Twist API Documentation

Contents

Sections


```
code = 1
# comment
'''
other
'''
```

Twist Overview

Twist is a name resolution system with flexible naming conventions that use human readable strings to implement tags, and thus names. Additionally, Twist uses references called links that help express intra-object relationships. The goal of Twist is to be able to map names to global object IDs. Features include search queries, semantic relationships through links and tags, versioning, and public/private, namespaces.

2.1 Twizzler

refer (link) to twizzler documentation

Twist Naming API

3.1 Namespaces

an object that stores mapping from name to object id

```
'''
 params:
   parent_ns - (optional) reference to namespace ns_name is in
   ns_name - string name refering to namespace
   cap - (optional) capability used to verify permissions
   flags - (optional) flags
 returns:
   child_ns - reference to namespace
   status - success/failure code
'''
open_namespace(parent_ns, ns_name, cap, flags) → status, child_ns
#   returns a reference by name relative to the current namespace
#     or parent_ns
#   reference to current namespace implicit to system (similar to env var)
#   internally calls Twist name resolver and invokes a name lookup

'''
 params:
   namespace - (optional) reference to namespace ns_name is in
   flags - (optional) flags
 returns:
   status - success/failure code
'''
close_namespace(namespace, flags) -> status
delete_namespace(namespace, flags)
```

Object Tags

Tags are tuples (`obj_id`, `ns_id`, `key`, `value`, `policy`) that are used to uniquely identify objects and organize them to make searches more efficient. They contain: a reference to the object (`obj_id`) they are tied to, a reference to the namespace (`ns_id`) they belong to, a description of the tag (`key`), the attribute `value`, and the `policy` used to access

control. The tag `key` is implemented as a human readable string, while the tag `value` is implemented as a byte string. Tags are protected by capabilities which are enforced based on the `policy`. Capabilities along with the `policy` flage are described in the [security section][security]. Users do not need to provide capabilities in some cases.

Tag Properties

There are two types of tags recognized by Twist: user defined tags and tags used by the system to implement features. The tags used by the system are tags whose keys are reserved, indicated by two underscores. Tag values, however, are user defined.

Examples of reserved tag keys are:

- `__name`
- `__version`
- `__type`
- `__link`

The `__type` key is the only tag whose value can only be `namespace` or `object_name`. It is used for the system to differentiate between objects that are namespaces and user objects.

The scope of an object depends on the namespace it belongs to. Tags can be globally tied to an object (e.g. `ns_id = 0`). Alternatively a tag can be a local instance relative to a particular namespace.

Tag Operations

- create_tag
  - adds a new tag entry to the namespace
- modify_tag
  - updates one or more tag fields
    - used to change key or update value
    - can change permission, namespace, object
- delete_tag
  - deletes the tag entry from namespace
- access_tag

Tag Queries

- find_object

  - by tags (generic query interface)
    * by owner
    * by name

- conjunctive queries for now (AND only)

4.1 create_tag

`create_tag(ns_id, obj_id, tag_key, tag_val, policy, cap, flags)` → `status`

Creates a new tag relative to `ns_id`.

Input Parameters

- `ns_id` - unique identifier that refers to namespace
- `obj_id` - unique identifier that refers to object
- `tag_key` - string used to describe attribute
- `tag_val` - tag value that describes object
- `policy` - (optional) permission bits for tag
- `cap` - (optional) capability used to verify permissions
- `flags` - (optional) flags:
  - Use default tag permissions. Ignores `policy` value.

Global tags are created if the value of `ns_id` is 0. Otherwise a local tag relative to the `ns_id` is created.

Return Value(s)

- `status` - success/failure code:
  - permission error
  - object does not exist
  - tag already exists

```
'''
 params:
   parent_ns - reference to namespace name will be stored
   obj_id - unique identifier that refers to object
   tag_key - string used to describe attribute
   tag_val - tag value that refers to object
   cap - (optional) capability used to verify permissions
   flags - (optional) flags:
       * tag is registered globally or locally
```

```
10          * creating or modifying (write) a tag
11          * change the policy
12   returns:
13     status - success/failure code:
14          * permission error
15          * tag does not exist (write)
16  '''
17  modify_tag(parent_ns, obj_id, key, value, cap, flags)
18  #   used to create/write tag attributes which refer to an
        object
19  #
20  #   policy for enforcing capabilities stored in metadata
21
22  who initially creates the policy var for a tag???
23
24  '''
25   params:
26     parent_ns - reference to namespace name will be stored
27     obj_id - unique identifier that refers to object
28     tag_key - string used to describe attribute
29     tag_val - tag value that refers to object
30     cap - (optional) capability used to verify permissions
31     flags - (optional) flags:
32          * global or local search
33   returns:
34     status - success/failure code:
35          * permission error
36          * tag does not exist
37  '''
38  read_tag(parent_ns, obj_id, key, value, cap, flags)
39  #   used to read value of a tag for a specific object
40  #   has implicit verify call for capability
41  #   policy for enforcing capabilities stored in metadata
42
43  '''
44   params:
45     parent_ns - reference to namespace name will be stored
46     obj_id - unique identifier that refers to object
47     tag_list - list of tags that will help identify object
48          * tags are simply kv-pairs provided by the user
49              * tag_key - string used to describe attribute
50              * tag_val - tag value that refers to object
51     cap - (optional) capability used to verify permissions
52     flags - (optional) flags:
53          * global or local search
54   returns:
```

```
55      status - success/failure code:
56          * permission error
57          * tag does not exist
58  '''
59  get_by_tags(parent_ns, tag_list, flags) → [name_1 ... name_n],
        status
60  #   used to return a list of candidate objects
61  #   (obj_name, obj_id) pairs that are tagged
62  #   with tags in tag_list
63  #
64  #
65
66  will this leak information?
67  not sure if this needs capaibilities used to read tags/objids
68  can we just trust the NRS to read this only??
69  some of these tags may be _global ... do we refer to this
        somehow??
```

4.2 Comments

- we can use the `get_by_tags` to essentially implement everything else
  - needs to be designed correctly, allow many types of queries
- some queries might have incomplete information
  - in the case where we do not know the object id
  - the user could only provide hints through tags
    - lists what objects do

4.3 Questions

- what other types of queries can be made other than conjunctive?
  - something like `find al*` (names that start with `al`)
    - regex-like query
    - range query
  - find object (s) owned by someone
- should `modify_tag` be changed, it seems to be doing so much
  - a single call can change many things
  - not sure if atomicity can be ensured
  - maybe a `move_tag` call for namespace, policy, and object changes

4.4 Object Names

Names are used to refer to objects, they are simply a special type of tag whose key is reserved by the system.

Name Operations

```
'''
 params:
   parent_ns - reference to namespace name will be stored
   obj_name - string that will be used to refer to obj_id
   obj_id - unique identifier that refers to object
   flags - (optional) flags
 returns:
   namespace - reference to namespace
   status - success/failure code
'''
add_name(parent_ns, obj_name, obj_id, cap, flags) → status
#   registers a (obj_name, obj_id) pair within parent_ns
#
#   initially this object in the namespace has no tags
#   associated with it, so it is created with a unique
#   version tag with value equal to the object id
#
#   internal call modify_tag()

# modify name?????

'''
params:
   parent_ns - (optional) reference to namespace name will be
      stored
   tag_list - list of tags that will help identify object
       * tags are simply kv-pairs provided by the user
       * one of those tags are the obj_name string
   flags - (optional) flags:
     * name refers to a namespace
 returns:
   obj_id - reference to object we are interested in
   status - success or failure code:
       * name might not exist
       * tag_list does not narrow down search
'''
get_name(parent_ns, tag_list, flags) → obj_id, status
#   looks up a name within a namespace to find an obj_id
#
```

```
# a unique set of tags will help identify an object
#   with an internal call to get_by_tags()

change_name()
#   used to rename an object registered in the ns
#   changes the mapping new -> id

delete_name()
#   delete name -> objid mapping
#   delete unique (non-global) tags in namespace
```

4.4.1 Comments

- writes which are duplicates: idempotent
- unique: objid, nsid, key, and value
  - value is needed because we might want to have two names point the same object
  - can we have a thing where we don'y need the value??
- should values in tags be versioned?? idk wait to think about that
- need a delete function
- links could be implemented as tags

4.4.2 Questions

- how do we make sure that we do not overwrite names?
- how do we make sure we map unique things
- do we need to enforce capabilties here?
  - for naming, they are essetially tags ...
  - I can see the case for change/delete
  - get_name is interesting
  - do we need special permissions to add a name?
    - atleast to the namespace
- not sure if we need seperate name and tag APIs

4.5 Links

Links are references to other objects. They can be used to express relationships between objects. Examples of relationships are:

- dependencies
- versioning
- indexing

links are tuples of `(srcid, dstid, flags)`. The used object ids instead of names.

- `srcid` and `dstid` are object ids
- `flags = type | dir`
  - `type`: indicates the type of link
    - what the link is pointing to (dstid)
  - what kind of relationship they share
  - `dir`: link direction bit
    - links are unidirectional

still thinking about this (might need cap argument)

Link Operations

- add_link(ns_id, src_id, dst_id, flags)
  - ns_id indicates permission group/namespace
  - flags tells us loca/global link
- delete_link(ns_id, src_id, dst_id, flags)
  - flags tells us if we are doing local/global
- get_links(ns_id, obj_id, flags)

4.5.1 Versions

Versions become read-only once they are marked. Linear and non-linear versioning is supported, however, versions are manage by the application. Still thinking about this.

Version Operations

```
'''
params:
   parent_ns - reference to namespace version will be stored
   obj_id - reference to object version belongs to
   tag_value - the name of this version
   cap - (optional) capability used to verify permissions
   flags - (optional) flags:
       * version can exist globally or locally
```

```
 9   returns:
10     obj_id - reference to new object (COW) of old
11     status - success or failure code
12   '''
13   mark_version(parent_ns, obj_id, tag_value, cap, flags)
14   #   marks this object as a read-only instance and
15   #   tags it with (__version, tag_value) kv
16   #   implicit call to modify_tag for object
17
18   delete_version()
```

4.5.2 Questions

- should versions global or local?
- what happens when you delete a version?

Twist Security

uses capabilities to enforce access to:

- modify a namespace
- read/write tags
- delete things

5.1 Tag Permissions

`policy` var delegates how capabilities are enforced

- Read/write are the only access permissions
- Delete is special for removing tags
- Object access supersedes all other accesses (namespace, tag)
- Policy bits
  - `r | w | z | o | n`
  - `(r/w/z)` read/write/delete
    - public permissions for tag
  - (o) object access
    - Only (r/w/z) permission bits considered

        * Object's permission bits always considered
        * If o and n bit is set, then we need access to obj and ns
    - (n) namespace access
        * Only (r/w/z) permission bits considered
        * If z bit is set, then only local tags can be deleted

- What does a 0 mean for n and o?
    - Access to this resource is not necessary to access tag
    - It might be allowed (e.g. object access)
- What does a 1 mean for n and o?
    - Access to this resource is necessary to access tag
- List of policy permissions (from least to most restrictive)
    - Public permission bits set
    - n = 1 and access to ns
    - o = 1 and access to obj
    - n = 1 and o = 1, access to ns and obj needed
- What does r/w/z access mean?
    - r: you can read the tag
    - w: you can modify the tag
        * Change permissions
        * Change value attr
        * Change policy
    - z: you can delete the tag
- Access to a tag calculated as:

```
access_bits(object, cap) = ro | wo | zo
access_bits(namespace, cap) = rn | wn | zn
access_bits(tag.policy) = rt | wt | zt
r = rt | ro & ~(n & o) | ro & rn & (n & o) | rn & (n & ~o)
w = # same logic as above
z = zt | zo & ~(n & o) | zo & zn & (n & o) | zn & (n & ~o) &
   tag.local
#  Can delete if tag is local and you can remove namespace
```

5.2 Link Permissions

Need security API for this. Still thinking about it.

links exists in shared permission groups

- per object global metadata lives somewhere
  - tags + links
- different groups exists for different permissions
  - (r/w/z) read/write/delete
    - each of these indicate the default permissions
  - a group may have no default permissions
    - in the case of tags protected by capabilities
    - plus policy which is enforced by API
  - protected by capabilities as well since they
    - live in the same object
  - permission groups are objects
    - stores metadata entries
    - has default permissions (read-only) and
      - protected by capabilities
      - only API can change this (need rw access)
- local instances of links or tags may have their own policy

what does r/w/z mean??

- r: read the metadata
- w: change the metadata
- z: delete the metadata

5.2.1 Questions

- not sure if tags should be in their own protection groups since the enforcement of permissions are semantically different?