HCP SDK Documentation

Release 0.9.1-6

Thorsten Simons

February 18, 2015

1 Preface **1**
1.1 Intention . . . 1
1.2 About HCP . . . 1
1.3 Focus . . . 1
1.4 Coding for HCP . . . 2

2 State of Implementation **3**

3 Installation **5**
3.1 Dependencies . . . 5
3.2 Installation . . . 5
3.3 Contribute . . . 5
3.4 Support . . . 6

4 `hcpsdk` — object access **7**
4.1 Methods . . . 7
4.2 Classes . . . 7
4.3 Exceptions . . . 11
4.4 Example . . . 11

5 SSL certificate verification **13**

6 `hcpsdk.ips` — name resolution **15**
6.1 Classes . . . 15
6.2 Functions . . . 16
6.3 Exceptions . . . 16

7 `hcpsdk.namespace` — namespace information **17**
7.1 Classes . . . 17
7.2 Example . . . 19

8 `hcpsdk.pathbuilder` — unique object names **21**
8.1 Intended Use . . . 21
8.2 Classes . . . 21
8.3 Exceptions . . . 22

9 `hcpsdk.mapi` — MAPI access **25**
9.1 Classes . . . 25
9.2 Exceptions . . . 27
9.3 Example . . . 27

10 Code samples **29**
10.1 Simple object I/O without replica 29
10.2 Simple object I/O without replica, with SSL certificate verification 34

11 License **37**

12 About **39**

13 Appendixes **41**
13.1 Appendix 1 - Default Namespace 41

14 Glossary **43**

Python Module Index **45**

Index **47**

CHAPTER

ONE

PREFACE

1.1 Intention

There are several needs that led to the creation of the HCP SDK:

- Blueprint implementation of a connector module to access HCPs authenticated *namespaces* in a language that is easy enough to understand for any developers, whatever language she/he normally uses, to provide a base for own development.
- Showcase for coding guidelines outlined below.
- Demonstration material for developer trainings.
- And last, but not least, a replacement for the various modules the author used in the past for his own coding projects.

1.2 About HCP

Hitachi Content Platform (HCP) is a distributed object storage system designed to support large, growing repositories of fixed-content data. An HCP system consists of both hardware (physical or virtual) and software.

HCP stores objects that include both data and metadata that describes that data. HCP distributes these objects across the storage space. HCP represents objects either as URLs or as files in a standard file system. An HCP repository is partitioned into namespaces. Each *namespace* consists of a distinct logical grouping of objects with its own directory structure. *Namespaces* are owned and managed by *tenants*.

HCP provides access to objects through a variety of industry-standard protocols, as well as through a native http[s]/*reST* interface.

1.3 Focus

hcpsdk primarily focuses on HCP version 3 and above, and the *authenticated Namespaces* invented with version 3.

For using **hcpsdk** with the *Default Namespace*, see *Appendix 1* (page 41).

Using the **hcpsdk.mapi.replication** class needs functionality invented with HCP version 7.

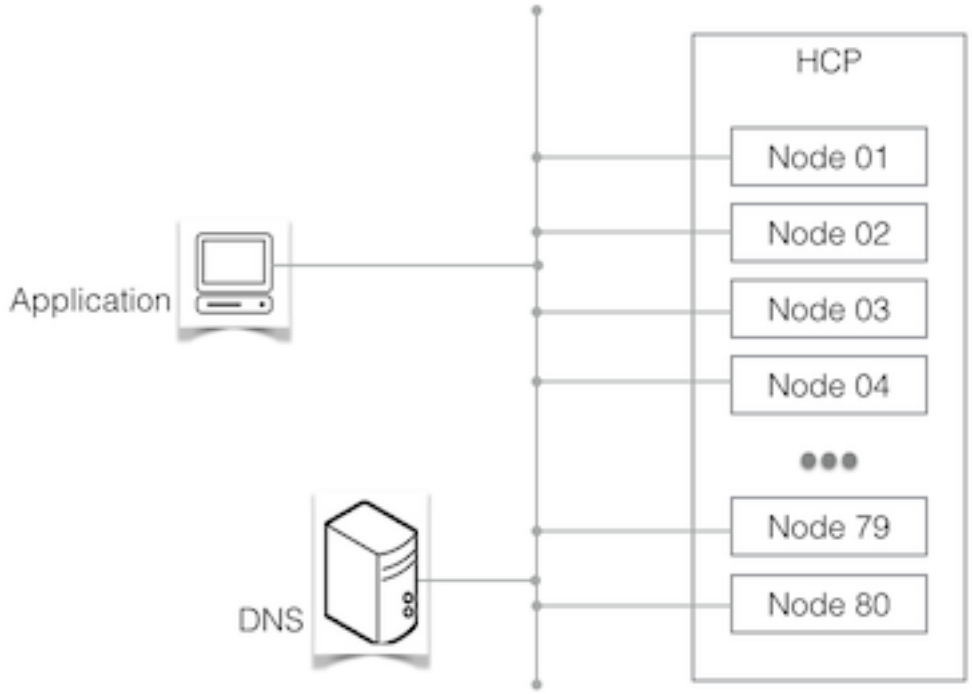

Figure 1.1: A simple HCP environment

1.4 Coding for HCP

Even as HCP might behave like a web server at first glance, it has some very different characteristics when it comes to coding against it, using one of the http/*reST* based interfaces (native http/*reST*, *HS3* and *HSwift*). This isn't really relevant for an application doing a single request from time to time, but it is critical for an application designated for high load / high performance HCP access.

To create an application optimized for optimal HCP access for the latter case:

1. Use threading or multiprocessing to access HCP using multiple connections in parallel
2. Use all available nodes, in conjunction with (1.)
3. Keep connections persistent, as connection setup is an expensive operation, especially when using https. Nevertheless, release connections if unused for some time, as one should not block an HCP connection slot permanently without using it.
4. If there's no urgent need for an human-readable structure, use a structure optimized for HCP, as demonstrated with the *hcpsdk.pathbuilder — unique object names* (page 21) subpackage

There are some additional suggestions not aiming at performance, but for reliability:

5. If there is no load balancer in the data path to HCP, cache HCPs IP addresses in the application and use them to access all nodes in a round-robin fashion. Refresh the cached address pool from time to time and on a failed Connection, too. *Depending on how HCP has been integrated with the corporate DNS, this can lower network traffic overhead significantly.*
6. If there is a replication target HCP, make the application replica-aware - at least, allow the application to read from the replica.
7. As a last resort, make sure the application can survive some time of not being able to connect to HCP by caching content locally to a certain degree (this is not covered by this SDK).

CHAPTER

TWO

STATE OF IMPLEMENTATION

Release 0.9.1-6

- Handle HCP as a *Target* object, responsible for IP address resolution (by-passing the local *DNS* cache, per default) and assigning IP addresses out of an internally cached pool to *Connection* objects. As of today, it handles the native http/*reST* interface. Support for *HS3* and *HSwift* is planned.

  Support for automated usage of a replicated HCP will be implemented soon, with various usage strategies available.

- Supports verification of SSL certificates presented by HCP when using https against a private CA chain file or the system's trusted CA store. Default is not to verify certificates.

- Provide *Connection* objects related to *Target* objects, responsible for traffic handling, service time measurement as well as handling of errors and timeouts. Connections are persistent for a configurable idle time, and are automatically re-connected on next use, if they timed out on idle.

- Easy access to *namespace* information and statistics.

- The *pathbuilder* (page 21) subpackage builds a path/name combination to be used to store an object into HCP, keeping the number of needed folders low.

- Provide convenience methods for the *Management API (MAPI)*. This is a bit limited today, but will be extended primarily on the authors needs. Available today:

  – Replication link information, link failover/failback.

CHAPTER

THREE

INSTALLATION

3.1 Dependencies

hcpsdk depends on these packages:

- dnspython3[1] - Used for non-cached name resolution when bypassing the system's resolver.
- sphinx[2] - Used to build this documentation from source code and *.rst files.
- alabaster[3] - the html template used for documentation

Thanks for creating those great packages!

3.2 Installation

Install **hcpsdk** by running:

```
pip install hcpsdk
```

-or-

- get the source from GitHub (https://github.com/Simont3/hcpsdk/archive/master.zip)
- unzip the archive
- run `python3 setup.py install`

-or-

- Fork at Github (https://github.com/Simont3/hcpsdk)

3.3 Contribute

- Fork at Github (https://github.com/Simont3/hcpsdk)
- Submit a pull request

[1] http://www.dnspython.org
[2] http://sphinx-doc.org
[3] https://github.com/bitprophet/alabaster

3.4 Support

If you find any bugs, please let us know via the Issue Tracker[4]; if you have comments or suggestions, send an email to mailto:sw@snomis.de

[4] https://github.com/simont3/hcpsdk/issues

CHAPTER

FOUR

HCPSDK — OBJECT ACCESS

hcpsdk provides access to HCP through http[s]/*reST* dialects.

Setup is easy (*see example below* (page 11)):

1. Instantiate an *Authorization* object with the required credentials.

   This class will be queried by *Target* objects for authorization tokens.

2. **Optional:** create an *SSL context* if you want to have certificates presented by HCP validated.

3. Instantiate a *Target* class with HCPs *Full Qualified Domain Name*, the port to be used, the *Authorization* object, optionally the *SSL context* created in 2. and -eventually- the *FQDN* of a replica HCP.

4. Instantiate one or more *Connection* objects.

   These objects are the workhorses of the *hcpsdk* - they are providing the access methods needed. You'll need to consult the respective HCP manuals to find out how to frame the required requests.

   Connection objects will open a session to HCP as soon as needed, but not before. After use, they keep the session open for an adjustable amount of time, helping to speed up things for an subsequent request.

   Don't forget to close *Connection* objects when finished with them!

4.1 Methods

hcpsdk.**version**()
Return the full version of the HCPsdk (0.9.1-6).

4.2 Classes

class hcpsdk.**NativeAuthorization**(*user*, *password*)
Authorization for native http/REST access to HCP.

Parameters

- **user** – the data access user
- **password** – his password

class hcpsdk.**DummyAuthorization**
Dummy authorization for the *Default Namespace*.

class hcpsdk.**Target**(*fqdn*, *authorization*, *port=443*, *dnscache=False*, *sslcontext=None*, *interface='I_NATIVE'*, *replica_fqdn=None*, *replica_strategy=None*)

This is the a central access point to an HCP target (and its replica, eventually). It caches the FQDN and the port and queries the provided *Authorization* object for the required authorization token.

Parameters

- **fqdn** – ([namespace.]tenant.hcp.loc)
- **authorization** – an instance of one of BaseAuthorization's subclasses
- **port** – port number (443, 8000 and 9090 are seen as ports using ssl)
- **dnscache** – if True, use the system resolver (which **might** do local caching), else use an internal resolver, bypassing any cache available
- **sslcontext** – the context used to handle https requests; defaults to no certificate verification
- **interface** – the HCP interface to use (I_NATIVE)
- **replica_fqdn** – the replica HCP's FQDN
- **replica_strategy** – ORed combination of the RS_* modes

Raises HcpsdkError

Class constants:

I_NATIVE
HCP's http/REST dialect for access to HCPs authenticated *namespaces*.

I_DUMMY
HCP's http dialect for access to HCPs *Default Namespace*.

RS_READ_ALLOWED
Allow to read from replica (always)

RS_READ_ON_FAILOVER
Automatically read from replica when failed over

RS_WRITE_ALLOWED
Allow write to replica (always, **A/A links only**)

RS_WRITE_ON_FAILOVER
Allow write to replica when failed over

Read-only class attributes:

fqdn
The *FQDN* for which the Target was initialized (string).

port
The port used by the Target (int).

ssl
Target initialized for SSL (bool).

addresses
The IP addresses used by this Target (list).

headers
The http headers prepared for this Target (dictionary).

replica
The replica Target, if available (an *hcpsdk.Target* object or None).

Class methods:

getaddr()
Convenience method to get an IP address out of the pool.

Returns an IP address (as string)

class hcpsdk.**Connection**(*target*, *timeout=30*, *idletime=30*, *retries=0*, *debuglevel=0*)
This class represents a Connection to HCP, caching the related parameters.

Parameters

- **target** – an initialized Target object
- **timeout** – the timeout for this Connection (secs)
- **idletime** – the time the Connection shall stay persistence when idle (secs)
- **retries** – the number of retries until giving up on a Request
- **debuglevel** – 0..9 -see-> http.client.HTTPconnection[1]

***Connection()* retries *request()s* if:**

1. the underlying connection has been closed by HCP before *idletime* has passed (the request will be retried using the existing connection context) or
2. a timeout emerges during an active request, in which case the connection is closed, *Target()* is urged to refresh its cache of IP addresses, a fresh IP address is acquired from the cache and the connection is setup from scratch.

Read-only class attributes:

address
The IP address used for this Connection.

Response
Exposition of the http.client.Response object for the last Request.

response_status
The HTTP status code of the last Request.

response_reason
The corresponding HTTP status message.

connect_time
The time the last connect took.

service_time1
The time the last action on a Request took. This can be the initial part of PUT/GET/etc. or a single (possibly incomplete) read from a Response.

service_time2
Duration of the complete Request up to now. Sum of all `service_time1` during handling a Request.

Class methods:

[1] https://docs.python.org/3/library/http.client.html?highlight=http.client#http.client.HTTPConnection.set_debuglevel

request(*method*, *url*, *body=None*, *params=None*, *headers=None*)

Wraps the *http.client.HTTP[s]Connection.Request()* method to be able to catch any exception that might happen plus to be able to trigger hcpsdk.Target to do a new DNS query.

Url and *params* will be urlencoded, by default.

Beside of *method*, all arguments are valid for the convenience methods, too.

Parameters

- **method** – any valid http method (GET,HEAD,PUT,POST,DELETE)
- **url** – the url to access w/o the server part (i.e: /rest/path/object)
- **body** – the payload to send (see *http.client* documentation for details)
- **params** – a dictionary with parameters to be added to the Request:

  ```
  {'verbose': 'true', 'retention': 'A+10y', ...}
  ```

  or a list of 2-tuples:

  ```
  [('verbose', 'true'), ('retention', 'A+10y'), ...]
  ```

- **headers** – a dictionary holding additional key/value pairs to add to the auto-prepared header

Returns the original *Response* object received from *http.client.HTTP[s]Connection.requests()*.

getheader(**args*, ***kwargs*)

Used to get a single *Response* header. Wraps *http.client.Response.getheader()*. Arguments are simply passed through.

getheaders()

Used to get a the *Response* headers. Wraps *http.client.Response.getheaders()*.

PUT(*url*, *body=None*, *params=None*, *headers=None*)

Convenience method for Request() - PUT an object. Cleans up and leaves the Connection ready for the next Request. For parameter description see *Request()*.

GET(*url*, *params=None*, *headers=None*)

Convenience method for Request() - GET an object. You need to fully *.read()* the requested content from the Connection before it can be used for another Request. For parameter description see *Request()*.

HEAD(*url*, *params=None*, *headers=None*)

Convenience method for Request() - HEAD - get metadata of an object. Cleans up and leaves the Connection ready for the next Request. For parameter description see *Request()*.

POST(*url*, *params=None*, *headers=None*)

Convenience method for Request() - POST metadata. Cleans up and leaves the Connection ready for the next Request. For parameter description see *Request()*.

DELETE(*url*, *params=None*, *headers=None*)

Convenience method for Request() - DELETE an object. Cleans up and leaves the Connection ready for the next Request. For parameter description see *Request()*.

read(*amt=None*)

Read amt # of bytes (or all, if amt isn't given) from a *Response*.

Parameters **amt** – number of bytes to read

Returns the requested number of bytes; fewer (or zero) bytes signal end of transfer, which means that the Connection is ready for another Request.

close()
Close the Connection. **It is essential to close the Connection**, as open connections might keep the program from terminating for at max *timeout* seconds, due to the fact that the timer used to keep the Connection persistent runs in a separate thread, which will be canceled on *close()*.

4.3 Exceptions

exception hcpsdk.**HcpsdkError**(*reason*)
Used to signal a generic error in **hcpsdk**.

reason
An error description.

exception hcpsdk.**HcpsdkTimeoutError**(*reason*)
Used to signal a Connection timeout.

reason
An error description.

exception hcpsdk.**HcpsdkCertificateError**(*reason*)
Raised if the *SSL context* wasn't able to verify a certificate presented by HCP.

reason
An error description.

exception hcpsdk.**HcpsdkReplicaInitError**(*reason*)
Used to signal that the Target for a replica HCP couldn't be initialized (typically, this is a name resolution problem). **If this exception is raised, the primary Target's init failed, too.** You'll need to retry!

reason
An error description.

4.4 Example

```
>>> import hcpsdk
>>> hcpsdk.version()
'0.9.0-2'
>>> auth = hcpsdk.NativeAuthorization('n', 'n01')
>>> t = hcpsdk.Target('n1.m.hcp1.snomis.local', auth, port=443)
>>> c = hcpsdk.Connection(t)
>>> c.connect_time
'0.000000000010'
>>>
>>> r = c.PUT('/rest/hcpsdk/test1.txt', body='This is an example', params={'index': 'tru
>>> c.response_status, c.response_reason
(201, 'Created')
>>>
>>> r = c.HEAD('/rest/hcpsdk/test1.txt')
>>> c.response_status, c.response_reason
```

```
(200, 'OK')
>>> c.getheaders()
[('Date', 'Sat, 31 Jan 2015 20:34:53 GMT'),
 ('Server', 'HCP V7.1.0.10'),
 ('X-RequestId', '38AD86EF250DEB35'),
 ('X-HCP-ServicedBySystem', 'hcp1.snomis.local'),
 ('X-HCP-Time', '1422736493'),
 ('X-HCP-SoftwareVersion', '7.1.0.10'),
 ('ETag', '"68791e1b03badd5e4eb9287660f67745"'),
 ('Cache-Control', 'no-cache,no-store'),
 ('Pragma', 'no-cache'),
 ('Expires', 'Thu, 01 Jan 1970 00:00:00 GMT'),
 ('Content-Type', 'text/plain'),
 ('Content-Length', '18'),
 ('X-HCP-Type', 'object'),
 ('X-HCP-Size', '18'),
 ('X-HCP-Hash', 'SHA-256 47FB563CC8F86DC37C86D08BC542968F7986ACD81C97BF76DB7AD744407FE11
 ('X-HCP-VersionId', '91055133849537'),
 ('X-HCP-IngestTime', '1422736466'),
 ('X-HCP-RetentionClass', ''),
 ('X-HCP-RetentionString', 'Deletion Allowed'),
 ('X-HCP-Retention', '0'),
 ('X-HCP-IngestProtocol', 'HTTP'),
 ('X-HCP-RetentionHold', 'false'),
 ('X-HCP-Shred', 'false'),
 ('X-HCP-DPL', '2'),
 ('X-HCP-Index', 'true'),
 ('X-HCP-Custom-Metadata', 'false'),
 ('X-HCP-ACL', 'false'),
 ('X-HCP-Owner', 'n'),
 ('X-HCP-Domain', ''),
 ('X-HCP-UID', ''),
 ('X-HCP-GID', ''),
 ('X-HCP-CustomMetadataAnnotations', ''),
 ('X-HCP-Replicated', 'false'),
 ('X-HCP-ReplicationCollision', 'false'),
 ('X-HCP-ChangeTimeMilliseconds', '1422736466446.00'),
 ('X-HCP-ChangeTimeString', '2015-01-31T21:34:26+0100'),
 ('Last-Modified', 'Sat, 31 Jan 2015 20:34:26 GMT')
]
>>>
>>> r = c.GET('/rest/hcpsdk/test1.txt')
>>> c.response_status, c.response_reason
(200, 'OK')
>>> c.read()
b'This is an example'
>>> c.service_time2
0.0005471706390380859
>>>
>>> r = c.DELETE('/rest/hcpsdk/test1.txt')
>>> c.response_status, c.response_reason
(200, 'OK')
>>> c.service_time2
0.0002570152282714844
>>>
>>> c.close()
>>>
```

CHAPTER

FIVE

SSL CERTIFICATE VERIFICATION

Warning: **hcpsdk** doesn't verify SSL certificates presented by HCP, per default.

For the case that SSL certificate verification is desired, **hcpsdk** allows to do so without excessive effort:

- Make sure the SSL certificate presented by HCP contains the IP addresses (!) of all HCP nodes as *Subject Alternative Names*.
- Create an *SSL context* and assign it to the **Target** object during creation. Each **Connection** created using that **Target** will automatically inherit the *SSL context*.

Here are some hints:

- This example creates an *SSL context* with the recommended security settings for client sockets, including automatic certificate verification against the system's trusted CA store:

```
>>> context = ssl.create_default_context()
>>> auth = hcpsdk.NativeAuthorization('n', 'n01')
>>> t = hcpsdk.Target('n1.m.hcp1.snomis.local', auth,
                      port=443, sslcontext=context)
```

- Alternatively, you can create an *SSL context* that verifies certificates against a local CA file:

```
>>> context = ssl.create_default_context(cafile='myCA.pem')
>>> auth = hcpsdk.NativeAuthorization('n', 'n01')
>>> t = hcpsdk.Target('n1.m.hcp1.snomis.local', auth,
                       port=443, sslcontext=context)
```

If you want to have more control about the protocol and/or the cipher suites in use, follow the Python documentation about SSL context creation[1].

[1] https://docs.python.org/3/library/ssl.html?highlight=ssl.sslcontext#ssl.SSLContext

CHAPTER

SIX

HCPSDK.IPS — NAME RESOLUTION

hcpsdk.ips provides name resolution service and IP address caching. Used by *hcpsdk* internally; exposed here as it might be useful alone.

6.1 Classes

class hcpsdk.ips.**Circle**(*fqdn*, *port=443*, *dnscache=False*)

Resolve an FQDN (through **query()**), cache the acquired IP addresses and yield them round-robin.

Parameters

- **fqdn** – the FQDN to be resolved
- **port** – the port to be used by the **hcpsdk.Target** object
- **dnscache** – if True, use the system resolver (which **might** do local caching), else use an internal resolver, bypassing any cache available

Read-only class attributes:

_addresses

List of the cached IP addresses

Class methods:

refresh()

Force a fresh DNS query and rebuild the cached list of IP addresses

class hcpsdk.ips.**Response**(*fqdn*, *cache*)

DNS query Response object, returned by the **query()** function.

Parameters

- **fqdn** – the FQDN for the Response
- **cache** – Response from a query by-passing the local DNS cache (False) or using the system resolver (True)

Read-only class attributes:

ips

List of resolved IP addresses (as strings)

fqdn

The *FQDN* for which the resolve happened.

cache
False if the local *DNS* cache has been by-passed, True if the system-default resolver was used.

raised
Empty string when no Exception were raised, otherwise the Exception's error message.

6.2 Functions

hcpsdk.ips.**query**(*fqdn*, *cache=False*)
Submit a DNS query, using *socket.getaddrinfo()* if cache=True, or *dns.resolver.query()* if cache=False.

Parameters

- **fqdn** – a FQDN to query DNS -or- a *Request* object
- **cache** – if True, use the system resolver (which might do local caching), else use an internal resolver, bypassing any cache available

Returns an **hcpsdk.ips.Response** object

Raises should never raise, as Exceptions are signaled through the **Response.raised** attribute

6.3 Exceptions

exception hcpsdk.ips.**IpsError**(*reason*)
Signal an error in 'ips' - typically a name resolution problem.

reason
An error description.

CHAPTER

SEVEN

HCPSDK.NAMESPACE — NAMESPACE INFORMATION

hcpsdk.namespace provides access to the actual Namespace's parameters and statistics. The **hcpsdk.Target** object must have been instantiated with a *Namespace FQDN*.

7.1 Classes

class hcpsdk.namespace.**Info**(*target*, *debuglevel=0*)

Class to access namespaces metadata information

Parameters

- **target** – an **hcpsdk.Target** object
- **debuglevel** – 0..9 (propagated to *http.client*)

nsstatistics()

Query for namespace statistic information

Returns a dict holding the stats

Raises hcpsdk.HcpsdkError()

returned dictionary (example):

```
{'customMetadataObjectBytes': 13542405,
 'customMetadataObjectCount': 380107,
 'namespaceName': 'n1',
 'objectCount': 402403,
 'shredObjectBytes': 0,
 'shredObjectCount': 0,
 'softQuotaPercent': 85,
 'totalCapacityBytes': 53687091200,
 'usedCapacityBytes': 13792362496}
```

listaccessiblens(*all=False*)

List the settings of the actual (or all accessible namespace(s).

Parameters **all** – list all accessible namespaces if True, else list the actual one, only.

Returns a dict holding a dict per namespace

returned dictionary (example):

```
{'n2': {'defaultIndexValue': True,
        'defaultRetentionValue': 0,
        'defaultShredValue': False,
        'description': ['replicated', 'search', 'versioning'],
        'dpl': 2,
        'hashScheme': 'SHA-256',
        'name': 'n2',
        'nameIDNA': 'n2',
        'retentionMode': 'enterprise',
        'searchEnabled': True,
        'versioningEnabled': True},
 'n1': {'defaultIndexValue': True,
        'defaultRetentionValue': 0,
        'defaultShredValue': False,
        'description': ['replicated', 'search', 'no versioning'],
        'dpl': 2,
        'hashScheme': 'SHA-256',
        'name': 'n1',
        'nameIDNA': 'n1',
        'retentionMode': 'enterprise',
        'searchEnabled': True,
        'versioningEnabled': False}}
```

listretentionclasses()

List the Retention Classes available for the actual namespace.

Returns a dict holding a dict per Retention Class

returned dictionary (example):

```
{'initial_unspecified': {'autoDelete': False,
                         'description': 'Retention Class with an initial '
                                        'unspecified value.',
                         'name': 'initial_unspecified',
                         'value': -2},
 'deletion_prohibited': {'autoDelete': False,
                         'description': 'Class which prohibits deletion.',
                         'name': 'deletion_prohibited',
                         'value': -1},
 'TAX_DATA': {'autoDelete': True,
              'description': 'Class for tax data - actually 10 years.',
              'name': 'TAX_DATA',
              'value': 'A+10y'}}
```

listpermissions()

List the namespace and user permissions for the actual namespace.

Returns a dict holding a dict per permission domain

returned dictionary (example):

```
{'namespacePermissions': {'browse': True,
                          'changeOwner': True,
                          'delete': True,
                          'privileged': True,
```

```
                                    'purge': True,
                                    'read': True,
                                    'readAcl': True,
                                    'search': True,
                                    'write': True,
                                    'writeAcl': True},
             'namespaceEffectivePermissions': {'browse': True,
                                               'changeOwner': True,
                                               'delete': True,
                                               'privileged': True,
                                               'purge': True,
                                               'read': True,
                                               'readAcl': True,
                                               'search': True,
                                               'write': True,
                                               'writeAcl': True},
             'userPermissions': {'browse': True,
                                 'changeOwner': True,
                                 'delete': True,
                                 'privileged': True,
                                 'purge': True,
                                 'read': True,
                                 'readAcl': True,
                                 'search': True,
                                 'write': True,
                                 'writeAcl': True},
             'userEffectivePermissions': {'browse': True,
                                          'changeOwner': True,
                                          'delete': True,
                                          'privileged': True,
                                          'purge': True,
                                          'read': True,
                                          'readAcl': True,
                                          'search': True,
                                          'write': True,
                                          'writeAcl': True}}
```

7.2 Example

```
>>> import hcpsdk.namespace
>>> from pprint import pprint
>>> auth = hcpsdk.NativeAuthorization('n', 'n01')
>>> t = hcpsdk.Target('n1.m.hcp1.snomis.local', auth, port=443)
>>> n = hcpsdk.namespace.Info(t)
>>> pprint(n.nsstatistics())
{'customMetadataObjectBytes': 0,
 'customMetadataObjectCount': 0,
 'namespaceName': 'n1',
 'objectCount': 0,
 'shredObjectBytes': 0,
 'shredObjectCount': 0,
 'softQuotaPercent': 85,
 'totalCapacityBytes': 53687091200,
 'usedCapacityBytes': 0}
>>>
```

CHAPTER

EIGHT

HCPSDK.PATHBUILDER — UNIQUE OBJECT NAMES

Due to its internals, bulk ingest activity into HCP delivers best possible performance if multiple parallel writes are directed to different folders take place.

This subpackage offers functionality to create an unique object name along with a pseudo-random path to a folder to store the object in.

The object name generated is an UUID version 1, as defined in RFC 4122[1]. The algorithm uses (one of) the servers MAC addresses, along with the system time to create the UUID.

8.1 Intended Use

Applications typically utilize a database to keep reference pointers to objects they stored to HCP. Instead of storing a full path as a reference to each object (i.e.: `https://ns.tenant.hcp.domain.com/rest/mypath/myfile`), applications should define a *data pool* that describes the storage target (i.e.: `https://ns.tenant.hcp.domain.com, startpath='/rest/myapp'`).

When storing a reference, an application should store the name generated by **hcpsdk.pathbuilder.PathBuilder.getunique()**, along with a reference to the *data pool* used.

Benefits are:

- Reasonable space usage in the applications database
- The actual content address (the full URL) is easily computable from the *data pool* and **hcpsdk.pathbuilder.PathBuilder.getpath(reference)**
- In case the applications data needs to be migrated to a different storage system or a different *namespace* within HCP, it's just a matter of migrating the data in the background and then changing the *data pool* definition, keeping application disturbence extremly low.

8.2 Classes

class hcpsdk.pathbuilder.**PathBuilder**(*initialpath='/rest/hcpsdk'*, *annotation=False*)

Conversion of a filename into a unique object name and a proper path for HCPs needs. Reconversion of a unique object name to the path where the object can be found in HCP.

Parameters

[1] http://tools.ietf.org/pdf/rfc4122.pdf

- **initialpath** – the leading part of the path
- **annotation** – if True, create an XML structure to be used as custom metadata annotation containing a tag with the original filename of the object.

getunique (*filename*)

Build a unique path / object name scheme.

The path is build from **initialpath** given during class instantiation plus byte 4 and 3 of the UUID in hexadecimal.

If **annotation** is True during class instantiation, there will be a third element in the returned tuple, containing an XML structure that can be used as custom metadata annotation for the object.

Parameters **filename** – the filename to be transformed

Returns a tuple consisting of path, unique object name and -eventually- an annotation string.

Raises hcpsdk.pathbuilder.pathbuilderError

Example:

```
>>> from hcpsdk.pathbuilder import PathBuilder
>>> p = PathBuilder(initialpath='/rest/mypath', annotation=True)
>>> o = p.getunique('testfile.txt')
>>> o
('/rest/mypath/b4/ec', '8ac8ecb4-9f1e-11e4-a524-98fe94437d8c',
 '<?xml version='1.0' encoding='utf-8'?>
    <hcpsdk_fileobject
        filename="testfile.txt"
        path="/rest/mypath/b4/ec"
        uuid="8ac8ecb4-9f1e-11e4-a524-98fe94437d8c"
 />')
>>>
```

getpath (*objectname*)

From a unique object name, retrieve the path in which the object was stored.

Parameters **objectname** – an unique object name

Returns the full path to the object (including its name)

Raises hcpsdk.pathbuilder.pathbuilderError

Example:

```
>>> p.getpath('8ac8ecb4-9f1e-11e4-a524-98fe94437d8c')
'/rest/mypath/b4/ec/8ac8ecb4-9f1e-11e4-a524-98fe94437d8c'
>>>
```

8.3 Exceptions

exception hcpsdk.pathbuilder.**PathBuilderError** (*reason*)

Used to signal an error during unique object name generation or object name to path mapping.

reason
An error description.

CHAPTER

NINE

HCPSDK.MAPI — MAPI ACCESS

hcpsdk.mapi provides access to selected *MAPI* functions.

9.1 Classes

class hcpsdk.mapi.**Replication**(*target*, *debuglevel=0*)

Access replication link information, modify the replication link state.

Parameters

- **target** – an hcpsdk.Target object
- **debuglevel** – 0..9 (used in *http.client*)

Class constants:

Link types:

R_ACTIVE_ACTIVE
Active/Active link

R_OUTBOUND
Outbound link (active/passive)

R_INBOUND
Inbound link (active/passive)

Link activities:

R_SUSPEND
Suspend a link (all link types)

R_RESUME
Resume a suspended link (all link types)

R_RESTORE
Restore a link (all link types)

R_FAILOVER
Initiate a failover (all link types)

R_FAILBACK
Initiate a failback (ACTIVE/ACTIVE links only)

R_BEGINRECOVERY
Begin recovery (INBOUND links only)

R_COMPLETERECOVERY
Complete recovery (INBOUND links only)

Class methodes:

getreplicationsettings()
Query MAPI for the general settings of the replication service.

Returns a dict containing the settings

Raises HcpsdkError

returned dictionary (example):

```
{'allowTenantsToMonitorNamespaces': 'true',
 'enableDNSFailover': 'true',
 'enableDomainAndCertificateSynchronization': 'true',
 'network': '[hcp_system]'}
```

getlinklist()
Query MAPI for a list of replication links.

Returns a list with the names of replication links

Raises HcpsdkError

returned list (example):

```
['hcp1-a-a-hcp2']
```

getlinkdetails(*link*)
Query MAPI for the details of a replication link.

Parameters **link** – the name of the link as retrieved by **getlinklist()**

Returns a dict holding the details

Raises HcpsdkError

the returned dictionary (example):

```
{'compression': 'false',
 'Connection': {'localHost': '192.168.0.52, 192.168.0.53, 192.168.0.54, '
                             '192.168.0.55',
                'localPort': '5748',
                'remoteHost': '192.168.0.56, 192.168.0.57, 192.168.0.58, '
                              '192.168.0.59',
                'remotePort': '5748'},
 'description': 'active/active link between HCP1 and HCP2',
 'encryption': 'false',
 'failoverSettings': {'local': {'autoFailover': 'false',
                                'autoFailoverMinutes': '120'},
                      'remote': {'autoFailover': 'false',
                                 'autoFailoverMinutes': '120'}},
 'id': 'b9c488db-f641-486e-a8b4-56810faf23cd',
 'name': 'hcp1-a-a-hcp2',
```

```
            'priority': 'OLDEST_FIRST',
            'statistics': {'bytesPending': '0',
                           'bytesPendingRemote': '0',
                           'bytesPerSecond': '0.0',
                           'bytesReplicated': '0',
                           'errors': '0',
                           'errorsPerSecond': '0.0',
                           'objectsPending': '0',
                           'objectsPendingRemote': '0',
                           'objectsReplicated': '0',
                           'operationsPerSecond': '0.0',
                           'upToDateAsOfMillis': '1419975449113',
                           'upToDateAsOfString': '2014-12-30T22:37:29+0100'},
            'status': 'GOOD',
            'statusMessage': 'Synchronizing data',
            'suspended': 'false',
            'type': 'ACTIVE_ACTIVE'}
```

setreplicationlinkstate(*linkname*, *action*, *linktype=None*)

Failover and failback a replication link.

Parameters

- **linkname** – name of the link to change the state
- **linktype** – one of [R_ACTIVE_ACTIVE, R_OUTBOUND, R_INBOUND]; not required for [R_SUSPEND, R_RESUME, R_RESTORE]
- **action** – one of [R_SUSPEND, R_RESUME, R_RESTORE, R_FAILOVER, R_FAILBACK, R_BEGINRECOVERY, R_COMPLETERECOVERY]

Raises HcpsdkError

9.2 Exceptions

exception hcpsdk.mapi.**ReplicationSettingsError**(*reason*)

Indicate an invalid action for the given link type (R_BEGINRECOVERY or R_COMPLETERECOVERY on a R_ACTIVE_ACTIVE link, R_FAILBACK on an R_OUTBOUND or R_INBOUND link).

reason

An error description.

9.3 Example

```
>>> import hcpsdk.mapi
>>> from pprint import pprint
>>>
>>> auth = hcpsdk.NativeAuthorization('service', 'service01')
>>> t = hcpsdk.Target('admin.hcp1.snomis.local', auth, port=9090)
>>> r = hcpsdk.mapi.Replication(t)
>>> l = r.getlinklist()
```

```
>>> l
['hcp1--<-->--hcp2']
>>> d = r.getlinkdetails(l[0])
>>> pprint(d)
{'compression': 'false',
 'connection': {'localHost': '192.168.0.52, 192.168.0.53, 192.168.0.54, '
                             '192.168.0.55',
                'localPort': '5748',
                'remoteHost': '192.168.0.56, 192.168.0.57, 192.168.0.58, '
                              '192.168.0.59',
                'remotePort': '5748'},
 'description': 'active/active replication between HCP1 and HCP2',
 'encryption': 'false',
 'failoverSettings': {'local': {'autoFailover': 'false',
                                'autoFailoverMinutes': '120'},
                      'remote': {'autoFailover': 'false',
                                 'autoFailoverMinutes': '120'}},
 'id': '81b6df01-2bda-4094-aed8-0c47e68bd820',
 'name': 'hcp1--<-->--hcp2',
 'priority': 'OLDEST_FIRST',
 'statistics': {'bytesPending': '0',
                'bytesPendingRemote': '0',
                'bytesPerSecond': '0.0',
                'bytesReplicated': '0',
                'errors': '0',
                'errorsPerSecond': '0.0',
                'objectsPending': '0',
                'objectsPendingRemote': '0',
                'objectsReplicated': '0',
                'operationsPerSecond': '0.0',
                'upToDateAsOfMillis': '1422701963994',
                'upToDateAsOfString': '2015-01-31T11:59:23+0100'},
 'status': 'GOOD',
 'statusMessage': 'Synchronizing data',
 'suspended': 'false',
 'type': 'ACTIVE_ACTIVE'}
>>>
```

CHAPTER

TEN

CODE SAMPLES

10.1 Simple object I/O without replica

10.1.1 Code sample

This code sample shows basic usage of the SDK - ingest an object, retrieve its metadata, read and delete it. It also shows how to retrieve request timers and how to enable debug logging.

First, we import the needed packages and setup a few constants with the parameters needed to access HCP. We also make sure that this program only runs if called as such:

```
import sys
from os.path import normpath
from pprint import pprint
import hcpsdk

# HCP Connection details - you'll need to adopt this to your environment!
# -- primary HCP
P_FQDN = 'n1.m.hcp1.snomis.local'
P_USER = 'n'
P_PASS = 'n01'
P_PORT = 443
# -- file to be used for the test (read-only)
P_FILE = normpath('../testfiles/128kbfile')
# -- debug mode
P_DEBUG = True

if __name__ == '__main__':
```

We need to create an authorization object, which converts the user credentials into the authorization token needed for HCP access.

```
    # Setup an authorization object:
    auth = hcpsdk.NativeAuthorization(P_USER, P_PASS)
    print('*I_NATIVE* authorization initialized')
    print('')
```

Now, we initialize a **Target** object with the parameters and the authorization object created in the steps before. Notice that we do this within a try/except clause, as we need to be able to react on errors that might happen during initialization.

```
    # Setup an HCP Target object:
    try:
        t = hcpsdk.Target(P_FQDN, auth, port=P_PORT)
```

```
except hcpsdk.HcpsdkError as e:
    sys.exit('init of *Target* failed - {}'.format(e))
else:
    print('Target *t* was initialized with IP addresses: {}'.format(t.addresses
```

At next, we initialize a **Connection** object, using the **Target** created before. Notice that there is no IP address assigned to the Connection at this time! This is because a connection will acquire an IP address not earlier than needed.

```
# Setup a Connection object:
try:
    c = hcpsdk.Connection(t)
except hcpsdk.HcpsdkError as e:
    sys.exit('init of *Target* failed - {}'.format(e))
else:
    print('Connection *c* uses IP address: {}'.format(c.address))
    print('')
```

Now that we have a **Connection** and its corresponding **Target**, let's write an object (the 128kb file); we'll also set some policies for it, using the *params* argument. Again, notice the exception handling! Now, we have an IP address assigned. If all's well, print the hash value HCP calculated for our object:

```
# Ingest an object:
try:
    with open(P_FILE, 'r') as infile:
        r = c.PUT('/rest/hcpsdk/sample_primary_only.txt',
                  body=infile,
                  params={'index': 'true', 'shred': 'true'})
except hcpsdk.HcpsdkTimeoutError as e:
    sys.exit('PUT timed out - {}'.format(e))
except hcpsdk.HcpsdkError as e:
    sys.exit('PUT failed - {}'.format(e))
except OSError as e:
    sys.exit('failure on {} - {}'.format(P_FILE, e))
else:
    if c.response_status == 201:
        print('PUT Request was successful')
        print('used IP address: {}'.format(c.address))
        print('hash = {}'.format(c.getheader('X-HCP-Hash')))
        print('connect time:     {:0.12f} seconds'.format(c.connect_time))
        print('Request duration: {:0.12f} seconds'.format(c.service_time2))
        print('')
    else:
        sys.exit('PUT failed - {}-{}'.format(c.response_status,
                                             c.response_reason))
```

OK, as all was well so far, let's see if our object is really there - we'll do an *HEAD* Request and if successful, print the returned headers, as they contain the objects metadata:

```
# Check an object for existence and get its metadata:
try:
    r = c.HEAD('/rest/hcpsdk/sample_primary_only.txt')
except hcpsdk.HcpsdkTimeoutError as e:
    sys.exit('HEAD timed out - {}'.format(e))
except hcpsdk.HcpsdkError as e:
    sys.exit('HEAD failed - {}'.format(e))
else:
```

```
    if c.response_status == 200:
        print('HEAD Request was successful - one of the headers:')
        print('Server: {}'.format(c.getheader('Server')))
        print('used IP address: {}'.format(c.address))
        print('Request duration: {:0.12f} seconds'.format(c.service_time2))
        print('')
    else:
        sys.exit('HEAD failed - {}-{}'.format(c.response_status,
                                              c.response_reason))
```

We'll read the object back and print the first few bytes of its content:

```
# Read an object:
try:
    r = c.GET('/rest/hcpsdk/sample_primary_only.txt')
except hcpsdk.HcpsdkTimeoutError as e:
    sys.exit('GET timed out - {}'.format(e))
except hcpsdk.HcpsdkError as e:
    sys.exit('GET failed - {}'.format(e))
else:
    if c.response_status == 200:

        print('GET Request was successful - here\'s the content:')
        print('{}...'.format(c.read()[:40]))
        print('used IP address: {}'.format(c.address))
        print('Request duration: {:0.12f} seconds'.format(c.service_time2))
        print('')
    else:
        sys.exit('GET failed - {}-{}'.format(c.response_status,
                                             c.response_reason))
```

Clean up by deleting the object again:

```
# Delete the object:
try:
    r = c.DELETE('/rest/hcpsdk/sample_primary_only.txt')
except hcpsdk.HcpsdkTimeoutError as e:
    sys.exit('DELETE timed out - {}'.format(e))
except hcpsdk.HcpsdkError as e:
    sys.exit('DELETE failed - {}'.format(e))
else:
    if c.response_status == 200:
        print('DELETE Request was successful')
        print('used IP address: {}'.format(c.address))
        print('Request duration: {:0.12f} seconds'.format(c.service_time2))
        print('')
    else:
        sys.exit('DELETE failed - {}-{}'.format(c.response_status,
                                                c.response_reason))
```

And finally, don't forget to close the **Connection**! This will cleanly cancel the timer thread that keeps an idle connection open (persistent). Not doing so will lead to the program not finishing until the timer expires!

```
# Close the Connection:
finally:
    # noinspection PyUnboundLocalVariable
    c.close()
```

As the SDK is pre-configured for DEBUG logging using Pythons native logging facility, you simply enable it by activating a logger, set to level DEBUG. In this example, we simply set P_DEBUG to True, which will enable the logging facility:

```
if P_DEBUG:
    import logging
    logging.basicConfig(level=logging.DEBUG, style='{', format='{levelname:>5s}
    # noinspection PyShadowingBuiltins
    print = pprint = logging.info
```

10.1.2 Sample code output

Without debug messages

```
running *simple_primary_only.py*
*I_NATIVE* authorization initialized

Target *t* was initialized with IP addresses: ['192.168.0.53', '192.168.0.54', '192.168.
Connection *c* uses IP address: None

PUT Request was successful
used IP address: 192.168.0.54
hash = SHA-256 A2706A20394E48179A86C71E82C360C2960D3652340F9B9FDB355A42E3AC7691
connect time:     0.001283884048 seconds
Request duration: 0.079370975494 seconds

HEAD Request was successful - one of the headers:
Server: HCP V7.1.0.10
used IP address: 192.168.0.54
Request duration: 0.000217914581 seconds

GET Request was successful - here's the content:
b'0123456789abcdef0123456789abcdef01234567'...
used IP address: 192.168.0.54
Request duration: 0.019832849503 seconds

DELETE Request was successful
used IP address: 192.168.0.54
Request duration: 0.000179052353 seconds
```

With debug messages

```
 INFO running *simple_primary_only.py*
DEBUG *I_NATIVE* authorization initialized for user: n
DEBUG pre version 6:     Cookie: hcp-ns-auth=bg==:1dc7fed37e11b35093d311ef66928ad9
DEBUG version 6+: Authorization: HCP bg==:1dc7fed37e11b35093d311ef66928ad9
 INFO *I_NATIVE* authorization initialized
 INFO
DEBUG (re-) loaded IP address cache: ['192.168.0.52', '192.168.0.53', '192.168.0.54', '1
DEBUG issued IP address: 192.168.0.52
DEBUG Target initialized: n1.m.hcp1.snomis.local:443 - SSL = True
 INFO Target *t* was initialized with IP addresses: ['192.168.0.52', '192.168.0.53', '19
DEBUG Connection object initialized: IP None (n1.m.hcp1.snomis.local) - timeout: 30 - id
 INFO Connection *c* uses IP address: None
```

```
 INFO
DEBUG tried to cancel a non-existing idletimer (pretty OK)
DEBUG URL = /rest/hcpsdk/sample_primary_only.txt
DEBUG Connection needs to be opened
DEBUG issued IP address: 192.168.0.53
DEBUG Connection open: IP 192.168.0.53 (n1.m.hcp1.snomis.local) - connect_time: 0.001631
DEBUG PUT Request for /rest/hcpsdk/sample_primary_only.txt - service_time1 = 0.078655004
DEBUG tried to cancel a non-existing idletimer (pretty OK)
DEBUG idletimer started: <Timer(Thread-1, started 4350545920)>
 INFO PUT Request was successful
 INFO used IP address: 192.168.0.53
 INFO hash = SHA-256 A2706A20394E48179A86C71E82C360C2960D3652340F9B9FDB355A42E3AC7691
 INFO connect time:     0.001631975174 seconds
 INFO Request duration: 0.078655004501 seconds
 INFO
DEBUG idletimer canceled: <Timer(Thread-1, started 4350545920)>
DEBUG URL = /rest/hcpsdk/sample_primary_only.txt
DEBUG HEAD Request for /rest/hcpsdk/sample_primary_only.txt - service_time1 = 0.0001850
DEBUG tried to cancel a non-existing idletimer (pretty OK)
DEBUG idletimer started: <Timer(Thread-2, started 4350545920)>
 INFO HEAD Request was successful - one of the headers:
 INFO Server: HCP V7.1.0.10
 INFO used IP address: 192.168.0.53
 INFO Request duration: 0.000185012817 seconds
 INFO
DEBUG idletimer canceled: <Timer(Thread-2, started 4350545920)>
DEBUG URL = /rest/hcpsdk/sample_primary_only.txt
DEBUG GET Request for /rest/hcpsdk/sample_primary_only.txt - service_time1 = 0.00018692
DEBUG tried to cancel a non-existing idletimer (pretty OK)
DEBUG idletimer started: <Timer(Thread-3, started 4350545920)>
 INFO GET Request was successful - here's the content:
DEBUG (partial?) read: service_time1 = 0.022004127502441406 secs
 INFO b'0123456789abcdef0123456789abcdef01234567'...
 INFO used IP address: 192.168.0.53
 INFO Request duration: 0.022191047668 seconds
 INFO
DEBUG idletimer canceled: <Timer(Thread-3, started 4350545920)>
DEBUG URL = /rest/hcpsdk/sample_primary_only.txt
DEBUG DELETE Request for /rest/hcpsdk/sample_primary_only.txt - service_time1 = 0.000180
DEBUG tried to cancel a non-existing idletimer (pretty OK)
DEBUG idletimer started: <Timer(Thread-4, started 4350545920)>
 INFO DELETE Request was successful
 INFO used IP address: 192.168.0.53
 INFO Request duration: 0.000180006027 seconds
 INFO
DEBUG idletimer canceled: <Timer(Thread-4, started 4350545920)>
DEBUG Connection object closed: IP 192.168.0.53 (n1.m.hcp1.snomis.local)
```

10.2 Simple object I/O without replica, with SSL certificate verification

10.2.1 Code sample

This code sample is exactly the same as the one shown as *Simple object I/O without replica* (page 29), except that we verify the SSL certificate presented by HCP against a CA certificate chain we have locally on file. So, described here are only the differences to the first code sample.

We need to import **ssl.create_default_context** and define a file that holds our CA certificate chain:

```
import sys
from os.path import normpath
from ssl import create_default_context
from pprint import pprint
import hcpsdk

# HCP Connection details - you'll need to adopt this to your environment!
# -- primary HCP
P_FQDN = 'n1.m.hcp1.snomis.local'
P_USER = 'n'
P_PASS = 'n01'
P_PORT = 443
# -- file to be used for the test (read-only)
P_FILE = normpath('../testfiles/128kbfile')
# -- file holding a private CA certificate chain
P_CAFILE = normpath('../../../tests/certs/failCertificate.pem')
# -- debug mode
P_DEBUG = True

if __name__ == '__main__':
```

Now, we create an *SSL context* and use it when instantiating our **Target** object:

```
    # Setup an authorization object:
    auth = hcpsdk.NativeAuthorization(P_USER, P_PASS)
    print('*I_NATIVE* authorization initialized')
    print('')

    # Create an SSL context for server authentication, using a local CAfile
    ctxt = create_default_context(cafile=P_CAFILE)

    # Setup an HCP Target object:
    try:
        t = hcpsdk.Target(P_FQDN, auth, port=P_PORT, sslcontext=ctxt)
    except hcpsdk.HcpsdkError as e:
        sys.exit('init of *Target* failed - {}'.format(e))
    else:
        print('Target *t* was initialized with IP addresses: {}'.format(t.addresses
```

10.2.2 Sample code output

Certificate verification success, with debug messages

```
 INFO running *simple_primary_only.py*
DEBUG *I_NATIVE* authorization initialized for user: n
DEBUG pre version 6:     Cookie: hcp-ns-auth=bg==:1dc7fed37e11b35093d311ef66928ad9
DEBUG version 6+: Authorization: HCP bg==:1dc7fed37e11b35093d311ef66928ad9
 INFO *I_NATIVE* authorization initialized
 INFO
DEBUG (re-) loaded IP address cache: ['192.168.0.54', '192.168.0.55', '192.168.0.52', '1
DEBUG issued IP address: 192.168.0.54
DEBUG Target initialized: n1.m.hcp1.snomis.local:443 - SSL = True
 INFO Target *t* was initialized with IP addresses: ['192.168.0.54', '192.168.0.55', '19
DEBUG Connection object initialized: IP None (n1.m.hcp1.snomis.local) - timeout: 30 - id
DEBUG SSLcontext = <ssl.SSLContext object at 0x101a2c638>
 INFO Connection *c* uses IP address: None
 INFO
DEBUG tried to cancel a non-existing idletimer (pretty OK)
DEBUG URL = /rest/hcpsdk/sample_primary_only.txt
DEBUG Connection needs to be opened
DEBUG issued IP address: 192.168.0.55
DEBUG Connection open: IP 192.168.0.55 (n1.m.hcp1.snomis.local) - connect_time: 3.004074
DEBUG PUT Request for /rest/hcpsdk/sample_primary_only.txt - service_time1 = 0.039813041
DEBUG tried to cancel a non-existing idletimer (pretty OK)
DEBUG idletimer started: <Timer(Thread-1, started 4350545920)>
 INFO PUT Request was successful
 INFO used IP address: 192.168.0.55
 INFO hash = SHA-256 A2706A20394E48179A86C71E82C360C2960D3652340F9B9FDB355A42E3AC7691
 INFO connect time:     0.000030040741 seconds
 INFO Request duration: 0.039813041687 seconds
 INFO
DEBUG idletimer canceled: <Timer(Thread-1, started 4350545920)>
DEBUG URL = /rest/hcpsdk/sample_primary_only.txt
DEBUG HEAD Request for /rest/hcpsdk/sample_primary_only.txt - service_time1 = 0.00040006
DEBUG tried to cancel a non-existing idletimer (pretty OK)
DEBUG idletimer started: <Timer(Thread-2, started 4350545920)>
 INFO HEAD Request was successful - one of the headers:
 INFO Server: HCP V7.1.0.10
 INFO used IP address: 192.168.0.55
 INFO Request duration: 0.000400066376 seconds
 INFO
DEBUG idletimer canceled: <Timer(Thread-2, started 4350545920)>
DEBUG URL = /rest/hcpsdk/sample_primary_only.txt
DEBUG GET Request for /rest/hcpsdk/sample_primary_only.txt - service_time1 = 0.00018382
DEBUG tried to cancel a non-existing idletimer (pretty OK)
DEBUG idletimer started: <Timer(Thread-3, started 4350545920)>
 INFO GET Request was successful - here's the content:
DEBUG (partial?) read: service_time1 = 0.03570699691772461 secs
 INFO b'0123456789abcdef0123456789abcdef01234567'...
 INFO used IP address: 192.168.0.55
 INFO Request duration: 0.035890817642 seconds
 INFO
DEBUG idletimer canceled: <Timer(Thread-3, started 4350545920)>
DEBUG URL = /rest/hcpsdk/sample_primary_only.txt
DEBUG DELETE Request for /rest/hcpsdk/sample_primary_only.txt - service_time1 = 0.000231
DEBUG tried to cancel a non-existing idletimer (pretty OK)
```

```
DEBUG idletimer started: <Timer(Thread-4, started 4350545920)>
 INFO DELETE Request was successful
 INFO used IP address: 192.168.0.55
 INFO Request duration: 0.000231027603 seconds
 INFO
DEBUG idletimer canceled: <Timer(Thread-4, started 4350545920)>
DEBUG Connection object closed: IP 192.168.0.55 (n1.m.hcp1.snomis.local)
```

Certificate verification failed, with debug messages

(**P_CAFILE** has been changed to a file holding a non-matching CA chain)

```
 INFO running *simple_primary_only.py*
DEBUG *I_NATIVE* authorization initialized for user: n
DEBUG pre version 6:     Cookie: hcp-ns-auth=bg==:1dc7fed37e11b35093d311ef66928ad9
DEBUG version 6+: Authorization: HCP bg==:1dc7fed37e11b35093d311ef66928ad9
 INFO *I_NATIVE* authorization initialized
 INFO
DEBUG (re-) loaded IP address cache: ['192.168.0.53', '192.168.0.54', '192.168.0.55', '1
DEBUG issued IP address: 192.168.0.53
DEBUG Target initialized: n1.m.hcp1.snomis.local:443 - SSL = True
 INFO Target *t* was initialized with IP addresses: ['192.168.0.53', '192.168.0.54', '19
DEBUG Connection object initialized: IP None (n1.m.hcp1.snomis.local) - timeout: 30 - id
DEBUG SSLcontext = <ssl.SSLContext object at 0x102220638>
 INFO Connection *c* uses IP address: None
 INFO
DEBUG tried to cancel a non-existing idletimer (pretty OK)
DEBUG URL = /rest/hcpsdk/sample_primary_only.txt
DEBUG Connection needs to be opened
DEBUG issued IP address: 192.168.0.54
DEBUG Connection open: IP 192.168.0.54 (n1.m.hcp1.snomis.local) - connect_time: 2.598762
DEBUG Request raised exception: [SSL: CERTIFICATE_VERIFY_FAILED] certificate verify fail
DEBUG tried to cancel a non-existing idletimer (pretty OK)
DEBUG Connection object closed: IP 192.168.0.54 (n1.m.hcp1.snomis.local)
PUT failed - [SSL: CERTIFICATE_VERIFY_FAILED] certificate verify failed (_ssl.c:600)
```

CHAPTER

ELEVEN

LICENSE

The MIT License (MIT)

Copyright (c) 2014-2015 Thorsten Simons (sw@snomis.de[1])

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

[1] sw@snomis.de

CHAPTER

TWELVE

ABOUT

About the Developer

The developer serves for Hitachi Data Systems since 2007, with a main focus on **Hitachi Content Platform** and its family of related products. Being a presales consultant with HDS Germany for more than six years, he actually works for the HCP engineering department as an HCP Technologist for the EMEA region.

Prior to HDS, he served for eight years as a presales manager for a major storage vendor in Germany. Before that, he worked for ten years as a software developer, system programmer, project manager and technical architect for a major German manufacturing company.

In his spare time, he develops tools around HCP that make his own (and hopefully) others life easier.

You can contact him per email at sw@snomis.de[1]

[1] sw@snomis.de

CHAPTER

THIRTEEN

APPENDIXES

13.1 Appendix 1 - Default Namespace

hcpsdk is primarily targeting the *authenticated Namespaces* invented with HCP version 3.

Nevertheless, it is possible to use **hcpsdk** with the legacy *Default Namespace* by taking notice of a few differences:

- As there is no user authentication with the Default Namespace, use the *hcpsdk.DummyAuthorization* class as the *authorization* argument when instantiating an *hcpsdk.Target*
- Different from authenticated Namespaces, the root folder for requests is `/fcfs_data` (instead of `/rest`)
- An *HEAD* request for an object stored in the Default Namespace will yield very limited object metadata, only. If you need more, you need to request the metadata by a *GET* from `/fcfs_metadata/your/path/to/object/core-metadata.xml`
- The Default Namespace can have a single annotation (custom metadata), only. You need to request it by a *GET* from `/fcfs_metadata/your/path/to/object/custom-metadata.xml`
- Several actions you can trigger by a *POST* request to an authenticated Namespace need to be executed by a *PUT* to one of the files in `/fcfs_metadata/your/path/to/object/`.

Note: Consult the manual **Using the Default Namespace** available from the HCP System and Tenant Management Consoles for details about working with the Default Namespace.

13.1.1 Example

```
>>> import hcpsdk
>>>
>>> auth = hcpsdk.DummyAuthorization()
>>> t = hcpsdk.Target('default.default.hcp1.snomis.local', auth, port=443)
>>> c = hcpsdk.Connection(t)
>>> c.connect_time
'0.000000000010'
>>>
>>> r = c.PUT('/fcfs_data/hcpsdk/test1.txt', body='This is an example', params={'index':
>>> c.response_status, c.response_reason
(201, 'Created')
```

```
>>>
>>> r = c.HEAD('/fcfs_data/hcpsdk/test1.txt')
>>> c.response_status, c.response_reason
(200, 'OK')
>>> c.getheaders()
[('Date', 'Wed, 18 Feb 2015 16:48:49 GMT'),
 ('Server', 'HCP V7.1.0.10'),
 ('X-ArcServicedBySystem', 'hcp1.snomis.local'),
 ('X-ArcClusterTime', '1424278129'),
 ('X-RequestId', '6BB17FCE72FECA84'),
 ('X-HCP-ServicedBySystem', 'hcp1.snomis.local'),
 ('X-HCP-Time', '1424278129'),
 ('X-HCP-SoftwareVersion', '7.1.0.10'),
 ('ETag', '"68791e1b03badd5e4eb9287660f67745"'),
 ('Cache-Control', 'no-cache,no-store'),
 ('Pragma', 'no-cache'),
 ('Expires', 'Thu, 01 Jan 1970 00:00:00 GMT'),
 ('Content-Type', 'text/plain'),
 ('Content-Length', '18'),
 ('X-ArcPermissionsUidGid', 'mode=0100400; uid=0; gid=0'),
 ('X-ArcTimes', 'ctime=1424278066; mtime=1424278066; atime=1424278065'),
 ('X-ArcSize', '18')]
>>>
>>> r = c.GET('/fcfs_data/hcpsdk/test1.txt')
>>> c.response_status, c.response_reason
(200, 'OK')
>>> c.read()
b'This is an example'
>>> c.service_time2
0.00039696693420410156
>>>
>>> r = c.DELETE('/fcfs_data/hcpsdk/test1.txt')
>>> c.response_status, c.response_reason
(200, 'OK')
>>> c.service_time2
0.0001819133758544922
>>>
>>> c.close()
```

CHAPTER

FOURTEEN

GLOSSARY

Data Access Account A local user within a *Tenant*

Default Namespace The legacy Namespace, a relict from the time before HCP version 3. Doesn't support user authentication. Seldomly used in our days, deprecated. (*default.default.hcp.domain.com* or *www.hcp.domain.com*)

DNS Domain Name System - used to translate an *FQDN* to IP addresses

FQDN Full Qualified Domain Name (*namespace.tenant.hcp.domain.com*)

HS3 Amazon S3 compatible interface

HSwift OpenStack Swift compatible interface

MAPI Management API - a *reSTful* interface to manage HCP

Namespace A Namespace is a addressable data store, separated from other namespaces, having an individual filesystem-like structure, several access protocols along with a set of other configurables.

reST *representational State Transfer*

An architectural approach for client/server communication for performance, scalability, simplicity, reliability and more. See the Wikipedia entry[1] for more details.

Tenant A Tenant within HCP is an administrative entity that allows to configure and manage a set of *namespaces* within a configurable storage quota, along with the user account required to access data.

[1] http://en.wikipedia.org/wiki/Representational_state_transfer

h

hcpsdk, 7
hcpsdk.ips, 15
hcpsdk.mapi, 25
hcpsdk.namespace, 17
hcpsdk.pathbuilder, 21

Symbols

_addresses (hcpsdk.ips.Circle attribute), 15

A

address (hcpsdk.Connection attribute), 9
addresses (hcpsdk.Target attribute), 8

C

cache (hcpsdk.ips.Response attribute), 15
Circle (class in hcpsdk.ips), 15
close() (hcpsdk.Connection method), 11
connect_time (hcpsdk.Connection attribute), 9
Connection (class in hcpsdk), 9

D

Data Access Account, **43**
Default Namespace, **43**
DELETE() (hcpsdk.Connection method), 10
DNS, **43**
DummyAuthorization (class in hcpsdk), 7

F

FQDN, **43**
fqdn (hcpsdk.ips.Response attribute), 15
fqdn (hcpsdk.Target attribute), 8

G

GET() (hcpsdk.Connection method), 10
getaddr() (hcpsdk.Target method), 9
getheader() (hcpsdk.Connection method), 10
getheaders() (hcpsdk.Connection method), 10
getlinkdetails() (hcpsdk.mapi.Replication method), 26
getlinklist() (hcpsdk.mapi.Replication method), 26
getpath() (hcpsdk.pathbuilder.PathBuilder method), 22
getreplicationsettings() (hcpsdk.mapi.Replication method), 26
getunique() (hcpsdk.pathbuilder.PathBuilder method), 22

H

hcpsdk (module), 7
hcpsdk.ips (module), 15
hcpsdk.mapi (module), 25
hcpsdk.namespace (module), 17
hcpsdk.pathbuilder (module), 21
HcpsdkCertificateError, 11
HcpsdkError, 11
HcpsdkReplicaInitError, 11
HcpsdkTimeoutError, 11
HEAD() (hcpsdk.Connection method), 10
headers (hcpsdk.Target attribute), 8
HS3, **43**
HSwift, **43**

I

I_DUMMY (hcpsdk.Target attribute), 8
I_NATIVE (hcpsdk.Target attribute), 8
Info (class in hcpsdk.namespace), 17
ips (hcpsdk.ips.Response attribute), 15
IpsError, 16

L

listaccessiblens() (hcpsdk.namespace.Info method), 17
listpermissions() (hcpsdk.namespace.Info method), 18
listretentionclasses() (hcpsdk.namespace.Info method), 18

M

MAPI, **43**

N

Namespace, **43**
NativeAuthorization (class in hcpsdk), 7
nsstatistics() (hcpsdk.namespace.Info method), 17

P

PathBuilder (class in hcpsdk.pathbuilder), 21
PathBuilderError, 22

port (hcpsdk.Target attribute), 8
POST() (hcpsdk.Connection method), 10
PUT() (hcpsdk.Connection method), 10

Q

query() (in module hcpsdk.ips), 16

R

R_ACTIVE_ACTIVE (hcpsdk.mapi.Replication attribute), 25
R_BEGINRECOVERY (hcpsdk.mapi.Replication attribute), 25
R_COMPLETERECOVERY (hcpsdk.mapi.Replication attribute), 25
R_FAILBACK (hcpsdk.mapi.Replication attribute), 25
R_FAILOVER (hcpsdk.mapi.Replication attribute), 25
R_INBOUND (hcpsdk.mapi.Replication attribute), 25
R_OUTBOUND (hcpsdk.mapi.Replication attribute), 25
R_RESTORE (hcpsdk.mapi.Replication attribute), 25
R_RESUME (hcpsdk.mapi.Replication attribute), 25
R_SUSPEND (hcpsdk.mapi.Replication attribute), 25
raised (hcpsdk.ips.Response attribute), 16
read() (hcpsdk.Connection method), 10
reason (hcpsdk.HcpsdkCertificateError attribute), 11
reason (hcpsdk.HcpsdkError attribute), 11
reason (hcpsdk.HcpsdkReplicaInitError attribute), 11
reason (hcpsdk.HcpsdkTimeoutError attribute), 11
reason (hcpsdk.ips.IpsError attribute), 16
reason (hcpsdk.mapi.ReplicationSettingsError attribute), 27
reason (hcpsdk.pathbuilder.PathBuilderError attribute), 22
refresh() (hcpsdk.ips.Circle method), 15
replica (hcpsdk.Target attribute), 8
Replication (class in hcpsdk.mapi), 25
ReplicationSettingsError, 27
request() (hcpsdk.Connection method), 9
Response (class in hcpsdk.ips), 15
Response (hcpsdk.Connection attribute), 9
response_reason (hcpsdk.Connection attribute), 9
response_status (hcpsdk.Connection attribute), 9
reST, **43**
RS_READ_ALLOWED (hcpsdk.Target attribute), 8
RS_READ_ON_FAILOVER (hcpsdk.Target attribute), 8
RS_WRITE_ALLOWED (hcpsdk.Target attribute), 8
RS_WRITE_ON_FAILOVER (hcpsdk.Target attribute), 8

S

service_time1 (hcpsdk.Connection attribute), 9
service_time2 (hcpsdk.Connection attribute), 9
setreplicationlinkstate() (hcpsdk.mapi.Replication method), 27
ssl (hcpsdk.Target attribute), 8

T

Target (class in hcpsdk), 7
Tenant, **43**

V

version() (in module hcpsdk), 7