](#)

Parameters

- path** (*bytes*)

class _WriteStream(*shell_pool*, *path*)

Bases: [WriteStream](#)

async write(*data*)

Append a chunk of data to the stream.

Parameters

data – The data.

async commit()

Commit the written data.

_abc_impl = <_abc._abc_data object>

async _stream()

Return a new stream.

This method is implemented by subclasses and only used internally. See `stream()`.

_abc_impl = <_abc._abc_data object>

class _FileMainStreamReadStreamable(*shell_pool, path*)

Bases: [ReadStreamable](#)

Parameters

path (*bytes*)

class _ReadStream(*shell_pool, path*)

Bases: [ReadStream](#)

async read(*max_len*)

Read the next chunk of data from the stream and return it, or return `None` if the end of the stream has been reached.

Parameters

max_len – The maximum chunk length.

_abc_impl = <_abc._abc_data object>

async _stream()

Return a new stream.

This method is implemented by subclasses and only used internally. See `stream()`.

_abc_impl = <_abc._abc_data object>

_abc_impl = <_abc._abc_data object>

class Shell

Bases: `ABC`

A shell allows to execute arbitrary command lines. See e.g. [exec\(\)](#).

Before usage, it needs to be entered (`with-block`). It must only be entered once!

class ExecutionResult(*exit_code, out*)

Bases: `object`

The execution result of a shell command line.

Parameters

- **exit_code** (*int*)

- **out** (*bytes*)

exit_code: int

The exit code.

out: bytes

The output of this command (does not include 'err').

abstractmethod _shell_cmdline()

Return the command line for starting this shell.

This is implemented by subclasses and only used internally.

Return type

Sequence[str]

async exec_raw(cmd_line)

Execute a command line and return the process object for further communication.

This shell must be currently entered in order to execute commands.

Note: This is a low-level method for particular use cases. Usually `exec()` should be used instead.

If problems occur, e.g. because it just lost the connection to the remote side, it will raise `parzzley.fs.Site.ConnectionLostError`. See also `verify_alive()` for later checks whether the shell is still alive.

Parameters

cmd_line (bytes) – The command line to execute.

Return type

Popen

async exec(cmd_line, *, fail_on_error=True, input=b'')

Execute a command line and return the execution result.

This shell must be currently entered in order to execute commands.

If problems occur in the low-level implementation, e.g. because it just lost the connection to the remote side, it will raise `parzzley.fs.Site.ConnectionLostError`.

Parameters

- **cmd_line** (str | bytes | Iterable[str | bytes]) – The command line to execute.
- **fail_on_error** (bool) – Whether to fail (with an `IOError`) if the exit-code is nonzero.
- **input** (bytes) – The input to stream to the command.

Return type

ExecutionResult

verify_alive()

Check whether this shell is still alive (e.g. has not lost connection to its remote side) and raise a `parzzley.fs.Site.ConnectionLostError` if not.

You only need this method for `exec_raw()`.

__write_stdin(b)

_abc_impl = <_abc._abc_data object>

_abc_impl = <_abc._abc_data object>

`parzzley.builtin.fs._find_unsafe(string, pos=0, endpos=9223372036854775807)`

Scan through string looking for a match, and return a corresponding match object instance.

Return None if no position in the string matches.

`parzzley.builtin.fs._shell_quote(s)`

Return a shell-escaped version of the given byte string.

Parameters

s (bytes) – The byte string.

Return type
bytes

Submodules

parzzley.builtin.fs.local module

See [Backend](#).

class parzzley.builtin.fs.local.**Backend**

Bases: [Backend](#)

Local filesystem backend implementation.

async connect(*, path)

Connect and return a site backend for given arguments (or `None` if this is impossible for some reasons).

This method does not raise any exceptions.

Parameters

- **kwargs** – Additional arguments for the site backend (specific to its type).
- **path** (*Path* | *str*)

async disconnect(site_backend)

Disconnect a given site backend (returned earlier by [connect\(\)](#)).

Must only be called once for each site backend.

This method does not raise any exceptions.

Parameters

site_backend – The site backend to disconnect. This is what the last [connect\(\)](#) call returned.

class _SiteBackend(path)

Bases: [SiteBackend](#)

item_type_by_cookie(cookie)

Return the item type by the main stream cookie.

Parameters

cookie – The main stream cookie.

async move_item(item, to_item)

Move an item.

If the source is a normal file and the destination already exists and is also a normal file, the destination will be overwritten.

Parameters

- **item** – The source item location.
- **to_item** – The destination location.

async create_item(item, item_type)

Create a new item.

See `Site.create_item()` about what exception it must raise in which situations.

Parameters

- **item** – The item location.
- **item_type** – The new item type.

async cookie(*item*, *stream_name*)

Return the item cookie for an item. See also `Site.cookie()`.

Parameters

- **item** – The item location.
- **stream_name** – The stream name.

async remove_item(*item*, *recursive*)

Remove an item.

Parameters

- **item** – The item location.
- **recursive** – For a directory, whether to remove all content inside it as well if not empty.

async child_names(*item*)

Return the names of all children of an item.

Parameters

- item** – The item location.

async read_streamable(*item*, *stream_name*="")

Return a read-streamable for an item.

Parameters

- **item** – The item location.
- **stream_name** – The stream name.

async write_streamable(*item*, *stream_name*="")

Return a write-streamable for an item.

Parameters

- **item** – The item location.
- **stream_name** – The stream name.

async wait_for_changes(*on_changed*)

Until canceled, watch this site for any changes.

There will be on filtering, so it will also report changes e.g. on the control site directory.

Parameters

- on_changed** – The function to call whenever changes were observed.

class _FileMainStreamReadStreamable(*full_path*)

Bases: [ReadStreamable](#)

class _ReadStream(*full_path*)

Bases: [ReadStream](#)

async read(*max_len*)

Read the next chunk of data from the stream and return it, or return None if the end of the stream has been reached.

Parameters

- max_len** – The maximum chunk length.

_abc_impl = **<_abc._abc_data object>**

async _stream()

Return a new stream.

This method is implemented by subclasses and only used internally. See `stream()`.

_abc_impl = **<_abc._abc_data object>**

```
class _FileMainStreamWriteStreamable(full_path)
    Bases: WriteStreamable

    class _WriteStream(full_path)
        Bases: WriteStream

        async write(data)
            Append a chunk of data to the stream.
            Parameters
            data – The data.

        async commit()
            Commit the written data.

        _abc_impl = <_abc._abc_data object>

    async _stream()
        Return a new stream.

        This method is implemented by subclasses and only used internally. See stream().

        _abc_impl = <_abc._abc_data object>

    _abc_impl = <_abc._abc_data object>

_abc_impl = <_abc._abc_data object>
```

parzzley.builtin.fs.ssh module

See *Backend*.

```
class parzzley.builtin.fs.ssh.Backend
    Bases: ShellBasedBackend

    ssh filesystem backend implementation.

    _shell(**kwargs)
        Return a shell for the given arguments.

        Parameters
        kwargs – The backend arguments.

    class _Shell(*, host, port=22, user, id_file, skip_host_key_check=False)
        Bases: Shell

        Parameters
        • host (str)
        • port (int)
        • user (str)
        • id_file (Path)
        • skip_host_key_check (bool)

    _shell_cmdline()
        Return the command line for starting this shell.

        This is implemented by subclasses and only used internally.
```

```

    _abc_impl = <_abc._abc_data object>
    _abc_impl = <_abc._abc_data object>

```

parzzley.builtin.log_formatters package

Parzzley builtin log formatter implementations. See also *parzzley.sync.logger*.

Submodules

parzzley.builtin.log_formatters.html module

See *Formatter*.

class parzzley.builtin.log_formatters.html.**Formatter**

Bases: *Formatter*

HTML log formatter.

format(*sync_run*, *entries*)

Format the given entries to a textual representation.

Parameters

- **sync_run** – The sync run (or None if this report is not associated to a particular sync run).
- **entries** – The entries.

__header(*sync_run*)

Parameters

sync_run (*SyncRun* | *None*)

Return type

str

__entries(*entries*)

Parameters

entries (*Iterable*[*Entry*])

Return type

str

__title(*sync_run*)

Parameters

sync_run (*SyncRun*)

Return type

str

__style()

Return type

str

```

    _abc_impl = <_abc._abc_data object>

```

parzzley.builtin.log_formatters.json module

See *Formatter*.

class parzzley.builtin.log_formatters.json.**Formatter**

Bases: *Formatter*

JSON log formatter.

format(*sync_run*, *entries*)

Format the given entries to a textual representation.

Parameters

- **sync_run** – The sync run (or None if this report is not associated to a particular sync run).
- **entries** – The entries.

_abc_impl = <_abc._abc_data object>

parzzley.builtin.log_outs package

Parzzley builtin logger output channel implementations. See also *parzzley.sync.logger*.

Submodules

parzzley.builtin.log_outs.shell module

See *Out*.

class parzzley.builtin.log_outs.shell.**Out**(*, *cmdline*)

Bases: *Out*

Shell logger output channel.

Parameters

cmdline (*str*)

emit_text(*sync_run*, *s*)

Emit the text via this output channel.

Parameters

- **sync_run** – The sync run (or None if this report is not associated to a particular sync run).
- **s** – The text to emit.

_abc_impl = <_abc._abc_data object>

parzzley.config package

Parzzley configuration.

class parzzley.config.**Aspect**(*, *type_name*, *arguments=None*)

Bases: object

Configuration for one aspect instance.

At runtime, this will be transferred to an instance of an implementation of *parzzley.sync.aspect.Aspect*.

Parameters

- **type_name** (*str*) – The aspect type name.

- **arguments** (*dict[str, str] | None*) – The aspect arguments.

property type_name: **str**

The aspect type name.

property arguments: **dict[str, str]**

The aspect arguments.

class `parzzley.config.Site`(*name*, *, *kind*, *arguments=None*, *aspects=()*, *warn_after=None*)

Bases: `object`

Configuration for one site instance.

At runtime, this will eventually (with some indirections) be transferred to an instance of `parzzley.fs.Site`.

Parameters

- **name** (*str*) – The site name.
- **kind** (*str*) – The site kind.
- **arguments** (*dict[str, str] | None*) – The site arguments.
- **aspects** (*Iterable[Aspect]*) – The aspects configured specifically for this site.
- **warn_after** (*timedelta | None*) – The warn-after duration. See `warn_after`.

property name: **str**

The site name.

property kind: **str**

The site kind.

See `parzzley.fs.register_backend()`.

property warn_after: **timedelta | None**

The warn-after duration.

When Parzzley was not able to successfully synchronize this site for that duration, it will emit a warning message.

property arguments: **dict[str, str]**

The site arguments.

property aspects: **Sequence[Aspect]**

The aspects configured specifically for this site.

class `parzzley.config.Volume`(*name*, *, *sites*, *aspects*, *interval*)

Bases: `object`

Configuration for one sync volume.

At runtime, this will be transferred to an instance of `parzzley.sync.Volume`.

Parameters

- **name** (*str*) – The volume name.
- **sites** (*Iterable[Site]*) – The sites configured for this volume.
- **aspects** (*Iterable[Aspect]*) – The aspects configured for all sites.
- **interval** (*timedelta*) – The configured sync interval.

property name: `str`

The volume name.

property sites: `Sequence[Site]`

The sites configured for this volume.

property aspects: `Sequence[Aspect]`

The aspects configured for all sites.

property interval: `timedelta`

The configured sync interval.

class `parzzley.config.Logging(*, min_severity, max_severity, out, formatter, exclude)`

Bases: `object`

Configuration for a logger.

At runtime, this will be transferred to an instance of an implementation of `parzzley.sync.logger.Logging`.

Parameters

- **min_severity** (`str` / `None`) – The minimal severity.
- **max_severity** (`str` / `None`) – The maximal severity.
- **out** (`Iterable[Logging.Out]`) – The logger output channels.
- **formatter** (`Logging.Formatter`) – The log formatter.
- **exclude** (`Iterable[Logging.Exclude]`) – The log exclusions.

property min_severity: `str` | `None`

The minimal severity.

property max_severity: `str` | `None`

The maximal severity.

property out: `Sequence[Out]`

The logger output channels.

property formatter: `Formatter`

The log formatter.

property exclude: `Sequence[Exclude]`

The logger exclusions.

class `Out(*, kind, arguments=None)`

Bases: `object`

Configuration for a logger output channel.

At runtime, this will be transferred to an instance of an implementation of `parzzley.sync.logger.Out`.

Parameters

- **kind** (`str`) – The kind of logger output channel.
- **arguments** (`dict[str, str]` / `None`) – The logger output channel argument.

property kind: `str`

The kind of logger output channel.

property arguments: `dict[str, str]`

The logger output channel argument.

class `Formatter`(**, kind, arguments=None*)

Bases: `object`

Configuration for a log formatter.

At runtime, this will be transferred to an instance of an implementation of `parzzley.sync.logger.Formatter`.

Parameters

- **kind** (*str*) – The kind of log formatter.
- **arguments** (*dict[str, str] | None*) – The log formatter argument.

property kind: `str`

The kind of log formatter.

property arguments: `dict[str, str]`

The log formatter argument.

class `Exclude`(**, conditions*)

Bases: `object`

Configuration for a logger exclusion.

At runtime, this will be transferred to an instance of an implementation of `parzzley.sync.logger.Exclude`.

Parameters

conditions (*Iterable[Logging.Exclude.BaseCondition]*) – Exclusion conditions.

property conditions: `Sequence[BaseCondition]`

Exclusion conditions. A message will be excluded if any of these conditions are true.

class `BaseCondition`

Bases: `object`

Configuration for a logger exclusion condition. Base class of some subclasses.

At runtime, this will be transferred to an instance of an implementation of `parzzley.sync.logger.Exclude.Condition`.

class `Condition`(**, arguments=None*)

Bases: `BaseCondition`

Configuration for a simple logger exclusion condition.

Parameters

arguments (*dict[str, str] | None*) – The condition arguments.

property arguments: `dict[str, str]`

The condition arguments.

class `CombinedCondition`(**, combination, conditions*)

Bases: `BaseCondition`

Configuration for a combined logger exclusion condition.

Parameters

- **combination** (*str*) – The combination. Either 'AND' or 'OR'.
- **conditions** (*Iterable[Logging.Exclude.BaseCondition]*) – The conditions to combine.

property combination: `str`

The combination. Either 'AND' or 'OR'.

property conditions: `Sequence[BaseCondition]`

The conditions to combine.

class NegateCondition(**, condition*)

Bases: `BaseCondition`

Configuration for a negating logger exclusion condition.

Parameters

condition (`Logging.Exclude.BaseCondition`) – The condition to negate.

property condition: `BaseCondition`

The condition to negate.

class parzzley.config.Configuration(**, volumes, loggings*)

Bases: `object`

Represents one complete Parzzley configuration.

Parameters

- **volumes** (`Iterable[Volume]`) – The sync volumes.

- **loggings** (`Iterable[Logging]`) – The loggings.

property volumes: `Sequence[Volume]`

The sync volumes.

property loggings: `Sequence[Logging]`

The loggings.

Subpackages

`parzzley.config.file_formats` package

File formats for Parzzley configuration files.

See also the submodules.

class parzzley.config.file_formats.FileFormat

Bases: `ABC`

A file format for Parzzley configuration files.

parse_file(*config_file*)

Read and return the configuration from the given configuration file.

Parameters

config_file (`Path`) – The configuration file to parse.

Return type

Any

`_abc_impl = <_abc._abc_data object>`

`parzzley.config.file_formats.register_file_format`(*format_name*)

Return a decorator that registers a file format for Parzzley configuration files.

Parameters

format_name (`str`) – The format name.

`parzzley.config.file_formats.parse(path)`

Parse a Parzzley configuration file.

Parameters

path (*Path* | *str*) – The configuration file.

Return type

Any

`parzzley.config.file_formats.read_configuration(parzzley_config_dir)`

Parse an entire Parzzley configuration directory.

Parameters

parzzley_config_dir (*Path* | *str*) – The configuration directory.

Return type

[Configuration](#)

`parzzley.config.file_formats.timedelta(s)`

Translate a timedelta specification (usually a string when it comes from configuration files) to a timedelta.

Parameters

s (*str* | *float* | *timedelta*) – The timedelta specification.

Return type

timedelta

`parzzley.config.file_formats.log_severity(s)`

Translate a log severity specification (usually a string when it comes from configuration files) to a log severity.

Parameters

s (*str* | [Severity](#)) – The log severity specification.

Return type

[Severity](#)

`parzzley.config.file_formats._file_format(format_name)`

Return the file format by name.

Parameters

format_name (*str*) – The format name. See [register_file_format\(\)](#).

Return type

[FileFormat](#) | *None*

Submodules

`parzzley.config.file_formats.xml` module

Support for Parzzley configuration XML files.

class `parzzley.config.file_formats.xml.XmlFileFormat`

Bases: [FileFormat](#)

XML file format.

parse_file(*config_file*)

Read and return the configuration from the given configuration file.

Parameters

config_file – The configuration file to parse.

__sync_volume(*xml_elem*, *config_file*)

Parameters

xml_elem (*Element*)

Return type

[Volume](#)

__sync_site(*xml_elem*, *config_file*)

Parameters

xml_elem (*Element*)

Return type

[Site](#)

__sync_aspects(*xml_elem*)

Parameters

xml_elem (*Element*)

Return type

[Aspect](#)

__logging(*xml_elem*, *config_file*)

Parameters

xml_elem (*Element*)

Return type

[Logging](#)

__log_formatter(*xml_elem*)

Parameters

xml_elem (*Element*)

Return type

[Formatter](#)

__logger_out(*xml_elem*)

Parameters

xml_elem (*Element*)

Return type

[Out](#)

__logger_exclusion(*xml_elem*, *config_file*)

Parameters

xml_elem (*Element*)

Return type

[Exclude](#)

__logger_exclusion_condition(*xml_elem*, *config_file*)

Parameters

xml_elem (*Element*)

Return type

[BaseCondition](#)

```
_abc_impl = <_abc._abc_data object>
```

Submodules

parzzley.config.loader module

Configuration loader.

```
parzzley.config.loader.load_volume(volume_config)
```

Return a volume for a volume configuration.

Parameters

volume_config ([Volume](#)) – The volume configuration.

Return type

[Volume](#)

```
parzzley.config.loader.load_site_setup(site_config)
```

Return a site setup for a site configuration.

Parameters

site_config ([Site](#)) – The site configuration.

Return type

[SiteSetup](#)

```
parzzley.config.loader.load_aspect(aspect_config)
```

Return an aspect for an aspect configuration.

Parameters

aspect_config ([Aspect](#)) – The aspect configuration.

Return type

[Aspect](#)

```
parzzley.config.loader.load_logging(logger_config)
```

Return a logging for a logging configuration.

Parameters

logger_config ([Logging](#)) – The logging configuration.

Return type

[Logging](#)

```
parzzley.config.loader.load_log_formatter(log_formatter_config)
```

Return a log formatter for a log formatter configuration.

Parameters

log_formatter_config ([Formatter](#)) – The log formatter configuration.

Return type

[Formatter](#)

```
parzzley.config.loader.load_logger_out(logger_out_config)
```

Return a logger output channel for a logger output channel configuration.

Parameters

logger_out_config ([Out](#)) – The logger output channel configuration.

Return type

[Out](#)

`parzzley.config.loader.load_log_exclusion(log_exclusion_config)`

Return a logger exclusion for a logger exclusion configuration.

Parameters

`log_exclusion_config` ([Exclude](#)) – The logger exclusion configuration.

Return type

[Exclude](#)

`parzzley.config.loader.load_log_exclusion_condition(log_exclusion_condition_config)`

Return a logger exclusion condition for a logger exclusion condition configuration.

Parameters

`log_exclusion_condition_config` ([BaseCondition](#)) – The logger exclusion condition configuration.

Return type

[Condition](#)

parzzley.fs package

Parzzley filesystem abstraction.

All Parzzley sync operations operate on this API instead of direct OS filesystem APIs. So, Parzzley can sync not only parts of the local filesystem, but anything that Parzzley has a backend for (in particular this means full support for ssh-based locations, incl. some features that sshfs would not offer).

- Actual filesystem operations can be done with [Site](#) instances. Where needed, Parzzley API usually provides you with such an instance. So the following steps just provide background information.
- In order to get a [Site](#), you usually take a [SiteSetup](#) and [SiteSetup.connect\(\)](#) it.
- In order to get a [SiteSetup](#), you need to specify a [Backend](#) and some additional, backend-specific arguments.
- A [Backend](#) implements a particular kind of filesystem. Directly though, it just implements an interface that establishes the connection to that filesystem and returns a [Backend.SiteBackend](#) which is the actual implementation of filesystem operations.

Note: [Site](#) and [Backend.SiteBackend](#) look very similar. They are not the same, though. The latter one is the implementation of a particular kind of filesystem and not to be used directly, but mostly used by the former one. [Site](#) is the API to be used for filesystem operations outside of this module. It adds various additional checks and convenience features to what the site backend itself would provide.

For backend implementations, see [parzzley.builtin.fs](#).

class `parzzley.fs.ItemType(*values)`

Bases: Enum

Filesystem item type.

FILE = 1

A regular file.

DIRECTORY = 2

A directory.

SYMLINK = 3

A symlink.

ALIEN = 4

An alien file, i.e. anything that does not match any of the former item types. Could be pipes, devices files, ... These items will be mostly ignored by Parzzley, because they are unsupported. Parzzley will (with usual configuration) at least report conflicts if an item has a non-alien type on some sites, but are alien on others.

NONE = 5

Does not exist. Note: For items that exist but are not one of the known types, see [ALIEN](#).

class parzzley.fs.Item

Bases: ABC

Represents a particular location (usually in some volume). This is like a path, but not it your local filesystem.

abstract property path: bytes

This item's path (segments split by '/'; never at the start or end of a path). A volume's root directory has item path ''.

Paths are byte strings. They are usually UTF-8 encoded strings. It might be encoded in an invalid way, though, so it must only be decoded to a string for presentation purposes and this decoding must not fail for invalid input. Paths never contain the NULL character, though.

Site backends that internally deal with file names encoded in another encoding than UTF-8 (or have a different set of allowed characters) have to translate between these representations; also in a way that does not fail for invalid input.

abstract property parent: [Item](#) | None

The parent item (or None as the root item's parent).

abstract property name: bytes | None

This item's name, i.e. the last segment of [path](#) (or None for the root item).

abstractmethod _descendant(*relative_paths)

Parameters

relative_paths (bytes)

Return type

[Item](#)

_abc_impl = <_abc._abc_data object>

class parzzley.fs._Item(path, descendant_of)

Bases: [Item](#)

Parameters

- **path** (bytes / [Item](#))
- **descendant_of** ([Item](#) / None)

property path

This item's path (segments split by '/'; never at the start or end of a path). A volume's root directory has item path ''.

Paths are byte strings. They are usually UTF-8 encoded strings. It might be encoded in an invalid way, though, so it must only be decoded to a string for presentation purposes and this decoding must not fail for invalid input. Paths never contain the NULL character, though.

Site backends that internally deal with file names encoded in another encoding than UTF-8 (or have a different set of allowed characters) have to translate between these representations; also in a way that does not fail for invalid input.

property parent

The parent item (or None as the root item's parent).

property name

This item's name, i.e. the last segment of *path* (or None for the root item).

_descendant(**relative_paths*)

_abc_impl = <**_abc._abc_data** object>

parzzley.fs.item(**paths_or_items*)

Return an item for a given item-like object.

Parameters

paths_or_items (*bytes* | *Item*) – Paths or item objects.

Return type

Item

class parzzley.fs.Site(*name*, *site_backend*)

Bases: object

A filesystem site.

All kinds of filesystem operations are provided by means of its methods. Any pieces of Parzzley that execute some filesystem operations (i.e. virtually all aspects - see *parzzley.sync.aspect* - and much more) work with these.

The actual effect might be somewhere in the local OS filesystem, but could also be somewhere else, e.g. somewhere in some remote machine's filesystem.

See also *Site.Exception* about error handling.

Do not use directly. See *parzzley.fs*.

Parameters

- **name** (*str* | *None*) – The site name. See also *name*!
- **site_backend** (*Backend.SiteBackend*) – The site backend.

property name: *str* | *None*

The site name.

Each 'real' site, i.e. each one that is directly associated to a site configuration from the end user, has a non-empty site name that is unique in the scope of the volume configuration it is owned by.

In some places, there can be artificial sites, e.g. for the purpose of internal state data storage. You should not rely on their names in any way, as they typically do not have one at all.

property root_directory: *Item*

The site root directory. This is always an item with path "".

async create_item(*item_*, *item_type*, *, *recursive=False*)

Create a new item.

If there already is an item at the same location, raise *Site.ItemExistsError*.

Parameters

- **item** – The item location.
- **item_type** (*ItemType*) – The new item type.

- **recursive** (*bool*) – For a directory, whether to create super-directories on demand as well.
- **item_** (*bytes* / *Item*)

Return type

None

async remove_item(*item_*, *, *recursive=False*)

Remove an item.

Parameters

- **item** – The item location.
- **recursive** (*bool*) – For a directory, whether to remove all content inside it as well if not empty.
- **item_** (*bytes* / *Item*)

Return type

None

item_type_by_cookie(*cookie*)Return the item type by the main stream cookie. If you do not already have it, use [item_type\(\)](#) instead.**Parameters****cookie** (*object*) – The main stream cookie.**Return type**[ItemType](#)**async cookie**(*item_*, *stream_name=""*)

Return the cookie for a stream of an item.

Stream cookies are mostly an opaque data structure. They fulfill two purposes:

- The main stream's cookie must somehow carry the item type. Each site backend knows how to extract it from such a cookie. This will happen when [item_type_by_cookie\(\)](#) is called.
- But primarily, comparing stream cookies from the same item and site and from different times allows to detect changes (without interpreting their actual content in any way, just by simple equality-comparisons).

Each stream cookie must be one of Python's simple types (`str`, `float`, `int`, `bool` and `None`), or a tuple or list (lists will be translated to tuples automatically, though).**Parameters**

- **item** – The item location.
- **stream_name** (*str*) – The stream name (default: the main stream).
- **item_** (*bytes* / *Item*)

Return type

object

async item_type(*item_*)Return the item type for an item. If you only want to check whether an item exists, see [item_exists\(\)](#) instead.**Parameters**

- **item** – The item location.

- **item_** (*bytes* / *Item*)

Return type
ItemType

async item_exists(*item_*)

Return whether an item exists.

Parameters

- **item** – The item location.
- **item_** (*bytes* / *Item*)

Return type
bool

async move_item(*item_*, *to_item*, *, *to_site=None*)

Move an item.

If the source is a normal file and the destination already exists and is also a normal file, the destination will be overwritten.

Parameters

- **item** – The source item location.
- **to_item** (*bytes* / *Item*) – The destination location.
- **to_site** (*Site* / *None*) – The destination site (default: the same site). Note: Do not use. It is only allowed under particular circumstances and only used internally by Parzzley for specific functionality.
- **item_** (*bytes* / *Item*)

Return type
None

async child_names(*item_*)

Return the names of all children of an item (sorted). See also [children\(\)](#).

Note: This will never contain the `..parzzley.control` directory on root level (which is used internally by Parzzley for persistence of some sync states).

Parameters

- **item** – The item location.
- **item_** (*bytes* / *Item*)

Return type
Sequence[str]

async children(*item_*)

Return all child items of an item (sorted by name). See also [child_names\(\)](#).

Note: This will never contain the `..parzzley.control` directory on root level (which is used internally by Parzzley for persistence of some sync states).

Parameters

- **item** – The item location.
- **item_** (*bytes* / *Item*)

Return type*Sequence*[*Item*]**async read_streamable**(*item_*, *stream_name*="")

Return a read-streamable for an item.

Parameters

- **item** – The item location.
- **stream_name** (*str*) – The stream name (default: the main stream).
- **item_** (*bytes* / *Item*)

Return type*ReadStreamable***async write_streamable**(*item_*, *stream_name*="")

Return a write-streamable for an item.

Parameters

- **item** – The item location.
- **stream_name** (*str*) – The stream name (default: the main stream).
- **item_** (*bytes* / *Item*)

Return type*WriteStreamable***async read_bytes**(*item_*, *stream_name*="")

Read binary content from an item.

Note: If the content can exceed a few megabytes of size, you should better use *read_streamable()*.**Parameters**

- **item** – The item location.
- **stream_name** (*str*) – The stream name (default: the main stream).
- **item_** (*bytes* / *Item*)

Return type*bytes***async write_bytes**(*item_*, *content*, *stream_name*="")

Write binary content to an item.

Note: If the content can exceed a few megabytes of size, you should better use *write_streamable()*.**Parameters**

- **item** – The item location.
- **content** (*bytes*) – The content to write.
- **stream_name** (*str*) – The stream name (default: the main stream).
- **item_** (*bytes* / *Item*)

Return type*None*

async wait_for_changes(*on_changed*)

Until canceled, watch this site for any changes.

Access to the control site directory will be filtered and not reported. Also watch the control site explicitly if needed.

Parameters

on_changed (*Callable*[[*Item*], *None*]) – The function to call whenever changes were observed.

Return type

None

async create_temp_item(*item_type*)

Create and return a temporary item in some hidden temporary place that gets cleaned up automatically.

Parameters

item_type (*ItemType*) – The item type.

Return type

Item

sub_site(*root_dir*)

Return sub site, i.e. a site that represents a subtree of this site.

Note: Do not use. It is only used internally by Parzzley for specific functionality.

Parameters

root_dir (*bytes* / *Item*) – The root directory of the new site.

Return type

Site

sub_site_location(*sub_site*)

Return the location of the root directory of a given subsite in this site.

Note: Do not use. It is only used internally by Parzzley for specific functionality.

Parameters

sub_site (*Site*) – The sub site.

Return type

Item

async control_site()

Return the control site.

Note: Do not use. It is only used internally by Parzzley for specific functionality.

Return type

Site

async temp_site()

Return a sub-site that is able to store arbitrary temporary items and gets cleaned up automatically.

Note: Do not use. It is only used internally by Parzzley for specific functionality.

Return type

Site

__translate_arbitrary_exception_to_site_exception()

static `sanitized_cookie(cookie)`

Return the sanitized variant of a given cookie (e.g. convert all lists to tuples).

Parameters

cookie (*Any*) – The cookie to sanitize.

Return type

Any

exception `Exception`

Bases: `Exception`

Base class for arbitrary errors during a site operation.

Note: All operations on a [Site](#) will only throw `ValueError` (if an input value is invalid) or [Site.Exception](#) (or one of its subclasses).

Operations in a [Backend.SiteBackend](#) are free to raise arbitrary exceptions, but any exception that is not a [Site.Exception](#) will transparently be wrapped up by one when used by a [Site](#).

exception `ItemExistsError`

Bases: [Exception](#)

The item already exists.

exception `ConnectionLostError`

Bases: [Exception](#)

The site has been physically disconnected meanwhile.

class `parzzley.fs.Backend`

Bases: `ABC`

Base class for implementations of a particular kind of filesystem.

This part of the filesystem API is only relevant for the implementation of filesystems. Anywhere else, use [Site](#) instead.

While this interface defines some abstract methods, the majority implementation happens in the [Backend.SiteBackend](#) implementation that [connect\(\)](#) returns.

`CONTROL_SITE_ROOT_DIR_NAME = b'..parzzley.control'`

class `SiteBackend`

Bases: `ABC`

Base class for site backends for a particular kind of filesystem.

abstractmethod `async create_item(item, item_type)`

Create a new item.

See [Site.create_item\(\)](#) about what exception it must raise in which situations.

Parameters

- **item** ([Item](#)) – The item location.
- **item_type** ([ItemType](#)) – The new item type.

Return type

`None`

abstractmethod `async remove_item(item, recursive)`

Remove an item.

Parameters

- **item** ([Item](#)) – The item location.

- **recursive** (*bool*) – For a directory, whether to remove all content inside it as well if not empty.

Return type

None

abstractmethod item_type_by_cookie(cookie)

Return the item type by the main stream cookie.

Parameters

- **cookie** (*Any*) – The main stream cookie.

Return type[ItemType](#)**abstractmethod async cookie(item, stream_name)**Return the item cookie for an item. See also [Site.cookie\(\)](#).**Parameters**

- **item** ([Item](#)) – The item location.
- **stream_name** (*str*) – The stream name.

Return type

object

abstractmethod async move_item(item, to_item)

Move an item.

If the source is a normal file and the destination already exists and is also a normal file, the destination will be overwritten.

Parameters

- **item** ([Item](#)) – The source item location.
- **to_item** ([Item](#)) – The destination location.

Return type

None

abstractmethod async child_names(item)

Return the names of all children of an item.

Parameters

- **item** ([Item](#)) – The item location.

Return type[Iterable](#)[*str*]**abstractmethod async read_streamable(item, stream_name)**

Return a read-streamable for an item.

Parameters

- **item** ([Item](#)) – The item location.
- **stream_name** (*str*) – The stream name.

Return type[ReadStreamable](#)**abstractmethod async write_streamable(item, stream_name)**

Return a write-streamable for an item.

Parameters

- **item** ([Item](#)) – The item location.
- **stream_name** (*str*) – The stream name.

Return type[WriteStreamable](#)**abstractmethod async wait_for_changes(on_changed)**

Until canceled, watch this site for any changes.

There will be on filtering, so it will also report changes e.g. on the control site directory.

Parameters

on_changed (*Callable*[[*Item*], *None*]) – The function to call whenever changes were observed.

Return type

None

is_sub_site_of()

Return information about its relation to an outer site, if it is a sub site.

Do not override this method.

Return type

tuple[*SiteBackend*, *Item*] | *None*

_abc_impl = <_abc._abc_data object>

abstractmethod async connect(***kwargs*)

Connect and return a site backend for given arguments (or *None* if this is impossible for some reasons).

This method does not raise any exceptions.

Parameters

kwargs – Additional arguments for the site backend (specific to its type).

Return type

SiteBackend | *None*

abstractmethod async disconnect(*site_backend*)

Disconnect a given site backend (returned earlier by *connect()*).

Must only be called once for each site backend.

This method does not raise any exceptions.

Parameters

site_backend (*SiteBackend*) – The site backend to disconnect. This is what the last *connect()* call returned.

Return type

None

_abc_impl = <_abc._abc_data object>

class parzzley.fs.SiteSetup(*name, backend, arguments*)

Bases: *object*

A site setup specifies a name, a backend and additional arguments. It can eventually generate a *Site* out of that (see *connect()*).

Usually not used directly. See *parzzley.config.loader.load_site_setup()* or even *parzzley.config.loader.load_volume()*.

Parameters

- **name** (*str*) – The site name.
- **backend** (*Backend*) – The backend.
- **arguments** (*dict*[*str*, *str*]) – The additional arguments.

property name: *str*

The site name.

connect()

Connect this site setup. To be used in a *with*-block.

This method does not raise any exceptions, but the context manager might return `None` instead of a site if there were problems during connecting.

This context is allowed to be entered multiple times; even nested. It will automatically reuse the existing connection when an active context gets entered again and only disconnect the backend after the ‘last’ context exits.

Return type

AsyncContextManager[[Site](#) | `None`, `bool` | `None`]

parzzley.fs.register_backend(backend_type)

Decorator that registers a filesystem backend with its parent module name as kind name (e.g. "baz" if the backend is implemented in a package like `foo.bar.baz`), so it can be located later by [backend_by_kind\(\)](#).

Parameters

backend_type (*type*[[Backend](#)]) – The backend type.

parzzley.fs.backend_by_kind(kind)

Return a filesystem backend by its kind name (as registered by [register_backend\(\)](#)).

Parameters

kind (*str*)

Return type

[Backend](#) | `None`

parzzley.fs.TSiteInput = parzzley.fs.Site | parzzley.fs.SiteSetup | str

A site specification that can at least be transformed to a site name by [site_name\(\)](#).

parzzley.fs.site_name(site)

Return a site name for a given site specification.

Parameters

site ([Site](#) | [SiteSetup](#) | *str* | `None`) – The site specification.

Return type

str | `None`

class parzzley.fs._SubTreeSiteBackend(origin_site_backend, root_item)

Bases: [SiteBackend](#)

Parameters

- **origin_site_backend** ([SiteBackend](#))
- **root_item** (*bytes* | [Item](#))

item_type_by_cookie(cookie)

Return the item type by the main stream cookie.

Parameters

cookie – The main stream cookie.

async cookie(item, stream_name)

Return the item cookie for an item. See also [Site.cookie\(\)](#).

Parameters

- **item** – The item location.

- **stream_name** – The stream name.

async remove_item(*item*, *recursive*)

Remove an item.

Parameters

- **item** – The item location.
- **recursive** – For a directory, whether to remove all content inside it as well if not empty.

async move_item(*item*, *to_item*)

Move an item.

If the source is a normal file and the destination already exists and is also a normal file, the destination will be overwritten.

Parameters

- **item** – The source item location.
- **to_item** – The destination location.

async child_names(*item*)

Return the names of all children of an item.

Parameters

- item** – The item location.

async create_item(*item*, *item_type*)

Create a new item.

See [Site.create_item\(\)](#) about what exception it must raise in which situations.

Parameters

- **item** – The item location.
- **item_type** – The new item type.

async read_streamable(*item*, *stream_name*)

Return a read-streamable for an item.

Parameters

- **item** – The item location.
- **stream_name** – The stream name.

async write_streamable(*item*, *stream_name*)

Return a write-streamable for an item.

Parameters

- **item** – The item location.
- **stream_name** – The stream name.

async wait_for_changes(*on_changed*)

Until canceled, watch this site for any changes.

There will be on filtering, so it will also report changes e.g. on the control site directory.

Parameters

- on_changed** – The function to call whenever changes were observed.

is_sub_site_of()

Return information about its relation to an outer site, if it is a sub site.

Do not override this method.

_abc_impl = <_abc._abc_data object>

Submodules

parzzley.fs.stream module

Filesystem streams.

class parzzley.fs.stream.ReadStream

Bases: ABC

Base class for read streams.

A read stream is able to provide arbitrary binary content in chunk-wise manner (see [read\(\)](#)).

After it is consumed once, it cannot be used anymore. It should not be used directly for any API, but only by internal code. See [ReadStreamable](#) instead.

abstractmethod async read(max_len)

Read the next chunk of data from the stream and return it, or return None if the end of the stream has been reached.

Parameters

max_len (*int*) – The maximum chunk length.

Return type

bytes | None

_abc_impl = <_abc._abc_data object>

class parzzley.fs.stream.WriteStream

Bases: ABC

Base class for write streams.

A write stream is able to receive arbitrary binary content in chunk-wise manner. Once everything is written (see [write\(\)](#)), one has to [commit\(\)](#). Otherwise, everything written before might only exist in some hidden, temporary place (client code must never rely on that, though).

After it is committed once, it cannot be used anymore. It should not be used directly for any API, but only by internal code. See [WriteStreamable](#) instead.

abstractmethod async write(data)

Append a chunk of data to the stream.

Parameters

data (*bytes*) – The data.

Return type

None

abstractmethod async commit()

Commit the written data.

Return type

None

```
_abc_impl = <_abc._abc_data object>
```

```
class parzzley.fs.stream.ReadStreamable
```

Bases: ABC

Base class for read-streamables.

A read-streamable can create [ReadStream](#) instances (so, in contrast to that one, it can be used multiple times).

```
stream()
```

Return a context manager (with-block) that provides a [ReadStream](#) for the actual read operation.

Return type

[AsyncGenerator](#)[[ReadStream](#), None]

```
async read_bytes()
```

Convenience function that reads the complete binary content at once.

Return type

bytes

```
abstractmethod async _stream()
```

Return a new stream.

This method is implemented by subclasses and only used internally. See [stream\(\)](#).

Return type

[ReadStream](#)

```
_abc_impl = <_abc._abc_data object>
```

```
class parzzley.fs.stream.WriteStreamable
```

Bases: ABC

Base class for write-streamables.

A write-streamable can create [WriteStream](#) instances (so, in contrast to that one, it can be used multiple times).

```
stream()
```

Return a context manager (with-block) that provides a [WriteStream](#) for the actual write operation.

Return type

[AsyncGenerator](#)[[WriteStream](#), None]

```
async write_bytes(content)
```

Convenience function that writes a complete binary content at once.

Parameters

content (bytes) – The binary content to write.

Return type

None

```
abstractmethod async _stream()
```

Return a new stream.

This method is implemented by subclasses and only used internally. See [stream\(\)](#).

Return type

[WriteStream](#)

```
_abc_impl = <_abc._abc_data object>
```

```
class parzzley.fs.stream.MemoryReadStreamable(content)
```

Bases: [ReadStreamable](#)

A memory backed read-streamable.

Parameters

content (*bytes*) – The content of this stream.

```
class _ReadStream(content)
```

Bases: [ReadStream](#)

Parameters

content (*bytes*)

```
async read(max_len)
```

Read the next chunk of data from the stream and return it, or return None if the end of the stream has been reached.

Parameters

max_len – The maximum chunk length.

```
_abc_impl = <_abc._abc_data object>
```

```
async _stream()
```

Return a new stream.

This method is implemented by subclasses and only used internally. See `stream()`.

```
_abc_impl = <_abc._abc_data object>
```

```
class parzzley.fs.stream.MemoryWriteStreamable(commit_func)
```

Bases: [WriteStreamable](#)

A memory backed write-streamable.

Parameters

commit_func (*Callable*)

```
class _WriteStream(commit_func)
```

Bases: [WriteStream](#)

Parameters

commit_func (*Callable*) – The commit function to call when a stream finished writing.

```
async write(data)
```

Append a chunk of data to the stream.

Parameters

data – The data.

```
async commit()
```

Commit the written data.

```
_abc_impl = <_abc._abc_data object>
```

```
async _stream()
```

Return a new stream.

This method is implemented by subclasses and only used internally. See `stream()`.

```
_abc_impl = <_abc._abc_data object>
```

parzzley.fs.utils module

Low-level utils for filesystem implementations.

async `parzzley.fs.utils.site_for_key(site, key)`

Return an inner site for an existing one by a key. The key is used for generating a subdirectory name on the given site.

Parameters

- **site** ([Site](#)) – The original site.
- **key** (*str*) – The key.

Return type

[Site](#)

`parzzley.fs.utils.serialize_bytes_dict(xattrs)`

Convert a dictionary with extended attributes to serialized binary content (as used for [deserialize_bytes_dict\(\)](#)).

Parameters

xattrs (*dict[bytes, bytes]*) – Dictionary with extended attributes.

Return type

bytes

`parzzley.fs.utils.deserialize_bytes_dict(xattrs_bin)`

Convert a serialized binary content (as returned by [serialize_bytes_dict\(\)](#)) to a dictionary with extended attributes.

Parameters

xattrs_bin (*bytes*) – Serialized binary content.

Return type

dict[bytes, bytes]

parzzley.service package

Parzzley service is a loop.

See [Service](#).

class `parzzley.service.Service(*, config_dir)`

Bases: `object`

Parzzley service is a loop - usually running in background - that executes the synchronization(s) from its configuration in regular intervals and whenever triggered from outside. See e.g. [run\(\)](#).

Parameters

config_dir (*Path* / *str*) – The Parzzley config directory.

property volumes: `Sequence[Volume]`

All sync volumes processed by this service.

stopSoon()

Stop the service as soon as possible.

This method does not wait for that to actually happen but returns instantly (non-blocking).

This can be called from anywhere in any situation. Once called, the service will stop as soon as possible and will never be able to run again (i.e. you need to create a new instance if you want to run it again later).

Return type

None

syncSoon(volume_name)

Request the service loop to execute the sync for a given volume as soon as possible.

This method does not wait for that to actually happen but returns instantly (non-blocking).

Parameters

volume_name (*str*) – The name of the volume to sync.

Return type

None

run()

Run the service loop. This will potentially block forever (but see [*stopSoon\(\)*](#)).

This will run the configured sync volumes in their regular intervals or whenever [*syncSoon\(\)*](#) was called.

Return type

None

class _LoopThread(*, name)

Bases: Thread

This constructor should always be called with keyword arguments. Arguments are:

group should be None; reserved for future extension when a ThreadGroup class is implemented.

target is the callable object to be invoked by the run() method. Defaults to None, meaning nothing is called.

name is the thread name. By default, a unique name is constructed of the form “Thread-N” where N is a small decimal number.

args is a list or tuple of arguments for the target invocation. Defaults to ().

kwargs is a dictionary of keyword arguments for the target invocation. Defaults to {}.

If a subclass overrides the constructor, it must make sure to invoke the base class constructor (Thread.__init__()) before doing anything else to the thread.

Parameters

name (*str*)

stopSoon()

Request this thread to stop as soon as possible and return directly after doing the request (non-blocking).

Return type

None

async _action()**Return type**

None

async _stopping()**Return type**

None

property stop_requested: bool

Whether this thread was requested to stop (or even already stopped).

`_sleep(duration)`

Parameters

`duration` (*float*)

Return type

None

`_LoopThread__run()`

`async _LoopThread__run__async()`

`class _VolumeThread(service, manager, volume, force_sync_requests)`

Bases: [`_LoopThread`](#)

This constructor should always be called with keyword arguments. Arguments are:

group should be None; reserved for future extension when a ThreadGroup class is implemented.

target is the callable object to be invoked by the run() method. Defaults to None, meaning nothing is called.

name is the thread name. By default, a unique name is constructed of the form “Thread-N” where N is a small decimal number.

args is a list or tuple of arguments for the target invocation. Defaults to ().

kwargs is a dictionary of keyword arguments for the target invocation. Defaults to {}.

If a subclass overrides the constructor, it must make sure to invoke the base class constructor (Thread.__init__()) before doing anything else to the thread.

Parameters

- **`volume`** ([`Volume`](#))
- **`force_sync_requests`** (*list[str]*)

`async _action()`

`async _stopping()`

`class _RunningSyncTask(prepare_sync_context: AsyncContextManager[ForwardRef('parzzley.sync.manager.Manager.PreparedSyncSetup'), bool | None], prepared_sync_setup: 'parzzley.sync.manager.Manager.PreparedSyncSetup', sync_control: 'parzzley.sync.control.SyncControl')`

Bases: `object`

Parameters

- **`prepare_sync_context`** ([`AsyncContextManager`](#)[[`PreparedSyncSetup`](#), *bool | None*])
- **`prepared_sync_setup`** ([`PreparedSyncSetup`](#))
- **`sync_control`** ([`SyncControl`](#))

`prepare_sync_context:` [`AsyncContextManager`](#)[[`PreparedSyncSetup`](#), *bool | None*]

`prepared_sync_setup:` [`PreparedSyncSetup`](#)

`sync_control:` [`SyncControl`](#)

`async _VolumeThread__action__finished_sync_run()`

`_VolumeThread__action__outdated_sites()`

Return type

Sequence[[`Site`](#)]

async `_VolumeThread__action__start_sync(outdated_sites)`

Parameters

outdated_sites (*Sequence*[[Site](#)])

_VolumeThread__action__sync_forced_flag()

Return type

None

class `_MonitorSiteForChangesThread(service, volume, site)`

Bases: [_LoopThread](#)

This constructor should always be called with keyword arguments. Arguments are:

group should be None; reserved for future extension when a ThreadGroup class is implemented.

target is the callable object to be invoked by the run() method. Defaults to None, meaning nothing is called.

name is the thread name. By default, a unique name is constructed of the form “Thread-N” where N is a small decimal number.

args is a list or tuple of arguments for the target invocation. Defaults to ().

kwargs is a dictionary of keyword arguments for the target invocation. Defaults to {}.

If a subclass overrides the constructor, it must make sure to invoke the base class constructor (Thread.__init__()) before doing anything else to the thread.

Parameters

site ([Site](#))

async `_action()`

_MonitorSiteForChangesThread__mark_changed()

parzzley.sync package

Parzzley syncing.

This module only defines some common structures and interfaces. See submodules for more. In order to execute syncing, see [parzzley.sync.manager](#).

class `parzzley.sync.Volume(name, site_setups, aspects_per_site)`

Bases: object

A sync volume.

A sync volume basically defines some sites (see [parzzley.fs.Site](#)), i.e. some local or remote filesystems and aspects (see [parzzley.sync.aspect.Aspect](#)). A synchronization takes place on a particular volume.

Parameters

- **name** (*str*) – The volume name.
- **site_setups** (*Iterable*[[parzzley.fs.SiteSetup](#)]) – The sites associated to this volume (as site setups).
- **aspects_per_site** (*dict*[*str*, *Iterable*[[parzzley.sync.aspect.Aspect](#)]]) – The aspects per site.

property name: *str*

The volume name.

property site_setups: `Sequence[parzzley.fs.SiteSetup]`

The sites associated to this volume (as site setups).

aspects(*site_name*)

Return the aspects associated to a given site of this volume.

This includes all aspects that were configured particularly on the given site, but also all aspects that were configured for the entire volume.

Parameters

site_name (*str*) – The site name.

Return type

`Sequence[parzzley.sync.aspect.Aspect]`

parzzley.sync.load_implementations_from_package(*package_name*)

Load custom implementations (for aspects, filesystem backends, ...).

Parameters

package_name (*str*) – The Python package to load implementations from.

Return type

None

Subpackages

[parzzley.sync.aspect](#) package

Aspect API. See [Aspect](#).

class `parzzley.sync.aspect.Aspect`(**inner_aspects*)

Bases: object

Aspects are plugins for the Parzzley sync engine. The engine itself does not much more than hosting and controlling these aspects. All visible effects from a sync run (i.e. all its actual duties) are realized by aspects; not by the engine itself.

A typical aspect is a subclass of [Aspect](#) that includes some event handlers. See also [event_handler\(\)](#) and [parzzley.sync.aspect.events](#) for more details.

Parameters

inner_aspects ([Aspect](#))

_all_event_handlers()

parzzley.sync.aspect.event_handler(**conditions*, *more_names*=(), *beforehand*=(), *beforehand_optional*=(), *afterwards*=(), *afterwards_optional*=())

Decorate an aspect method to be an event handler.

Parameters

- **conditions** – Conditions that need to be satisfied for execution. See [parzzley.sync.aspect.only_if](#).
- **more_names** (*Iterable[str]*) – Additional names of this event handler. Used for dependency specification together with the following parameters.
- **beforehand** (*Iterable[str | Callable]*) – Event handlers that must be executed before this one.
- **beforehand_optional** (*Iterable[str | Callable]*) – Event handlers that must be executed before this one if they exist.

- **afterwards** (*Iterable[str | Callable]*) – Event handlers that must be executed after this one.
- **afterwards_optional** (*Iterable[str | Callable]*) – Event handlers that must be executed after this one if they exist.

`parzzley.sync.aspect._dep_str(dep)`

Parameters

dep (*str | Callable*)

Return type

str

`parzzley.sync.aspect._dep_list(orig_list)`

Parameters

orig_list (*Iterable[str | Callable]*)

Return type

Sequence[str]

`parzzley.sync.aspect.register_aspect(aspect_type)`

Make an aspect class available to be used, e.g. in volume configurations.

Parameters

aspect_type (*type[Aspect]*) – The aspect class to register.

Return type

type[Aspect]

`parzzley.sync.aspect.aspect_type_by_name(name)`

Return an aspect type by the type name.

Parameters

name (*str*) – The aspect type name.

Return type

type[Aspect] | None

Subpackages

`parzzley.sync.aspect.events` package

Aspect events are part of the plug-in mechanism that encapsulates its behavior in aspects.

In the course of a sync run, the Parzzley engine will trigger those events in a particular order. Aspects implement various parts of Parzzley functionality by handling some of those events (while the engine itself does not much more than triggering these events).

The Parzzley documentation also contains an overview of aspect events and how they actually perform their duty of filesystem synchronization.

class `parzzley.sync.aspect.events._Event`

Bases: `ABC`

Base class for all events that can be handled by aspects. They are fired by the sync engine in order to execute the actual sync logic that is configured. See also [parzzley.sync.aspect](#) for the basic idea.

abstract property site: [parzzley.fs.Site](#)

The current site.

This is the site which this aspect is associated with, so it usually operates on this site.

_abc_impl = <**_abc._abc_data** object>

class `parzzley.sync.aspect.events.Data`(*initial_value=None*, *, *per_site=False*, *per_item=False*, *per_stream=False*)

Bases: object

Event data variable for storage of arbitrary information during a sync run (or smaller scopes).

Parameters

- **initial_value** – The initial value.
- **per_site** (*bool*) – Whether this data is stored separately per site (giving it a smaller scope).
- **per_item** (*bool*) – Whether this data is stored separately per item (giving it a smaller scope).
- **per_stream** (*bool*) – Whether this data is stored separately per stream (giving it a smaller scope).

get(*event*, *, *key_data=None*)

Return the current value.

Parameters

- **event** ([_Event](#)) – The event that carries the given situation.
- **key_data** (*dict* | *None*) – Additional selector key. Do not use.

set(*event*, *value*, *key_data=None*)

Set the value.

Parameters

- **event** ([_Event](#)) – The event that carries the given situation.
- **value** (*object*) – The new value.
- **key_data** (*dict* | *None*) – Additional selector key. Do not use.

Return type

None

Subpackages

[parzzley.sync.aspect.events.item](#) package

Item events. See also [parzzley.sync.aspect.events](#).

class `parzzley.sync.aspect.events.item._WithSetItemInfo`

Bases: ABC

abstractmethod **set_item_info**(*item_type*, *stream_cookie*, *, *for_site=None*)

Set item info for this site or another.

Parameters

- **item_type** ([parzzley.fs.ItemType](#)) – The item type.
- **stream_cookie** – The stream cookie.

- **for_site** (*parzzley.fs.TSiteInput* / *None*) – The site to store the item info for (default: the current site).

Return type

None

_abc_impl = <**_abc._abc_data** object>

class *parzzley.sync.aspect.events.item._WithStreamCookie*

Bases: *ABC*

abstractmethod *stream_cookie*(*for_site=None*)

Return the stream cookie of the main stream, usually determined in *parzzley.sync.aspect.events.item.stream.DetermineCookie*.

For details about cookies, see *parzzley.fs.Site.cookie()*.

This is the same as *stream_cookie* of stream events related to the main stream (i.e. "").

Parameters

for_site (*parzzley.fs.TSiteInput* / *None*) – For which site? Default is the event's current site.

Return type

parzzley.fs.Site.TCookie

_abc_impl = <**_abc._abc_data** object>

class *parzzley.sync.aspect.events.item._WithWorkingItems*

Bases: *ABC*

abstractmethod *working_items*(***, *for_site=None*)

Return all working items. Those are either the ones added by *add_working_item()* earlier, or, if none are added, it is a list containing only *item* (if it exists on item master and is not a directory).

Parameters

for_site (*parzzley.fs.TSiteInput* / *None*) – For which site? Default is the event's current site.

Return type

Iterable[*parzzley.fs.Item*]

_abc_impl = <**_abc._abc_data** object>

class *parzzley.sync.aspect.events.item._Event*

Bases: *_EventAfterStreamSupportDetermined*, *ABC*

Base class for an aspect event on an *parzzley.fs.Item*.

Such events potentially happen once for each visited file, directory, etc.

abstract property *item*: *parzzley.fs.Item*

The current item.

abstractmethod *mark_full_retry_needed*()

Mark the current item for a full retry (e.g. because ongoing changes were detected during update).

Return type

None

abstract property is_marked_full_retry_needed: bool

Whether the current item is marked for a full retry (see [mark_full_retry_needed\(\)](#)).

_abc_impl = <_abc._abc_data object>

class parzzley.sync.aspect.events.item.DecideToSkip

Bases: [_Event](#), ABC

Occurs in order to decide if to skip an item according to criteria like path exclusions.

See [skip_item\(\)](#).

It is the earliest item event, happening before [Prepare](#) and various preparation steps for each of the supported streams.

These and all other following item events will only occur if this event does not decide to skip the current item.

abstract property is_item_marked_to_skip: bool

Whether the current item is marked for being skipped at least this time (i.e. by [skip_item\(\)](#)).

abstract property is_item_marked_to_skip_permanently: bool

Whether the current item is marked for being skipped permanently (i.e. by [skip_item\(\)](#)).

abstractmethod skip_item(*, only_now=False)

Mark the current item for being skipped.

Parameters

only_now (*bool*) – Whether to skip this item now, but keep its internal state data for later sync runs.

Return type

None

sites_involved_in_current_item_last_successful_sync()

Return the name of the sites that were involved in the last sync run which has successfully synced the current item (this is the last sync run if it was successful and has not skipped the current item), or None if it was not synced yet.

Return type

Sequence[str] | None

_abc_impl = <_abc._abc_data object>

class parzzley.sync.aspect.events.item.Prepare

Bases: [_Event](#), ABC

Occurs in order to prepare item sync.

It is a very early item event, happening after [DecideToSkip](#) and before various preparation steps for each of the supported streams.

Typical implementations use this event e.g. for handling (parts of) the item-retry logic.

_abc_impl = <_abc._abc_data object>

class parzzley.sync.aspect.events.item._EventAfterItemTypeKnown

Bases: [_Event](#), ABC

Base class for item events that occur after the item type was determined.

abstractmethod `item_type(for_site=None)`

Return the just determined type for the current item.

Parameters

for_site (`parzzley.fs.TSiteInput | None`) – For which site? Default is the event's current site.

Return type

`parzzley.fs.ItemType`

`_abc_impl = <_abc._abc_data object>`

class `parzzley.sync.aspect.events.item.DetermineType`

Bases: `_Event`, `_WithStreamCookie`, `ABC`

Occurs in order to determine the item type (usually derived from the main stream cookie).

This event should solely be handled in Parzzley infrastructure aspects, not in custom ones!

See `set_item_type()` and `stream_cookie()`.

It happens during the preparation steps of the main stream, after `parzzley.sync.aspect.events.item.stream.DetermineCookie` and before `parzzley.sync.aspect.events.item.stream.DetectChanges` of the main stream.

The typical implementation just queries `parzzley.fs.Size.item_type_by_cookie` using the cookie from `parzzley.sync.aspect.events.item.stream.DetermineCookie`.

abstractmethod `set_item_type(item_type, *, for_site=None)`

Declare the item type of the current item on this site.

Parameters

- **item_type** (`parzzley.fs.ItemType`) – The item type.
- **for_site** (`parzzley.fs.TSiteInput | None`) – For which site? Default is the event's current site.

Return type

`None`

`_abc_impl = <_abc._abc_data object>`

class `parzzley.sync.aspect.events.item.SkipDueToConflicts`

Bases: `_Event`, `ABC`

Occurs when an item was skipped due to conflicts.

It happens after the stream preparation is done, before `RefreshItemsBook`.

class `_Conflict(stream_name: str, description: str, details: str | None)`

Bases: `object`

Parameters

- **stream_name** (`str`)
- **description** (`str`)
- **details** (`str | None`)

stream_name: `str`

description: `str`

```

    details: str | None

    abstract property all_conflicts: Iterable[_Conflict]
        Conflicts.

    _abc_impl = <_abc._abc_data object>

class parzzley.sync.aspect.events.item._EventAfterMainStreamMasterSiteDetermined
    Bases: _EventAfterItemTypeKnown, ABC

    property master_site: parzzley.fs.Site
        The site that was determined to keep the 'right' (usually: most recent) data for the main stream.
        This is the same as master_site of stream events related to the main stream (i.e. "").

    abstractmethod content_version(for_site=None, *, only_past)
        Return the version number of the current item's main stream, for this site or any other one.
        This is the same as content_version of stream events related to the main stream (i.e. ""). See there for more
        details.

        Parameters
            • for_site (parzzley.fs.TSiteInput | None) – For which site? Default is the event's
              current site.
            • only_past (bool) – Depending on this flag, this will either always be the sn of a sync
              run from the past, i.e. it will always be less than the current sync run's sn, or it returns the
              current sync run's sn if there were changes detected since the last sync run.

        Return type
            int

    abstractmethod _master_site(of_stream=None)
        Return the master site for the given stream (or the item's main stream).

        Parameters
            of_stream (str | None)

        Return type
            parzzley.fs.Site

    _abc_impl = <_abc._abc_data object>

class parzzley.sync.aspect.events.item.PrepareUpdating
    Bases: _EventAfterMainStreamMasterSiteDetermined, _WithSetItemInfo, ABC

    Occurs in order to prepare item updates.

    A handler may apply local changes to the item. If it does, set_item_info() must be called afterward.

    It happens as first step after the stream preparation is done and no conflicts occurred, before
    SetUpWorkingItems.

    _abc_impl = <_abc._abc_data object>

class parzzley.sync.aspect.events.item.SetUpWorkingItems
    Bases: _EventAfterMainStreamMasterSiteDetermined, _WithStreamCookie, _WithWorkingItems,
    ABC

    Occurs in order to maybe create working items in non-master sites that can receive the main content and secondary
    streams for an item and can be committed (ideally atomically) later.

```

See [add_working_item\(\)](#).

It happens as early after the stream preparation is done and no conflicts occurred, after [PrepareUpdating](#) and before the per-stream update events.

A typical implementation would create a temporary working item if on this site the main stream needs an update from the master site. In other situations, i.e. if this site's main stream is already up-to-date, there will be no temporary working item. In later events, other streams will directly apply their updates in place instead.

abstractmethod `working_items(*, for_site=None)`

Return all working items. Those are either the ones added by [add_working_item\(\)](#) earlier, or, if none are added, it is a list containing only `item` (if it exists on item master and is not a directory).

Parameters

for_site (`parzzley.fs.TSiteInput` | `None`) – For which site? Default is the event's current site.

Return type

`Iterable[parzzley.fs.Item]`

abstractmethod `add_working_item(working_item, *, for_site=None)`

Add a working item.

Parameters

- **working_item** (`parzzley.fs.TItemInput`) – The working item.
- **for_site** (`parzzley.fs.TSiteInput` | `None`) – For which site? Default is the event's current site.

Return type

`None`

abstractmethod `remove_working_item(working_item, *, for_site=None)`

Remove a working item.

Parameters

- **working_item** (`parzzley.fs.TItemInput`) – The working item.
- **for_site** (`parzzley.fs.TSiteInput` | `None`) – For which site? Default is the event's current site.

Return type

`None`

`_abc_impl = <_abc._abc_data object>`

class `parzzley.sync.aspect.events.item.ApplyUpdate`

Bases: [_EventAfterMainStreamMasterSiteDetermined](#), [ABC](#)

Occurs in order to finish/apply stream updates (particularly these that were not in-place updates). All apply implementations should be atomic, i.e. whenever they actually apply an update, they should either fail, leaving the destination untouched, or succeed completely. They should never e.g. cleanup the destination but then maybe fail to bring the new copy in place!

If the item was touched by the update, [update_cookies\(\)](#) must be called.

It happens after the stream update events occurred and is one of the latest events of item processing, before [RefreshItemsBook](#).

abstractmethod `update_cookies(stream_cookies, *, for_site=None)`

Set new cookies as they are after the update for this site.

If the given dictionary contains only some of the supported stream names, the other stream cookies will stay untouched (like `dict.update()`)!

Parameters

- **stream_cookies** – The new stream cookies.
- **for_site** (`parzzley.fs.TSiteInput | None`) – For which site? Default is the event's current site.

Return type

None

abstractmethod `streams_with_destination_streamables(for_site=None)`

Return the list of stream names for which there are any destination streamables on this site or any other one.

Parameters

for_site (`parzzley.fs.TSiteInput | None`) – For which site? Default is the event's current site.

Return type

Sequence[str]

`_abc_impl = <_abc._abc_data object>`

class `parzzley.sync.aspect.events.item.RefreshItemsBook`

Bases: `_EventAfterItemTypeKnown`, `ABC`

Occurs in order to refresh item books after item stream updates.

This event should solely be handled in Parzzley infrastructure aspects, not in custom ones!

See `updated_cookie()` and `set_item_book_entry()`.

It happens after the stream update, i.e. after `ApplyUpdate` (or after a conflict was detected).

The typical implementation primarily stores the “updated cookie” (determined in `ApplyUpdate`), so it can be used for decisions in the next sync run.

abstractmethod `updated_cookie(stream_name, *, for_site=None)`

The updated cookie for a given stream that needs to be stored.

Parameters

- **stream_name** (`str`) – The stream name.
- **for_site** (`parzzley.fs.TSiteInput | None`) – For which site? Default is the event's current site.

Return type

`parzzley.fs.TCookie`

abstractmethod `set_item_book_entry(item_type, cookies, *, for_site=None)`

Write an entry to the item book. This info can be used in later sync runs.

Parameters

- **item_type** (`parzzley.fs.ItemType`) – The new item type to store.
- **cookies** (`dict[str, object]`) – The new stream cookies to store.

- **for_site** (*parzzley.fs.TSiteInput* / *None*) – For which site? Default is the event's current site.

Return type

None

_abc_impl = <_abc._abc_data object>**Submodules****parzzley.sync.aspect.events.item.dir module**

Directory item events. See also *parzzley.sync.aspect.events*.

class parzzley.sync.aspect.events.item.dir._Event

Bases: *_EventAfterItemTypeKnown*, ABC

Base class for an aspect event on an *parzzley.fs.Item* which is a directory.

These extend the stuff from *parzzley.sync.aspect.events.item* by directory specific logic, like directory traversal.

_abc_impl = <_abc._abc_data object>**class** parzzley.sync.aspect.events.item.dir.Prepare

Bases: *_Event*, ABC

Occurs at the beginning of a directory sync processing, after *parzzley.sync.aspect.events.item.stream.TransferUpdate* of the main stream, before *List*.

_abc_impl = <_abc._abc_data object>**class** parzzley.sync.aspect.events.item.dir.List

Bases: *_Event*, ABC

Occurs in order to get a list of the directory's children, after *Prepare* and before *Iterated*.

This event should solely be handled in Parzzley infrastructure aspects, not in custom ones!

There will be a sequence of per-item (and so also per-stream) events for each determined child item directly after this event.

The typical implementation retrieves that list directly from the site.

abstractmethod add_child_names(*child_names*)

Add child names to the list of all children to be processed for this directory.

Parameters

child_names (*Iterable[str]*) – The names of the children to be added.

Return type

None

_abc_impl = <_abc._abc_data object>**class** parzzley.sync.aspect.events.item.dir.Iterated

Bases: *_Event*, ABC

Occurs at the end of a directory sync, after all children are processed, and before the item updating continues to *parzzley.sync.aspect.events.item.ApplyUpdate*.

_abc_impl = <_abc._abc_data object>

parzzley.sync.aspect.events.item.stream module

Item stream events. See also [parzzley.sync.aspect.events](#).

class `parzzley.sync.aspect.events.item.stream._WithStreamCookie`

Bases: ABC

abstractmethod `stream_cookie(for_site=None)`

Return the stream cookie of the current stream, usually determined in [parzzley.sync.aspect.events.item.stream.DetermineCookie](#).

Stream cookies are mostly an opaque data structure (beyond the fact that the main stream's cookie must somehow carry the item type), but comparing stream cookies from the same item and site and from different times allows to detect changes.

For the main stream (i.e. ""), this is the same as *stream_cookie* of some item events.

Parameters

for_site (`parzzley.fs.TSiteInput` | `None`)

Return type

`parzzley.fs.Site.TCookie`

`_abc_impl = <_abc._abc_data object>`

class `parzzley.sync.aspect.events.item.stream._WithSourceStreamableGetters`

Bases: ABC

abstractmethod `source_streamable(for_site=None)`

Return the source streamable for this site. It is used as the source of the stream update step.

Usually the source streamable is from the stream master site (this is the main reason to determine it before).

It was determined earlier in [CollectSourceStreamables](#).

Parameters

for_site (`Site` | `SiteSetup` | `str` | `None`) – For which site? Default is the event's current site.

Return type

`ReadStreamable` | `None`

abstractmethod `stream_pipes(for_site=None)`

Return all stream pipes.

They are used by the stream update step as a mechanism to apply changes to the stream while it is transferring from the source to the destinations.

Parameters

for_site (`parzzley.fs.TSiteInput` | `None`) – For which site? Default is the event's current site.

Return type

`Iterable[CollectDestinationStreamables.Pipe]`

`_abc_impl = <_abc._abc_data object>`

class `parzzley.sync.aspect.events.item.stream._WithDestinationStreamableGetters`

Bases: ABC

abstractmethod destination_streamables()

Return all destination streamables. They are used as destinations of the stream update step.

Usually there is one destination streamable for each site that is not the stream master site. There is usually no destination streamable for the master site, as that is the streaming source instead.

Return type

Sequence[[WriteStreamable](#)]

_abc_impl = <_abc._abc_data object>

class `parzzley.sync.aspect.events.item.stream._WithMasterSiteTableSetter`

Bases: `ABC`

abstractmethod set_master_site(*master_site=None*)

Set the master site for the current item stream.

This is usually the site where the other sites will get updated from (regarding the current stream).

Parameters

master_site ([Site](#) / [SiteSetup](#) / *str* / *None*) – The master site.

Return type

None

abstractmethod set_sync_run_sn_of_last_change(*sn, *, for_site=None*)

Set the serial number of the sync run that has seen the last change (for this site or any other).

Parameters

- **sn** (*int*) – The sync run’s serial number.
- **for_site** ([Site](#) / [SiteSetup](#) / *str* / *None*) – For which site? Default is the event’s current site.

Return type

None

_abc_impl = <_abc._abc_data object>

class `parzzley.sync.aspect.events.item.stream._Event`

Bases: [_EventAfterItemTypeKnown](#), `ABC`

Base class for an aspect event on a stream of a [parzzley.fs.Item](#).

abstract property stream_name: *str*

The current stream name.

abstractmethod _master_site(*of_stream=None*)

Return the master site for the given stream (or the current stream).

Parameters

of_stream (*str* / *None*)

Return type

[Site](#)

_abc_impl = <_abc._abc_data object>

class `parzzley.sync.aspect.events.item.stream._EventAfterItemInfoCollected`

Bases: [_Event](#), `ABC`

Base class for stream events that occur after information about the stream was collected.

```
_abc_impl = <_abc._abc_data object>
```

```
class parzzley.sync.aspect.events.item.stream._EventAfterMasterSiteDetermined
```

Bases: [_EventAfterItemInfoCollected](#), ABC

Base class for stream events that occur after the master site for this stream was determined.

property master_site: [Site](#)

The stream master site (for a particular stream on a particular filesystem item) is the site that is determined to contain the ‘right’ data for this stream, i.e. usually the most recently modified one.

It is determined after the change detection phase, i.e. mostly during [parzzley.sync.aspect.events.item.stream.DetermineMasterSiteTable](#).

For the main stream (i.e. ""), this is the same as *master_site* of some item events.

abstractmethod content_version(*for_site=None, *, only_past*)

Return the version number of the current item stream, for this site or any other one.

This number always refers to a particular sync run. So, it can be the serial number of an earlier sync run, meaning that the site has successfully synchronized this item last time in that sync run. Note: This does not include sync runs that skipped this file from the beginning or due to conflicts.

If this item was marked as needing to get refreshed, this will return 0 (meaning: it has the lowest possible version number and so definitely needs to get updated).

If *only_past* is False and there were changes detected, this will return the serial number of the current sync run.

For the main stream (i.e. ""), this is the same as *content_version* of some item events.

Parameters

- **for_site** ([Site](#) / [SiteSetup](#) / *str* / *None*) – For which site? Default is the event’s current site.
- **only_past** (*bool*) – Whether to exclude the current sync run and only look into the past.

Return type

int

```
_abc_impl = <_abc._abc_data object>
```

```
class parzzley.sync.aspect.events.item.stream.DetermineComparator
```

Bases: [_Event](#), ABC

Occurs in order to determine the comparator for the current item stream.

This is the first step of stream preparation. It happens before [DetermineCookie](#).

Without any changes to the comparator, it will do byte-wise comparison. Aspects can override it for particular streams if the comparison is more complicated than that.

abstract property comparator: [Callable](#)[[[ReadStreamable](#), [ReadStreamable](#)], *bool*] | *None*

The current comparator (or None for the default, byte-wise one). See also [set_comparator\(\)](#).

abstractmethod set_comparator(*comparator*)

Set the comparator.

Parameters

comparator – The new comparator.

`_abc_impl = <_abc._abc_data object>`

class `parzzley.sync.aspect.events.item.stream.DetermineCookie`

Bases: `_Event`, `ABC`

Occurs in order to determine the cookie for the current item stream.

This event should solely be handled in Parzzley infrastructure aspects, not in custom ones!

This is an early step of stream preparation, after `DetermineComparator`. If the current stream is the main stream, `parzzley.sync.aspect.events.item.DetermineType` happens afterward. Then the stream preparation continues to the change detection phase (`DetectChanges`).

The typical implementation just queries the cookie from the site.

abstractmethod `set_stream_cookie(stream_cookie, *, for_site=None)`

Set the stream cookie for the current item stream.

Parameters

- **stream_cookie** – The stream cookie.
- **for_site** (`Site` / `SiteSetup` / `str` / `None`) – For which site? Default is the event's current site.

`_abc_impl = <_abc._abc_data object>`

class `parzzley.sync.aspect.events.item.stream.DetectChanges`

Bases: `_EventAfterItemInfoCollected`, `_WithStreamCookie`, `ABC`

Occurs in order to detect changes of the current item stream at the current site.

The detection (as it is implemented by Parzzley builtin aspects) basically works by comparing the recent stream cookie with the last one (but the full solution is more complex, e.g. for directories, removed items, ...).

See `mark_changed()`.

It happens after the stream cookie (and the item type) is known and before `DetermineMasterSiteTable`. This is all part of the stream preparation.

abstractmethod `stream_cookie_after_last_sync(for_site=None)`

Return the cookie this item stream had after last sync.

Parameters

for_site (`Site` / `SiteSetup` / `str` / `None`) – For which site? Default is the event's current site.

Return type

tuple

abstractmethod `item_type_after_last_sync(for_site=None)`

Return the type this item had after last sync.

This is only useful for the main stream (i.e. "").

Parameters

for_site (`Site` / `SiteSetup` / `str` / `None`) – For which site? Default is the event's current site.

Return type

`ItemType`

abstractmethod `mark_changed(site=None)`

Mark this site as having changed. This might make this site the stream master site later, so the other sites get updated from it.

Parameters

site (`Site` / `SiteSetup` / `str` / `None`) – For which site? Default is the event’s current site.

Return type

`None`

abstractmethod `mark_all_last_involved_sites_changed_if_this_is_not_one_of_them(site=None)`

Mark all lastly involved sites as having changed (see `mark_changed()`) if the given site is not one of them.

This prevents data loss in some situations and will eventually lead to conflicts instead.

Parameters

site (`Site` / `SiteSetup` / `str` / `None`) – For which site? Default is the event’s current site.

Return type

`None`

abstractmethod `mark_refresh_needed(site=None)`

Mark this site as definitely outdated. In some way this is the opposite to `mark_changed()`. This is only used in some special contexts and not part of the basic mechanism.

Parameters

site (`Site` / `SiteSetup` / `str` / `None`) – For which site? Default is the event’s current site.

Return type

`None`

`_abc_impl = <_abc._abc_data object>`

class `parzzley.sync.aspect.events.item.stream.DetermineMasterSiteTable`

Bases: `_EventAfterMasterSiteDetermined`, `_WithMasterSiteTableSetter`, `ABC`

Occurs after changes were detected, in order to determine the master site for the current item stream. This is the site where the other sites will get updated from (regarding the current stream).

This event should solely be handled in Parzzley infrastructure aspects, not in custom ones!

It happens after `DetectChanges` and before `MasterSiteDetermined`. This is all part of the stream preparation.

The typical implementation determines that based on the information gathered in `DetectChanges`.

abstractmethod `is_site_marked_as_changed(for_site=None)`

Return whether this site was marked as changed (usually by `DetectChanges.mark_changed()`).

Parameters

for_site (`Site` / `SiteSetup` / `str` / `None`) – For which site? Default is the event’s current site.

Return type

`bool`

abstractmethod `is_site_marked_as_needing_refreshed(for_site=None)`

Return whether this site was marked as needing to get refreshed (usually by `DetectChanges.mark_refresh_needed()`).

Parameters

for_site ([Site](#) / [SiteSetup](#) / *str* / *None*) – For which site? Default is the event's current site.

Return type

bool

_abc_impl = <_abc._abc_data object>

class `parzzley.sync.aspect.events.item.stream.MasterSiteDetermined`

Bases: [_EventAfterMasterSiteDetermined](#), [_WithSetItemInfo](#), ABC

Occurs after the master site for the current item stream was determined.

It happens after [DetermineMasterSiteTable](#) and before stream conflict handling. This is all part of the stream preparation.

A handler may apply local changes to the item. If it does, `set_item_info()` must be called afterward.

_abc_impl = <_abc._abc_data object>

class `parzzley.sync.aspect.events.item.stream.CollectSourceStreamables`

Bases: [_EventAfterMasterSiteDetermined](#), [_WithStreamCookie](#), [_WithSourceStreamableGetters](#), [_WithWorkingItems](#), ABC

Occurs in order to collect read-streamable objects for the current item stream.

See [set_source_streamable\(\)](#).

It occurs as a step of the stream preparation, after [MasterSiteDetermined](#) and before conflict handling.

The typical implementation directly gets the source streamable from the site. For some streams, there may be additional aspects that do further things, e.g. adding stream pipes for some arbitrary stream translations.

class `Pipe(inner_streamable)`

Bases: [ReadStreamable](#), ABC

An abstract pipe.

Pipes are used by the stream update process as a mechanism to apply changes to the stream while it is transferring from the source to the destinations.

In order to implement a new pipe, see also [CollectSourceStreamables.FullContentPipe](#).

Parameters

inner_streamable (`parzzley.fs.stream.ReadStreamable`)

class `_ReadStream(inner_streamable, pipe_func)`

Bases: [ReadStream](#)

Parameters

inner_streamable (`parzzley.fs.stream.ReadStreamable`)

async read(*max_len*)

Read the next chunk of data from the stream and return it, or return *None* if the end of the stream has been reached.

Parameters

max_len – The maximum chunk length.

_abc_impl = <_abc._abc_data object>

abstractmethod `async pipe(data)`

Receive and process a chunk of data and return a processed chunk of data (return `None` to signal the end).

If a pipe just returns the original data (return `data`), this pipe would be a no-op.

Parameters

data (*bytes* | *None*) – The chunk of source data. This is `None` at the end of the stream.

Return type

bytes | *None*

async `_stream()`

Return a new stream.

This method is implemented by subclasses and only used internally. See `stream()`.

`_abc_impl = <_abc._abc_data object>`

class `FullContentPipe(*args, **kwargs)`

Bases: *Pipe*, *ABC*

An abstract pipe that operates on the full content of a stream as a single chunk of data.

This makes some transformation tasks a lot easier, but it requires to keep the full content in memory. It can only be used for streams that never exceed reasonable length limits.

async `pipe(data)`

Receive and process a chunk of data and return a processed chunk of data (return `None` to signal the end).

If a pipe just returns the original data (return `data`), this pipe would be a no-op.

Parameters

data – The chunk of source data. This is `None` at the end of the stream.

abstractmethod `async pipe_content(data)`

Receive and process the full content of a stream and return the processed content.

Parameters

data (*bytes*) – The stream's entire content.

Return type

bytes

`_abc_impl = <_abc._abc_data object>`

abstractmethod `set_source_streamable(source_streamable, *, for_site=None)`

Set the source streamable. This usually happens for the stream master site.

Parameters

- **source_streamable** (*ReadStreamable*) – The source streamable.
- **for_site** (*Site* | *SiteSetup* | *str* | *None*) – For which site? Default is the event's current site.

Return type

None

abstractmethod `add_stream_pipe(pipe, *, for_site=None)`

Add a pipe. Each pipe can apply changes to the stream while it is transferring from the source to the destinations.

Parameters

- **pipe** (*type*[[Pipe](#)]) – The pipe to add.
- **for_site** ([Site](#) | [SiteSetup](#) | *str* | *None*) – For which site? Default is the event's current site.

Return type

None

_abc_impl = *<_abc._abc_data object>*

class `parzzley.sync.aspect.events.item.stream.DetermineConflicts`

Bases: [_EventAfterMasterSiteDetermined](#), [_WithStreamCookie](#), [_WithSourceStreamableGetters](#), [ABC](#)

Occurs in order to find conflicts for the current item stream.

See [add_conflict\(\)](#) and [changed_dangerously_late\(\)](#).

This is part of the stream preparation, after the stream's master site is determined and source streamables are collected. If it detects any conflicts, there will be a chance to get them resolved by [TryResolveConflicts](#) and [ApplyConflictResolution](#). If there will be unresolved conflicts afterward, most of the further item processing gets skipped, and the next event will be [parzzley.sync.aspect.events.item.SkipDueToConflicts](#). Otherwise, the actual stream update will begin.

class `_Conflict`(*description: str, details: str | None*)

Bases: `object`

Parameters

- **description** (*str*)
- **details** (*str* | *None*)

description: *str*

details: *str* | *None*

abstract property conflicts: `Iterable[_Conflict]`

All conflicts found by this event so far.

abstractmethod `changed_dangerously_late`(*for_site=None*)

Returns whether this site has seen changes late enough to be potentially in conflict with something.

Precisely, it returns whether the content version number (see `content_version()`) of this site is at least as large as the one from the elected master site.

This always indicates a potential conflict, because there might be a change on the given site as well, which would be lost once the master site actually updates all the other sites.

Parameters

- **for_site** ([Site](#) | [SiteSetup](#) | *str* | *None*) – For which site? Default is the event's current site. For the master site, it will by nature always return `True`.

Return type

bool

abstractmethod `add_conflict`(*description, details=None*)

Add a conflict.

Parameters

- **description** (*str*) – The conflict description.
- **details** (*str* | *None*) – Conflict details.

Return type

None

_abc_impl = <_abc._abc_data object>**class** parzzley.sync.aspect.events.item.stream.TryResolveConflictsBases: [_EventAfterMasterSiteDetermined](#), ABCOccurs in order to try to resolve conflicts found by [DetermineConflicts](#).See [conflicts\(\)](#) and [resolve_conflicts\(\)](#).**abstractmethod** [resolve_conflicts](#)(*master_site*)

Resolve all conflicts.

Parameters**master_site** ([Site](#) | [SiteSetup](#) | *str*) – The new stream master site.**Return type**

None

abstract property [conflicts](#): [Iterable](#)[[_Conflict](#)]

All conflicts.

_abc_impl = <_abc._abc_data object>**class** parzzley.sync.aspect.events.item.stream.ApplyConflictResolutionBases: [_EventAfterMasterSiteDetermined](#), [_WithMasterSiteTableSetter](#), ABCOccurs in order to apply the resolution determined by [TryResolveConflicts](#).

This event should solely be handled in Parzzley infrastructure aspects, not in custom ones!

See [conflicts\(\)](#) and [resolve_conflicts\(\)](#).The typical implementation applies the resolution from [TryResolveConflicts](#).**abstract property** [conflict_resolution](#): *str* | None

The name of the new stream master site if the conflicts were resolved successfully, otherwise None.

abstract property [conflicts](#): [Iterable](#)[[_Conflict](#)]

All conflicts.

abstractmethod [remove_conflict](#)(*conflict*)

Remove a conflict.

Parameters**conflict** ([_Conflict](#)) – The conflict to remove.**Return type**

None

_abc_impl = <_abc._abc_data object>**class** parzzley.sync.aspect.events.item.stream.CollectDestinationStreamablesBases: [_EventAfterMasterSiteDetermined](#), [_WithStreamCookie](#), [_WithDestinationStreamableGetters](#), [_WithWorkingItems](#), ABC

Occurs in order to collect write-streamable objects for the current item stream.

See [add_destination_streamable\(\)](#) and [working_items\(\)](#).It occurs as step of the stream update procedure, after [parzzley.sync.aspect.events.item.SetupWorkingItems](#) was triggered and before [TransferUpdate](#).

A typical implementation would, for the main stream, check if the current site would actually need an update from the master site (i.e. its main stream has actually been changed). If not, it would do nothing. If so, it would at first make sure that either a temporary working item is in place for this item on this site (see [parzzley.sync.aspect.events.item.SetUpWorkingItems](#)), or the original location is at least part of all working items. Then it just adds one destination streamable per working items. For the other streams, it would do nearly the same, but if there is a temporary working item, it will definitely pass its check and do its setup, even if the stream content did not change (the temporary working item will later replace the original, so every stream must be transferred to it).

abstractmethod `add_destination_streamable(destination_streamable)`

Add a destination streamable. This usually happens for each site but the stream master site.

Parameters

destination_streamable ([WriteStreamable](#)) – The destination streamable to add.

Return type

None

abstractmethod `disable_change_detection()`

Disable the change detection (that would do a retry if the source changes during transfer).

Return type

None

`_abc_impl = <_abc._abc_data object>`

class `parzzley.sync.aspect.events.item.stream.TransferUpdate`

Bases: [_EventAfterMasterSiteDetermined](#), [_WithStreamCookie](#), [_WithSourceStreamableGetters](#), [_WithDestinationStreamableGetters](#), [ABC](#)

Occurs in order to update the current item stream.

Event handlers should not make visible changes here but keep it back until [parzzley.sync.aspect.events.item.ApplyUpdate](#). You also have to set the new item info there!

It is the main part of the stream update routine, happening after [CollectDestinationStreamables](#) and before [parzzley.sync.aspect.events.item.ApplyUpdate](#) (and before per-directory processing happens if the current item is a directory).

abstractmethod `is_change_detection_disabled()`

Return whether the change detection (that would do a retry if the source changes during transfer) was disabled.

Return type

bool

`_abc_impl = <_abc._abc_data object>`

parzzley.sync.aspect.events.sync_run package

Sync run events. See also [parzzley.sync.aspect.events](#).

These events usually occur (at most) once for each sync run.

class `parzzley.sync.aspect.events.sync_run._Event`

Bases: [_Event](#), [ABC](#)

Base class for an aspect event on an entire [parzzley.sync.run.SyncRun](#).

abstract property log: [Logger](#)

The logger.

abstract property sync_run: [SyncRun](#)

The current sync run.

abstract property all_sites: [Iterable\[Site\]](#)

All sites involved in the current sync run.

abstractmethod mark_effective()

Mark the current sync run as effective.

This should be called whenever it had any (potential) effect. The client can e.g. check if it makes sense to sync other sites as well (because they would get these changes then as well).

Return type

None

_abc_impl = <_abc._abc_data object>

class parzzley.sync.aspect.events.sync_run.**Prepare**

Bases: [_Event](#), ABC

Occurs at the beginning of a sync run, before [DetermineStreamSupport](#).

Aspects can listen to that event in order to do arbitrary initialization steps, e.g. for data structures used in later events.

abstractmethod async remove_site_from_items_books(site)

Remove a site from the items books, so it will be considered as never seen before.

Do not use. It is typically used by infrastructure.

Parameters

site ([Site](#) / [SiteSetup](#) / *str*)

Return type

None

_abc_impl = <_abc._abc_data object>

class parzzley.sync.aspect.events.sync_run.**DetermineStreamSupport**

Bases: [_Event](#), ABC

Occurs in order to determine which item streams are supported (and for which item types) for the current site.

See [set_stream_support\(\)](#).

It occurs at the beginning of a sync run, after [Prepare](#). There will be [ValidateStreamSupport](#) occurring afterward, in order to turn the per-site stream support info into a general one (all later steps will only use the general one).

abstractmethod set_stream_support(stream_name, item_types, *, for_site=None, cookies_are_move_stable=False)

Declare that this site supports the given stream for the given item types.

Parameters

- **stream_name** (*str*) – The name of the supported stream.
- **item_types** ([Iterable\[ItemType\]](#)) – The exact list of item types supported for this stream.

- **for_site**(*Site* / *SiteSetup* / *str* / *None*) – For which site? Default is the event’s current site.
- **cookies_are_move_stable**(*bool*) – Whether the cookie stays the same when an item gets moved.

Return type

None

_abc_impl = <_abc._abc_data object>

class parzzley.sync.aspect.events.sync_run.ValidateStreamSupport

Bases: *_Event*, ABC

Occurs in order to check if the stream support determined by *DetermineStreamSupport* is valid and to derive global stream support info from the per-site stream support info determined there.

This event should solely be handled in Parzzley infrastructure aspects, not in custom ones!

Handlers raise an exception if there are problems with the determined stream support, e.g. if there are fatal conflicts between the sites.

See *determined_supported_streams()*, *supported_item_types()* and *set_final_stream_support()*.

It occurs at the beginning of a sync run, after *DetermineStreamSupport* and before the (per-item; recursive) processing of the root directory.

parzzley.builtin.aspects.streaming.GlobalStreamSupport is the typical implementation. It checks if all sites have determined the same supported streams (with the same item types). It will then take this as the global result or will fail otherwise.

abstractmethod **determined_supported_streams**(***, *for_site=None*)

Return the streams supported by the current site, as determined by *DetermineStreamSupport*.

Parameters

for_site(*Site* / *SiteSetup* / *str* / *None*) – For which site? Default is the event’s current site.

Return type

Iterable[*str*]

abstractmethod **supported_item_types**(*stream_name*, ***, *for_site=None*)

Return the item types supported by the current site for the given stream, as determined by *DetermineStreamSupport*.

Parameters

- **stream_name**(*str*) – The name of the supported stream.
- **for_site**(*Site* / *SiteSetup* / *str* / *None*) – For which site? Default is the event’s current site.

Return type

Iterable[*ItemType*]

abstractmethod **cookies_are_move_stable**(*stream_name*, ***, *for_site=None*)

Return whether cookies are move-stable on the current site for the given stream, as determined by *DetermineStreamSupport*.

Parameters

- **stream_name**(*str*) – The name of the supported stream.

- **for_site** (*Site* / *SiteSetup* / *str* / *None*) – For which site? Default is the event's current site.

Return type

bool

abstractmethod set_final_stream_support (*stream_name*, *item_types*, *cookies_are_move_stable*)

Declare that the current sites support the given stream for the given item types.

Parameters

- **stream_name** (*str*) – The name of the supported stream.
- **item_types** (*Iterable[ItemType]*) – The exact list of item types supported for this stream.
- **cookies_are_move_stable** (*bool*) – Whether the cookie stays the same when an item gets moved.

Return type

None

_abc_impl = <_abc._abc_data object>

class parzzley.sync.aspect.events.sync_run._EventAfterStreamSupportDetermined

Bases: *_Event*, ABC

Base class for an aspect event after the stream support has been determined.

class StreamSupportInfo

Bases: ABC

Information about the stream support.

abstract property supported_streams: *Sequence[str]*

Names of all supported streams.

abstractmethod supported_item_types (*stream_name*)

Return the item types supported by the given stream.

Parameters

stream_name (*str*) – The name of the supported stream.

Return type

Sequence[ItemType]

abstractmethod cookies_are_move_stable (*stream_name*)

Return whether the cookies for the given stream are move-stable.

Parameters

stream_name (*str*) – The name of the supported stream.

Return type

bool

_abc_impl = <_abc._abc_data object>

abstract property stream_support_info: *StreamSupportInfo*

Information about the stream support.

Determined by *ValidateStreamSupport*.

_abc_impl = <_abc._abc_data object>

class parzzley.sync.aspect.events.sync_run.Finish

Bases: [_Event](#), ABC

Occurs at the end of a successful sync run, after the processing of the root directory, before [Close](#).

_abc_impl = <_abc._abc_data object>

class parzzley.sync.aspect.events.sync_run.Close

Bases: [_Event](#), ABC

Occurs at the end of a sync run (after [Finish](#)), no matter if successful or not.

abstract property error: [Exception](#) | None

Critical error that aborted the sync run prematurely (or None).

_abc_impl = <_abc._abc_data object>

Submodules

parzzley.sync.aspect.only_if module

Aspect event execution conditions.

See [Condition](#).

class parzzley.sync.aspect.only_if.Condition

Bases: ABC

Base class for aspect event execution conditions.

These conditions can be applied to an aspect event handler in order to skip it in some situations (with less code than explicit checks in the event handler body).

abstractmethod is_met(*event*)

Return whether this condition is met (i.e. the event handler is not to be skipped by this condition) in a given situation.

Parameters

event – The event that carries the given situation.

Return type

bool

_abc_impl = <_abc._abc_data object>

class parzzley.sync.aspect.only_if._ItemExistsCondition(*value=True*)

Bases: [Condition](#), ABC

Base class for conditions that check whether the item exists on some site.

Parameters

value (*bool*) – Whether to apply the check in a positive way. Set to False to negate the condition.

is_met(*event*)

Return whether this condition is met (i.e. the event handler is not to be skipped by this condition) in a given situation.

Parameters

event – The event that carries the given situation.

Return type

bool

abstractmethod `_site(event)`

The site to check by this condition.

Parameters**event** – The event that carries the given situation.`_abc_impl = <_abc._abc_data object>`**class** `parzzley.sync.aspect.only_if._ItemIsTypeCondition(*types, no=None)`Bases: [Condition](#), ABC

Base class for conditions that check against the item type on some site.

Parameters

- **types** ([ItemType](#)) – The allowed types.
- **no** ([Iterable](#)[[ItemType](#)] | [None](#)) – Alternatively to **types**, the non-allowed types.

is_met(event)

Return whether this condition is met (i.e. the event handler is not to be skipped by this condition) in a given situation.

Parameters**event** – The event that carries the given situation.**abstractmethod** `_site(event)`

The site to check by this condition.

Parameters**event** – The event that carries the given situation.`_abc_impl = <_abc._abc_data object>`**class** `parzzley.sync.aspect.only_if._IsItemTypeSupportedByStreamCondition(value=True)`Bases: [Condition](#), ABC

Base class for conditions that check whether the item type on some site is supported by the current stream.

Parameters**value** ([bool](#)) – Whether to apply the check in a positive way. Set to [False](#) to negate the condition.**is_met(event)**

Return whether this condition is met (i.e. the event handler is not to be skipped by this condition) in a given situation.

Parameters**event** – The event that carries the given situation.**abstractmethod** `_site(event)`

The site to check by this condition.

Parameters**event** – The event that carries the given situation.`_abc_impl = <_abc._abc_data object>`

```
class parzzley.sync.aspect.only_if.ItemExistsHere(value=True)
```

Bases: `_ItemExistsCondition`

Condition that checks whether the item exists on the current site.

Parameters

value (*bool*) – Whether to apply the check in a positive way. Set to `False` to negate the condition.

_site(*event*)

The site to check by this condition.

Parameters

event – The event that carries the given situation.

_abc_impl = `<_abc._abc_data object>`

```
class parzzley.sync.aspect.only_if.ItemHereIsType(*types, no=None)
```

Bases: `_ItemIsTypeCondition`

Condition that checks against the item type on the current site.

Parameters

- **types** (*ItemType*) – The allowed types.
- **no** (*Iterable[ItemType] | None*) – Alternatively to `types`, the non-allowed types.

_site(*event*)

The site to check by this condition.

Parameters

event – The event that carries the given situation.

_abc_impl = `<_abc._abc_data object>`

```
class parzzley.sync.aspect.only_if.ItemExistsOnMasterSite(value=True, of_stream=None)
```

Bases: `_ItemExistsCondition`

Condition that check whether the item exists on the master site (of the current stream or another one).

Parameters

- **value** (*bool*) – Whether to apply the check in a positive way. Set to `False` to negate the condition.
- **of_stream** (*str | None*) – Which stream to check (default: the current one).

_site(*event*)

The site to check by this condition.

Parameters

event – The event that carries the given situation.

_abc_impl = `<_abc._abc_data object>`

```
class parzzley.sync.aspect.only_if.ItemOnMasterSiteIsType(*types, no=None, of_stream=None)
```

Bases: `_ItemIsTypeCondition`

Condition that checks against the item type on the master site (of the current stream or another one).

Parameters

- **types** (*ItemType*) – The allowed types.

- **no** (*Iterable[ItemType]* / *None*) – Alternatively to *types*, the non-allowed types.
- **of_stream** (*str* / *None*) – Which stream to check (default: the current one).

_site(*event*)

The site to check by this condition.

Parameters

event – The event that carries the given situation.

_abc_impl = <**_abc._abc_data** object>

class parzzley.sync.aspect.only_if.**ThisIsMasterSite**(*value=True*, *, *of_stream=None*)

Bases: *Condition*

Condition that checks whether the current site is the master site (of the current stream or another one).

Parameters

- **value** (*bool*) – Whether to apply the check in a positive way. Set to *False* to negate the condition.
- **of_stream** (*str* / *None*) – Which stream to check (default: the current one).

is_met(*event*)

Return whether this condition is met (i.e. the event handler is not to be skipped by this condition) in a given situation.

Parameters

event – The event that carries the given situation.

_abc_impl = <**_abc._abc_data** object>

class parzzley.sync.aspect.only_if.**IsStream**(*stream_name*)

Bases: *Condition*

Condition that checks against the current stream name.

Parameters

stream_name (*str*) – The stream name.

is_met(*event*)

Return whether this condition is met (i.e. the event handler is not to be skipped by this condition) in a given situation.

Parameters

event – The event that carries the given situation.

_abc_impl = <**_abc._abc_data** object>

class parzzley.sync.aspect.only_if.**IsItemTypeHereSupportedByStream**(*value=True*)

Bases: *_IsItemTypeSupportedByStreamCondition*

Condition that checks whether the item type on the current site is supported by the current stream.

Parameters

value (*bool*) – Whether to apply the check in a positive way. Set to *False* to negate the condition.

_site(*event*)

The site to check by this condition.

Parameters

event – The event that carries the given situation.

_abc_impl = <_abc._abc_data object>

class parzzley.sync.aspect.only_if.IsItemTypeOnStreamMasterSiteSupportedByStream(*value=True*)

Bases: *_IsItemTypeSupportedByStreamCondition*

Condition that checks whether the item type on the master site (of the current stream) is supported by the current stream.

Parameters

value (*bool*) – Whether to apply the check in a positive way. Set to *False* to negate the condition.

_site(*event*)

The site to check by this condition.

Parameters

event – The event that carries the given situation.

_abc_impl = <_abc._abc_data object>

parzzley.sync.engine package

The Parzzley engine. See [Engine](#). Used internally by other parts of the API ([parzzley.sync](#)).

Subpackages are used internally by the engine.

class parzzley.sync.engine.Engine(*, *sync_run, sites, loggings*)

Bases: *object*

The Parzzley engine executes a sync run on a volume.

At first, it collects all the aspects that are defined for each site or for the entire volume (i.e. for all sites). Then it runs the volume synchronization by triggering a sequence of events (see [parzzley.sync.aspect.events](#)). The engine itself has no own logic beyond that. All the actual work is done by the event handlers provided by the defined aspects.

Do not use directly.

Parameters

- **sync_run** ([parzzley.sync.run.SyncRun](#)) – The sync run.
- **sites** (*dict[Site, AsyncContextManager[Site | None, bool | None]]*) – The connected sites.
- **loggings** (*Iterable[Logging]*)

property sync_control: [SyncControl](#)

The sync control.

start()

Start the engine.

This method must not be called more than once per instance.

Return type

None

async _exec()

Return type

None

async __exec__post(sync_run_event)

Return type

None

async __sync(sync_run_event)

Return type

None

async __sync__post(sync_run_event, sync_run_label)

async __sync_item(item, sync_run_event)

Parameters

item ([Item](#))

Return type

None

async __sync_item__collect_info_and_determine_master_sites_for_streams(item_event)

Parameters

item_event ([ItemEvent](#))

Return type

None

async __sync_item__source_streamables(item_event)

Parameters

item_event ([ItemEvent](#))

Return type

None

async __sync_item__conflicts(item_event)

Parameters

item_event ([ItemEvent](#))

Return type

None

async __sync_item__skipped_by_conflicts(item_event)

Parameters

item_event ([ItemEvent](#))

Return type

None

async __sync_item__update_streams(item_event)

Parameters

item_event ([ItemEvent](#))

Return type

None

async `__sync_item__directory(item_event)`

Return type

None

async `__sync_item__populate_items_book(item_event)`

Parameters

item_event ([ItemEvent](#))

Return type

None

async `__prepare_items_book()`

Return type

None

`__fall_back_items_book()`

Return type

None

async `__store_items_book()`

Return type

None

`__items_books_variable()`

`__catch_exceptions(sync_run_event, crash_reason)`

Parameters

crash_reason (*str*)

`__check_canceled()`

Return type

None

`_cancel()`

Return type

None

exception `_Canceled`

Bases: `BaseException`

class `_SyncControl(sync_run_id)`

Bases: [SyncControl](#)

Parameters

sync_run_id (*str*)

set_finished(*, *was_successful*, *was_effective*)

Mark the sync run associated to this control as finished.

Parameters

- **was_successful** (*bool*) – Whether the sync run was successful.
- **was_effective** (*bool*) – Whether the sync run was effective (see [was_effective](#)).

property sync_run_id

This sync run's identifier (always a new unique one for each run!).

property is_finished

Whether the associated sync run is already finished (either successfully or not).

property was_successful

Whether the associated sync run is already finished and was successful.

For a finished but unsuccessful run, there is at least one critical alert (see [alerts](#)).

property was_effective

Whether the associated sync run had any effects, i.e. synced any changes.

Note: This is only a hint. You must not rely on it for any critical decisions!

wait_finished()

Wait until the associated sync run is finished. When this function returns, [is_finished](#) is set.

_abc_impl = <_abc._abc_data object>

class parzzley.sync.engine._ItemsBookBase(*after_last_sync_item_data*)

Bases: ABC, Generic

Parameters

after_last_sync_item_data (*dict[bytes, ItemInfoT]*)

classmethod [from_serializable_data](#)(*data, *args*)

Return an items book from serializable data. See also [to_serializable_data\(\)](#).

Parameters

- **data** (*Iterable | None*) – Serializable items book data.
- **args** – Additional constructor arguments.

Return type

Self

to_serializable_data()

Return serializable data for this items book. See also [from_serializable_data\(\)](#).

Return type

Sequence[Any]

abstractmethod classmethod [_item_info_from_serializable_data](#)(*data*)

Return an item info for the given serializable data (as returned by [_item_info_to_serializable_data\(\)](#)).

Parameters

data (*Sequence[Any]*) – Serializable data.

Return type

ItemInfoT

abstractmethod classmethod [_item_info_to_serializable_data](#)(*item_info*)

Return serializable data for a given item info (as interpreted by [_item_info_from_serializable_data\(\)](#)).

Parameters

item_info (*ItemInfoT*) – The item info.

Return type*Iterable[Any]***classmethod** `_state_args_from_serializable_data(data)`

Return additional state arguments for the given serializable data (as returned by `_state_args_to_serializable_data()`).

Parameters**data** (*Sequence[Any]*) – Serializable data.**Return type***Iterable[Any]***_state_args_to_serializable_data()**

Return serializable data for additional state arguments (as interpreted by `_state_args_from_serializable_data()`).

Return type*Any***take_last_item_book_entries_as_fallback()**

Take the item book entries from last time for any item that has not got a new info so far.

This method is to be used when a sync run got aborted for some reason, so we do not lose information for not seen items.

Return type*None***_abc_impl** = `<_abc._abc_data object>`

```
class parzzley.sync.engine._GlobalItemsBook(after_last_sync_item_data, sync_run_no,
                                             connected_site_names,
                                             connected_site_names_by_sync_run_no)
```

Bases: `_ItemsBookBase[_GlobalItemsBook._ItemInfo]`

Parameters

- **after_last_sync_item_data** (*dict[str, _GlobalItemsBook._ItemInfo]*)
- **sync_run_no** (*int*)
- **connected_site_names** (*Iterable[str]*)
- **connected_site_names_by_sync_run_no** (*dict[int, Sequence[str]]*)

mark_successfully_synced(item)

Mark the given item as successfully synchronized in this sync run.

Parameters**item** (*bytes* / *Item*) – The item to mark.**Return type***None***mark_skipped_now(item)**

Mark the given item as skipped now.

Parameters**item** (*bytes* / *Item*) – The item to mark.**Return type***None*

last_successful_sync_no(*item*)

Return the serial number of the sync run that has successfully synced the given item last time (or None if it was not successfully synced yet).

Parameters

item (*bytes* / *Item*) – The item.

Return type

int | None

sites_involved_in_sync_no(*sync_run_no*)

Return the names of the sites that were involved in a given sync run.

This must be one of the serial numbers returned by [last_successful_sync_no\(\)](#).

Parameters

sync_run_no (*int*) – The serial number of the sync run.

Return type

Sequence[str]

classmethod _item_info_to_serializable_data(*item_info*)

Return serializable data for a given item info (as interpreted by [_item_info_from_serializable_data\(\)](#)).

Parameters

item_info – The item info.

classmethod _item_info_from_serializable_data(*data*)

Return an item info for the given serializable data (as returned by [_item_info_to_serializable_data\(\)](#)).

Parameters

data – Serializable data.

_state_args_to_serializable_data()

Return serializable data for additional state arguments (as interpreted by [_state_args_from_serializable_data\(\)](#)).

classmethod _state_args_from_serializable_data(*data*)

Return additional state arguments for the given serializable data (as returned by [_state_args_to_serializable_data\(\)](#)).

Parameters

data – Serializable data.

class _ItemInfo(*last_successful_sync_run_no: int*)

Bases: object

Parameters

last_successful_sync_run_no (*int*)

last_successful_sync_run_no: int

_abc_impl = <_abc._abc_data object>

class parzzley.sync.engine._PerSiteItemsBook(*after_last_sync_item_data, sync_run_no*)

Bases: [_ItemsBookBase](#)[[_PerSiteItemsBook._ItemInfo](#)]

A per-site items book stores site-specific state info about each item.

Data gets stored by means of `store_item_info()` as soon as a sync run has finished an item (actual persistence will only happen at the end of the sync run, though).

After that, you can look up an item's last type and last stream cookies for each site (assuming there is one items book per site), and also the serial number of the sync run that has done the last successful sync on it.

Parameters

- `after_last_sync_item_data` (`dict[str, _PerSiteItemsBook._ItemInfo]`)
- `sync_run_no` (`int`)

`item_type_after_last_sync(item)`

Return the item type that a given item had lastly.

Parameters

`item` (`bytes` / `Item`) – The item location.

Return type

`ItemType`

`stream_cookie_after_last_sync(item, stream_name)`

Return the cookie that a given stream of a given item had lastly.

Parameters

- `item` (`bytes` / `Item`) – The item location.
- `stream_name` (`str`) – The stream name.

Return type

`Any`

`last_change_run_sn(item)`

Return the serial number of the sync run that has seen the last change on this site (or `0` if that never happened).

Parameters

`item` (`bytes` / `Item`) – The item location.

Return type

`int`

`store_item_info(item, item_type, stream_cookies)`

Store item info.

Parameters

- `item` (`bytes` / `Item`) – The item location.
- `item_type` (`ItemType`) – The item type.
- `stream_cookies` (`dict[str, object]`) – The stream cookie for each stream.

Return type

`None`

`classmethod _item_info_to_serializable_data(item_info)`

Return serializable data for a given item info (as interpreted by `_item_info_from_serializable_data()`).

Parameters

`item_info` – The item info.

classmethod `_item_info_from_serializable_data(data)`

Return an item info for the given serializable data (as returned by `_item_info_to_serializable_data()`).

Parameters

data – Serializable data.

class `_ItemInfo(changed_in_sync_run_no: int, item_type: 'parzzley.fs.ItemType', streams: dict[str, object])`

Bases: object

Parameters

- **changed_in_sync_run_no** (*int*)
- **item_type** (*ItemType*)
- **streams** (*dict[str, object]*)

changed_in_sync_run_no: *int*

item_type: *ItemType*

streams: *dict[str, object]*

`_abc_impl = <_abc._abc_data object>`

Submodules

parzzley.sync.engine.event_runner module

Event runner. See [Runner](#).

class `parzzley.sync.engine.event_runner.Runner(aspects)`

Bases: object

Event runner.

This is a helper class used internally by the engine in order to run events (in particular, to execute event handlers registered by the configured aspects).

Parameters

aspects (*dict[parzzley.fs.Site, Iterable[parzzley.sync.aspect.Aspect]]*)

class `_DependencyControlledHandlerCollection(handler_nodes)`

Bases: object

Collection of event handler nodes and some operations on it that allow to execute them in an order that obeys their dependencies.

Parameters

handler_nodes (*list[Runner._HandlerNode]*) – The event handler nodes.

property is_empty: *bool*

Whether this collection is empty.

available_nodes()

Return all handler nodes that are available for execution and not marked as done yet.

Return type

Iterable[_Handler]

mark_done(*handlers)

Mark some handler nodes as done.

Parameters

handlers ([_Handler](#)) – The handler nodes.

Return type

None

```
class \_Handler(func: Callable[[ForwardRef('parzzley.sync.aspect.events._Event')], NoneType], site:
    'parzzley.fs.Site', conditions:
    Iterable[ForwardRef('parzzley.sync.aspect.only_if.Condition')], event_type:
    type['parzzley.sync.aspect.events._Event'], names: Iterable[str], beforehand: Iterable[str],
    beforehand_optional: Iterable[str], afterwards: Iterable[str], afterwards_optional:
    Iterable[str])
```

Bases: object

Parameters

- **func** (Callable[[[_Event](#)], None])
- **site** ([Site](#))
- **conditions** (Iterable[[Condition](#)])
- **event_type** (type[[_Event](#)])
- **names** (Iterable[str])
- **beforehand** (Iterable[str])
- **beforehand_optional** (Iterable[str])
- **afterwards** (Iterable[str])
- **afterwards_optional** (Iterable[str])

func: Callable[[[_Event](#)], None]

site: [Site](#)

conditions: Iterable[[Condition](#)]

event_type: type[[_Event](#)]

names: Iterable[str]

beforehand: Iterable[str]

beforehand_optional: Iterable[str]

afterwards: Iterable[str]

afterwards_optional: Iterable[str]

```
class \_HandlerNode(handler: 'Runner._Handler', depends_on_handlers: list['Runner._HandlerNode'])
```

Bases: object

Parameters

- **handler** ([_Handler](#))
- **depends_on_handlers** (list[[_HandlerNode](#)])

handler: [_Handler](#)

depends_on_handlers: `list[_HandlerNode]`

async run_event(*event_type*, *event*)

Run an event.

Parameters

- **event_type** (`type[_Event]`) – The event type.
- **event** (`_Event`) – The event instance.

Return type

None

__handlers_for_event_type(*event_type*)

Parameters

event_type (`type[_Event]`)

Return type

`_DependencyControlledHandlerCollection`

__all_handler_nodes()

Return type

`Sequence[_HandlerNode]`

__all_handler_nodes__find_dependencies(*handler_node*, *handler_nodes_by_name*)

Return type

None

parzzley.sync.engine.events_impl module

Implementation of `parzzley.sync.aspect.events` events, used by the engine.

Note: Instead of creating new event instances for each event type (and constantly moving data around), the engine works with one instance for the entire sync run, and only sometimes derives specific event objects from that, in a much coarser way than the events are originally defined. This simplifies engine implementation to a large degree.

`parzzley.sync.engine.events_impl.SyncRunEvent`(*sync_run*, *items_books*)

Create and return a new sync run event for a sync run.

Parameters

- **sync_run** – The sync run to associate to this event.
- **items_books** – The items books used in this sync run.

class `parzzley.sync.engine.events_impl.SyncRunEvent`(*site*, *sync_run*, *per_sync_run_data*, *items_books*, *logger=None*)

Bases: `DetermineStreamSupport`, `ValidateStreamSupport`, `Close`, `Finish`, `Prepare`

Implementation of all sync run events.

Parameters

- **site** (`parzzley.fs.Site` / `None`)
- **sync_run** (`parzzley.sync.run.SyncRun`)
- **logger** (`_Logger` / `None`)

`_DATA__STREAM_SUPPORT_PER_SITE = <parzzley.sync.aspect.events.Data object>`

`_DATA__STREAM_SUPPORT = <parzzley.sync.aspect.events.Data object>`

`_DATA__LOG_ENTRIES = <parzzley.sync.aspect.events.Data object>`

`_DATA__EFFECTIVE = <parzzley.sync.aspect.events.Data object>`

property `all_log_entries`

property `log`

The logger.

to_item_event(*item*)

to_event_for_site(*site*)

Parameters

site ([Site](#))

Return type

Self

property `site`

The current site.

This is the site which this aspect is associated with, so it usually operates on this site.

property `sync_run`

The current sync run.

_get_data(*data_obj*, *per_site*, *per_item*, *per_stream*, *key_data*, *initial_value*)

_set_data(*data_obj*, *per_site*, *per_item*, *per_stream*, *key_data*, *value*)

__event_data_wrapper(*key*, *initial_value*=None, *, *per_site*=False, *per_item*=False, *per_stream*=False)

__data_key_data(*key_data*)

_data_key_data()

_data__for_site(*for_site*)

property `_per_sync_run_data`

property `_items_books`

property `all_sites`

All sites involved in the current sync run.

mark_effective()

Mark the current sync run as effective.

This should be called whenever it had any (potential) effect. The client can e.g. check if it makes sense to sync other sites as well (because they would get these changes then as well).

property `was_effective`: `bool`

property `error`

Critical error that aborted the sync run prematurely (or None).

set_error(*error*)

Parameters

error ([Exception](#))

async remove_site_from_items_books(*site*)

Remove a site from the items books, so it will be considered as never seen before.

Do not use. It is typically used by infrastructure.

set_stream_support(*stream_name*, *item_types*, *, *for_site*=None, *cookies_are_move_stable*=False)

Declare that this site supports the given stream for the given item types.

Parameters

- **stream_name** – The name of the supported stream.
- **item_types** – The exact list of item types supported for this stream.
- **for_site** – For which site? Default is the event's current site.
- **cookies_are_move_stable** – Whether the cookie stays the same when an item gets moved.

__stream_support_per_site(*for_site*)

_stream_support()

property stream_support_info

determined_supported_streams(*, *for_site*=None)

Return the streams supported by the current site, as determined by `DetermineStreamSupport`.

Parameters

for_site – For which site? Default is the event's current site.

supported_item_types(*stream_name*, *, *for_site*=None)

Return the item types supported by the current site for the given stream, as determined by `DetermineStreamSupport`.

Parameters

- **stream_name** – The name of the supported stream.
- **for_site** – For which site? Default is the event's current site.

cookies_are_move_stable(*stream_name*, *, *for_site*=None)

Return whether cookies are move-stable on the current site for the given stream, as determined by `DetermineStreamSupport`.

Parameters

- **stream_name** – The name of the supported stream.
- **for_site** – For which site? Default is the event's current site.

set_final_stream_support(*stream_name*, *item_types*, *cookies_are_move_stable*)

Declare that the current sites support the given stream for the given item types.

Parameters

- **stream_name** – The name of the supported stream.
- **item_types** – The exact list of item types supported for this stream.

- **cookies_are_move_stable** – Whether the cookie stays the same when an item gets moved.

```
class _EventData(name, initial_value, native_data_name, scope)
```

Bases: `object`

```
get(event, key_data)
```

```
set(event, key_data, value)
```

```
_EventData__key(key_data)
```

```
_EventData__native_data(event)
```

```
class _StreamSupportInfo(info_dict)
```

Bases: `StreamSupportInfo`

```
property supported_streams
```

Names of all supported streams.

```
cookies_are_move_stable(stream_name)
```

Return whether the cookies for the given stream are move-stable.

Parameters

stream_name – The name of the supported stream.

```
supported_item_types(stream_name)
```

Return the item types supported by the given stream.

Parameters

stream_name – The name of the supported stream.

```
_abc_impl = <_abc._abc_data object>
```

```
_abc_impl = <_abc._abc_data object>
```

```
class parzzley.sync.engine.events_impl.ItemEvent(site, sync_run, per_sync_run_data, items_books,  
item, per_item_data, logger=None)
```

Bases: `SyncRunEvent`, `DecideToSkip`, `Prepare`, `DetermineType`, `RefreshItemsBook`, `ApplyUpdate`, `PrepareUpdating`, `SetUpWorkingItems`, `SkipDueToConflicts`

Implementation of all item events.

Parameters

- **item** (`parzzley.fs.Item`)

- **logger** (`_Logger` / `None`)

```
_DATA__ALL_CONFLICTS = <parzzley.sync.aspect.events.Data object>
```

```
_DATA__IS_MARKED_FOR_SKIP = <parzzley.sync.aspect.events.Data object>
```

```
_DATA__IS_MARKED_FOR_FULL_RETRY = <parzzley.sync.aspect.events.Data object>
```

```
_DATA__ITEM_TYPE = <parzzley.sync.aspect.events.Data object>
```

```
_DATA__NEW_ITEM_INFO = <parzzley.sync.aspect.events.Data object>
```

```
_DATA__PER_STREAM_DATAS = <parzzley.sync.aspect.events.Data object>
```

```
_DATA__WORKING_ITEMS = <parzzley.sync.aspect.events.Data object>
```


to_dir_event()

to_stream_event(*stream_name*)

property item

The current item.

property _per_item_data

stream_cookie(*for_site=None*)

Return the stream cookie of the main stream, usually determined in *parzzley.sync.aspect.events.item.stream.DetermineCookie*.

For details about cookies, see *parzzley.fs.Site.cookie()*.

This is the same as *stream_cookie* of stream events related to the main stream (i.e. "").

Parameters

for_site – For which site? Default is the event's current site.

property all_conflicts

Conflicts.

set_all_conflicts(*all_conflicts*)

update_cookies(*stream_cookies*, *, *for_site=None*)

Set new cookies as they are after the update for this site.

If the given dictionary contains only some of the supported stream names, the other stream cookies will stay untouched (like *dict.update()*)!

Parameters

- **stream_cookies** – The new stream cookies.
- **for_site** – For which site? Default is the event's current site.

streams_with_destination_streamables(*for_site=None*)

Return the list of stream names for which there are any destination streamables on this site or any other one.

Parameters

for_site – For which site? Default is the event's current site.

add_working_item(*working_item*, *, *for_site=None*)

Add a working item.

Parameters

- **working_item** – The working item.
- **for_site** – For which site? Default is the event's current site.

remove_working_item(*working_item*, *, *for_site=None*)

Remove a working item.

Parameters

- **working_item** – The working item.
- **for_site** – For which site? Default is the event's current site.

working_items(*, *for_site=None*)

Return all working items. Those are either the ones added by [add_working_item\(\)](#) earlier, or, if none are added, it is a list containing only [item](#) (if it exists on item master and is not a directory).

Parameters

for_site – For which site? Default is the event’s current site.

set_item_type(*item_type*, *, *for_site=None*)

Declare the item type of the current item on this site.

Parameters

- **item_type** – The item type.
- **for_site** ([Site](#) / [SiteSetup](#) / *str* / *None*) – For which site? Default is the event’s current site.

property is_marked_full_retry_needed

Whether the current item is marked for a full retry (see [mark_full_retry_needed\(\)](#)).

mark_full_retry_needed()

Mark the current item for a full retry (e.g. because ongoing changes were detected during update).

property is_item_marked_to_skip

Whether the current item is marked for being skipped at least this time (i.e. by [skip_item\(\)](#)).

property is_item_marked_to_skip_permanently

Whether the current item is marked for being skipped permanently (i.e. by [skip_item\(\)](#)).

skip_item(*, *only_now=False*)

Mark the current item for being skipped.

Parameters

only_now – Whether to skip this item now, but keep its internal state data for later sync runs.

sites_involved_in_current_item_last_successful_sync()

Return the name of the sites that were involved in the last sync run which has successfully synced the current item (this is the last sync run if it was successful and has not skipped the current item), or *None* if it was not synced yet.

item_type_after_last_sync(*for_site=None*)

item_type(*for_site=None*)

Return the just determined type for the current item.

Parameters

for_site – For which site? Default is the event’s current site.

_master_site(*of_stream=None*)

Return the master site for the given stream (or the item’s main stream).

content_version(*for_site=None*, *, *only_past*)

Return the version number of the current item’s main stream, for this site or any other one.

This is the same as *content_version* of stream events related to the main stream (i.e. *""*). See there for more details.

Parameters

- **for_site** – For which site? Default is the event’s current site.

- **only_past** – Depending on this flag, this will either always be the sn of a sync run from the past, i.e. it will always be less than the current sync run's sn, or it returns the current sync run's sn if there were changes detected since the last sync run.

set_item_info(*item_type*, *stream_cookie*, *, *for_site*=None)

Set item info for this site or another.

Parameters

- **item_type** – The item type.
- **stream_cookie** – The stream cookie.
- **for_site** – The site to store the item info for (default: the current site).

updated_cookie(*stream_name*, *, *for_site*=None)

The updated cookie for a given stream that needs to be stored.

Parameters

- **stream_name** – The stream name.
- **for_site** – For which site? Default is the event's current site.

set_item_book_entry(*item_type*, *cookies*, *, *for_site*=None)

Write an entry to the item book. This info can be used in later sync runs.

Parameters

- **item_type** – The new item type to store.
- **cookies** – The new stream cookies to store.
- **for_site** – For which site? Default is the event's current site.

__working_items__list(*for_site*)

_abc_impl = <_abc._abc_data object>

class parzzley.sync.engine.events_impl.**DirEvent**(*site*, *sync_run*, *per_sync_run_data*, *items_books*, *item*, *per_item_data*, *logger*=None)

Bases: [ItemEvent](#), [Iterated](#), [List](#), [Prepare](#)

Implementation of all directory events.

Parameters

- **item** ([parzzley.fs.Item](#))
- **logger** ([_Logger](#) / None)

_DATA__CHILD_NAMES = <parzzley.sync.aspect.events.Data object>

property child_names: set[str]

add_child_names(*child_names*)

Add child names to the list of all children to be processed for this directory.

Parameters

- **child_names** – The names of the children to be added.

_abc_impl = <_abc._abc_data object>

```
class parzzley.sync.engine.events_impl.StreamEvent(site, sync_run, per_sync_run_data, items_books,
                                                    item, per_item_data, stream_name,
                                                    per_stream_data, logger=None)
```

Bases: *ItemEvent*, *DetermineComparator*, *DetermineCookie*, *DetermineConflicts*,
TryResolveConflicts, *ApplyConflictResolution*, *DetectChanges*, *DetermineMasterSiteTable*,
MasterSiteDetermined, *CollectDestinationStreamables*, *CollectSourceStreamables*,
TransferUpdate

Implementation of all stream events.

Parameters

- **item** (*parzzley.fs.Item*)
- **stream_name** (*str*)
- **logger** (*_Logger* / *None*)

```
_DATA__CHANGED_SITES = <parzzley.sync.aspect.events.Data object>
```

```
_DATA__CONFLICTS = <parzzley.sync.aspect.events.Data object>
```

```
_DATA__CONFLICT_RESOLUTION = <parzzley.sync.aspect.events.Data object>
```

```
_DATA__DESTINATION_STREAMABLES = <parzzley.sync.aspect.events.Data object>
```

```
_DATA__DISABLE_CHANGE_DETECTION = <parzzley.sync.aspect.events.Data object>
```

```
_DATA__REFRESH_NEEDED_SITES = <parzzley.sync.aspect.events.Data object>
```

```
_DATA__SOURCE_STREAMABLE = <parzzley.sync.aspect.events.Data object>
```

```
_DATA__STREAM_COOKIES = <parzzley.sync.aspect.events.Data object>
```

```
_DATA__STREAM_COMPARATOR = <parzzley.sync.aspect.events.Data object>
```

```
_DATA__STREAM_MASTER_SITE = <parzzley.sync.aspect.events.Data object>
```

```
_DATA__STREAM_MASTER_TABLE = <parzzley.sync.aspect.events.Data object>
```

```
_DATA__STREAM_PIPES = <parzzley.sync.aspect.events.Data object>
```

property stream_name

The current stream name.

property _per_stream_data

is_site_marked_as_changed(*for_site=None*)

Return whether this site was marked as changed (usually by `DetectChanges.mark_changed()`).

Parameters

for_site – For which site? Default is the event’s current site.

is_site_marked_as_needing_refreshed(*for_site=None*)

Return whether this site was marked as needing to get refreshed (usually by `DetectChanges.mark_refresh_needed()`).

Parameters

for_site – For which site? Default is the event’s current site.

source_streamable(*for_site=None*)

Return the source streamable for this site. It is used as the source of the stream update step.

Usually the source streamable is from the stream master site (this is the main reason to determine it before).

It was determined earlier in `CollectSourceStreamables`.

Parameters

for_site – For which site? Default is the event’s current site.

set_source_streamable(*source_streamable*, *, *for_site=None*)

Set the source streamable. This usually happens for the stream master site.

Parameters

- **source_streamable** – The source streamable.
- **for_site** – For which site? Default is the event’s current site.

destination_streamables()

Return all destination streamables. They are used as destinations of the stream update step.

Usually there is one destination streamable for each site that is not the stream master site. There is usually no destination streamable for the master site, as that is the streaming source instead.

add_destination_streamable(*destination_streamable*)

Add a destination streamable. This usually happens for each site but the stream master site.

Parameters

destination_streamable – The destination streamable to add.

disable_change_detection()

Disable the change detection (that would do a retry if the source changes during transfer).

is_change_detection_disabled()

Return whether the change detection (that would do a retry if the source changes during transfer) was disabled.

stream_pipes(*for_site=None*)

Return all stream pipes.

They are used by the stream update step as a mechanism to apply changes to the stream while it is transferring from the source to the destinations.

Parameters

for_site – For which site? Default is the event’s current site.

add_stream_pipe(*pipe*, *, *for_site=None*)

Add a pipe. Each pipe can apply changes to the stream while it is transferring from the source to the destinations.

Parameters

- **pipe** – The pipe to add.
- **for_site** – For which site? Default is the event’s current site.

mark_changed(*site=None*)

Mark this site as having changed. This might make this site the stream master site later, so the other sites get updated from it.

Parameters

site – For which site? Default is the event’s current site.

mark_all_last_involved_sites_changed_if_this_is_not_one_of_them(*site=None*)

Mark all lastly involved sites as having changed (see [mark_changed\(\)](#)) if the given site is not one of them.

This prevents data loss in some situations and will eventually lead to conflicts instead.

Parameters

site – For which site? Default is the event’s current site.

mark_refresh_needed(*site=None*)

Mark this site as definitely outdated. In some way this is the opposite to [mark_changed\(\)](#). This is only used in some special contexts and not part of the basic mechanism.

Parameters

site – For which site? Default is the event’s current site.

resolve_conflicts(*master_site*)

Resolve all conflicts.

Parameters

master_site – The new stream master site.

property conflict_resolution

The name of the new stream master site if the conflicts were resolved successfully, otherwise *None*.

property conflicts

All conflicts found by this event so far.

changed_dangerously_late(*for_site=None*)

Returns whether this site has seen changes late enough to be potentially in conflict with something.

Precisely, it returns whether the content version number (see [content_version\(\)](#)) of this site is at least as large as the one from the elected master site.

This always indicates a potential conflict, because there might be a change on the given site as well, which would be lost once the master site actually updates all the other sites.

Parameters

for_site – For which site? Default is the event’s current site. For the master site, it will by nature always return *True*.

add_conflict(*description, details=None*)

Add a conflict.

Parameters

- **description** – The conflict description.
- **details** – Conflict details.

remove_conflict(*conflict*)

Remove a conflict.

Parameters

conflict – The conflict to remove.

set_master_site(*master_site=None*)

Set the master site for the current item stream.

This is usually the site where the other sites will get updated from (regarding the current stream).

Parameters

master_site – The master site.

set_sync_run_sn_of_last_change(*sn*, *, *for_site=None*)

Set the serial number of the sync run that has seen the last change (for this site or any other).

Parameters

- **sn** – The sync run’s serial number.
- **for_site** – For which site? Default is the event’s current site.

_master_site(*of_stream=None*)

Return the master site for the given stream (or the item’s main stream).

content_version(*for_site=None*, *, *only_past*)

Return the version number of the current item’s main stream, for this site or any other one.

This is the same as *content_version* of stream events related to the main stream (i.e. ""). See there for more details.

Parameters

- **for_site** – For which site? Default is the event’s current site.
- **only_past** – Depending on this flag, this will either always be the sn of a sync run from the past, i.e. it will always be less than the current sync run’s sn, or it returns the current sync run’s sn if there were changes detected since the last sync run.

stream_cookie(*for_site=None*)

Return the stream cookie of the main stream, usually determined in [parzzley.sync.aspect.events.item.stream.DetermineCookie](#).

For details about cookies, see [parzzley.fs.Site.cookie\(\)](#).

This is the same as *stream_cookie* of stream events related to the main stream (i.e. "").

Parameters

- **for_site** – For which site? Default is the event’s current site.

property stream_cookies: `dict[str, tuple]`

set_stream_cookie(*stream_cookie*, *, *for_site=None*)

Set the stream cookie for the current item stream.

Parameters

- **stream_cookie** – The stream cookie.
- **for_site** – For which site? Default is the event’s current site.

stream_cookie_after_last_sync(*for_site=None*)

Return the cookie this item stream had after last sync.

Parameters

- **for_site** – For which site? Default is the event’s current site.

_updated_cookie(*for_site=None*)

property comparator

The current comparator (or None for the default, byte-wise one). See also [set_comparator\(\)](#).

set_comparator(*comparator*)

Set the comparator.

Parameters

- **comparator** – The new comparator.

`_abc_impl = <_abc._abc_data object>`

class `parzzley.sync.engine.events_impl._Logger(event, item, stream_name)`

Bases: [Logger](#)

Parameters

- **event** ([SyncRunEvent](#))
- **item** ([parzzley.fs.Item](#) / `None`)
- **stream_name** (`str` / `None`)

`_PYTHON_LOGGER_FUNC_BY_SEVERITY = {Severity.DEBUG: <bound method Logger.debug of <Logger parzzley.sync (WARNING)>>, Severity.INFO: <bound method Logger.info of <Logger parzzley.sync (WARNING)>>, Severity.WARNING: <bound method Logger.warning of <Logger parzzley.sync (WARNING)>>, Severity.FATAL: <bound method Logger.error of <Logger parzzley.sync (WARNING)>>}`

`_log(severity, message, message_args)`

Log a message.

Parameters

- **severity** – The message severity.
- **message** – The message text (as Python logging format string; see `message_args`).
- **message_args** – The message arguments.

`_abc_impl = <_abc._abc_data object>`

parzzley.sync.logger package

Sync logging. See [Logger](#).

class `parzzley.sync.logger.Severity(*values)`

Bases: `Enum`

Message severity.

DEBUG = 1

Debug severity.

INFO = 2

Info severity.

WARNING = 3

Warning severity.

FATAL = 4

Fatal severity.

class `parzzley.sync.logger.Logger`

Bases: `ABC`

Base class for a sync logger, as used for logging e.g. by aspects and the sync engine itself.

abstractmethod `_log(severity, message, message_args)`

Log a message.

Parameters

- **severity** ([Severity](#)) – The message severity.
- **message** (*str*) – The message text (as Python logging format string; see `message_args`).
- **message_args** (*Sequence[Any]*) – The message arguments.

Return type

None

debug(*message*, **message_args*)

Log a message with ‘debug’ severity.

Parameters

- **message** (*str*) – The message text (as Python logging format string; see `message_args`).
- **message_args** (*Sequence[Any]*) – The message arguments.

Return type

None

info(*message*, **message_args*)

Log a message with ‘info’ severity.

Parameters

- **message** (*str*) – The message text (as Python logging format string; see `message_args`).
- **message_args** (*Sequence[Any]*) – The message arguments.

Return type

None

warning(*message*, **message_args*)

Log a message with ‘warning’ severity.

Parameters

- **message** (*str*) – The message text (as Python logging format string; see `message_args`).
- **message_args** (*Sequence[Any]*) – The message arguments.

Return type

None

fatal(*message*, **message_args*)

Log a message with ‘fatal’ severity.

Parameters

- **message** (*str*) – The message text (as Python logging format string; see `message_args`).
- **message_args** (*Sequence[Any]*) – The message arguments.

Return type

None

_abc_impl = <_abc._abc_data object>**class** `parzzley.sync.logger.Entry`(**severity*, *message*, *message_args*, *item*, *stream*)Bases: `object`

Represents a single sync log entry.

Parameters

- **severity** ([Severity](#)) – The entry severity.

- **message** (*str*) – The entry message (as Python logging format string; see `message_args`).
- **message_args** (*Iterable[Any]*) – The entry message arguments.
- **item** (`parzzley.fs.Item` / *None*) – The associated item.
- **stream** (*str* / *None*) – The associated stream name.

property severity: `Severity`

The entry severity.

property message: `str`

The entry message.

This is already the resolved final string, with the `message_args` applied.

property item: `parzzley.fs.Item` | `None`

The associated item.

property stream: `str` | `None`

The associated stream name.

class `parzzley.sync.logger.Logging`(*, *min_severity*, *max_severity*=*None*, *out*, *formatter*, *exclude*=())

Bases: `object`

Represents one logging setup (usually as specified in some configuration file).

This is mostly used by the engine in order to provide the sync reports (i.e. the final product of this entire module).

Parameters

- **min_severity** (`Severity` / *None*) – The minimal severity.
- **max_severity** (`Severity` / *None*) – The maximal severity.
- **out** (*Iterable[Out]*) – The logger output channels.
- **formatter** (`Formatter`) – The log formatter.
- **exclude** (*Iterable[Exclude]*) – The log exclusions.

emit(*sync_run*, *entries*)

Emit a list of entries, after filtering as specified, formatted by the formatter, to the specified logger output channels.

If errors are raised from the formatter or the output channels, this method will log these callstacks to Python logging, but does not forward these errors! It will just return normally instead.

Parameters

- **sync_run** (`parzzley.sync.run.SyncRun` / *None*) – The sync run (or *None* if this report is not associated to a particular sync run).
- **entries** (*Iterable[Entry]*) – The entries.

Return type

None

class `parzzley.sync.logger.Formatter`

Bases: `ABC`

Base class for an implementation of a log formatter.

abstractmethod `format(sync_run, entries)`

Format the given entries to a textual representation.

Parameters

- **sync_run** ([parzzley.sync.run.SyncRun](#) / *None*) – The sync run (or *None* if this report is not associated to a particular sync run).
- **entries** (*Iterable*[[Entry](#)]) – The entries.

Return type

str

`_abc_impl = <_abc._abc_data object>`

class `parzzley.sync.logger.Out`

Bases: `ABC`

Base class for an implementation of a logger output channel.

abstractmethod `emit_text(sync_run, s)`

Emit the text via this output channel.

Parameters

- **sync_run** ([parzzley.sync.run.SyncRun](#) / *None*) – The sync run (or *None* if this report is not associated to a particular sync run).
- **s** (*str*) – The text to emit.

Return type

None

`_abc_impl = <_abc._abc_data object>`

class `parzzley.sync.logger.Exclude(conditions)`

Bases: `object`

Represents an entry exclusion definition.

Parameters

conditions (*Iterable*[[Exclude.Condition](#)]) – The conditions. A message gets excluded if it matches at least one of these conditions.

is_met(*entry*)

Return whether this exclusion definition is met by the given log entry.

Parameters

entry ([Entry](#)) – The log entry.

Return type

bool

class `Condition`

Bases: `ABC`

Represents an exclusion condition.

See subclasses in [parzzley.sync.logger.conditions](#).

abstractmethod `is_met(entry)`

Return whether this condition is met by the given log entry.

Parameters**entry** ([Entry](#)) – The log entry.**Return type**

bool

`_abc_impl = <_abc._abc_data object>``parzzley.sync.logger.register_formatter(formatter_type)`

Make a log formatter class available to be used, e.g. in logger configurations, by its kind name. The kind name is the parent module name (e.g. "baz" if the type is implemented in a package like `foo.bar.baz`).

Parameters**formatter_type** (`type[Formatter]`) – The log formatter class to register.**Return type**`type[Formatter]``parzzley.sync.logger.formatter_type_by_kind(kind)`

Return a log formatter type by the kind name.

Parameters**kind** (`str`) – The log formatter kind.**Return type**`type[Formatter] | None``parzzley.sync.logger.register_out(out_type)`

Make a logger output channel class available to be used, e.g. in logger configurations, by its kind name. The kind name is the parent module name (e.g. "baz" if the type is implemented in a package like `foo.bar.baz`).

Parameters**out_type** (`type[Out]`) – The logger output channel class to register.**Return type**`type[Out]``parzzley.sync.logger.out_by_kind(kind)`

Return a logger output channel type by the kind name.

Parameters**kind** (`str`) – The logger output channel kind.**Return type**`type[Out] | None`

Submodules

`parzzley.sync.logger.conditions` module

Sync logger exclusion conditions. See [SimpleCondition](#).

```
class parzzley.sync.logger.conditions.SimpleCondition(*, message=None, message_re=None,
                                                    item=None, item_re=None,
                                                    severity_below=None, stream=None)
```

Bases: [Condition](#)

Represents a simple condition.

Parameters

- **message** (`str` | `None`)

- `message_re(str | None)`
- `item(str | None)`
- `item_re(str | None)`
- `severity_below(Severity | None)`
- `stream(str | None)`

is_met(entry)

Return whether this condition is met by the given log entry.

Parameters

entry – The log entry.

`_abc_impl = <_abc._abc_data object>`

class `parzzley.sync.logger.conditions.ConditionAllOf(*, conditions)`

Bases: `Condition`

Represents the AND-combination of its inner conditions.

Parameters

conditions (`Iterable[Condition]`)

is_met(entry)

Return whether this condition is met by the given log entry.

Parameters

entry – The log entry.

`_abc_impl = <_abc._abc_data object>`

class `parzzley.sync.logger.conditions.ConditionAnyOf(*, conditions)`

Bases: `Condition`

Represents the OR-combination of its inner conditions.

Parameters

conditions (`Iterable[Condition]`)

is_met(entry)

Return whether this condition is met by the given log entry.

Parameters

entry – The log entry.

`_abc_impl = <_abc._abc_data object>`

class `parzzley.sync.logger.conditions.ConditionNegate(*, condition)`

Bases: `Condition`

Represents the negation of its inner condition.

Parameters

condition (`Condition`)

is_met(entry)

Return whether this condition is met by the given log entry.

Parameters

entry – The log entry.

```
_abc_impl = <_abc._abc_data object>
```

Submodules

parzzley.sync.control module

Sync controls. See [SyncControl](#).

class parzzley.sync.control.SyncControl

Bases: ABC

Base class for sync controls. A sync control allows to monitor and control a running sync run.

These objects are returned e.g. by [parzzley.sync.manager.Manager.run_sync\(\)](#).

See e.g. [wait_finished\(\)](#) and [was_successful](#).

abstract property sync_run_id: str

This sync run's identifier (always a new unique one for each run!).

abstract property is_finished: bool

Whether the associated sync run is already finished (either successfully or not).

abstract property was_successful: bool

Whether the associated sync run is already finished and was successful.

For a finished but unsuccessful run, there is at least one critical alert (see [alerts](#)).

abstract property was_effective: bool

Whether the associated sync run had any effects, i.e. synced any changes.

Note: This is only a hint. You must not rely on it for any critical decisions!

abstractmethod wait_finished()

Wait until the associated sync run is finished. When this function returns, [is_finished](#) is set.

Return type

None

property is_cancel_requested: bool

Whether this sync run was requested to cancel. See also [cancel\(\)](#).

cancel()

Request this sync run to cancel.

Return type

None

```
_abc_impl = <_abc._abc_data object>
```

parzzley.sync.manager module

Parzzley manager. See [Manager](#).

class parzzley.sync.manager.Manager(*, id_hint, config)

Bases: object

A manager allows to execute the syncing of given configuration.

Its context must be entered (with-block) in order to use it. You must enter its context only once.

Each manager is associated to a ‘control site’ (basically a filesystem location - usually somewhere in the local filesystem’s /var directory - where Parzzley stores various internal state information). Sync executions of a particular volume should always use the same control site. A fresh control site will drop state information, which potentially incurs a much longer time for syncing and maybe further inconveniences like conflicts.

Do not use directly. See [for_config_directory\(\)](#).

Parameters

- **id_hint** (*str*) – If this is a new manager (i.e. there never was a manager using the same control site), the id hint will be used as part of its manager id (see [id](#)).
- **config** ([parzzley.config.Configuration](#)) – All specified configuration.

_MANAGERS = {}

property var_dir: [Path](#)

This manager’s data root directory.

static for_config_directory(*config_dir*)

Return a manager for a given config directory.

Parameters

config_dir (*Path* | *str*) – The Parzzley config directory.

Return type

[Manager](#)

property volumes: [Sequence](#)[[Volume](#)]

All specified volumes.

property id: *str*

The sync manager id.

This id is permanent for a given control site (as long as it does not lose its data) and globally unique.

async start_sync(*prepared_sync_setup*)

Start a sync (and return a control object for watching or canceling it). The actual sync run will happen in background.

You need to use [prepare_sync\(\)](#) before in order to get the required argument.

If you do not need all this control flexibility, see [run_sync\(\)](#) instead.

Parameters

prepared_sync_setup ([PreparedSyncSetup](#)) – The prepared sync setup to execute.

Return type

[SyncControl](#) | None

run_sync(*s*)

Run a sync (and return a control object that contains some auxiliary information).

Parameters

s ([Volume](#) | [PreparedSyncSetup](#)) – The volume to sync (or a prepared sync setup as returned by [prepare_sync\(\)](#)).

Return type

[SyncControl](#) | None

prepare_sync(*volume_config*, *, *require_one_of*=None)

For a given volume configuration, try to connect to all its sites and return a prepared sync setup as required for [start_sync\(\)](#).

You must always call this function right before starting a sync and use that fresh instance.

Parameters

- **volume_config** ([parzzley.config.Volume](#)) – The volume configuration.
- **require_one_of** ([Iterable\[Site | SiteSetup | str\]](#) | None) – If none of these sites are able to connect, then do not even try to connect the other ones, but continue with an empty connection list instead.

Return type

[AsyncGenerator\[PreparedSyncSetup, None\]](#)

async store_success_info(*volume_name*, *sites*)

See [parzzley.sync.run.SyncRun.store_success_info\(\)](#).

Parameters

- **volume_name** (*str*)
- **sites** ([Iterable\[Site | SiteSetup | str\]](#))

Return type

None

async _manager_control_site(*key*)

See [parzzley.sync.run.SyncRun.manager_control_site\(\)](#).

Parameters

key (*str*)

Return type

[Site](#)

volume_state_variable(*volume_name*, *name*, *, *initial_value*=None)

See [parzzley.sync.run.SyncRun.volume_state_variable\(\)](#).

Parameters

- **volume_name** (*str*)
- **name** (*str*)
- **initial_value** (*Any*)

Return type

[VolumeStateVariable](#)

__verify_entered()

__site_success_info_variable(*volume_name*)

Parameters

volume_name (*str*)

Return type

[VolumeStateVariable](#)

__last_warned_info_variable(*volume_name*)

Parameters

volume_name (*str*)

Return type

[VolumeStateVariable](#)

async __run_sync__async(*s*)

Parameters

s ([Volume](#) | [PreparedSyncSetup](#))

Return type

[SyncControl](#) | None

_TSiteTuple

alias of `tuple[parzzley.fs.Site|None, parzzley.fs.SiteContextManager]`

_TFullSiteTuple

alias of `tuple[parzzley.fs.SiteSetup, tuple[parzzley.fs.Site|None, parzzley.fs.SiteContextManager]]`

async __try_connect()

Parameters

site_setup ([SiteSetup](#))

Return type

`tuple[SiteSetup, tuple[Site | None, AsyncContextManager[Site | None, bool | None]]]`

async __emit_not_synced_warnings(*volume_config*, *loggings*)

Parameters

- **volume_config** ([Volume](#))
- **loggings** ([Iterable](#) [[Logging](#)])

Return type

None

class PreparedSyncSetup(*volume_config*, *volume*, *connected_sites*, *loggings*)

Bases: `object`

A prepared sync setup. Such objects are returned by `Manager.prepare_sync()` and used for `Manager.start_sync()`.

Parameters

- **volume_config** ([parzzley.config.Volume](#))
- **volume** ([parzzley.sync.Volume](#))
- **connected_sites** (`dict[parzzley.fs.SiteSetup, _TSiteTuple]`)
- **loggings** (`Sequence[parzzley.sync.logger.Logging]`)

volume_config: [parzzley.config.Volume](#)

The volume config to sync.

volume: [parzzley.sync.Volume](#)

The volume to sync.

connected_sites: `dict[parzzley.fs.SiteSetup, _TSiteTuple]`

All sites that were actually able to connect.

loggings: `Sequence[parzzley.sync.logger.Logging]`

The loggings.

class VolumeStateVariable(*manager_control_site, volume_name, name, initial_value*)

Bases: `object`

A persistent volume state variable. Used for storing arbitrary state data about a sync volume, in order to make it available for later sync runs on that volume.

Each variable could be used for keeping a single value, but also as a key/value-store (by specifying a key).

Data stored in this variable is persisted locally in a Parzzley data directory. It will stay there as long as the local filesystem stays intact.

Do not use directly. See also `parzzley.sync.run.SyncRun.volume_state_variable()`.

Parameters

- **manager_control_site** (`Callable[[str], Coroutine[Any, Any, parzzley.fs.Site]]`)
- **volume_name** (`str`)
- **name** (`str`)
- **initial_value** (`Any`)

async value(**, key=""*)

Return the current value of this variable.

Parameters

key (`str`) – The key. Leave to default for single value storage.

Return type

`Any`

async set_value(*value, *, key=""*)

Set the value of this variable.

This will happen atomically, i.e. it will either fail, leaving the variable in its old state, or succeed; even if the process would get killed meanwhile.

Parameters

- **value** (`Any`) – The new value.
- **key** (`str`) – The key. Leave to default for single value storage.

Return type

`None`

async __promote_early_file_to_final_file(*control_site, key*)

async __control_site()

parzzley.sync.run module

Sync runs. See `SyncRun`.

class parzzley.sync.run.SyncRun(**, manager, volume, sites, started_at*)

Bases: `object`

Representation of one sync run (i.e. one execution) on some volume, used inside the engine and aspects. For controlling a sync run from outside, see `parzzley.sync.control.SyncControl` instead.

In particular, see [store_success_info\(\)](#)!

Parameters

- **manager** – The associated manager.
- **volume** ([parzzley.sync.Volume](#)) – The volume to associate with this sync run.
- **sites** ([Iterable](#)[[parzzley.fs.Site](#)]) – The sites of the volume that were able to actually establish a connection.
- **started_at** ([datetime](#)) – The time when this sync run has been started.

property sn: int

The serial number of this sync run.

This will increase for each sync run on a given volume. The first sync run will be 1.

property volume_name: str

The name of the volume associated to this sync run.

property manager_id: str

This manager's identifier (the same for each run, unless it gets synced by another manager - e.g. after system reinstallation of the Parzzley machine).

property manager_var_dir: Path

The manager's data root directory.

property id: str

This sync run's identifier (always a new unique one for each run!).

property sites: Sequence[[Site](#)]

The connected sites associated to this sync run. Note: This only contains sites that actually could be connected to.

property started_at: datetime

The time when this sync run has been started.

aspects(*site_name*)

Return the aspects associated to this sync run for a given site.

Parameters

site_name (*str*) – The site name.

Return type

[Sequence](#)[[Aspect](#)]

volume_state_variable(*name*, *, *initial_value=None*)

Return a volume state variable (see also [parzzley.sync.manager.Manager.VolumeStateVariable](#)).

This variable will automatically contain the value(s) that were stored in earlier sync runs.

Calling this method with the same name on the same instance more than once is only allowed with the same settings (e.g. *initial_value*).

Data stored in this variable is persisted locally in a Parzzley data directory.

Parameters

- **name** (*str*) – The variable name.
- **initial_value** (*Any*) – The initial value.

Return type[VolumeStateVariable](#)**async site_control_site**(*for_site*, *key*, *, *retain_from_former_manager=False*)

Return the control site for a given key and site.

Multiple calls with the same combination of key and site will return the same site, even beyond a sync run. So control sites can be used for persistence of arbitrary internal status data.

Data will be stored on the site. It will be lost when the site itself (maybe on some remote device) gets replaced by an empty one, e.g. due to hardware replacement. See also [manager_control_site\(\)](#). It will, depending on its configuration, also be dropped whenever a new Parzzley manager takes over (usually after a reinstallation of the Parzzley machine).

Parameters

- **for_site** ([Site](#) / [SiteSetup](#) / *str*) – The storage site. All data will be stored on that site (but usually invisible to the user).
- **key** (*str*) – Arbitrary string.
- **retain_from_former_manager** (*bool*) – Whether to retain the related content even from a former manager (e.g. after a backup was restored on some site).

Return type[Site](#)**async manager_control_site**(*key*)

Return the local control site for a given key.

Multiple calls with the same combination of key and site will return the same site, even beyond a sync run. So control sites can be used for persistence of arbitrary internal status data.

Manager control sites are local. They will lose their content when the Parzzley machine gets replaced or loses its filesystem in other ways. See also [site_control_site\(\)](#).

Note: This provides much flexibility, but for most cases, it is recommended to use the simpler [volume_state_variable\(\)](#) instead.

Parameters

key (*str*) – Arbitrary string.

Return type[Site](#)**async store_success_info**(*sites*)

Store success info for this sync run.

To be called after the sync run has been completed successfully.

Parameters

sites (*Iterable*[[Site](#) / [SiteSetup](#) / *str*]) – The involved sites.

Return type

None

async __get_next_sn()**Return type**

int

parzzley.sync.utils module

Low-level utils related to syncing.

`parzzley.sync.utils.run_coroutine_in_new_thread(coro, *, thread_name=None, thread_name_postfix=None)`

Run a coroutine in a new thread and return its result or forward its exception.

Parameters

- **coro** – The coroutine to run.
- **thread_name** (*str* / *None*) – Optional thread name.
- **thread_name_postfix** (*str* / *None*) – Optional thread name postfix to append to the current thread name.

`parzzley.sync.utils.__run_coroutine_in_new_thread__run(coro, result)`

Return type

None

10.1.2 Submodules

10.1.3 parzzley.parzzley_cli module

The Parzzley CLI.

`parzzley.parzzley_cli.main()`

The CLI main routine.

Return type

None

`parzzley.parzzley_cli.parser(*, only_documentation=True)`

Return an argument parser.

Parameters

only_documentation (*bool*) – Whether to include only parameters that are relevant for documentation.

Return type

ArgumentParser

class `parzzley.parzzley_cli.Commands`

Bases: `object`

Parzzley CLI commands.

sync(*config_dir*, *volume_names*, **_)

Execute sync once.

Parameters

- **config_dir** (*str*) – The Parzzley configuration directory.
- **volume_names** (*Sequence[str]*) – The volumes to sync.

Return type

None

sync_loop(*config_dir*, **_)

Execute sync service loop.

Parameters

config_dir (*str*) – The Parzzley configuration directory.

Return type

None

`parzzley.parzzley_cli._setup_logging(*, debug=False)`

Parameters

debug (*bool*)

PYTHON MODULE INDEX

p

- parzzley, 27
- parzzley.asset, 27
- parzzley.asset.data, 27
- parzzley.asset.project_info, 27
- parzzley.builtin, 27
- parzzley.builtin.aspects, 27
- parzzley.builtin.aspects.attach_sites, 28
- parzzley.builtin.aspects.conflicts, 29
- parzzley.builtin.aspects.default_sync, 31
- parzzley.builtin.aspects.directory, 31
- parzzley.builtin.aspects.exclude_paths, 32
- parzzley.builtin.aspects.item_type, 33
- parzzley.builtin.aspects.main_stream, 33
- parzzley.builtin.aspects.posix_attributes, 34
- parzzley.builtin.aspects.pull_and_purge, 34
- parzzley.builtin.aspects.remove, 35
- parzzley.builtin.aspects.revision_tracking, 36
- parzzley.builtin.aspects.streaming, 37
- parzzley.builtin.aspects.sync_report, 40
- parzzley.builtin.aspects.xattrs, 40
- parzzley.builtin.fs, 41
- parzzley.builtin.fs.local, 46
- parzzley.builtin.fs.ssh, 48
- parzzley.builtin.log_formatters, 49
- parzzley.builtin.log_formatters.html, 49
- parzzley.builtin.log_formatters.json, 50
- parzzley.builtin.log_outs, 50
- parzzley.builtin.log_outs.shell, 50
- parzzley.config, 50
- parzzley.config.file_formats, 54
- parzzley.config.file_formats.xml, 55
- parzzley.config.loader, 57
- parzzley.fs, 58
- parzzley.fs.stream, 70
- parzzley.fs.utils, 73
- parzzley.parzzley_cli, 137
- parzzley.service, 73
- parzzley.sync, 76
- parzzley.sync.aspect, 77
- parzzley.sync.aspect.events, 78
- parzzley.sync.aspect.events.item, 79
- parzzley.sync.aspect.events.item.dir, 86
- parzzley.sync.aspect.events.item.stream, 87
- parzzley.sync.aspect.events.sync_run, 96
- parzzley.sync.aspect.only_if, 100
- parzzley.sync.control, 130
- parzzley.sync.engine, 104
- parzzley.sync.engine.event_runner, 111
- parzzley.sync.engine.events_impl, 113
- parzzley.sync.logger, 124
- parzzley.sync.logger.conditions, 128
- parzzley.sync.manager, 130
- parzzley.sync.run, 134
- parzzley.sync.utils, 137

INDEX

Symbols

<code>_DATA__ALL_CONFLICTS</code>	(parzz- ley.sync.engine.events_impl.ItemEvent tribute), 116	<code>_DATA__LOG_ENTRIES</code>	(parzz- ley.sync.engine.events_impl.SyncRunEvent tribute), 114
<code>_DATA__CHANGED_SITES</code>	(parzz- ley.sync.engine.events_impl.StreamEvent tribute), 120	<code>_DATA__LOG_REMOVAL</code>	(parzz- ley.builtin.aspects.remove.DirectRemove tribute), 35
<code>_DATA__CHECK_DONE</code>	(parzz- ley.builtin.aspects.streaming.SkipItemIfNoUpToDateSitesAreConnected tribute), 37	<code>_DATA__LOG_REMOVAL</code>	(parzz- ley.builtin.aspects.remove.TrashRemove tribute), 35
<code>_DATA__CHILD_NAMES</code>	(parzz- ley.sync.engine.events_impl.DirEvent tribute), 119	<code>_DATA__MASTER_SITE_TUPLE</code>	(parzz- ley.builtin.aspects.streaming.ComputeMasterSiteTable tribute), 38
<code>_DATA__CONFLICTS</code>	(parzz- ley.builtin.aspects.conflicts.TrackConflicts tribute), 30	<code>_DATA__NEEDS_TAGGING</code>	(parzz- ley.builtin.aspects.xattrs.XattrSynchronization tribute), 40
<code>_DATA__CONFLICTS</code>	(parzz- ley.sync.engine.events_impl.StreamEvent tribute), 120	<code>_DATA__NEW_ITEM_INFO</code>	(parzz- ley.sync.engine.events_impl.ItemEvent tribute), 116
<code>_DATA__CONFLICT_RESOLUTION</code>	(parzz- ley.sync.engine.events_impl.StreamEvent tribute), 120	<code>_DATA__PER_STREAM_DATAS</code>	(parzz- ley.sync.engine.events_impl.ItemEvent tribute), 116
<code>_DATA__CONTENT_EQUAL_TO_MASTER</code>	(parzz- ley.builtin.aspects.conflicts.DetectContentConflicts tribute), 29	<code>_DATA__REFRESH_NEEDED_SITES</code>	(parzz- ley.sync.engine.events_impl.StreamEvent tribute), 120
<code>_DATA__DESTINATION_STREAMABLES</code>	(parzz- ley.sync.engine.events_impl.StreamEvent tribute), 120	<code>_DATA__REPORT_DONE</code>	(parzz- ley.builtin.aspects.sync_report.SyncReport tribute), 40
<code>_DATA__DISABLE_CHANGE_DETECTION</code>	(parzz- ley.sync.engine.events_impl.StreamEvent tribute), 120	<code>_DATA__RETRY</code>	(parzz- ley.builtin.aspects.streaming.StickToOldMasterSitesC tribute), 39
<code>_DATA__EFFECTIVE</code>	(parzz- ley.sync.engine.events_impl.SyncRunEvent tribute), 114	<code>_DATA__REVISIONS_SITE</code>	(parzz- ley.builtin.aspects.revision_tracking.RevisionTracking tribute), 36
<code>_DATA__IS_MARKED_FOR_FULL_RETRY</code>	(parzz- ley.sync.engine.events_impl.ItemEvent tribute), 116	<code>_DATA__REVISION_ITEM</code>	(parzz- ley.builtin.aspects.revision_tracking.RevisionTracking tribute), 36
<code>_DATA__IS_MARKED_FOR_SKIP</code>	(parzz- ley.sync.engine.events_impl.ItemEvent tribute), 116	<code>_DATA__ROLLBACK_DIR</code>	(parzz- ley.builtin.aspects.directory.RemoveForReplacement tribute), 32
<code>_DATA__ITEM_TYPE</code>	(parzz- ley.sync.engine.events_impl.ItemEvent tribute), 116	<code>_DATA__SOURCE_STREAMABLE</code>	(parzz- ley.sync.engine.events_impl.StreamEvent tribute), 120

<code>_DATA__STREAM_COMPARATOR</code>	(parzz- ley.sync.engine.events_impl.StreamEvent attribute), 120	<code>_EventAfterStreamSupportDetermined</code>	(class in parzzley.sync.aspect.events.sync_run), 99
<code>_DATA__STREAM_COOKIES</code>	(parzz- ley.sync.engine.events_impl.StreamEvent attribute), 120	<code>_EventAfterStreamSupportDetermined.StreamSupportInfo</code>	(class in parzzley.sync.aspect.events.sync_run), 99
<code>_DATA__STREAM_MASTER_SITE</code>	(parzz- ley.sync.engine.events_impl.StreamEvent attribute), 120	<code>_EventData__key()</code>	(parzz- ley.sync.engine.events_impl.SyncRunEvent._EventData method), 116
<code>_DATA__STREAM_MASTER_TABLE</code>	(parzz- ley.sync.engine.events_impl.StreamEvent attribute), 120	<code>_EventData__native_data()</code>	(parzz- ley.sync.engine.events_impl.SyncRunEvent._EventData method), 116
<code>_DATA__STREAM_PIPES</code>	(parzz- ley.sync.engine.events_impl.StreamEvent attribute), 120	<code>_GlobalItemsBook</code>	(class in parzzley.sync.engine), 108
<code>_DATA__STREAM_SUPPORT</code>	(parzz- ley.sync.engine.events_impl.SyncRunEvent attribute), 114	<code>_GlobalItemsBook._ItemInfo</code>	(class in parzz- ley.sync.engine), 109
<code>_DATA__STREAM_SUPPORT_DICT</code>	(parzz- ley.builtin.aspects.streaming.GlobalStreamSupportItemIsTypeCondition attribute), 37	<code>_IsItemTypeSupportedByStreamCondition</code>	(class in parzzley.sync.aspect.only_if), 101
<code>_DATA__STREAM_SUPPORT_PER_SITE</code>	(parzz- ley.sync.engine.events_impl.SyncRunEvent attribute), 113	<code>_Item</code>	(class in parzzley.fs), 59
<code>_DATA__TEMP_ITEM</code>	(parzz- ley.builtin.aspects.streaming.WorkingItem attribute), 39	<code>_ItemExistsCondition</code>	(class in parzz- ley.sync.aspect.only_if), 100
<code>_DATA__UPDATE_LOG_INFO</code>	(parzz- ley.builtin.aspects.streaming.UpdateTransfer attribute), 39	<code>_ItemIsTypeCondition</code>	(class in parzz- ley.sync.aspect.only_if), 101
<code>_DATA__WORKING_ITEMS</code>	(parzz- ley.sync.engine.events_impl.ItemEvent at- tribute), 116	<code>_ItemsBookBase</code>	(class in parzzley.sync.engine), 107
<code>_EARLY_LAST_SYNC_RUN_SN_FILE</code>	(parzz- ley.builtin.aspects.attach_sites.AttachSites attribute), 28	<code>_LAST_SYNC_RUN_SN_FILE</code>	(parzz- ley.builtin.aspects.attach_sites.AttachSites attribute), 28
<code>_EXCEPTED_LAST_SYNC_RUN_SN_PER_SITE_KEY</code>	(parzzley.builtin.aspects.attach_sites.AttachSites attribute), 28	<code>_Logger</code>	(class in parzzley.sync.engine.events_impl), 124
<code>_Event</code>	(class in parzzley.sync.aspect.events), 78	<code>_LoopThread__run()</code>	(parzz- ley.service.Service._LoopThread method), 75
<code>_Event</code>	(class in parzzley.sync.aspect.events.item), 80	<code>_LoopThread__run__async()</code>	(parzz- ley.service.Service._LoopThread method), 75
<code>_Event</code>	(class in parzzley.sync.aspect.events.item.dir), 86	<code>_MANAGED_BY_FILE</code>	(parzz- ley.builtin.aspects.attach_sites.AttachSites attribute), 28
<code>_Event</code>	(class in parzz- ley.sync.aspect.events.item.stream), 88	<code>_MANAGERS</code>	(parzzley.sync.manager.Manager attribute), 131
<code>_Event</code>	(class in parzzley.sync.aspect.events.sync_run), 96	<code>_MOVABLE_ITEM_TYPES</code>	(parzz- ley.builtin.aspects.pull_and_purge.PullAndPurgeSyncSink attribute), 34
<code>_EventAfterItemInfoCollected</code>	(class in parzz- ley.sync.aspect.events.item.stream), 88	<code>_MonitorSiteForChangesThread__mark_changed()</code>	(parzzley.service.Service._MonitorSiteForChangesThread method), 76
<code>_EventAfterItemTypeKnown</code>	(class in parzz- ley.sync.aspect.events.item), 81	<code>_PYTHON_LOGGER_FUNC_BY_SEVERITY</code>	(parzz- ley.sync.engine.events_impl._Logger attribute), 124
<code>_EventAfterMainStreamMasterSiteDetermined</code>	(class in parzzley.sync.aspect.events.item), 83	<code>_PerSiteItemsBook</code>	(class in parzzley.sync.engine), 109
<code>_EventAfterMasterSiteDetermined</code>	(class in parzz- ley.sync.aspect.events.item.stream), 89	<code>_PerSiteItemsBook._ItemInfo</code>	(class in parzz- ley.sync.engine), 111
		<code>_SUPPORTED_ITEM_TYPES</code>	(parzz- ley.builtin.aspects.main_stream.MainStreamSynchronization attribute), 33
		<code>_SUPPORTED_ITEM_TYPES</code>	(parzz-

ley.builtin.aspects.posix_attributes.PosixAttributesSynchronization method), 113
attribute), 34 *__are_streams_equal()* (parzz-
ley.builtin.aspects.conflicts.DetectContentConflicts
 _SUPPORTED_ITEM_TYPES (parzz- *method*), 29
ley.builtin.aspects.revision_tracking.RevisionTracking
attribute), 37 *__catch_exceptions()* (parzzley.sync.engine.Engine
method), 106
 _SUPPORTED_ITEM_TYPES (parzz- *__check_canceled()* (parzzley.sync.engine.Engine
method), 106
ley.builtin.aspects.xattrs.XattrSynchronization
attribute), 40 *__check_whether_to_stay_attached()* (parzz-
ley.builtin.aspects.attach_sites.AttachSites
method), 28
 _ShellPool__new_shell() (parzz- *__control_site()* (parzz-
ley.builtin.fs.ShellBasedBackend._SiteBackend._ShellPool *ley.sync.manager.Manager.VolumeStateVariable*
method), 42 *method*), 134
 _Source (class in parzzley.builtin.aspects.conflicts), 30 *__data_key_data()* (parzz-
ley.sync.engine.events_impl.SyncRunEvent
 _SubTreeSiteBackend (class in parzzley.fs), 68 *method*), 114
 _TFullSiteTuple (parzzley.sync.manager.Manager *attribute*), 133
 _TSiteTuple (parzzley.sync.manager.Manager *attribute*), 133
 _VolumeThread__action__finished_sync_run() (parzz-
ley.service.Service._VolumeThread *method*), 75
 _VolumeThread__action__outdated_sites() (parzz-
ley.service.Service._VolumeThread *method*), 75
 _VolumeThread__action__start_sync() (parzz-
ley.service.Service._VolumeThread *method*), 75
 _VolumeThread__action__sync_forced_flag() (parzz-
ley.service.Service._VolumeThread *method*), 76
 _WORKING_LAST_SYNC_RUN_SN_FILE (parzz-
ley.builtin.aspects.attach_sites.AttachSites
attribute), 28
 _WithDestinationStreamableGetters (class in
 parzzley.sync.aspect.events.item.stream), 87
 _WithMasterSiteTableSetter (class in parzz-
 ley.sync.aspect.events.item.stream), 88
 _WithSetItemInfo (class in parzz-
 ley.sync.aspect.events.item), 79
 _WithSourceStreamableGetters (class in parzz-
 ley.sync.aspect.events.item.stream), 87
 _WithStreamCookie (class in parzz-
 ley.sync.aspect.events.item), 80
 _WithStreamCookie (class in parzz-
 ley.sync.aspect.events.item.stream), 87
 _WithWorkingItems (class in parzz-
 ley.sync.aspect.events.item), 80
 _XATTR_TAG_KEY (parzz-
ley.builtin.aspects.xattrs.XattrSynchronization
attribute), 40
 __all_handler_nodes() (parzz-
ley.sync.engine.event_runner.Runner *method*),
 113
 __all_handler_nodes__find_dependencies() (parzz-
 ley.sync.engine.event_runner.Runner

<i>ley.builtin.aspects.streaming.WorkingItem</i> method), 40	<i>__sync_volume()</i> (parzz- <i>ley.config.file_formats.xml.XmlFileFormat</i> method), 55
<i>__prepare_items_book()</i> (parzz- <i>ley.sync.engine.Engine</i> method), 106	<i>__take_over_site_control()</i> (parzz- <i>ley.builtin.aspects.attach_sites.AttachSites</i> method), 28
<i>__promote_early_file_to_final_file()</i> (parzz- <i>ley.sync.manager.Manager.VolumeStateVariable</i> method), 134	<i>__title()</i> (parzzley.builtin.log_formatters.html.Formatter method), 49
<i>__read_last_sync_run_sn()</i> (parzz- <i>ley.builtin.aspects.attach_sites.AttachSites</i> method), 28	<i>__translate_arbitrary_exception_to_site_exception()</i> (parzzley.fs.Site method), 64
<i>__run_coroutine_in_new_thread__run()</i> (in mod- ule parzzley.sync.util), 137	<i>__try_connect()</i> (parzzley.sync.manager.Manager method), 133
<i>__run_sync__async()</i> (parzz- <i>ley.sync.manager.Manager</i> method), 133	<i>__verify_entered()</i> (parzzley.sync.manager.Manager method), 132
<i>__site_success_info_variable()</i> (parzz- <i>ley.sync.manager.Manager</i> method), 132	<i>__working_items__list()</i> (parzz- <i>ley.sync.engine.events_impl.ItemEvent</i> method), 119
<i>__source_streamable()</i> (parzz- <i>ley.builtin.aspects.conflicts.DetectContentConflicts</i> method), 29	<i>__write_last_sync_run_sn()</i> (parzz- <i>ley.builtin.aspects.attach_sites.AttachSites</i> method), 28
<i>__store_current_sync_run()</i> (parzz- <i>ley.builtin.aspects.attach_sites.AttachSites</i> method), 28	<i>__write_stdin()</i> (parzz- <i>ley.builtin.fs.ShellBasedBackend.Shell</i> method), 45
<i>__store_items_book()</i> (parzzley.sync.engine.Engine method), 106	<i>_abc_impl</i> (parzzley.builtin.aspects.xattrs.XattrSynchronization._PatchDic attribute), 41
<i>__stream_support_per_site()</i> (parzz- <i>ley.sync.engine.events_impl.SyncRunEvent</i> method), 115	<i>_abc_impl</i> (parzzley.builtin.fs.ShellBasedBackend attribute), 45
<i>__style()</i> (parzzley.builtin.log_formatters.html.Formatter method), 49	<i>_abc_impl</i> (parzzley.builtin.fs.ShellBasedBackend.Shell attribute), 45
<i>__sync()</i> (parzzley.sync.engine.Engine method), 105	<i>_abc_impl</i> (parzzley.builtin.fs.ShellBasedBackend._SiteBackend attribute), 44
<i>__sync__post()</i> (parzzley.sync.engine.Engine method), 105	<i>_abc_impl</i> (parzzley.builtin.fs.ShellBasedBackend._SiteBackend._FileMain attribute), 44
<i>__sync_aspects()</i> (parzz- <i>ley.config.file_formats.xml.XmlFileFormat</i> method), 56	<i>_abc_impl</i> (parzzley.builtin.fs.ShellBasedBackend._SiteBackend._FileMain attribute), 44
<i>__sync_item()</i> (parzzley.sync.engine.Engine method), 105	<i>_abc_impl</i> (parzzley.builtin.fs.ShellBasedBackend._SiteBackend._FileMain attribute), 44
<i>__sync_item_collect_info_and_determine_master_site_info_streams()</i> (parzzley.sync.engine.Engine method), 105	<i>_abc_impl</i> (parzzley.builtin.fs.ShellBasedBackend._SiteBackend._FileMain attribute), 44
<i>__sync_item_conflicts()</i> (parzz- <i>ley.sync.engine.Engine</i> method), 105	<i>_abc_impl</i> (parzzley.builtin.fs.local.Backend attribute), 48
<i>__sync_item_directory()</i> (parzz- <i>ley.sync.engine.Engine</i> method), 105	<i>_abc_impl</i> (parzzley.builtin.fs.local.Backend._SiteBackend attribute), 48
<i>__sync_item_populate_items_book()</i> (parzz- <i>ley.sync.engine.Engine</i> method), 106	<i>_abc_impl</i> (parzzley.builtin.fs.local.Backend._SiteBackend._FileMainStrea attribute), 47
<i>__sync_item_skipped_by_conflicts()</i> (parzz- <i>ley.sync.engine.Engine</i> method), 105	<i>_abc_impl</i> (parzzley.builtin.fs.local.Backend._SiteBackend._FileMainStrea attribute), 47
<i>__sync_item__source_streamables()</i> (parzz- <i>ley.sync.engine.Engine</i> method), 105	<i>_abc_impl</i> (parzzley.builtin.fs.local.Backend._SiteBackend._FileMainStrea attribute), 48
<i>__sync_item_update_streams()</i> (parzz- <i>ley.sync.engine.Engine</i> method), 105	<i>_abc_impl</i> (parzzley.builtin.fs.local.Backend._SiteBackend._FileMainStrea attribute), 48
<i>__sync_site()</i> (parzz- <i>ley.config.file_formats.xml.XmlFileFormat</i> method), 56	<i>_abc_impl</i> (parzzley.builtin.fs.ssh.Backend attribute), 49
	<i>_abc_impl</i> (parzzley.builtin.fs.ssh.Backend._Shell attribute), 48

_abc_impl (parzzley.builtin.log_formatters.html.Formatter attribute), 49
 _abc_impl (parzzley.builtin.log_formatters.json.Formatter attribute), 50
 _abc_impl (parzzley.builtin.log_outs.shell.Out attribute), 50
 _abc_impl (parzzley.config.file_formats.FileFormat attribute), 54
 _abc_impl (parzzley.config.file_formats.xml.XmlFileFormat attribute), 56
 _abc_impl (parzzley.fs.Backend attribute), 67
 _abc_impl (parzzley.fs.Backend.SiteBackend attribute), 67
 _abc_impl (parzzley.fs.Item attribute), 59
 _abc_impl (parzzley.fs._Item attribute), 60
 _abc_impl (parzzley.fs._SubTreeSiteBackend attribute), 70
 _abc_impl (parzzley.fs.stream.MemoryReadStreamable attribute), 72
 _abc_impl (parzzley.fs.stream.MemoryReadStreamable._ReadStreamable attribute), 72
 _abc_impl (parzzley.fs.stream.MemoryWriteStreamable attribute), 72
 _abc_impl (parzzley.fs.stream.MemoryWriteStreamable._WriteStreamable attribute), 72
 _abc_impl (parzzley.fs.stream.ReadStream attribute), 70
 _abc_impl (parzzley.fs.stream.ReadStreamable attribute), 71
 _abc_impl (parzzley.fs.stream.WriteStream attribute), 70
 _abc_impl (parzzley.fs.stream.WriteStreamable attribute), 71
 _abc_impl (parzzley.sync.aspect.events._Event attribute), 79
 _abc_impl (parzzley.sync.aspect.events.item.ApplyUpdate attribute), 85
 _abc_impl (parzzley.sync.aspect.events.item.DecideToSkip attribute), 81
 _abc_impl (parzzley.sync.aspect.events.item.DetermineType attribute), 82
 _abc_impl (parzzley.sync.aspect.events.item.Prepare attribute), 81
 _abc_impl (parzzley.sync.aspect.events.item.PrepareUpdating attribute), 83
 _abc_impl (parzzley.sync.aspect.events.item.RefreshItemsBo attribute), 86
 _abc_impl (parzzley.sync.aspect.events.item.SetUpWorking attribute), 84
 _abc_impl (parzzley.sync.aspect.events.item.SkipDueToConf attribute), 83
 _abc_impl (parzzley.sync.aspect.events.item._Event attribute), 81
 _abc_impl (parzzley.sync.aspect.events.item._EventAfterItem attribute), 82
 _abc_impl (parzzley.sync.aspect.events.item._EventAfterMaster attribute), 82
 _abc_impl (parzzley.sync.aspect.events.item._WithSetItemInfo attribute), 80
 _abc_impl (parzzley.sync.aspect.events.item._WithStreamCookie attribute), 80
 _abc_impl (parzzley.sync.aspect.events.item._WithWorkingItems attribute), 80
 _abc_impl (parzzley.sync.aspect.events.item.dir.Iterated attribute), 86
 _abc_impl (parzzley.sync.aspect.events.item.dir.List attribute), 86
 _abc_impl (parzzley.sync.aspect.events.item.dir.Prepare attribute), 86
 _abc_impl (parzzley.sync.aspect.events.item.dir._Event attribute), 86
 _abc_impl (parzzley.sync.aspect.events.item.stream.ApplyConflictResolution attribute), 95
 _abc_impl (parzzley.sync.aspect.events.item.stream.CollectDestinationStream attribute), 96
 _abc_impl (parzzley.sync.aspect.events.item.stream.CollectSourceStream attribute), 94
 _abc_impl (parzzley.sync.aspect.events.item.stream.CollectSourceStream attribute), 93
 _abc_impl (parzzley.sync.aspect.events.item.stream.CollectSourceStream attribute), 93
 _abc_impl (parzzley.sync.aspect.events.item.stream.CollectSourceStream attribute), 92
 _abc_impl (parzzley.sync.aspect.events.item.stream.DetectChanges attribute), 91
 _abc_impl (parzzley.sync.aspect.events.item.stream.DetermineComparator attribute), 89
 _abc_impl (parzzley.sync.aspect.events.item.stream.DetermineConflicts attribute), 95
 _abc_impl (parzzley.sync.aspect.events.item.stream.DetermineCookie attribute), 90
 _abc_impl (parzzley.sync.aspect.events.item.stream.DetermineMasterSite attribute), 92
 _abc_impl (parzzley.sync.aspect.events.item.stream.MasterSiteDetermined attribute), 92
 _abc_impl (parzzley.sync.aspect.events.item.stream.TransferUpdate attribute), 96
 _abc_impl (parzzley.sync.aspect.events.item.stream.TryResolveConflicts attribute), 95
 _abc_impl (parzzley.sync.aspect.events.item.stream._Event attribute), 88
 _abc_impl (parzzley.sync.aspect.events.item.stream._EventAfterItemInfoCo attribute), 88
 _abc_impl (parzzley.sync.aspect.events.item.stream._EventAfterMasterSite attribute), 89
 _abc_impl (parzzley.sync.aspect.events.item.stream._WithDestinationStream attribute), 88
 _abc_impl (parzzley.sync.aspect.events.item.stream._WithMasterSiteTable attribute), 88
 _abc_impl (parzzley.sync.aspect.events.item.stream._WithSourceStream attribute), 88

attribute), 87
 _abc_impl (parzzley.sync.aspect.events.item.stream._WithStreamControl attribute), 87
 _abc_impl (parzzley.sync.aspect.events.sync_run.Close attribute), 100
 _abc_impl (parzzley.sync.aspect.events.sync_run.DetermineStreamInfo attribute), 98
 _abc_impl (parzzley.sync.aspect.events.sync_run.Finish attribute), 100
 _abc_impl (parzzley.sync.aspect.events.sync_run.Prepare attribute), 97
 _abc_impl (parzzley.sync.aspect.events.sync_run.ValidateStreamSupport attribute), 99
 _abc_impl (parzzley.sync.aspect.events.sync_run._Event attribute), 97
 _abc_impl (parzzley.sync.aspect.events.sync_run._EventAfterStream attribute), 99
 _abc_impl (parzzley.sync.aspect.events.sync_run._EventAfterStream attribute), 99
 _abc_impl (parzzley.sync.aspect.only_if.Condition attribute), 100
 _abc_impl (parzzley.sync.aspect.only_if.IsItemTypeHereSupported attribute), 104
 _abc_impl (parzzley.sync.aspect.only_if.IsItemTypeOnStream attribute), 104
 _abc_impl (parzzley.sync.aspect.only_if.IsStream attribute), 103
 _abc_impl (parzzley.sync.aspect.only_if.ItemExistsHere attribute), 102
 _abc_impl (parzzley.sync.aspect.only_if.ItemExistsOnMasterSite attribute), 102
 _abc_impl (parzzley.sync.aspect.only_if.ItemHereIsType attribute), 102
 _abc_impl (parzzley.sync.aspect.only_if.ItemOnMasterSite attribute), 103
 _abc_impl (parzzley.sync.aspect.only_if.ThisIsMasterSite attribute), 103
 _abc_impl (parzzley.sync.aspect.only_if._IsItemTypeSupportedByStream attribute), 101
 _abc_impl (parzzley.sync.aspect.only_if._ItemExistsCondition attribute), 101
 _abc_impl (parzzley.sync.aspect.only_if._ItemIsTypeCondition attribute), 101
 _abc_impl (parzzley.sync.control.SyncControl attribute), 130
 _abc_impl (parzzley.sync.engine.Engine._SyncControl attribute), 107
 _abc_impl (parzzley.sync.engine._GlobalItemsBook attribute), 109
 _abc_impl (parzzley.sync.engine._ItemsBookBase attribute), 108
 _abc_impl (parzzley.sync.engine._PerSiteItemsBook attribute), 111
 _abc_impl (parzzley.sync.engine.events_impl.DirEvent attribute), 119
 _abc_impl (parzzley.sync.engine.events_impl.ItemEvent attribute), 119
 _abc_impl (parzzley.sync.engine.events_impl.StreamEvent attribute), 124
 _abc_impl (parzzley.sync.engine.events_impl.SyncRunEvent attribute), 116
 _abc_impl (parzzley.sync.engine.events_impl.SyncRunEvent._StreamSupport attribute), 116
 _abc_impl (parzzley.sync.engine.events_impl._Logger attribute), 124
 _abc_impl (parzzley.sync.logger.Exclude.Condition attribute), 128
 _abc_impl (parzzley.sync.logger.Formatter attribute), 127
 _abc_impl (parzzley.sync.logger.Logger attribute), 125
 _abc_impl (parzzley.sync.logger.Out attribute), 127
 _abc_impl (parzzley.sync.logger.StreamSlipPastInfo attribute), 129
 _abc_impl (parzzley.sync.logger.conditions.ConditionAnyOf attribute), 129
 _abc_impl (parzzley.sync.logger.conditions.ConditionNegate attribute), 129
 _abc_impl (parzzley.sync.logger.conditions.SimpleCondition attribute), 129
 _action() (parzzley.service.Service._LoopThread method), 74
 _action() (parzzley.service.Service._MonitorSiteForChangesThread method), 76
 _action() (parzzley.service.Service._VolumeThread method), 75
 _all_event_handlers() (parzzley.sync.aspect.Aspect method), 77
 _cancel() (parzzley.sync.engine.Engine method), 106
 _conflict_directory_item() (in module parzzley.builtin.aspects.conflicts), 30
 _conflicts_storage_site() (in module parzzley.builtin.aspects.conflicts), 30
 _data_for_site() (parzzley.sync.engine.events_impl.SyncRunEvent method), 114
 _data_key_data() (parzzley.sync.engine.events_impl.SyncRunEvent method), 114
 _dep_list() (in module parzzley.sync.aspect), 78
 _dep_str() (in module parzzley.sync.aspect), 78
 _descendant() (parzzley.fs.Item method), 59
 _descendant() (parzzley.fs._Item method), 60
 _exec() (parzzley.sync.engine.Engine method), 104
 _file_format() (in module parzzley.config.file_formats), 55
 _find_unsafe() (in module parzzley.builtin.fs), 45
 _get_data() (parzzley.sync.engine.events_impl.SyncRunEvent method), 114

<code>_item_info_from_serializable_data()</code>	(parzzley.sync.engine._GlobalItemsBook class method), 109	<code>ley.builtin.fs.ShellBasedBackend.Shell method)</code> , 44
<code>_item_info_from_serializable_data()</code>	(parzzley.sync.engine._ItemsBookBase class method), 107	<code>_shell_cmdline()</code> (parzzley.builtin.fs.ssh.Backend._Shell method), 48
<code>_item_info_from_serializable_data()</code>	(parzzley.sync.engine._PerSiteItemsBook class method), 110	<code>_shell_quote()</code> (in module parzzley.builtin.fs), 45
<code>_item_info_to_serializable_data()</code>	(parzzley.sync.engine._GlobalItemsBook class method), 109	<code>_site()</code> (parzzley.sync.aspect.only_if.IsItemTypeHereSupportedByStream method), 103
<code>_item_info_to_serializable_data()</code>	(parzzley.sync.engine._ItemsBookBase class method), 107	<code>_site()</code> (parzzley.sync.aspect.only_if.IsItemTypeOnStreamMasterSiteSupportedByStream method), 104
<code>_item_info_to_serializable_data()</code>	(parzzley.sync.engine._PerSiteItemsBook class method), 110	<code>_site()</code> (parzzley.sync.aspect.only_if.ItemExistsHere method), 102
<code>_items_books</code> (parzzley.sync.engine.events_impl.SyncRunEvent property), 114		<code>_site()</code> (parzzley.sync.aspect.only_if.ItemExistsOnMasterSite method), 102
<code>_log()</code> (parzzley.sync.engine.events_impl._Logger method), 124		<code>_site()</code> (parzzley.sync.aspect.only_if.ItemHereIsType method), 102
<code>_log()</code> (parzzley.sync.logger.Logger method), 124		<code>_site()</code> (parzzley.sync.aspect.only_if.ItemOnMasterSiteIsType method), 103
<code>_manager_control_site()</code> (parzzley.sync.manager.Manager method), 132		<code>_site()</code> (parzzley.sync.aspect.only_if._IsItemTypeSupportedByStreamCondition method), 101
<code>_master_site()</code> (parzzley.sync.aspect.events.item._EventAfterMainStreamMasterSite method), 83		<code>_site()</code> (parzzley.sync.aspect.only_if._ItemExistsCondition method), 101
<code>_master_site()</code> (parzzley.sync.aspect.events.item.stream._Event method), 88		<code>_site()</code> (parzzley.sync.aspect.only_if._ItemIsTypeCondition method), 101
<code>_master_site()</code> (parzzley.sync.engine.events_impl.ItemEvent method), 118		<code>_sleep()</code> (parzzley.service.Service._LoopThread method), 74
<code>_master_site()</code> (parzzley.sync.engine.events_impl.StreamEvent method), 123		<code>_state_args_from_serializable_data()</code> (parzzley.sync.engine._GlobalItemsBook class method), 109
<code>_per_item_data</code> (parzzley.sync.engine.events_impl.ItemEvent property), 117		<code>_state_args_from_serializable_data()</code> (parzzley.sync.engine._ItemsBookBase class method), 108
<code>_per_stream_data</code> (parzzley.sync.engine.events_impl.StreamEvent property), 120		<code>_state_args_to_serializable_data()</code> (parzzley.sync.engine._GlobalItemsBook method), 109
<code>_per_sync_run_data</code> (parzzley.sync.engine.events_impl.SyncRunEvent property), 114		<code>_state_args_to_serializable_data()</code> (parzzley.sync.engine._ItemsBookBase method), 108
<code>_set_data()</code> (parzzley.sync.engine.events_impl.SyncRunEvent method), 114		<code>_stopping()</code> (parzzley.service.Service._LoopThread method), 74
<code>_setup_logging()</code> (in module parzzley.parzzley_cli), 138		<code>_stopping()</code> (parzzley.service.Service._VolumeThread method), 75
<code>_shell()</code> (parzzley.builtin.fs.ShellBasedBackend method), 42		<code>_stream()</code> (parzzley.builtin.fs.ShellBasedBackend._SiteBackend._FileMain method), 44
<code>_shell()</code> (parzzley.builtin.fs.ssh.Backend method), 48		<code>_stream()</code> (parzzley.builtin.fs.ShellBasedBackend._SiteBackend._FileMain method), 44
<code>_shell_cmdline()</code> (parzz-		<code>_stream()</code> (parzzley.builtin.fs.local.Backend._SiteBackend._FileMain method), 47
		<code>_stream()</code> (parzzley.builtin.fs.local.Backend._SiteBackend._FileMain method), 48
		<code>_stream()</code> (parzzley.fs.stream.MemoryReadStreamable method), 72
		<code>_stream()</code> (parzzley.fs.stream.MemoryWriteStreamable method), 72

[_stream\(\)](#) (*parzzley.fs.stream.ReadStreamable* method), 71
[_stream\(\)](#) (*parzzley.fs.stream.WriteStreamable* method), 71
[_stream\(\)](#) (*parzzley.sync.aspect.events.item.stream.CollectSourceStreamable* method), 93
[_stream_support\(\)](#) (*parzzley.sync.engine.events_impl.SyncRunEvent* method), 115
[_updated_cookie\(\)](#) (*parzzley.sync.engine.events_impl.StreamEvent* method), 123

A

[add_child_names\(\)](#) (*parzzley.sync.aspect.events.item.dir.List* method), 86
[add_child_names\(\)](#) (*parzzley.sync.engine.events_impl.DirEvent* method), 119
[add_conflict\(\)](#) (*parzzley.sync.aspect.events.item.stream.DetermineConflicts* method), 94
[add_conflict\(\)](#) (*parzzley.sync.engine.events_impl.StreamEvent* method), 122
[add_destination_streamable\(\)](#) (*parzzley.sync.aspect.events.item.stream.CollectDestinationStreamable* method), 96
[add_destination_streamable\(\)](#) (*parzzley.sync.engine.events_impl.StreamEvent* method), 121
[add_stream_pipe\(\)](#) (*parzzley.sync.aspect.events.item.stream.CollectSourceStreamable* method), 93
[add_stream_pipe\(\)](#) (*parzzley.sync.engine.events_impl.StreamEvent* method), 121
[add_working_item\(\)](#) (*parzzley.sync.aspect.events.item.SetUpWorkingItems* method), 84
[add_working_item\(\)](#) (*parzzley.sync.engine.events_impl.ItemEvent* method), 117
[add_working_item_for_revision\(\)](#) (*parzzley.builtin.aspects.revision_tracking.RevisionTracking* method), 37
[afterwards](#) (*parzzley.sync.engine.event_runner.Runner.Handler* attribute), 112
[afterwards_optional](#) (*parzzley.sync.engine.event_runner.Runner.Handler* attribute), 112
[ALIEN](#) (*parzzley.fs.ItemType* attribute), 58

[alien_detect_changes\(\)](#) (*parzzley.builtin.aspects.main_stream.MainStreamSynchronization* method), 33
[all_conflicts](#) (*parzzley.sync.aspect.events.item.SkipDueToConflicts* property), 83
[all_conflicts](#) (*parzzley.sync.engine.events_impl.ItemEvent* property), 117
[all_log_entries](#) (*parzzley.sync.engine.events_impl.SyncRunEvent* property), 114
[all_sites](#) (*parzzley.sync.aspect.events.sync_run.Event* property), 97
[all_sites](#) (*parzzley.sync.engine.events_impl.SyncRunEvent* property), 114
[apply_conflict_resolution\(\)](#) (*parzzley.builtin.aspects.conflicts.ApplyConflictResolution* method), 30
[ApplyConflictResolution](#) (class in *parzzley.builtin.aspects.conflicts*), 30
[ApplyConflictResolution](#) (class in *parzzley.sync.aspect.events.item.stream*), 95
[ApplyUpdate](#) (class in *parzzley.sync.aspect.events.item*), 84
[arguments](#) (*parzzley.config.Aspect* property), 51
[arguments](#) (*parzzley.config.Logging.Exclude.Condition* property), 53
[arguments](#) (*parzzley.config.Logging.Formatter* property), 53
[arguments](#) (*parzzley.config.Logging.Out* property), 52
[arguments](#) (*parzzley.config.Site* property), 51
[Aspect](#) (class in *parzzley.config*), 50
[Aspect](#) (class in *parzzley.sync.aspect*), 77
[aspect_type_by_name\(\)](#) (in module *parzzley.sync.aspect*), 78
[aspects](#) (*parzzley.config.Site* property), 51
[aspects](#) (*parzzley.config.Volume* property), 52
[aspects\(\)](#) (*parzzley.sync.run.SyncRun* method), 135
[aspects\(\)](#) (*parzzley.sync.Volume* method), 77
[attach_site\(\)](#) (*parzzley.builtin.aspects.attach_sites.AttachSites* method), 28
[AttachSites](#) (class in *parzzley.builtin.aspects.attach_sites*), 28
[available_nodes\(\)](#) (*parzzley.sync.engine.event_runner.Runner.DependencyControlledHandler* method), 111

B

[Backend](#) (class in *parzzley.builtin.fs.local*), 46
[Backend](#) (class in *parzzley.builtin.fs.ssh*), 48
[Backend](#) (class in *parzzley.fs*), 65
[Backend._Shell](#) (class in *parzzley.builtin.fs.ssh*), 48

Backend._SiteBackend (class in parzz- Close (class in parzzley.sync.aspect.events.sync_run),
ley.builtin.fs.local), 46 100

Backend._SiteBackend._FileMainStreamReadStreamableCollectDestination() (parzz-
(class in parzzley.builtin.fs.local), 47 ley.builtin.aspects.streaming.WorkingItem

Backend._SiteBackend._FileMainStreamReadStreamable._ReadStream() (class in parzzley.builtin.fs.local), 47 collect_destination() (parzz-
(class in parzzley.builtin.fs.local), 47 method), 41

Backend._SiteBackend._FileMainStreamWriteStreamable ley.builtin.aspects.xattrs.XattrSynchronization
(class in parzzley.builtin.fs.local), 47 method), 41

Backend._SiteBackend._FileMainStreamWriteStreamableCollectWriteSet() (parzz-
(class in parzzley.builtin.fs.local), 48 ley.builtin.aspects.xattrs.XattrSynchronization
method), 41

Backend.SiteBackend (class in parzzley.fs), 65

backend_by_kind() (in module parzzley.fs), 68

beforehand (parzzley.sync.engine.event_runner.Runner._Handler ley.sync.aspect.events.item.stream), 95
attribute), 112

beforehand_optional (parzz- CollectDestinationStreamables (class in parzz-
ley.sync.engine.event_runner.Runner._Handler ley.sync.aspect.events.item.stream), 92
attribute), 112

buffer (parzzley.builtin.aspects.conflicts._Source CollectSourceStreamables (class in parzz-
attribute), 31 ley.sync.aspect.events.item.stream), 92

C CollectSourceStreamables.FullContentPipe
(class in parzz-
ley.sync.aspect.events.item.stream), 93

cancel() (parzzley.sync.control.SyncControl method), CollectSourceStreamables.Pipe (class in parzz-
130 ley.sync.aspect.events.item.stream), 92

changed_dangerously_late() (parzz- combination (parzzley.config.Logging.Exclude.CombinedCondition
ley.sync.aspect.events.item.stream.DetermineConflicts property), 53
method), 94

changed_dangerously_late() (parzz- Commands (class in parzzley.parzzley_cli), 137
ley.sync.engine.events_impl.StreamEvent commit() (parzzley.builtin.fs.local.Backend._SiteBackend._FileMainStream
method), 122 method), 48

changed_in_sync_run_no (parzz- commit() (parzzley.builtin.fs.ShellBasedBackend._SiteBackend._FileMainS
ley.sync.engine._PerSiteItemsBook._ItemInfo method), 44
attribute), 111

child_names (parzzley.sync.engine.events_impl.DirEvent commit() (parzzley.fs.stream.MemoryWriteStreamable._WriteStream
property), 119 method), 72

child_names() (parzz- commit() (parzzley.fs.stream.WriteStream method), 70
ley.builtin.fs.local.Backend._SiteBackend commit_update() (parzz-
method), 47 ley.builtin.aspects.streaming.WorkingItem
method), 40

child_names() (parzz- comparator (parzzley.sync.aspect.events.item.stream.DetermineComparato
ley.builtin.fs.ShellBasedBackend._SiteBackend property), 89
method), 43

child_names() (parzzley.fs._SubTreeSiteBackend comparator (parzzley.sync.engine.events_impl.StreamEvent
method), 69 property), 123

child_names() (parzzley.fs.Backend.SiteBackend comparator() (parzzley.builtin.aspects.posix_attributes.PosixAttributesSy
method), 66 method), 34

child_names() (parzzley.fs.Site method), 62 ComputeMasterSiteTable (class in parzz-
child_names() (parzzley.fs.Site method), 62 ley.builtin.aspects.streaming), 38

cleanup_conflicts_storage_site() (parzz- Condition (class in parzzley.sync.aspect.only_if), 100
ley.builtin.aspects.conflicts.TrackConflicts condition (parzzley.config.Logging.Exclude.NegateCondition
method), 30 property), 54

cleanup_trash_bin() (parzz- ConditionAllOf (class in parzz-
ley.builtin.aspects.remove.CleanupTrashBin ley.sync.logger.conditions), 129
method), 36

CleanupTrashBin (class in parzz- ConditionAnyOf (class in parzz-
ley.builtin.aspects.remove), 36 ley.sync.logger.conditions), 129

ConditionNegate (class in parzz-
ley.sync.logger.conditions), 129

conditions (parzzley.config.Logging.Exclude prop-

erty), 53

conditions (parzzley.config.Logging.Exclude.CombinedCondition property), 54

conditions (parzzley.sync.engine.event_runner.Runner._Hooks attribute), 112

Configuration (class in parzzley.config), 54

conflict_resolution (parzz-
ley.sync.aspect.events.item.stream.ApplyConflictResolution property), 95

conflict_resolution (parzz-
ley.sync.engine.events_impl.StreamEvent property), 122

conflicts (parzzley.sync.aspect.events.item.stream.ApplyConflictResolution property), 95

conflicts (parzzley.sync.aspect.events.item.stream.DetermineConflicts property), 94

conflicts (parzzley.sync.aspect.events.item.stream.TryResolveConflicts property), 95

conflicts (parzzley.sync.engine.events_impl.StreamEvent property), 122

connect() (parzzley.builtin.fs.local.Backend method), 46

connect() (parzzley.builtin.fs.ShellBasedBackend method), 41

connect() (parzzley.fs.Backend method), 67

connect() (parzzley.fs.SiteSetup method), 67

connected_sites (parzz-
ley.sync.manager.Manager.PreparedSyncSetup attribute), 133

content_version() (parzz-
ley.sync.aspect.events.item._EventAfterMainStreamMasterSiteDetermined method), 83

content_version() (parzz-
ley.sync.aspect.events.item.stream._EventAfterMasterSiteDetermined method), 89

content_version() (parzz-
ley.sync.engine.events_impl.ItemEvent method), 118

content_version() (parzz-
ley.sync.engine.events_impl.StreamEvent method), 123

control_site() (parzzley.fs.Site method), 64

CONTROL_SITE_ROOT_DIR_NAME (parzzley.fs.Backend attribute), 65

cookie() (parzzley.builtin.fs.local.Backend._SiteBackend method), 46

cookie() (parzzley.builtin.fs.ShellBasedBackend._SiteBackend method), 43

cookie() (parzzley.fs._SubTreeSiteBackend method), 68

cookie() (parzzley.fs.Backend.SiteBackend method), 66

cookie() (parzzley.fs.Site method), 61

cookies_are_move_stable() (parzz-
ley.sync.aspect.events.sync_run._EventAfterStreamSupportInfo method), 99

cookies_are_move_stable() (parzz-
ley.sync.aspect.events.sync_run.ValidateStreamSupport method), 98

cookies_are_move_stable() (parzz-
ley.sync.engine.events_impl.SyncRunEvent method), 115

cookies_are_move_stable() (parzz-
ley.sync.engine.events_impl.SyncRunEvent._StreamSupportInfo method), 116

create_dir() (parzzley.builtin.aspects.directory.DirectoryCreation method), 32

create_item() (parzz-
ley.builtin.fs.local.Backend._SiteBackend method), 46

create_item() (parzz-
ley.builtin.fs.ShellBasedBackend._SiteBackend method), 42

create_item() (parzzley.fs._SubTreeSiteBackend method), 69

create_item() (parzzley.fs.Backend.SiteBackend method), 65

create_item() (parzzley.fs.Site method), 60

create_temp_item() (parzzley.fs.Site method), 64

D

Data (class in parzzley.sync.aspect.events), 79

DEBUG (parzzley.sync.logger.Severity attribute), 124

debug() (parzzley.sync.logger.Logger method), 125

decide_to_skip() (parzz-
ley.builtin.aspects.exclude_paths.ExcludePaths method), 37

decide_to_skip() (parzz-
ley.builtin.aspects.streaming.SkipItemIfNoUpToDateSitesAreConfirmed method), 37

DecideToSkip (class in parzz-
ley.sync.aspect.events.item), 81

DefaultBase (class in parzz-
ley.builtin.aspects.default_sync), 31

DefaultSync (class in parzz-
ley.builtin.aspects.default_sync), 31

depends_on_handlers (parzz-
ley.sync.engine.event_runner.Runner._HandlerNode attribute), 112

description (parzzley.sync.aspect.events.item.SkipDueToConflicts._Conflict attribute), 82

description (parzzley.sync.aspect.events.item.stream.DetermineConflicts attribute), 94

deserialize_bytes_dict() (in module parzz-
ley.fs.utils), 73

destination_streamables() (parzz-
ley.sync.aspect.events.item.stream._WithDestinationStreamableG method), 87

destination_streamables() (parzz-
ley.sync.engine.events_impl.StreamEvent method), 99

method), 121
 details (parzzley.sync.aspect.events.item.SkipDueToConflicts._Conflict attribute), 82
 details (parzzley.sync.aspect.events.item.stream.DetermineItemTypes attribute), 94
 detect_changes() (parzz- ley.builtin.aspects.directory.MarkChangedIfItemIsDirectoryByWasNotBefore method), 31
 detect_changes() (parzz- ley.builtin.aspects.main_stream.MainStreamSync/DetermineType method), 33
 detect_changes() (parzz- ley.builtin.aspects.posix_attributes.PosixAttributesSynchronizatio method), 34
 detect_changes() (parzz- ley.builtin.aspects.pull_and_purge.PullAndPurgeSyncSink method), 34
 detect_changes() (parzz- ley.builtin.aspects.remove.DetectRemoval method), 36
 detect_changes() (parzz- ley.builtin.aspects.xattrs.XattrSynchronizatio method), 40
 DetectChanges (class in parzz- ley.sync.aspect.events.item.stream), 90
 DetectContentConflicts (class in parzz- ley.builtin.aspects.conflicts), 29
 DetectItemTypeConflicts (class in parzz- ley.builtin.aspects.conflicts), 29
 DetectRemoval (class in parzz- ley.builtin.aspects.remove), 36
 determine_conflicts() (parzz- ley.builtin.aspects.conflicts.DetectContentConflicts method), 29
 determine_conflicts() (parzz- ley.builtin.aspects.conflicts.DetectItemTypeConflicts method), 29
 determine_cookie() (parzz- ley.builtin.aspects.streaming.GetCookie method), 38
 determine_item_type() (parzz- ley.builtin.aspects.item_type.DetermineItemTypes method), 33
 DetermineComparator (class in parzz- ley.sync.aspect.events.item.stream), 89
 DetermineConflicts (class in parzz- ley.sync.aspect.events.item.stream), 94
 DetermineConflicts._Conflict (class in parzz- ley.sync.aspect.events.item.stream), 94
 DetermineCookie (class in parzz- ley.sync.aspect.events.item.stream), 90
 determined_supported_streams() (parzz- ley.sync.aspect.events.sync_run.ValidateStreamSupport method), 98
 determined_supported_streams() (parzz- ley.sync.engine.events_impl.SyncRunEvent method), 115
 DetermineConflicts (class in parzz- ley.builtin.aspects.item_type), 33
 DetermineMasterSiteTable (class in parzz- ley.sync.aspect.events.item.stream), 91
 DetermineStreamSupport (class in parzz- ley.sync.aspect.events.sync_run), 97
 DetermineType (class in parzz- ley.sync.aspect.events.item), 82
 DIRECTORY (parzzley.fs.ItemType attribute), 58
 DirectoryCreation (class in parzz- ley.builtin.aspects.directory), 32
 DirectRemove (class in parzzley.builtin.aspects.remove), 35
 DirEvent (class in parzzley.sync.engine.events_impl), 119
 disable_change_detection() (parzz- ley.builtin.aspects.posix_attributes.PosixAttributesSynchronizatio method), 34
 disable_change_detection() (parzz- ley.builtin.aspects.xattrs.XattrSynchronizatio method), 41
 disable_change_detection() (parzz- ley.sync.aspect.events.item.stream.CollectDestinationStreamables method), 96
 disable_change_detection() (parzz- ley.sync.engine.events_impl.StreamEvent method), 121
 disconnect() (parzzley.builtin.fs.local.Backend method), 46
 disconnect() (parzzley.builtin.fs.ShellBasedBackend method), 41
 disconnect() (parzzley.builtin.fs.ShellBasedBackend._SiteBackend method), 43
 disconnect() (parzzley.builtin.fs.ShellBasedBackend._SiteBackend._Shell method), 42
 disconnect() (parzzley.fs.Backend method), 67
E
 emit() (parzzley.sync.logger.Logging method), 126
 emit_text() (parzzley.builtin.log_outs.shell.Out method), 50
 emit_text() (parzzley.sync.logger.Out method), 127
 Engine (class in parzzley.sync.engine), 104
 Engine._Canceled, 106
 Engine._SyncControl (class in parzzley.sync.engine), 106
 Entry (class in parzzley.sync.logger), 125
 error (parzzley.sync.aspect.events.sync_run.Close property), 100
 error (parzzley.sync.engine.events_impl.SyncRunEvent property), 114

event_handler() (in module parzzley.sync.aspect), 77
 event_type (parzzley.sync.engine.event_runner.Runner._Handler attribute), 112
 Exclude (class in parzzley.sync.logger), 127
 exclude (parzzley.config.Logging property), 52
 Exclude.Condition (class in parzzley.sync.logger), 127
 ExcludePaths (class in parzzley.builtin.aspects.exclude_paths), 32
 exec() (parzzley.builtin.fs.ShellBasedBackend.Shell method), 45
 exec_raw() (parzzley.builtin.fs.ShellBasedBackend.Shell method), 45
 exit_code (parzzley.builtin.fs.ShellBasedBackend.Shell.ExecutionResult attribute), 44

F

FATAL (parzzley.sync.logger.Severity attribute), 124
 fatal() (parzzley.sync.logger.Logger method), 125
 FILE (parzzley.fs.ItemType attribute), 58
 FileFormat (class in parzzley.config.file_formats), 54
 Finish (class in parzzley.sync.aspect.events.sync_run), 99
 finished (parzzley.builtin.aspects.conflicts._Source attribute), 31
 for_config_directory() (parzzley.sync.manager.Manager static method), 131
 format() (parzzley.builtin.log_formatters.html.Formatter method), 49
 format() (parzzley.builtin.log_formatters.json.Formatter method), 50
 format() (parzzley.sync.logger.Formatter method), 126
 Formatter (class in parzzley.builtin.log_formatters.html), 49
 Formatter (class in parzzley.builtin.log_formatters.json), 50
 Formatter (class in parzzley.sync.logger), 126
 formatter (parzzley.config.Logging property), 52
 formatter_type_by_kind() (in module parzzley.sync.logger), 128
 from_serializable_data() (parzzley.sync.engine._ItemsBookBase class method), 107
 func (parzzley.sync.engine.event_runner.Runner._Handler attribute), 112

G

get() (parzzley.sync.aspect.events.Data method), 79
 get() (parzzley.sync.engine.events_impl.SyncRunEvent.Event method), 116
 GetCookie (class in parzzley.builtin.aspects.streaming), 38

GlobalStreamSupport (class in parzzley.builtin.aspects.streaming), 37
 handler (parzzley.sync.engine.event_runner.Runner._HandlerNode attribute), 112

H

I

id (parzzley.sync.manager.Manager property), 131
 id (parzzley.sync.run.SyncRun property), 135
 INFO (parzzley.sync.logger.Severity attribute), 124
 info() (parzzley.sync.logger.Logger method), 125
 init() (parzzley.builtin.aspects.revision_tracking.RevisionTracking method), 37
 interval (parzzley.config.Volume property), 52
 is_cancel_requested (parzzley.sync.control.SyncControl property), 130
 is_change_detection_disabled() (parzzley.sync.aspect.events.item.stream.TransferUpdate method), 96
 is_change_detection_disabled() (parzzley.sync.engine.events_impl.StreamEvent method), 121
 is_empty (parzzley.sync.engine.event_runner.Runner._DependencyControl property), 111
 is_finished (parzzley.sync.control.SyncControl property), 130
 is_finished (parzzley.sync.engine.Engine._SyncControl property), 107
 is_item_marked_to_skip (parzzley.sync.aspect.events.item.DecideToSkip property), 81
 is_item_marked_to_skip (parzzley.sync.engine.events_impl.ItemEvent property), 118
 is_item_marked_to_skip_permanently (parzzley.sync.aspect.events.item.DecideToSkip property), 81
 is_item_marked_to_skip_permanently (parzzley.sync.engine.events_impl.ItemEvent property), 118
 is_marked_full_retry_needed (parzzley.sync.aspect.events.item._Event property), 80
 is_marked_full_retry_needed (parzzley.sync.engine.events_impl.ItemEvent property), 118
 is_met() (parzzley.sync.aspect.only_if_IsItemTypeSupportedByStreamControl method), 101
 is_met() (parzzley.sync.aspect.only_if_ItemExistsCondition method), 100
 is_met() (parzzley.sync.aspect.only_if_ItemIsTypeCondition method), 101

[is_met\(\)](#) ([parzzley.sync.aspect.only_if.Condition](#) method), 100
[is_met\(\)](#) ([parzzley.sync.aspect.only_if.IsStream](#) method), 103
[is_met\(\)](#) ([parzzley.sync.aspect.only_if.ThisIsMasterSite](#) method), 103
[is_met\(\)](#) ([parzzley.sync.logger.conditions.ConditionAllOf](#) method), 129
[is_met\(\)](#) ([parzzley.sync.logger.conditions.ConditionAnyOf](#) method), 129
[is_met\(\)](#) ([parzzley.sync.logger.conditions.ConditionNegate](#) method), 129
[is_met\(\)](#) ([parzzley.sync.logger.conditions.SimpleCondition](#) method), 129
[is_met\(\)](#) ([parzzley.sync.logger.Exclude](#) method), 127
[is_met\(\)](#) ([parzzley.sync.logger.Exclude.Condition](#) method), 127
[is_site_marked_as_changed\(\)](#) ([parzzley.sync.aspect.events.item.stream.DetermineMasterSiteType](#) method), 91
[is_site_marked_as_changed\(\)](#) ([parzzley.sync.engine.events_impl.StreamEvent](#) method), 120
[is_site_marked_as_needing_refreshed\(\)](#) ([parzzley.sync.aspect.events.item.stream.DetermineMasterSiteType](#) method), 91
[is_site_marked_as_needing_refreshed\(\)](#) ([parzzley.sync.engine.events_impl.StreamEvent](#) method), 120
[is_sub_site_of\(\)](#) ([parzzley.fs._SubTreeSiteBackend](#) method), 69
[is_sub_site_of\(\)](#) ([parzzley.fs.Backend.SiteBackend](#) method), 67
[IsItemTypeHereSupportedByStream](#) (class in [parzzley.sync.aspect.only_if](#)), 103
[IsItemTypeOnStreamMasterSiteSupportedByStream](#) (class in [parzzley.sync.aspect.only_if](#)), 104
[IsStream](#) (class in [parzzley.sync.aspect.only_if](#)), 103
[Item](#) (class in [parzzley.fs](#)), 59
[item](#) ([parzzley.sync.aspect.events.item._Event](#) property), 80
[item](#) ([parzzley.sync.engine.events_impl.ItemEvent](#) property), 117
[item](#) ([parzzley.sync.logger.Entry](#) property), 126
[item\(\)](#) (in module [parzzley.fs](#)), 60
[item_exists\(\)](#) ([parzzley.fs.Site](#) method), 62
[item_type](#) ([parzzley.sync.engine._PerSiteItemsBook._ItemInfo](#) attribute), 111
[item_type\(\)](#) ([parzzley.fs.Site](#) method), 61
[item_type\(\)](#) ([parzzley.sync.aspect.events.item._EventAfterItemTypeKnown](#) method), 81
[item_type\(\)](#) ([parzzley.sync.engine.events_impl.ItemEvent](#) method), 118
[item_type_after_last_sync\(\)](#) ([parzzley.sync.aspect.events.item.stream.DetectChanges](#) method), 90
[item_type_after_last_sync\(\)](#) ([parzzley.sync.engine._PerSiteItemsBook](#) method), 110
[item_type_after_last_sync\(\)](#) ([parzzley.sync.engine.events_impl.ItemEvent](#) method), 118
[item_type_by_cookie\(\)](#) ([parzzley.builtin.fs.local.Backend._SiteBackend](#) method), 46
[item_type_by_cookie\(\)](#) ([parzzley.builtin.fs.ShellBasedBackend._SiteBackend](#) method), 42
[item_type_by_cookie\(\)](#) ([parzzley.fs._SubTreeSiteBackend](#) method), 68
[item_type_by_cookie\(\)](#) ([parzzley.fs.Backend.SiteBackend](#) method), 66
[item_type_by_cookie\(\)](#) ([parzzley.fs.Site](#) method), 61
[ItemEvent](#) (class in [parzzley.sync.engine.events_impl](#)), 116
[ItemExistsHere](#) (class in [parzzley.sync.aspect.only_if](#)), 101
[ItemExistsOnMasterSite](#) (class in [parzzley.sync.aspect.only_if](#)), 102
[ItemHereIsType](#) (class in [parzzley.sync.aspect.only_if](#)), 102
[ItemOnMasterSiteIsType](#) (class in [parzzley.sync.aspect.only_if](#)), 102
[ItemType](#) (class in [parzzley.fs](#)), 58
[Iterated](#) (class in [parzzley.sync.aspect.events.item.dir](#)), 86

K

[kind](#) ([parzzley.config.Logging.Formatter](#) property), 53
[kind](#) ([parzzley.config.Logging.Out](#) property), 52
[kind](#) ([parzzley.config.Site](#) property), 51

L

[last_change_run_sn\(\)](#) ([parzzley.sync.engine._PerSiteItemsBook](#) method), 110
[last_successful_sync_no\(\)](#) ([parzzley.sync.engine._GlobalItemsBook](#) method), 108
[last_successful_sync_run_no](#) ([parzzley.sync.engine._GlobalItemsBook._ItemInfo](#) attribute), 109
[List](#) (class in [parzzley.sync.aspect.events.item.dir](#)), 86
[list_dir\(\)](#) ([parzzley.builtin.aspects.directory.ListDirectory](#) method), 31
[ListDirectory](#) (class in [parzzley.builtin.aspects.directory](#)), 31
[load_aspect\(\)](#) (in module [parzzley.config.loader](#)), 57

[load_builtin_implementations\(\)](#) (in module *parzzley.builtin*), 27
[load_implementations_from_package\(\)](#) (in module *parzzley.sync*), 77
[load_log_exclusion\(\)](#) (in module *parzzley.config.loader*), 57
[load_log_exclusion_condition\(\)](#) (in module *parzzley.config.loader*), 58
[load_log_formatter\(\)](#) (in module *parzzley.config.loader*), 57
[load_logger_out\(\)](#) (in module *parzzley.config.loader*), 57
[load_logging\(\)](#) (in module *parzzley.config.loader*), 57
[load_site_setup\(\)](#) (in module *parzzley.config.loader*), 57
[load_volume\(\)](#) (in module *parzzley.config.loader*), 57
[log](#) (*parzzley.sync.aspect.events.sync_run._Event* property), 96
[log](#) (*parzzley.sync.engine.events_impl.SyncRunEvent* property), 114
[log_removal\(\)](#) (*parzzley.builtin.aspects.remove.DirectRemove* method), 35
[log_removal\(\)](#) (*parzzley.builtin.aspects.remove.TrashRemove* method), 36
[log_severity\(\)](#) (in module *parzzley.config.file_formats*), 55
[log_update\(\)](#) (*parzzley.builtin.aspects.streaming.UpdateTransfer* method), 39
[Logger](#) (class in *parzzley.sync.logger*), 124
[Logging](#) (class in *parzzley.config*), 52
[Logging](#) (class in *parzzley.sync.logger*), 126
[Logging.Exclude](#) (class in *parzzley.config*), 53
[Logging.Exclude.BaseCondition](#) (class in *parzzley.config*), 53
[Logging.Exclude.CombinedCondition](#) (class in *parzzley.config*), 53
[Logging.Exclude.Condition](#) (class in *parzzley.config*), 53
[Logging.Exclude.NegateCondition](#) (class in *parzzley.config*), 54
[Logging.Formatter](#) (class in *parzzley.config*), 53
[Logging.Out](#) (class in *parzzley.config*), 52
[loggings](#) (*parzzley.config.Configuration* property), 54
[loggings](#) (*parzzley.sync.manager.Manager.PreparedSyncSetup* attribute), 134

M
[main\(\)](#) (in module *parzzley.parzzley_cli*), 137
[MainStreamSynchronization](#) (class in *parzzley.builtin.aspects.main_stream*), 33
[Manager](#) (class in *parzzley.sync.manager*), 130
[Manager.PreparedSyncSetup](#) (class in *parzzley.sync.manager*), 133
[Manager.VolumeStateVariable](#) (class in *parzzley.sync.manager*), 134
[manager_control_site\(\)](#) (*parzzley.sync.run.SyncRun* method), 136
[manager_id](#) (*parzzley.sync.run.SyncRun* property), 135
[manager_var_dir](#) (*parzzley.sync.run.SyncRun* property), 135
[mark_all_last_involved_sites_changed_if_this_is_not_one_of](#) (*parzzley.sync.aspect.events.item.stream.DetectChanges* method), 91
[mark_all_last_involved_sites_changed_if_this_is_not_one_of](#) (*parzzley.sync.engine.events_impl.StreamEvent* method), 121
[mark_changed\(\)](#) (*parzzley.sync.aspect.events.item.stream.DetectChanges* method), 90
[mark_changed\(\)](#) (*parzzley.sync.engine.events_impl.StreamEvent* method), 121
[mark_done\(\)](#) (*parzzley.sync.engine.event_runner.Runner._DependencyCor* method), 111
[mark_effective\(\)](#) (*parzzley.sync.aspect.events.sync_run._Event* method), 97
[mark_effective\(\)](#) (*parzzley.sync.engine.events_impl.SyncRunEvent* method), 114
[mark_full_retry_needed\(\)](#) (*parzzley.sync.aspect.events.item._Event* method), 80
[mark_full_retry_needed\(\)](#) (*parzzley.sync.engine.events_impl.ItemEvent* method), 118
[mark_refresh_needed\(\)](#) (*parzzley.sync.aspect.events.item.stream.DetectChanges* method), 91
[mark_refresh_needed\(\)](#) (*parzzley.sync.engine.events_impl.StreamEvent* method), 122
[mark_skipped_now\(\)](#) (*parzzley.sync.engine._GlobalItemsBook* method), 108
[mark_successfully_synced\(\)](#) (*parzzley.sync.engine._GlobalItemsBook* method), 108
[MarkChangedIfItemIsDirectoryButWasNotBefore](#) (class in *parzzley.builtin.aspects.directory*), 31
[master_site](#) (*parzzley.sync.aspect.events.item._EventAfterMainStreamMa* property), 83
[master_site](#) (*parzzley.sync.aspect.events.item.stream._EventAfterMasterS* property), 89
[master_site_table\(\)](#) (*parzz-*

ley.builtin.aspects.streaming.ComputeMasterSiteTable (method), 38
 MasterSiteDetermined (class in *parzzley.sync.aspect.events.item.stream*), 92
 max_severity (*parzzley.config.Logging* property), 52
 MemoryReadStreamable (class in *parzzley.fs.stream*), 72
 MemoryReadStreamable._ReadStream (class in *parzzley.fs.stream*), 72
 MemoryWriteStreamable (class in *parzzley.fs.stream*), 72
 MemoryWriteStreamable._WriteStream (class in *parzzley.fs.stream*), 72
 message (*parzzley.sync.logger.Entry* property), 126
 min_severity (*parzzley.config.Logging* property), 52
 module
 parzzley, 27
 parzzley.asset, 27
 parzzley.asset.data, 27
 parzzley.asset.project_info, 27
 parzzley.builtin, 27
 parzzley.builtin.aspects, 27
 parzzley.builtin.aspects.attach_sites, 28
 parzzley.builtin.aspects.conflicts, 29
 parzzley.builtin.aspects.default_sync, 31
 parzzley.builtin.aspects.directory, 31
 parzzley.builtin.aspects.exclude_paths, 32
 parzzley.builtin.aspects.item_type, 33
 parzzley.builtin.aspects.main_stream, 33
 parzzley.builtin.aspects.posix_attributes, 34
 parzzley.builtin.aspects.pull_and_purge, 34
 parzzley.builtin.aspects.remove, 35
 parzzley.builtin.aspects.revision_tracking, 36
 parzzley.builtin.aspects.streaming, 37
 parzzley.builtin.aspects.sync_report, 40
 parzzley.builtin.aspects.xattrs, 40
 parzzley.builtin.fs, 41
 parzzley.builtin.fs.local, 46
 parzzley.builtin.fs.ssh, 48
 parzzley.builtin.log_formatters, 49
 parzzley.builtin.log_formatters.html, 49
 parzzley.builtin.log_formatters.json, 50
 parzzley.builtin.log_outs, 50
 parzzley.builtin.log_outs.shell, 50
 parzzley.config, 50
 parzzley.config.file_formats, 54
 parzzley.config.file_formats.xml, 55
 parzzley.config.loader, 57
 parzzley.fs, 58
 parzzley.fs.stream, 70
 parzzley.fs.utils, 73
 parzzley.parzzley_cli, 137
 parzzley.service, 73
 parzzley.sync, 76
 parzzley.sync.aspect, 77
 parzzley.sync.aspect.events, 78
 parzzley.sync.aspect.events.item, 79
 parzzley.sync.aspect.events.item.dir, 86
 parzzley.sync.aspect.events.item.stream, 87
 parzzley.sync.aspect.events.sync_run, 96
 parzzley.sync.aspect.only_if, 100
 parzzley.sync.control, 130
 parzzley.sync.engine, 104
 parzzley.sync.engine.event_runner, 111
 parzzley.sync.engine.events_impl, 113
 parzzley.sync.logger, 124
 parzzley.sync.logger.conditions, 128
 parzzley.sync.manager, 130
 parzzley.sync.run, 134
 parzzley.sync.utils, 137
 move_item() (*parzzley.builtin.fs.local.Backend._SiteBackend* method), 46
 move_item() (*parzzley.builtin.fs.ShellBasedBackend._SiteBackend* method), 42
 move_item() (*parzzley.fs._SubTreeSiteBackend* method), 69
 move_item() (*parzzley.fs.Backend.SiteBackend* method), 66
 move_item() (*parzzley.fs.Site* method), 62
 move_to_trash() (*parzzley.builtin.aspects.remove.TrashRemove* method), 35

N

name (*parzzley.config.Site* property), 51
 name (*parzzley.config.Volume* property), 51
 name (*parzzley.fs._Item* property), 60
 name (*parzzley.fs.Item* property), 59
 name (*parzzley.fs.Site* property), 60
 name (*parzzley.fs.SiteSetup* property), 67
 name (*parzzley.sync.Volume* property), 76
 names (*parzzley.sync.engine.event_runner.Runner._Handler* attribute), 112
 NONE (*parzzley.fs.ItemType* attribute), 59

O

Out (class in *parzzley.builtin.log_outs.shell*), 50
 Out (class in *parzzley.sync.logger*), 127
 out (*parzzley.builtin.fs.ShellBasedBackend.Shell.ExecutionResult* attribute), 44
 out (*parzzley.config.Logging* property), 52
 out_by_kind() (in module *parzzley.sync.logger*), 128

P

- `parent` (*parzzley.fs.Item* property), 60
- `parent` (*parzzley.fs.Item* property), 59
- `parse()` (in module *parzzley.config.file_formats*), 54
- `parse_file()` (*parzzley.config.file_formats.FileFormat* method), 54
- `parse_file()` (*parzzley.config.file_formats.xml.XmlFileFormat* method), 55
- `parser()` (in module *parzzley.parzzley_cli*), 137
- `parzzley`
 - module, 27
- `parzzley.asset`
 - module, 27
- `parzzley.asset.data`
 - module, 27
- `parzzley.asset.project_info`
 - module, 27
- `parzzley.builtin`
 - module, 27
- `parzzley.builtin.aspects`
 - module, 27
- `parzzley.builtin.aspects.attach_sites`
 - module, 28
- `parzzley.builtin.aspects.conflicts`
 - module, 29
- `parzzley.builtin.aspects.default_sync`
 - module, 31
- `parzzley.builtin.aspects.directory`
 - module, 31
- `parzzley.builtin.aspects.exclude_paths`
 - module, 32
- `parzzley.builtin.aspects.item_type`
 - module, 33
- `parzzley.builtin.aspects.main_stream`
 - module, 33
- `parzzley.builtin.aspects.posix_attributes`
 - module, 34
- `parzzley.builtin.aspects.pull_and_purge`
 - module, 34
- `parzzley.builtin.aspects.remove`
 - module, 35
- `parzzley.builtin.aspects.revision_tracking`
 - module, 36
- `parzzley.builtin.aspects.streaming`
 - module, 37
- `parzzley.builtin.aspects.sync_report`
 - module, 40
- `parzzley.builtin.aspects.xattrs`
 - module, 40
- `parzzley.builtin.fs`
 - module, 41
- `parzzley.builtin.fs.local`
 - module, 46
- `parzzley.builtin.fs.ssh`
 - module, 48
- `parzzley.builtin.log_formatters`
 - module, 49
- `parzzley.builtin.log_formatters.html`
 - module, 49
- `parzzley.builtin.log_formatters.json`
 - module, 50
- `parzzley.builtin.log_outs`
 - module, 50
- `parzzley.builtin.log_outs.shell`
 - module, 50
- `parzzley.config`
 - module, 50
- `parzzley.config.file_formats`
 - module, 54
- `parzzley.config.file_formats.xml`
 - module, 55
- `parzzley.config.loader`
 - module, 57
- `parzzley.fs`
 - module, 58
- `parzzley.fs.stream`
 - module, 70
- `parzzley.fs.utils`
 - module, 73
- `parzzley.parzzley_cli`
 - module, 137
- `parzzley.service`
 - module, 73
- `parzzley.sync`
 - module, 76
- `parzzley.sync.aspect`
 - module, 77
- `parzzley.sync.aspect.events`
 - module, 78
- `parzzley.sync.aspect.events.item`
 - module, 79
- `parzzley.sync.aspect.events.item.dir`
 - module, 86
- `parzzley.sync.aspect.events.item.stream`
 - module, 87
- `parzzley.sync.aspect.events.sync_run`
 - module, 96
- `parzzley.sync.aspect.only_if`
 - module, 100
- `parzzley.sync.control`
 - module, 130
- `parzzley.sync.engine`
 - module, 104
- `parzzley.sync.engine.event_runner`
 - module, 111
- `parzzley.sync.engine.events_impl`
 - module, 113
- `parzzley.sync.logger`
 - module, 113

module, 124
 parzzley.sync.logger.conditions
 module, 128
 parzzley.sync.manager
 module, 130
 parzzley.sync.run
 module, 134
 parzzley.sync.utils
 module, 137
 path (parzzley.fs._Item property), 59
 path (parzzley.fs.Item property), 59
 pipe() (parzzley.sync.aspect.events.item.stream.CollectSourceStreamable method), 93
 pipe() (parzzley.sync.aspect.events.item.stream.CollectSourceStreamable method), 92
 pipe_content() (parzzley.builtin.aspects.xattrs.XattrSynchronization._PatchDictPipe method), 41
 pipe_content() (parzzley.sync.aspect.events.item.stream.CollectSourceStreamable method), 93
 PosixAttributesSynchronization (class in parzzley.builtin.aspects.posix_attributes), 34
 Prepare (class in parzzley.sync.aspect.events.item), 81
 Prepare (class in parzzley.sync.aspect.events.item.dir), 86
 Prepare (class in parzzley.sync.aspect.events.sync_run), 97
 prepare_item() (parzzley.builtin.aspects.streaming.StickToOldMasterSiteBackend method), 39
 prepare_sync() (parzzley.sync.manager.Manager method), 131
 prepare_sync_context (parzzley.service.Service._VolumeThread._RunningSyncTask attribute), 75
 prepare_sync_run() (parzzley.builtin.aspects.directory.RollbackCrashedTransaction method), 32
 prepare_updating() (parzzley.builtin.aspects.directory.RemoveForReplacement method), 32
 prepared_sync_setup (parzzley.service.Service._VolumeThread._RunningSyncTask attribute), 75
 PrepareUpdating (class in parzzley.sync.aspect.events.item), 83
 PullAndPurgeSyncSink (class in parzzley.builtin.aspects.pull_and_purge), 34
 PullAndPurgeSyncSource (class in parzzley.builtin.aspects.pull_and_purge), 35
 R
 read() (parzzley.builtin.fs.local.Backend._SiteBackend._FileMainStreamable method), 47
 read() (parzzley.builtin.fs.ShellBasedBackend._SiteBackend._FileMainStreamable method), 44
 read() (parzzley.fs.stream.MemoryReadStreamable._ReadStream method), 72
 read() (parzzley.fs.stream.ReadStream method), 70
 read() (parzzley.sync.aspect.events.item.stream.CollectSourceStreamables method), 92
 read_bytes() (parzzley.fs.Site method), 63
 read_bytes() (parzzley.fs.stream.ReadStreamable method), 71
 read_conflict() (parzzley.config.file_formats module parzzley.config.file_formats), 55
 read_more() (parzzley.builtin.aspects.conflicts._Source method), 31
 read_streamable() (parzzley.builtin.aspects.conflicts._Source method), 47
 read_streamable() (parzzley.builtin.fs.local.Backend._SiteBackend method), 47
 read_streamable() (parzzley.builtin.fs.ShellBasedBackend._SiteBackend method), 43
 read_streamable() (parzzley.fs._SubTreeSiteBackend method), 69
 read_streamable() (parzzley.fs.Backend.SiteBackend method), 66
 read_streamable() (parzzley.fs.Site method), 63
 readme_pdf() (in module parzzley.asset.data), 27
 ReadStream (class in parzzley.fs.stream), 70
 ReadStreamable (class in parzzley.fs.stream), 71
 refresh_items_book() (parzzley.builtin.aspects.streaming.SetItemsBookEntry method), 38
 RefreshItemsBook (class in parzzley.sync.aspect.events.item), 85
 register_aspect() (in module parzzley.sync.aspect), 78
 register_backend() (in module parzzley.fs), 68
 register_file_format() (in module parzzley.config.file_formats), 54
 register_formatter() (in module parzzley.sync.logger), 128
 register_out() (in module parzzley.sync.logger), 128
 remove() (parzzley.builtin.aspects.pull_and_purge.PullAndPurgeSyncSource method), 35
 remove_conflict() (parzzley.sync.aspect.events.item.stream.ApplyConflictResolution method), 95
 remove_conflict() (parzzley.sync.engine.events_impl.StreamEvent method), 122
 remove_dir() (parzzley.builtin.aspects.remove.DirectRemove method), 35
 remove_item() (parzzley.builtin.fs.local.Backend.SiteBackend method), 47

method), 47

remove_item() (parzzley.builtin.fs.ShellBasedBackend._SiteBackend method), 43

remove_item() (parzzley.fs._SubTreeSiteBackend method), 69

remove_item() (parzzley.fs.Backend.SiteBackend method), 65

remove_item() (parzzley.fs.Site method), 61

remove_non_dir() (parzzley.builtin.aspects.remove.DirectRemove method), 35

remove_rollback_data_after_finished() (parzzley.builtin.aspects.directory.RemoveForReplacement method), 32

remove_site_from_items_books() (parzzley.sync.aspect.events.sync_run.Prepare method), 97

remove_site_from_items_books() (parzzley.sync.engine.events_impl.SyncRunEvent method), 115

remove_working_item() (parzzley.sync.aspect.events.item.SetupWorkingItems method), 84

remove_working_item() (parzzley.sync.engine.events_impl.ItemEvent method), 117

RemoveForReplacement (class in parzzley.builtin.aspects.directory), 32

rename_already_existing() (parzzley.builtin.aspects.pull_and_purge.PullAndPurgeSyncSink method), 35

resolve_conflicts() (parzzley.sync.aspect.events.item.stream.TryResolveConflicts method), 95

resolve_conflicts() (parzzley.sync.engine.events_impl.StreamEvent method), 122

resolve_conflicts_by_hint_file() (parzzley.builtin.aspects.conflicts.TryResolveConflictsByHint method), 30

resolve_conflicts_on_directories() (parzzley.builtin.aspects.posix_attributes.PosixAttributesSynchronization method), 34

revert_content_equal_to_master_flag() (parzzley.builtin.aspects.streaming.WorkingItem method), 40

RevisionTracking (class in parzzley.builtin.aspects.revision_tracking), 36

RollbackCrashedTransfers (class in parzzley.builtin.aspects.directory), 32

root_directory (parzzley.fs.Site property), 60

run() (parzzley.service.Service method), 74

run_coroutine_in_new_thread() (in module parzzley.sync.utils), 137

run_event() (parzzley.sync.engine.event_runner.Runner method), 113

run_sync() (parzzley.sync.manager.Manager method), 131

Runner (class in parzzley.sync.engine.event_runner), 111

Runner._DependencyControlledHandlerCollection (class in parzzley.sync.engine.event_runner), 111

Runner._Handler (class in parzzley.sync.engine.event_runner), 112

Runner._HandlerNode (class in parzzley.sync.engine.event_runner), 112

S

sanitized_cookie() (parzzley.fs.Site static method), 64

serialize_bytes_dict() (in module parzzley.fs.utils), 73

Service (class in parzzley.service), 73

Service._LoopThread (class in parzzley.service), 74

Service._MonitorSiteForChangesThread (class in parzzley.service), 76

Service._VolumeThread (class in parzzley.service), 75

Service._VolumeThread._RunningSyncTask (class in parzzley.service), 75

set() (parzzley.sync.aspect.events.Data method), 79

set() (parzzley.sync.engine.events_impl.SyncRunEvent._EventData method), 116

set_all_conflicts() (parzzley.sync.engine.events_impl.ItemEvent method), 117

set_comparator() (parzzley.sync.aspect.events.item.stream.DetermineComparator method), 89

set_comparator() (parzzley.sync.engine.events_impl.StreamEvent method), 123

set_error() (parzzley.sync.engine.events_impl.SyncRunEvent method), 114

set_final_stream_support() (parzzley.sync.aspect.events.sync_run.ValidateStreamSupport method), 99

set_final_stream_support() (parzzley.sync.engine.events_impl.SyncRunEvent method), 115

set_finished() (parzzley.sync.engine.Engine._SyncControl method), 106

set_item_book_entry() (parzzley.sync.aspect.events.item.RefreshItemsBook method), 85

set_item_book_entry() (parzzley.sync.engine.events_impl.ItemEvent

`method`), 119
`set_item_info()` (parzz-
`ley.sync.aspect.events.item._WithSetItemInfo`
`method`), 79
`set_item_info()` (parzz-
`ley.sync.engine.events_impl.ItemEvent`
`method`), 119
`set_item_type()` (parzz-
`ley.sync.aspect.events.item.DetermineType`
`method`), 82
`set_item_type()` (parzz-
`ley.sync.engine.events_impl.ItemEvent`
`method`), 118
`set_master_site()` (parzz-
`ley.sync.aspect.events.item.stream._WithMasterSite`
`method`), 88
`set_master_site()` (parzz-
`ley.sync.engine.events_impl.StreamEvent`
`method`), 122
`set_source_streamable()` (parzz-
`ley.sync.aspect.events.item.stream.CollectSourceStreamable`
`method`), 93
`set_source_streamable()` (parzz-
`ley.sync.engine.events_impl.StreamEvent`
`method`), 121
`set_stream_cookie()` (parzz-
`ley.sync.aspect.events.item.stream.DetermineCookie`
`method`), 90
`set_stream_cookie()` (parzz-
`ley.sync.engine.events_impl.StreamEvent`
`method`), 123
`set_stream_support()` (parzz-
`ley.sync.aspect.events.sync_run.DetermineStreamSupport`
`method`), 97
`set_stream_support()` (parzz-
`ley.sync.engine.events_impl.SyncRunEvent`
`method`), 115
`set_sync_run_sn_of_last_change()` (parzz-
`ley.sync.aspect.events.item.stream._WithMasterSite`
`method`), 88
`set_sync_run_sn_of_last_change()` (parzz-
`ley.sync.engine.events_impl.StreamEvent`
`method`), 122
`set_value()` (parzzley.sync.manager.Manager.VolumeStateVariable
`method`), 134
`SetItemsBookEntry` (class in parzz-
`ley.builtin.aspects.streaming`), 37
`SetUpWorkingItems` (class in parzz-
`ley.sync.aspect.events.item`), 83
`Severity` (class in parzzley.sync.logger), 124
`severity` (parzzley.sync.logger.Entry property), 126
`shell()` (parzzley.builtin.fs.ShellBasedBackend._SiteBackend
`method`), 42
`ShellBasedBackend` (class in parzzley.builtin.fs), 41
`ShellBasedBackend._SiteBackend` (class in parzz-
`ley.builtin.fs`), 42
`ShellBasedBackend._SiteBackend._FileMainStreamReadStreamab`
`(class in parzzley.builtin.fs)`, 44
`ShellBasedBackend._SiteBackend._FileMainStreamReadStreamab`
`(class in parzzley.builtin.fs)`, 44
`ShellBasedBackend._SiteBackend._FileMainStreamWriteStreamab`
`(class in parzzley.builtin.fs)`, 43
`ShellBasedBackend._SiteBackend._FileMainStreamWriteStreamab`
`(class in parzzley.builtin.fs)`, 43
`ShellBasedBackend._SiteBackend._ShellPool`
`(class in parzzley.builtin.fs)`, 42
`ShellBasedBackend.Shell` (class in parzz-
`ley.builtin.fs`), 44
`ShellBasedBackend.Shell.ExecutionResult` (class
in parzzley.builtin.fs), 44
`SimpleCondition` (class in parzz-
`ley.sync.logger.conditions`), 128
`Site` (class in parzzley.config), 51
`Site` (class in parzzley.fs), 60
`site` (parzzley.sync.aspect.events._Event property), 78
`site` (parzzley.sync.engine.event_runner.Runner._Handler
attribute), 112
`site` (parzzley.sync.engine.events_impl.SyncRunEvent
property), 114
`Site.ConnectionLostError`, 65
`Site.Exception`, 65
`Site.ItemExistsError`, 65
`site_control_site()` (parzzley.sync.run.SyncRun
method), 136
`site_for_key()` (in module parzzley.fs.utils), 73
`site_name()` (in module parzzley.fs), 68
`SiteSetup` (parzzley.sync.Volume property), 76
`sites` (parzzley.config.Volume property), 52
`sites` (parzzley.sync.run.SyncRun property), 135
`sites_involved_in_current_item_last_successful_sync()`
(parzzley.sync.aspect.events.item.DecideToSkip
method), 81
`sites_involved_in_current_item_last_successful_sync()`
(parzzley.sync.engine.events_impl.ItemEvent
method), 118
`sites_involved_in_sync_no()` (parzz-
`ley.sync.engine._GlobalItemsBook` method),
109
`SiteSetup` (class in parzzley.fs), 67
`skip_item()` (parzzley.sync.aspect.events.item.DecideToSkip
method), 81
`skip_item()` (parzzley.sync.engine.events_impl.ItemEvent
method), 118
`SkipDueToConflicts` (class in parzz-
`ley.sync.aspect.events.item`), 82
`SkipDueToConflicts._Conflict` (class in parzz-
`ley.sync.aspect.events.item`), 82
`SkipItemIfNoUpToDateSitesAreConnected` (class in

parzzley.builtin.aspects.streaming), 37
 sn (*parzzley.sync.run.SyncRun* property), 135
 source_streamable() (*parzzley.builtin.aspects.streaming.SourceStreamable* method), 38
 source_streamable() (*parzzley.sync.aspect.events.item.stream._WithSourceStreamable* method), 87
 source_streamable() (*parzzley.sync.engine.events_impl.StreamEvent* method), 120
 SourceStreamable (class in *parzzley.builtin.aspects.streaming*), 38
 start() (*parzzley.sync.engine.Engine* method), 104
 start_sync() (*parzzley.sync.manager.Manager* method), 131
 started_at (*parzzley.sync.run.SyncRun* property), 135
 StickToOldMasterSitesOnItemRetry (class in *parzzley.builtin.aspects.streaming*), 39
 stop_requested (*parzzley.service.Service._LoopThread* property), 74
 stop_soon() (*parzzley.service.Service* method), 73
 stop_soon() (*parzzley.service.Service._LoopThread* method), 74
 store_conflict_data() (*parzzley.builtin.aspects.conflicts.TrackConflicts* method), 30
 store_item_info() (*parzzley.sync.engine._PerSiteItemsBook* method), 110
 store_success_info() (*parzzley.sync.manager.Manager* method), 132
 store_success_info() (*parzzley.sync.run.SyncRun* method), 136
 store_working_item() (*parzzley.builtin.aspects.revision_tracking.RevisionTracker* method), 37
 stream (*parzzley.builtin.aspects.conflicts._Source* attribute), 31
 stream (*parzzley.sync.logger.Entry* property), 126
 stream() (*parzzley.fs.stream.ReadStreamable* method), 71
 stream() (*parzzley.fs.stream.WriteStreamable* method), 71
 stream_cookie() (*parzzley.sync.aspect.events.item._WithStreamCookie* method), 80
 stream_cookie() (*parzzley.sync.aspect.events.item.stream._WithStreamCookie* method), 87
 stream_cookie() (*parzzley.sync.engine.events_impl.ItemEvent* method), 117
 stream_cookie() (*parzzley.sync.engine.events_impl.StreamEvent* method), 123
 stream_cookie_after_last_sync() (*parzzley.sync.aspect.events.item.stream.DetectChanges* method), 90
 stream_cookie_after_last_sync() (*parzzley.sync.engine._PerSiteItemsBook* method), 110
 stream_cookie_after_last_sync() (*parzzley.sync.engine.events_impl.StreamEvent* method), 123
 stream_cookies (*parzzley.sync.engine.events_impl.StreamEvent* property), 123
 stream_name (*parzzley.sync.aspect.events.item.SkipDueToConflicts._Conflict* attribute), 82
 stream_name (*parzzley.sync.aspect.events.item.stream._Event* property), 88
 stream_name (*parzzley.sync.engine.events_impl.StreamEvent* property), 120
 stream_pipes() (*parzzley.sync.aspect.events.item.stream._WithSourceStreamableGetters* method), 87
 stream_pipes() (*parzzley.sync.engine.events_impl.StreamEvent* method), 121
 stream_support() (*parzzley.builtin.aspects.main_stream.MainStreamSynchronization* method), 33
 stream_support() (*parzzley.builtin.aspects.posix_attributes.PosixAttributesSynchronization* method), 34
 stream_support() (*parzzley.builtin.aspects.xattrs.XattrSynchronization* method), 40
 stream_support_info (*parzzley.sync.aspect.events.sync_run._EventAfterStreamSupportDeterminer* property), 99
 stream_support_info (*parzzley.sync.engine.events_impl.SyncRunEvent* property), 115
 StreamEvent (class in *parzzley.sync.engine.events_impl*), 119
 streams (*parzzley.sync.engine._PerSiteItemsBook._ItemInfo* attribute), 111
 streams_with_destination_streamables() (*parzzley.sync.aspect.events.item.ApplyUpdate* method), 85
 streams_with_destination_streamables() (*parzzley.sync.engine.events_impl.ItemEvent* method), 117
 sub_site() (*parzzley.fs.Site* method), 64
 sub_site_location() (*parzzley.fs.Site* method), 64

supported_item_types() (parzz- temp_working_item() (parzz-
 ley.sync.aspect.events.sync_run._EventAfterStreamSupportDefBuiltInStreamsSupportingWorkingItem
 method), 99 method), 39

supported_item_types() (parzz- ThisIsMasterSite (class in parzz-
 ley.sync.aspect.events.sync_run.ValidateStreamSupport ley.sync.aspect.only_if), 103
 method), 98 timedelta() (in module parzzley.config.file_formats),
 55

supported_item_types() (parzz- to_dir_event() (parzz-
 ley.sync.engine.events_impl.SyncRunEvent ley.sync.engine.events_impl.ItemEvent
 method), 115 method), 116

supported_item_types() (parzz- to_stream_event_for_site() (parzz-
 ley.sync.engine.events_impl.SyncRunEvent._StreamSupportInfo ley.sync.engine.events_impl.SyncRunEvent
 method), 116 method), 114

supported_streams (parzz- to_stream_event_for_site() (parzz-
 ley.sync.aspect.events.sync_run._EventAfterStreamSupportDefBuiltInStreamsSupportingWorkingItem
 property), 99 ley.sync.engine.events_impl.SyncRunEvent
 method), 114

supported_streams (parzz- to_stream_event_for_site() (parzz-
 ley.sync.engine.events_impl.SyncRunEvent._StreamSupportInfo ley.sync.engine.events_impl.SyncRunEvent
 property), 116 method), 114

SYMLINK (parzzley.fs.ItemType attribute), 58 to_stream_event_for_site() (parzz-
 107

sync() (parzzley.parzzley_cli.Commands method), 137 to_stream_event_for_site() (parzz-
 117

sync_control (parzzley.service.Service._VolumeThread._RunningSyncTask ley.sync.engine.events_impl.ItemEvent
 attribute), 75 method), 117

sync_control (parzzley.sync.engine.Engine property), 104 TrackConflicts (class in parzz-
 ley.builtin.aspects.conflicts), 29

sync_loop() (parzzley.parzzley_cli.Commands transfer_update() (parzz-
 method), 137 ley.builtin.aspects.streaming.UpdateTransfer
 method), 39

sync_run (parzzley.sync.aspect.events.sync_run._Event TransferUpdate (class in parzz-
 property), 97 ley.sync.aspect.events.item.stream), 96

sync_run (parzzley.sync.engine.events_impl.SyncRunEvent TrashRemove (class in parzzley.builtin.aspects.remove),
 property), 114 35

sync_run_event() (in module parzz- TryResolveConflicts (class in parzz-
 ley.sync.engine.events_impl), 113 ley.sync.aspect.events.item.stream), 95

sync_run_id (parzzley.sync.control.SyncControl prop- TryResolveConflictsByHint (class in parzz-
 erty), 130 ley.builtin.aspects.conflicts), 30

sync_run_id (parzzley.sync.engine.Engine._SyncControl TSiteInput (in module parzzley.fs), 68
 property), 106 type_name (parzzley.config.Aspect property), 51

sync_soon() (parzzley.service.Service method), 74

SyncControl (class in parzzley.sync.control), 130

SyncReport (class in parzz-
 ley.builtin.aspects.sync_report), 40

SyncRun (class in parzzley.sync.run), 134

SyncRunEvent (class in parzz-
 ley.sync.engine.events_impl), 113

SyncRunEvent._EventData (class in parzz-
 ley.sync.engine.events_impl), 116

SyncRunEvent._StreamSupportInfo (class in parzz-
 ley.sync.engine.events_impl), 116

T

take_last_item_book_entries_as_fallback() (parzzley.sync.engine._ItemsBookBase
 method), 108

temp_site() (parzzley.fs.Site method), 64

U

update_cookies() (parzz-
 ley.sync.aspect.events.item.ApplyUpdate
 method), 84

update_cookies() (parzz-
 ley.sync.engine.events_impl.ItemEvent
 method), 117

updated_cookie() (parzz-
 ley.sync.aspect.events.item.RefreshItemsBook
 method), 85

updated_cookie() (parzz-
 ley.sync.engine.events_impl.ItemEvent
 method), 119

UpdateTransfer (class in parzz-
 ley.builtin.aspects.streaming), 38

V

`validate_stream_support()` (parzzley.builtin.aspects.streaming.GlobalStreamSupport method), 37

`ValidateStreamSupport` (class in parzzley.sync.aspect.events.sync_run), 98

`value()` (parzzley.sync.manager.Manager.VolumeStateVariable method), 134

`var_dir` (parzzley.sync.manager.Manager property), 131

`verify_alive()` (parzzley.builtin.fs.ShellBasedBackend.Shell method), 45

`Volume` (class in parzzley.config), 51

`Volume` (class in parzzley.sync), 76

`volume` (parzzley.sync.manager.Manager.PreparedSyncSetup attribute), 133

`volume_config` (parzzley.sync.manager.Manager.PreparedSyncSetup attribute), 133

`volume_name` (parzzley.sync.run.SyncRun property), 135

`volume_state_variable()` (parzzley.sync.manager.Manager method), 132

`volume_state_variable()` (parzzley.sync.run.SyncRun method), 135

`volumes` (parzzley.config.Configuration property), 54

`volumes` (parzzley.service.Service property), 73

`volumes` (parzzley.sync.manager.Manager property), 131

W

`wait_finished()` (parzzley.sync.control.SyncControl method), 130

`wait_finished()` (parzzley.sync.engine.Engine._SyncControl method), 107

`wait_for_changes()` (parzzley.builtin.fs.local.Backend._SiteBackend method), 47

`wait_for_changes()` (parzzley.builtin.fs.ShellBasedBackend._SiteBackend method), 43

`wait_for_changes()` (parzzley.fs._SubTreeSiteBackend method), 69

`wait_for_changes()` (parzzley.fs.Backend.SiteBackend method), 66

`wait_for_changes()` (parzzley.fs.Site method), 63

`warn_after` (parzzley.config.Site property), 51

`WARNING` (parzzley.sync.logger.Severity attribute), 124

`warning()` (parzzley.sync.logger.Logger method), 125

`was_effective` (parzzley.sync.control.SyncControl property), 130

`was_effective` (parzzley.sync.engine.Engine._SyncControl property), 107

`was_effective` (parzzley.sync.engine.events_impl.SyncRunEvent property), 114

`was_successful` (parzzley.sync.control.SyncControl property), 130

`was_successful` (parzzley.sync.engine.Engine._SyncControl property), 107

`WORKER_COUNT` (parzzley.builtin.fs.ShellBasedBackend._SiteBackend._Shell attribute), 42

`working_items()` (parzzley.sync.aspect.events.item._WithWorkingItems method), 80

`working_items()` (parzzley.sync.aspect.events.item.SetupWorkingItems method), 84

`working_items()` (parzzley.sync.engine.events_impl.ItemEvent method), 117

`WorkingItem` (class in parzzley.builtin.aspects.streaming), 39

`write()` (parzzley.builtin.fs.local.Backend._SiteBackend._FileMainStream method), 48

`write()` (parzzley.builtin.fs.ShellBasedBackend._SiteBackend._FileMainStream method), 43

`write()` (parzzley.fs.stream.MemoryWriteStreamable._WriteStream method), 72

`write()` (parzzley.fs.stream.WriteStream method), 70

`write_bytes()` (parzzley.fs.Site method), 63

`write_bytes()` (parzzley.fs.stream.WriteStreamable method), 71

`write_report()` (parzzley.builtin.aspects.sync_report.SyncReport method), 40

`write_streamable()` (parzzley.builtin.fs.local.Backend._SiteBackend method), 47

`write_streamable()` (parzzley.builtin.fs.ShellBasedBackend._SiteBackend method), 43

`write_streamable()` (parzzley.fs._SubTreeSiteBackend method), 69

`write_streamable()` (parzzley.fs.Backend.SiteBackend method), 66

`write_streamable()` (parzzley.fs.Site method), 63

`WriteStream` (class in parzzley.fs.stream), 70

`WriteStreamable` (class in parzzley.fs.stream), 71

X

`XattrSynchronization` (class in parzzley.builtin.aspects.xattrs), 40

`XattrSynchronization._PatchDictPipe` (class in parzzley.builtin.aspects.xattrs), 41

XmlFileFormat (class in *parzz-*
ley.config.file_formats.xml), [55](#)