



Bazaar Architecture Overview

Release 2.1.5dev

Bazaar Developers

January 07, 2018

Contents

1	Essential Domain Classes	i
2	Transport	ii
2.1	Filenames vs URLs	ii
3	Repository	ii
3.1	Stacked Repositories	ii

This document describes the key classes and concepts within Bazaar. It is intended to be useful to people working on the Bazaar codebase, or to people writing plugins.

If you have any questions, or if something seems to be incorrect, unclear or missing, please talk to us in `irc://irc.freenode.net/#bzzr`, or write to the Bazaar mailing list. To propose a correction or addition to this document, send a merge request or new text to the mailing list.

The current version of this document is available in the file `doc/developers/overview.txt` in the source tree, and available online within the developer documentation, <<http://doc.bazaar-vcs.org/developers/>>.

1 Essential Domain Classes

The core domain objects within the bazaar model are:

- Transport
- Branch
- Repository
- WorkingTree

Transports are explained below. See <http://bazaar-vcs.org/Classes/> for an introduction to the other key classes.

2 Transport

The `Transport` layer handles access to local or remote directories. Each `Transport` object acts as a logical connection to a particular directory, and it allows various operations on files within it. You can *clone* a transport to get a new `Transport` connected to a subdirectory or parent directory.

Transports are not used for access to the working tree. At present working trees are always local and they are accessed through the regular Python file I/O mechanisms.

2.1 Filenames vs URLs

Transports work in terms of URLs. Take note that URLs are by definition only ASCII - the decision of how to encode a Unicode string into a URL must be taken at a higher level, typically in the `Store`. (Note that `Stores` also escape filenames which cannot be safely stored on all filesystems, but this is a different level.)

The main reason for this is that it's not possible to safely roundtrip a URL into Unicode and then back into the same URL. The URL standard gives a way to represent non-ASCII bytes in ASCII (as %-escapes), but doesn't say how those bytes represent non-ASCII characters. (They're not guaranteed to be UTF-8 – that is common but doesn't happen everywhere.)

For example, if the user enters the URL `http://example/%e0`, there's no way to tell whether that character represents “latin small letter a with grave” in iso-8859-1, or “latin small letter r with acute” in iso-8859-2, or malformed UTF-8. So we can't convert the URL to Unicode reliably.

Equally problematic is if we're given a URL-like string containing (unescaped) non-ASCII characters (such as the accented a). We can't be sure how to convert that to a valid (i.e. ASCII-only) URL, because we don't know what encoding the server expects for those characters. (Although it is not totally reliable, we might still accept these and assume that they should be put into UTF-8.)

A similar edge case is that the URL `http://foo/sweet%2Fsour` contains one directory component whose name is “sweet/sour”. The escaped slash is not a directory separator, but if we try to convert the URL to a regular Unicode path, this information will be lost.

This implies that `Transports` must natively deal with URLs. For simplicity they *only* deal with URLs; conversion of other strings to URLs is done elsewhere. Information that `Transports` return, such as from `list_dir`, is also in the form of URL components.

3 Repository

Repositories store committed history: file texts, revisions, inventories, and graph relationships between them.

3.1 Stacked Repositories

A repository can be configured to refer to a list of “fallback” repositories. If a particular revision is not present in the original repository, it refers the query to the fallbacks.

Compression deltas don't span physical repository boundaries. So the first commit to a new, empty repository with fallback repositories will store a full text of the inventory, and of every new file text.

At runtime, repository stacking is actually configured by the branch, not the repository. So doing `a_bzrdir.open_repository()` gets you just the single physical repository, while

`a_bzrdir.open_branch().repository` gets one configured with a stacking. Therefore, to permanently change the fallback repository stored on disk, you must use `Branch.set_stacked_on_url`.

Changing away from an existing stacked-on URL will copy across any necessary history so that the repository remains usable.

A repository opened from an HPSS server is never stacked on the server side, because this could cause complexity or security problems with the server acting as a proxy for the client. Instead, the branch on the server exposes the stacked-on URL and the client can open that.