Links

- W3C wiki page http://www.w3.org/2012/ldp/wiki/ISSUE-32
- Affordances in Wikipedia http://en.wikipedia.org/wiki/Affordance

The wiki page takes an overly broad view (and that was intentional, as it says). It really lists every point of implementation variability one would find using the RFC 2119 keywords in the specification, at the point in time when it was created.

A less surprising (for those more steeped in the Affordance term), and happily simpler, approach is to raise the level of abstraction some. That doesn't make the other questions go away, but it lets us address them in more manageable chunks.

Since the term "affordance" itself now has different meanings and inconsistent usage in practice (source: Wikipedia), probably best to set one out here (same source): "…refer to just those action possibilities that are readily perceivable by an actor. … It makes the concept dependent not only on the physical capabilities of an actor, but also the actor's goals, plans, values, beliefs, and past experiences."

Straw-man:

People want to do a fairly small and simple number of things with "collections" and their "members", that directly map to bootstrapping (discovery) plus interaction (CRUD operations)

Task	**Collection**	**Member**
Find-existing	LDP is silent	List-members on a collection
Create-new	LDP is silent	POST content to collection
List-properties	GET	GET
Update-properties	PUT/PATCH	PUT/PATCH
Delete	DELETE	DELETE
List-members	GET membership triples	n/a
Add-member (existing resource)	PUT/PATCH	n/a
Remove-member	PUT/PATCH	n/a
Delete-all-members (recursively)	Under debate	n/a

Since everything other than list-properties is optional, there are many combinations. Not all of them are going to be commonly used, although your perception of **how** commonly used any one is will depend on where you sit. We could simplify in several ways, including

1. Create "profiles", commonly (based on consensus) used combinations and provide ways to introspect those.
2. Allow introspection of each choice separately, and stay away from the combinations.

Note that the same argument applies at the lower level of abstraction, i.e. "specification options", as signaled by SHOULD, MAY, etc.

Common (?) combinations:

The names are just placeholders. I do not care much what values the working group ultimately agrees to.

You should read text displayed in ~~strikeout~~ font as MAY; read normal text as MUST. Using that reading, the profile gives clients a guarantee of what IS supported (it's a "client usage profile" describing what implementation features a client could successfully use for interacting with a given container), and if the client wants anything beyond that then it must introspect at the lower level of abstraction or test for a more "powerful" profile. It means, for example, that a "completely open" (final table on final page) container would also satisfy the profiles for "read-only" and "strictly managed membership" containers.

Disclaimer: anything "guaranteed" by a profile might still fail for a given client. E.g. even if the server supports "completely open" and advertises that, a particular client's permissions or its choice of media type etc. might effectively limit its access to "read-only" (and this will vary by resource, even on a single server). Particularly kind implementations might choose to adjust the profiles advertised but it would be very hard to get it perfectly right, so profiles devolve to hints rather than strong guarantees of what will work on any given interaction.

Read-only

This covers cases like query results.

Task	**Collection**	**Member**
Find-existing	LDP is silent	List-members on a collection
~~Create-new~~	~~LDP is silent~~	~~POST content to collection~~
List-properties	GET	GET
~~Update-properties~~	~~PUT/PATCH~~	~~PUT/PATCH~~
~~Delete~~	~~DELETE~~	~~DELETE~~
List-members	GET membership triples	n/a
~~Add-member (existing resource)~~	~~PUT/PATCH~~	n/a
~~Remove-member~~	~~PUT/PATCH~~	n/a
~~Delete-all-members (recursively)~~	~~Under debate~~	n/a

Strictly Managed Membership (“managed/closed containers”?)

The only way to add members is to create them, and the collection’s lifecycle limits the lifetime of its members. This covers (I think) what we’ve been calling (strong/composite) containers.

Task	**Collection**	**Member**
Find-existing	LDP is silent	List-members on a collection
Create-new	LDP is silent	POST content to collection
List-properties	GET	GET
Update-properties (non-membership only)	PUT/PATCH	PUT/PATCH
Delete	DELETE	DELETE
List-members	GET membership triples	n/a
~~Add-member (existing resource)~~	~~PUT/PATCH~~	n/a
~~Remove-member~~	~~PUT/PATCH~~	n/a
Delete-all-members (recursively)	DELETE collection handles it	n/a

Completely Open ("read-write"? just "containers" would be nice here OTOH keeping the unqualified form for what the specification defines, and calling these "open" or whatever specifically is less editorial work and might be good for adoption anyway.)

Supports "everything". Might not hold true at lower level of abstraction, e.g. not all media types (known + unknown/extension) would be required.

Task	**Collection**	**Member**
Find-existing	LDP is silent	List-members on a collection
Create-new	LDP is silent	POST content to collection
List-properties	GET	GET
Update-properties	PUT/PATCH	PUT/PATCH
Delete	DELETE	DELETE
List-members	GET membership triples	n/a
Add-member (existing resource)	PUT/PATCH	n/a
Remove-member	PUT/PATCH	n/a
Delete-all-members (recursively)	Under debate	n/a

Specification options that operate at a lower level of abstraction must be introspected separately (and consistently, regardless of the profile) across all profiles. I'll worry about how to signal profile compliance as part of handling that lower level.

All other combinations of supported features are "just" unqualified "LDP containers".