
# HCP SDK Documentation

***Release 0.9.5-1***

**Thorsten Simons**

**Jun 29, 2023**

# Table of Contents

CHAPTER 1

# 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.

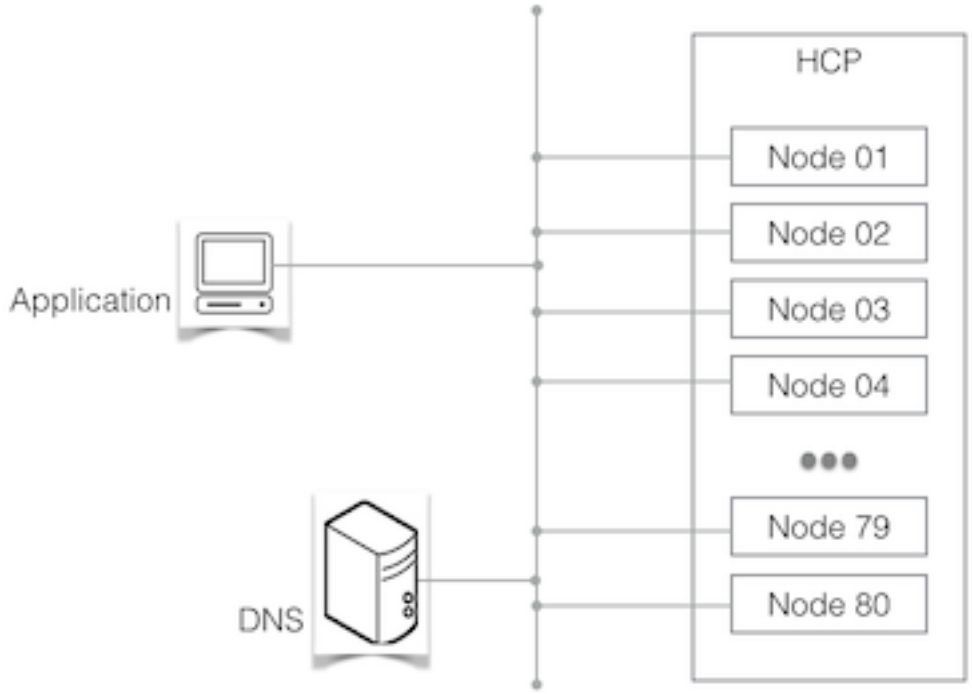


Fig. 1: A simple HCP environment

## 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 67).

## 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 29) 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).

## 1.5 Using the proper URL

**hcpsdk** tries to be as open as possible - that's why it doesn't pretend URLs prefixed for the various gateways HCP offers. The same is true for *headers* that are required for various request (espacially for HSwift and MAPI). It's up to you to use the correct URL for the gateway used, as well as to add *headers* where required. You might want to consult the HCP manuals for the details.

For convenience, here are the mainly used URLs:

- **native http(s)/REST to an authenticated Namespace:**

  `FQDN: namespace.tenant.hcp.dom.com`
  `URL: /rest/<your>/<folders>/object`

- **native http(s) to the default Namespace:**

  `FQDN: default.default.hcp.dom.com`
  `URL: /fcfs_data/<your>/<folders>/object`
  *or, if metadata access is needed:*
  `URL: /fcfs_metadata/<your>/<folders>/object/metafile`

- **HSwift http(s)/REST to an authenticated Namespace:**

  `FQDN: api.hcp.dom.com`
  `URL: /swift/v1/<tenant>/<namespace>/<your>/<folders>/object`

- **http(s)/REST to MAPI:**

  `FQDN: admin.hcp.dom.com` (when using a system level user)
  -or-
  `FQDN: <tenant>.hcp.dom.com` (when using a tenant level user)
  `URL: /mapi/<endpoint>`

CHAPTER 2

# State of Implementation

**Release** 0.9.5-1

- 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 29) 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:

  - Chargeback report download (requires at least HCP 5.0)

    New in version 0.9.4.

  - Log file download (requires at least HCP 7.2)

    New in version 0.9.3.

  - Replication link information, link failover/failback (requires at leat HCP 7.0)

  - Tenant management (mostly listing Tenants, yet)

    New in version 0.9.4.

CHAPTER 3

# Installation

## 3.1 Dependencies

**hcpsdk** depends on these packages:

- dnspython[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)

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

## 3.3 Contribute

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

## 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[5]

[4] https://github.com/simont3/hcpsdk/issues
[5] sw@snomis.de

CHAPTER 4

# Thread safety

These classes **are** thread-safe:

- *hcpsdk.Target()* (page 15)

  Within an application, create one *Target()* object per HCP you need to connect to.

- *hcpsdk.ips.Circle()* (page 23)

  This class is intended as an internal class for *hcpsdk.Target()*, so normally there is no need to instantiate it directly.

These classes **are not** thread-safe:

- *hcpsdk.Connection()* (page 16)

  A *Connection()* should be used within a single thread, only (or you need to provide your own locks to orchestrate usage).

- *hcpsdk.namespace.Info()* (page 25)
- *hcpsdk.pathbuilder.PathBuilder()* (page 30)
- *hcpsdk.mapi.Logs()* (page 36)
- *hcpsdk.mapi.Replication()* (page 44)

  These classes create their own *hcpsdk.Connection()* uppon the provided *hcpsdk.Target()*, each of them needs to stay within a single thread:

CHAPTER 5

# 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[6].

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

CHAPTER 6

# hcpsdk — object access

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

Setup is easy (*see example below* (page 20)):

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!

## 6.1 Functions

### 6.1.1 version

hcpsdk.**version**()
Return the full version of the HCPsdk (0.9.5-1).

### 6.1.2 checkport

hcpsdk.**checkport**(*target*, *port*)
Check if an *hcpsdk.Target()* object is initialized with the correct port.

**Parameters**

- **target** – the *hcpsdk.Target()* object to check
- **port** – the needed port

**Returns** nothing

**Raises** *hcpsdk.HcpsdkPortError* in case the port is invalid

## 6.2 Constants

**Ports**

hcpsdk.**P_HTTP**
Port 80 - insecure Namespace access

hcpsdk.**P_HTTPS**
Port 443 - secure Namespace access

hcpsdk.**P_SEARCH**
Port 8888 - search facility

hcpsdk.**P_MAPI**
Port 9090 - Management API access

**Interfaces**

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

hcpsdk.**I_HSWIFT**
HCP's http dialect for access to HCPs *HSwift* gateway.

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

## 6.3 Classes

### 6.3.1 NativeAuthorization

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

**Parameters**

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

### 6.3.2 NativeADAuthorization

**class** hcpsdk.**NativeADAuthorization**(*user*, *password*)
Authorization for native http/REST access to HCP using an Active Directory user.

Supported with HCP 7.2.0 and later. The user needs to be a member of the Active Directory domain in which HCP is joined.

**Parameters**

- **user** – an Active Directory user
- **password** – his password

New in version 0.9.4.0.

### 6.3.3 LocalSwiftAuthorization

**class** hcpsdk.**LocalSwiftAuthorization**(*user*, *password*)
Authorization for local *HSwift* access to HCP (w/o Keystone).

**Parameters**

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

New in version 0.9.3.10.

### 6.3.4 DummyAuthorization

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

### 6.3.5 Target

**class** hcpsdk.**Target**(*fqdn*, *authorization*, *port=443*, *dnscache=False*, *sslcontext=<ssl.SSLContext object>*, *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** – one of the port constants (*hcpsdk.P_**)
- **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** – OR'ed combination of the RS_* modes

**Raises** *ips.IpsError* if DNS query fails, *HcpsdkError* in all other fault cases

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

**Returns** an IP address (as string)

**fqdn**
The FQDN for which this object was initialized (r/o)

**interface**
The HCP interface used (r/o)

**port**
The target port in use (r/o)

**ssl**
Indicates if SSL is used (r/o)

**sslcontext**
The assigned SSL context (r/o)

**addresses**
The list of resolved IP addresses for this target (r/o)

**headers**
The calculated authorization headers (r/o)

**replica**
The target object for the HCP replica, if set (r/o)

**replica_strategy**
The replica strategy selected (r/o)

### 6.3.6 Connection

**class** hcpsdk.**Connection**(*target*, *timeout=30*, *idletime=30*, *retries=0*, *debuglevel=0*, *sock_keepalive=False*, *tcp_keepalive=60*, *tcp_keepintvl=60*, *tcp_keepcnt=3*)
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[7]
- **sock_keepalive** – enable TCP keepalive, if True
- **tcp_keepalive** – idle time used when SO_KEEPALIVE is enable

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

- **tcp_keepintvl** – interval between keepalives
- **tcp_keepcnt** – number of keepalives before close

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

a) the underlying connection has been closed by HCP before *idletime* has passed (the request will be retried using the existing connection context) or

b) 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.

You should rarely need this, but if you have a device in the data path that limits the time an idle connection can be open, this might be of help:

Setting *sock_keepalive* to *True* enables TCP keep-alive for this connection. *tcp_keepalive* defines the idle time before a keep-alive packet is first sent, *tcp_keepintvl* is the time between keep-alive packets and *tcp_keepcnt* is the number of keep-alive packets to be sent before failing the connection in case the remote end doesn't answer. See `man tcp` for the details.

New in version 0.9.4.3.

**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); url quoting will be done if necessary, but existing quoting will not be touched
- **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()*.

**Raises** one of the *hcpsdk.Hcpsdk[..]Error*s or *hcpsdk.ips.IpsError* in case an IP address cache refresh failed

**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*, *body=None*, *params=None*, *headers=None*)
Convenience method for Request() - POST metadata. Does no clean-up, as a POST can have a response body! 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.

**Raises** *HcpsdkTimeoutError* in case a socket.timeout was catched, *HcpsdkError* in all other cases.

**close**()
Close the Connection.

> **Warning: 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()*.

**address**
The IP address for which this object was initialized (r/o)

**con**
The internal connection object (r/o)

**Response**
Deprecated since version 0.9.4.2.

Use **response** instead!

**response**
Exposition of the http.client.Response object for the last Request (r/o)

New in version 0.9.4.2.

**response_status**
The HTTP status code of the last Request (r/o)

**response_reason**
The corresponding HTTP status message (r/o)

**connect_time**
The time in seconds the last connect took (r/o)

**service_time1**
The time in seconds 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 (r/o)

**service_time2**
Duration in secods of the complete Request up to now. Sum of all `service_time1` during handling a Request (r/o)

## 6.4 Exceptions

**exception** hcpsdk.**HcpsdkError**(*reason*)
Raised on generic errors in **hcpsdk**.

**Parameters** **reason** – an error description

**exception** hcpsdk.**HcpsdkCantConnectError**(*reason*)
Raised if a connection couldn't be established.

**Parameters** **reason** – an error description

**exception** hcpsdk.**HcpsdkTimeoutError**(*reason*)
Raised if a Connection timed out.

**Parameters** **reason** – an error description

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

**Parameters** **reason** – an error description

**exception** hcpsdk.**HcpsdkPortError**(*reason*)
Raised if the Target is initialized with an invalid port.

**Parameters** **reason** – an error description

**exception** hcpsdk.**HcpsdkReplicaInitError**(*reason*)
Raised if the setup of the internal *Target* for the replica HCP failed (typically, this is a name resolution problem). **If this exception is raised, the primary Target's init failed, too.** You'll need to retry!

**Parameters** **reason** – an error description

## 6.5 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': 'true'})
>>> 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 47FB563CC8F86DC37C86D08BC542968F7986ACD81C97B'
                'F76DB7AD744407FE117'),
 ('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'),
```

(continues on next page)

(continued from previous page)

```
 ('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 7

# 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.

## 7.1 Functions

### 7.1.1 query

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

## 7.2 Classes

### 7.2.1 Circle

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

Resolve an FQDN (using **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

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

**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

### 7.2.2 Response

**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.

## 7.3 Exceptions

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

**Parameters** **reason** – an error description

CHAPTER 8

# 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*.

**Warning:** Using this class with HCP prior to version 6 will deliver partial or no results when **hcpsdk.Target** is configured for the *Default Namespace*.

## 8.1 Classes

### 8.1.1 Info

**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,
```

(continues on next page)

(continued from previous page)

```
 '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 unspecifie
↪'
                                        'd value.',
                         'name': 'initial_unspecified',
                         'value': -2},
```

(continues on next page)

(continued from previous page)

```
 '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,
```

(continues on next page)

(continued from previous page)

```
                                    'read': True,
                                    'readAcl': True,
                                    'search': True,
                                    'write': True,
                                    'writeAcl': True}}
```

## 8.2 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 9

# `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[8]. The algorithm uses (one of) the servers MAC addresses, along with the system time to create the UUID.

## 9.1 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] http://tools.ietf.org/pdf/rfc4122.pdf

## 9.2 Classes

### 9.2.1 PathBuilder

**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**

- **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'
>>>
```

## 9.3 Exceptions

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

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

**Parameters** **reason** – an error description

# CHAPTER 10

# hcpsdk.mapi — MAPI access

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

**Note:** HCP needs to have the Management API (MAPI) enabled to make use of the classes in this module.

The *hcpsdk.Target()* object used needs to be initialized with *port=hcpsdk.P_MAPI*.

## 10.1 MAPI - Chargeback Resources

New in version 0.9.4. This class allows to request chargback reports from HCP.

HCP needs to have MAPI enabled for the system itself as well as for every *Tenant* to collect from.
A system-level user can collect summed-up reports from all *Tenants* that have MAPI enabled; he can collect *Namespace*-level reports from all *Tenants* that have granted system-level admin access.
A *Tenant*-level admin can collect reports for *Namespaces* as well as a summed-up report for the *Tenant* itself.

### 10.1.1 Classes

#### 10.1.1.1 Chargeback

**class** hcpsdk.mapi.**Chargeback**(*target*, *timeout=600*, *debuglevel=0*)
Access to HCP chargeback reports

**Parameters**

- **target** – an hcpsdk.Target object pointing to an HCP FQDN starting with **admin.** for access from a system level account or **<tenant>.** for a tenant level account
- **timeout** – the connection timeout; relatively high per default, as generating the report can take longer than **hcpsdk**s default of 30 seconds on a busy system
- **debuglevel** – 0..9 (used in *http.client*)

**Class constants:**

Collection periods:

**CBG_DAY**

**CBG_HOUR**

**CBG_TOTAL**

Output formats:

**CBM_CSV**

**CBM_JSON**

**CBM_XML**

**Class methodes**

**request**(*tenant=None*, *start=None*, *end=None*, *granularity='total'*, *fmt='application/json'*)
Request a chargeback report for a Tenant.

**Parameters**

- **tenant** – the *Tenant* to collect from
- **start** – starttime (a datetime object)
- **end** – endtime (a datetime object)
- **granularity** – one out of CBG_ALL
- **fmt** – output format, one out of CBM_ALL

**Returns** a file-like object in text-mode containing the report

**close**()
Close the underlying *hcpsdk.Connection* object.

### 10.1.2 Exceptions

**exception** hcpsdk.mapi.**ChargebackError**(*reason*)
Base Exception used by the *hcpsdk.mapi.Chargeback()* class.

**Parameters** **reason** – An error description

### 10.1.3 Sample Code

Note that the last record in this example shows the *Tenants* overall values, while the the record before shows statistics for a single *Namespace*.

```
>>> import hcpsdk
>>> auth = hcpsdk.NativeAuthorization('service', 'service01')
>>> tgt = hcpsdk.Target('admin.hcp73.archivas.com', auth,
                        port=hcpsdk.P_MAPI)
>>> cb = hcpsdk.mapi.Chargeback(tgt)
>>> result = cb.request(tenant='m',
                        granularity=hcpsdk.mapi.Chargeback.CBG_TOTAL,
                        fmt=hcpsdk.mapi.Chargeback.CBM_JSON)
>>> print(result.read())
{
  "chargebackData" : [ {
    "systemName" : "hcp73.archivas.com",
    "tenantName" : "m",
    "namespaceName" : "n1",
    "startTime" : "2015-11-04T15:27:29+0100",
    "endTime" : "2015-12-17T20:35:33+0100",
    "valid" : false,
    "deleted" : "false",
    "bytesOut" : 0,
    "reads" : 0,
    "writes" : 0,
    "deletes" : 0,
    "tieredObjects" : 0,
    "tieredBytes" : 0,
    "metadataOnlyObjects" : 0,
    "metadataOnlyBytes" : 0,
    "bytesIn" : 0,
    "storageCapacityUsed" : 25306468352,
    "ingestedVolume" : 25303387299,
    "objectCount" : 7219
  }, {
    "systemName" : "hcp73.archivas.com",
    "tenantName" : "m",
    "startTime" : "2015-11-04T15:27:29+0100",
    "endTime" : "2015-12-17T20:35:33+0100",
    "valid" : false,
    "deleted" : "false",
    "bytesOut" : 2156,
    "reads" : 2,
    "writes" : 1,
    "deletes" : 1,
    "tieredObjects" : 0,
    "tieredBytes" : 0,
    "metadataOnlyObjects" : 0,
    "metadataOnlyBytes" : 0,
    "bytesIn" : 5944,
    "storageCapacityUsed" : 25607081984,
    "ingestedVolume" : 25427708304,
    "objectCount" : 65607
  } ]
}
>>> cb.close()
```

## 10.2 MAPI - Log Resources

New in version 0.9.3. This class allows to request internal logs from HCP, which will then prepare a zip-file to be downloaded later on.

---

**Note:** To be able to use this class, HCP needs to run at least **version 7.2**.

---

The usage pattern is this:

- *prepare()*
- check the *status()* until *readyForStreaming* is True, then
- *download()* the logs
- save the zip'ed logs or process them, what ever is needed.
- *close()* the underlying *Connection()* object

### 10.2.1 Classes

#### 10.2.1.1 Logs

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

Access to HCP internal logfiles (ACCESS, SYSTEM, SERVICE, APPLICATION)

**Parameters**

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

**Raises**

***hcpsdk.HcpsdkPortError* in case *target* is** initialized with an incorrect port for use by this class.

**Class constants:**

Log types:

**L_ACCESS**
The log files *http_gateway_request.log.x*, *mapi_gateway_request.log.x* and *admin_request.log.x* contain all HTTP access messages that includes all of the HTTP based (SMC, TMC, MAPI, MQE, REST, HS3, HSwift) gateway logs. MAPI and Admin will contain all the system level access information while http gateway contains TMC and namespace access info.

**L_SYSTEM**
This is a snapshot of the running system when the logs were captured. This includes the messages that are generated by HCP OS, the firewall and routing rules, the fiber channel info and many others and are always important as a first step for support to go over.

**L_SERVICE**
These are logs generated when performing any of the service procedures such as when installing HCP software, adding a node, adding a LUN, node recovery etc.

**L_APPLICATION**
These are logs from all the HCP specific services such as the JVM, the volume manager, storman etc.

**L_ALL**
A list containing all other L_* log types.

**Class attributes**

**suggestedfilename**
When a *download* has been started, this attribute holds the filename suggested by HCP.

**Class methodes:**

**mark**(*message*)
Mark HCPs internal log with a message.

**Parameters** **message** – the message to be logged

**Raises** *LogsError*

**prepare**(*startdate=None*, *enddate=None*, *snodes=[]*)
Command HCP to prepare logs from *startdate* to *enddate* for later download.

**Parameters**

- **startdate** – 1st day to collect (as a *datetime.date* object)
- **enddate** – last day to collect (as a *datetime.date* object)
- **snodes** – list of S-series nodes to collect from

**Returns** a tuple of datetime.date(startdate), datetime.date(enddate) and str(prepared XML)

**Raises** *ValueError* arguments are invalid or one of the *LogsError* if an operation failed

**status**()
Query HCP for the status of the request log download.

**Returns**

a *collection.OrderedDict*:

```
{
 readyForStreaming: bool,
 streamingInProgress: bool,
 started: bool,
 error: bool,
 content: list # one or more of: L_ACCESS, L_SYSTEM,
               # L_SERVICE, L_APPLICATION)
}
```

**Raises** re-raises whatever is raised below

**download**(*hdl=None*, *nodes=[]*, *snodes=[]*, *logs=[]*, *progresshook=None*, *hidden=True*)
Download the requested logs.

**Parameters**

- **hdl** – a file (or file-like) handle open for binary read/write or *None*, in which case a temporary file will be created
- **nodes** – list of node-IDs (int), all if empty
- **snodes** – list of S-node names (str), none if empty
- **logs** – list of logs (*L_**), all if empty
- **progresshook** – a function taking a single argument (the # of bytes received) that will be called after each chunk of bytes downloaded
- **hidden** – the temporary file created will be hidden if possible (see tempfile.TemporaryFile()[9])

**Returns** a 2-tuple: the file handle holding the received logs, positioned at byte 0 and the filename suggested by HCP

**Raises** *LogsError* or *LogsNotReadyError*

**cancel**()
Cancel a log request.

**Returns** *True* if cancel was successfull

**Raises** *LogsError* in case the cancel failed

**close**()
Close the underlying *hcpsdk.Connection()*.

### 10.2.2 Exceptions

**exception** hcpsdk.mapi.**LogsError**(*reason*)
Base Exception used by the *hcpsdk.mapi.Logs()* class.

**Parameters** **reason** – An error description

**exception** hcpsdk.mapi.**LogsNotReadyError**(*reason*)
Raised by *Logs.download()* in case there are no logs ready to be downloaded.

**Parameters** **reason** – An error description

**exception** hcpsdk.mapi.**LogsInProgessError**(*reason*)
Raised by *Logs.download()* in case the log download is already in progress.

**Parameters** **reason** – An error description

### 10.2.3 Sample Code

The following example code creates a simple command processor that allows to prepare and download logs from HCP.

[9] https://docs.python.org/3/library/tempfile.html#tempfile.TemporaryFile

```
# -*- coding: utf-8 -*-
# The MIT License (MIT)
#
# Copyright (c) 2014-2023 Thorsten Simons (sw@snomis.eu)
#
# 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.

import sys
from pprint import pprint
import logging
import cmd
from datetime import date, timedelta
import hcpsdk

USR = 'logmon'
PWD = 'logmon01'
TGT = 'admin.hcp72.archivas.com'
PORT = hcpsdk.P_MAPI


class LogsShell(cmd.Cmd):
    intro = 'HCP Log Download Shell.   Type help or ? to list␣
↪commands.\n'
    prompt = '==> '

    def preloop(self):
        self.nodes = []
        self.snodes = []
        self.logs = []
        self.start = date.today() - timedelta(days=7)
```

(continues on next page)

(continued from previous page)

```
        self.end = date.today()

    def do_what(self, arg):
        'what - show what is selected for download'
        print('start date:      {}'.format(self.start.strftime('
↪%Y/%m/%d')))
        print('end date:        {}'.format(self.end.strftime('%Y/
↪%m/%d')))

        print('selected nodes:  {}'.format(' '.join(self.nodes)␣
↪or \
                                           ' '.join([x.split('.
↪')[3]
                                                     for x in l.
↪target.addresses])))
        print('selected snodes: {}'.format(' '.join(self.snodes)␣
↪or 'none'))
        print('selected logs:   {}'.format(' '.join(self.logs) or␣
↪\
                                           ' '.join(hcpsdk.mapi.
↪Logs.L_ALL)))

    def do_status(self, arg):
        'status - query HCP for the log preparation status'
        log.debug('do_status() called')
        try:
            pprint(l.status())
        except Exception as e:
            print(e)

    def do_prepare(self, arg):
        'prepare - trigger HCP to prepare the logs for later␣
↪download'
        try:
            l.prepare(snodes=self.snodes, startdate=self.start,
                      enddate=self.end)
        except Exception as e:
            print('prepare failed: {}'.format(e))
        else:
            print('preparing for nodes: {}\n'
                  '              snodes: {}\n'
                  '          date range: {} - {}'
                  .format(' '.join(self.nodes) or 'all',
                          ' '.join(self.snodes) or 'none',
                          self.start.strftime('%Y/%m/%d'),
                          self.end.strftime('%Y/%m/%d')))

    def do_nodes(self, arg):
        'nodes [node_id,]* - select the nodes to download logs␣
↪from\n'\
        '                    (no argument selects all nodes)'
        if arg:
            self.nodes = []
            n = [x.split('.')[3] for x in l.target.addresses]
            for x in arg.split(','):
                if x in n:
```

(continues on next page)

(continued from previous page)

```
                    self.nodes.append(x)
                else:
                    print('invalid: {}'.format(x))
        else:
            self.nodes = []

    def do_snodes(self, arg):
        'snodes [snode_name,]* - select the S-nodes to download␣
↪logs from\n'\
        '                        (no argument disables S-Node log␣
↪download)'
        if arg:
            self.snodes = arg.split(',')
        else:
            self.snodes = []

    def do_logs(self, arg):
        'logs [ACCESS,|SYSTEM,|SERVICE,|APPLICATION]* - select␣
↪log types\n'\
        '                                                nothing␣
↪selects all'
        if arg:
            self.logs = []
            for x in [y.upper() for y in arg.split(',')]:
                if x in hcpsdk.mapi.Logs.L_ALL:
                    self.logs.append(x)
                else:
                    print('invalid: {}'.format(x))
        else:
            self.logs = []

    def do_start(self, arg):
        'start YYYY/MM/DD - select start date (default is a week␣
↪ago)'
        try:
            d = arg.split('/')
            self.start = date(int(d[0]),int(d[1]),int(d[2]))
        except Exception as e:
            print('invalid input - YYYY/MM/DD required...')

    def do_end(self, arg):
        'end YYYY/MM/DD - select end date (default is today)'
        try:
            d = arg.split('/')
            self.end = date(int(d[0]),int(d[1]),int(d[2]))
        except Exception as e:
            print('invalid input - YYYY/MM/DD required...')


    def do_download(self, filename, ):
        'download <filename> - donwload logs and store into file
↪<filename>'
        if not filename:
            print('Error: filename missing!')
            return
```

(continues on next page)

(continued from previous page)

```
        print('downloading for nodes: {}\n'
              '                snodes: {}\n'
              '                  logs: {}\n'
              '            date range: {} - {}\n'
              .format(' '.join(self.nodes) or \
                      ' '.join([x.split('.')[3] for x in l.target.
↪addresses]),
                      ' '.join(self.snodes) or 'none',
                      ' '.join(self.logs) or \
                      ' '.join(hcpsdk.mapi.Logs.L_ALL,),
                      self.start.strftime('%Y/%m/%d'),
                      self.end.strftime('%Y/%m/%d')))
        try:
            with open(filename, 'w+b') as outhdl:
                hdl, n = l.download(hdl=outhdl, nodes=self.nodes,
                                    snodes=self.snodes, logs=self.
↪logs,

↪progresshook=showdownloadprogress)
                print('\nHCP suggested filename: {}'.format(n))
        except Exception as e:
            print('Error: {}'.format(e))
        else:
            print('download finished')

    def do_cancel(self, arg):
        'cancel - abort a log preparation'
        try:
            l.cancel()
        except Exception as e:
            print('cancel failed: {}'.format(e))
        else:
            print('cancel done')

    def do_mark(self, arg):
        'mark - mark HCPs log with a message'
        try:
            l.mark(arg)
        except Exception as e:
            print('mark failed: {}'.format(e))
        else:
            print('log marked')

    def do_quit(self, arg):
        'quit - exit the HCP Logs Shell'
        self.close()
        print('Bye...')
        return True

    def do_debug(self, arg):
        'debug - Toggle debug output'
        if log.getEffectiveLevel() != logging.DEBUG:
            log.setLevel(logging.DEBUG)
            log.debug('debug enabled')
            print('debug enabled')
        else:
```

(continues on next page)

(continued from previous page)

```
            log.setLevel(logging.CRITICAL)
            print('debug disabled')

    def emptyline(self):
        'run the status command if no command is given'
        return self.onecmd('status')

    def close(self):
        'close the underlying *hcpsdk.Logs()* object'
        l.close()


def showdownloadprogress(nBytes):
    '''
    Print a simple progress meter
    '''
    sz = ["B", "kB", "MB", "GB", "TB", "PB", "EB", "ZB", "YB"]
    i = 0
    while nBytes > 1023:
                    nBytes = nBytes / 1024
                    i = i + 1
    print("\rreceived: {0:.2f} {1:}".format(nBytes, sz[i]), end='
↪')


if __name__ == '__main__':

    auth = hcpsdk.NativeAuthorization(USR,PWD)
    try:
        t = hcpsdk.Target(TGT, auth, port=PORT)
    except hcpsdk.ips.IpsError as e:
        sys.exit('Can\'t resolve "{}"\n\t==> {}'.format(TGT, e))
    l = hcpsdk.mapi.Logs(t, debuglevel=0)

    # create console handler with a higher log level
    sh = logging.StreamHandler(sys.stderr)
    fh = logging.Formatter("[%(levelname)-8s]: %(message)s")
    sh.setFormatter(fh)
    log = logging.getLogger()
    log.addHandler(sh)
    # this makes the logger silent, until *debug* has activated
    log.setLevel(logging.CRITICAL)


    LogsShell().cmdloop()
```

### 10.2.4 Sample Output

```
HCP Log Download Shell.   Type help or ? to list commands.

==> logs access
==> snodes snode70
==> start 2015/09/23
==> what
```

(continues on next page)

(continued from previous page)

```
start date:      2015/09/23
end date:        2015/09/25
selected nodes:  176
selected snodes: snode70
selected logs:   ACCESS
==> prepare
preparing for nodes: all
             snodes: snode70
         date range: 2015/09/23 - 2015/09/25
==> status
{'readyForStreaming': False,
 'streamingInProgress': False,
 'error': False,
 'started': True,
 'content': ['ACCESS', 'SYSTEM', 'SERVICE', 'APPLICATION']}
==> status
{'readyForStreaming': True,
 'streamingInProgress': False,
 'error': False,
 'started': True,
 'content': ['ACCESS', 'SYSTEM', 'SERVICE', 'APPLICATION']}
==> download logs.zip
downloading for nodes: 176
               snodes: snode70
                 logs: ACCESS
           date range: 2015/09/23 - 2015/09/25

received: 11.41 MB
download finished
==> cancel
cancel done
==> quit
Bye...
```

## 10.3 MAPI - Replication management

This class allows to query HCP for replication links, their settings and state. It also allows to trigger a replication link failover anf failback.

**Note:** To be able to use this class, HCP needs to run at least **version 7.0**.

### 10.3.1 Classes

#### 10.3.1.1 Replication

**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*)

Alter the state of 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

### 10.3.2 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).

**Parameters** **reason** – An error description

### 10.3.3 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',
```

(continues on next page)

(continued from previous page)

```
                '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'}
>>>
```

## 10.4 MAPI - Tenant Resources

New in version 0.9.4. This class allows to work with the *Tenant* resources in HCP.

HCP needs to have MAPI enabled at the system-level.

### 10.4.1 Functions

#### 10.4.1.1 listtenants

hcpsdk.mapi.**listtenants**(*target*, *timeout=60*, *debuglevel=0*)

Get a list of available Tenants

**Parameters**

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

**Returns** a list() of *Tenant()* objects

**Raises** *hcpsdk.HcpsdkPortError* in case *target* is initialized with a port different that *P_MAPI*

### 10.4.2 Classes

#### 10.4.2.1 Tenant

**class** hcpsdk.mapi.**Tenant**(*target*, *name*, *timeout=60*, *debuglevel=0*)

A class representing a Tenant

**Parameters**

- **target** – an hcpsdk.Target object
- **name** – the Tenants name

- **timeout** – the connection timeout in seconds
- **debuglevel** – 0..9 (used in *http.client*)

**Class attributes**

**name**
The name of the Tenant represented by this object.

**Class methods**

**info**(*cache=True*)
Get the settings of the Tenant

**Parameters** **cache** – a bool indicating if cached information shall be used

**Returns** a dict holding the Tenants settings

**close**()
Close the underlying *hcpsdk.Connection()* object

### 10.4.3 Exceptions

**exception** hcpsdk.mapi.**TenantError**(*reason*)
Base Exception used by the *hcpsdk.mapi.Tenant()* class

**Parameters** **reason** – An error description

### 10.4.4 Sample Code

```
>>> from pprint import pprint
>>> import hcpsdk
>>> auth = hcpsdk.NativeAuthorization('service', 'service01')
>>> tgt = hcpsdk.Target('admin.hcp73.archivas.com', auth, port=hcpsdk.P_
↪MAPI)
>>> tenants = hcpsdk.mapi.listtenants(tgt)
>>> pprint(tenants)
[<hcpsdk.mapi.tenant.Tenant object at 0x1032ce6a0>,
 <hcpsdk.mapi.tenant.Tenant object at 0x1032ce748>,
 <hcpsdk.mapi.tenant.Tenant object at 0x1032ce7b8>,
 <hcpsdk.mapi.tenant.Tenant object at 0x1032ce828>,
 <hcpsdk.mapi.tenant.Tenant object at 0x1032ce898>,
 <hcpsdk.mapi.tenant.Tenant object at 0x1032ce908>]
>>> tenants[0].name
'Default'
>>> pprint(tenants[0].info())
{'snmpLoggingEnabled': False,
 'syslogLoggingEnabled': False,
 'systemVisibleDescription': '',
 'tenantVisibleDescription': ''}
```

**Warning:**

The *Default Tenant* has far less properties compared to the *usual Tenants* HCP provides.
See *Appendix 1* (page 67).

```
>>> tenants[1].name
'm'
>>> pprint(tenants[1].info())
{'administrationAllowed': True,
 'authenticationTypes': {'authenticationType': ['LOCAL', 'EXTERNAL']},
 'complianceConfigurationEnabled': True,
 'dataNetwork': '[hcp_system]',
 'hardQuota': '200.00 GB',
 'managementNetwork': '[hcp_system]',
 'maxNamespacesPerUser': 100,
 'name': 'm',
 'namespaceQuota': 'None',
 'replicationConfigurationEnabled': True,
 'searchConfigurationEnabled': True,
 'servicePlanSelectionEnabled': True,
 'snmpLoggingEnabled': False,
 'softQuota': 85,
 'syslogLoggingEnabled': False,
 'systemVisibleDescription': 'Der üblicherweise als erstes erzeugte Tenant.
↪..',
 'tags': {'tag': []},
 'tenantVisibleDescription': '',
 'versioningConfigurationEnabled': True}
>>> for t in tenants:
...     t.close()
...
>>>
```

# CHAPTER 11

# Code samples

## 11.1 Simple object I/O without replica

### 11.1.1 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:
```

(continues on next page)

(continued from previous page)

```
            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:
```

(continues on next page)

(continued from previous page)

```
            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}
↪{msg}')
        # noinspection PyShadowingBuiltins
        print = pprint = logging.info
```

### 11.1.2 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',
                                               '192.168.0.55',
                                               '192.168.0.52']
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==:1dc7fed37e11b35093d311...
DEBUG version 6+: Authorization: HCP bg==:1dc7fed37e11b35093d311ef66928...
 INFO *I_NATIVE* authorization initialized
 INFO
```

(continues on next page)

(continued from previous page)

```
DEBUG (re-) loaded IP address cache: ['192.168.0.52', '192.168.0.53',
                                      '192.168.0.54', '192.168.0.55'],
                                      dnscache = False
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',
                                                     '192.168.0.54',
                                                     '192.168.0.55']
DEBUG Connection object initialized: IP None (n1.m.hcp1.snomis.local)
                                     - timeout: 30
                                     - idletime: 30.0 - retries: 3
 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: 0.0016319751739501953
DEBUG PUT Request for /rest/hcpsdk/sample_primary_only.txt
                       - service_time1 = 0.07865500450134277
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 = 0.0001850128173828125
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 = 0.000186920166015625
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)>
```

(continues on next page)

(continued from previous page)

```
DEBUG URL = /rest/hcpsdk/sample_primary_only.txt
DEBUG DELETE Request for /rest/hcpsdk/sample_primary_only.txt
                         - service_time1 = 0.0001800060272216797
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)
```

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

### 11.2.1 Code sample

This `code sample` is exactly the same as the one shown as *Simple object I/O without replica* (page 51), 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('')
```

(continues on next page)

(continued from previous page)

```
    # 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))
```

### 11.2.2 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==:1dc7fed37e11b35093d311...
DEBUG version 6+: Authorization: HCP bg==:1dc7fed37e11b35093d311ef66928...
 INFO *I_NATIVE* authorization initialized
 INFO
DEBUG (re-) loaded IP address cache: ['192.168.0.54', '192.168.0.55',
                                      '192.168.0.52', '192.168.0.53'],
                                      dnscache = False
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',
                                                     '192.168.0.52',
                                                     '192.168.0.53']
DEBUG Connection object initialized: IP None (n1.m.hcp1.snomis.local)
                                     - timeout: 30 - idletime: 30.0
                                     - retries: 3
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: 3.0040740966796875e-05
DEBUG PUT Request for /rest/hcpsdk/sample_primary_only.txt
                       - service_time1 = 0.03981304168701172
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
```

(continues on next page)

(continued from previous page)

```
↪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 = 0.0004000663757324219
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 = 0.0001838207244873047
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 = 0.00023102760314941406
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==:1dc7fed37e11b35093d311...
DEBUG version 6+: Authorization: HCP bg==:1dc7fed37e11b35093d311ef66928...
 INFO *I_NATIVE* authorization initialized
 INFO
DEBUG (re-) loaded IP address cache: ['192.168.0.53', '192.168.0.54',
                                      '192.168.0.55', '192.168.0.52'],
                                      dnscache = False
```

(continues on next page)

(continued from previous page)

```
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',
                                                    '192.168.0.55',
                                                    '192.168.0.52']
DEBUG Connection object initialized: IP None (n1.m.hcp1.snomis.local)
                                     - timeout: 30 - idletime: 30.0
                                     - retries: 3
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: 2.5987625122070312e-05
DEBUG Request raised exception: [SSL: CERTIFICATE_VERIFY_FAILED]
                                certificate verify failed (_ssl.c:600)
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 12

# License

The MIT License (MIT)

# Copyright (c) 2014-2023 Thorsten Simons (sw@snomis.eu)

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.

CHAPTER 13

# 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

CHAPTER 14

# Trademarks

Hitachi is a registered trademark of Hitachi, Ltd., in the United States and other countries. Hitachi Data Systems is a registered trademark and service mark of Hitachi, Ltd., in the United States and other countries.

Archivas, Hitachi Content Platform, Hitachi Content Platform Anywhere and Hitachi Data Ingestor are registered trademarks of Hitachi Data Systems Corporation.

All other trademarks, service marks, and company names in this document or web site are properties of their respective owners.

CHAPTER 15

# Appendixes

## 15.1 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.

### 15.1.1 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': 'true'})
>>> 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 16

## Release History

**0.9.5-1 2023-06-29**

- fixed a bug that caused HTTPS connections to fail

**0.9.4-12 2018-07-25**

- fixed a bug which caused GETs to fail with *hcpsdk.HcpsdkError: faulty read: 'NoneType' object has no attribute 'read'* caused by idletimer cutting off the connection while transferring data

**0.9.4-11 2018-03-27**

- fixed a bug in hcpsdk.Connection.request() which caused blanks not to be url-encoded

**0.9.4-10 2018-03-06**

- fixed a bug in hcpsdk.Connection.request() where some characters were not properly encoded

**0.9.4-9 2017-08-26**

- Better __repr__ and __str__ for hcpsdk.__init__.py

**0.9.4-8 2017-07-14**

- Fixed a bug where URLs containing '+' also have to be url-encoded

**0.9.4-7 2017-07-07**

- Fixed a bug where already url-encoded URLs were url-encoded, again

**0.9.4-6 2017-03-31**

- Fixed a bug in NativeADAuthorization that caused 403 errors (thanks to Kevin L)

**0.9.4-5 2016-12-19**

- Fixed a bug in setup.py that caused dnspython3 still be called in

**0.9.4-4 2016-12-06**

- Changed the pre-requisite from dnspython3 to dnspython >= 1.15.0. (dnspython 1.15.0 supports Python 2 and 3)

**0.9.4-3 2016-02-28**

- Added the posibility to enable TCP keep-alive for *hcpsdk.Connection*s
- Documentation slightly reformatted

**0.9.4-2 2016-02-04**

- Re-worked the representation of *Target()'s* and *Connection()'s* (mainly) read-only attributes; fixed documentation source accordingly
- Renamed attribute *Response* to *response* in *hcpsdk.Connection(). Response* is deprecated, but will stay available for backward compatibility

**0.9.4-1 2015-12-18**

- Fixed doctring of *hcpsdk.mapi.replication.setreplicationlinkstate()*

**0.9.4-0 2015-12-17**

- Added *hcpsdk.mapi.chargeback*
- Added *hcpsdk.mapi.tenant* (with limited functionality)
- Added support for Active Directory users (*hcpsdk.NativeADAuthorization()*)
- Added trademarks to documentation

**0.9.3.11 2015-12-06**

- Fixed a bug that lead *Connection.service_time2* to be too low

**0.9.3.10 2015-12-01**

- Added support for local Swift Authorization (w/o Keystone and
- an article about the proper use of URLs

**0.9.3-9 2015-11-08**

- Documentation fixes

**0.9.3-7 2015-11-08**

- Fixed return value of *mapi.log.download()*
- Made output of *mapilogshell.py* more precise

**0.9.3-6 2015-10-20 0.9.3-5 2015-10-20**

- Removed alabaster from the list of dependencies for source installs

**0.9.3-4 2015-10-12 0.9.3-3 2015-10-12**

- Fixed a bug in setup.py to enable pip install on Windows

**0.9.3-2 2015-10-12**

- Changed README.md and setup.py to adopt to Gitlab and readthedocs requirements

**0.9.3-1 2015-10-11**

- Documentation fixes

**0.9.3-0 2015-10-10**

- Added access to the *logs* endpoint invented with HCP 7.2 (*hcpsdk.mapi.Logs()*)

- Provide an example script, based on cmd.cmd() to manually explore the log download MAPI. See documentation.
- Splitted *hcpsdk.mapi* classes into separate packages (*logs* and *replication*, yet), while maintaining accessibility through *hcpsdk.mapi.Logs* and *hcpsdk.mapi.Replication*, respectively.
- Corrected the documentation buildout for Exceptions
- Fixed a bug that prevented internal connections to be closed when idletimer fired
- Fixed the logger initialization per class throughout the entire code base
- Added exception handling for *ConnectionRefusedError*

**0.9.2-38 2015-08-31**

- Added documentation about thread-safety.

**0.9.2-37 2015-08-18**

- Handle OSError by retry-ing the call in *hcpsdk.Connection.request()*
- *Connection.close()* now clears the underlying socket reference
- *Connection.close()* no more generates debug message if the connection wasn't open.
- repr(*hcpsdk.Target*) and repr(*hcpsdk.Connection*) now returns the memory address of the respecive object for debug purposes.

**0.9.2-36 2015-07-15**

- Disabled character '+' as save when urlencoding parameters in *hcpsdk.Connection.request()*

**0.9.2-35 2015-06-23**

- Removed some of the annoying debug messages stating idleTimer stopped/started

**0.9.2-34 2015-06-23**

- Now showing the number of bytes read in debug output in *hcpsdk.Connection.read()*

**0.9.2-32 2015-06-02**

- Now catching BrokenPipeError during *hcpsdk.Connection.request()*, leading to as many retries as requested for the connection.

**0.9.2-31 2015-05-26**

- Fixed pip installer pre-requisites

**0.9.2-30 2015-05-24**

- Fixed pip installer pre-requisites
- Documentation for *hcpsdk.namespace*: added hint about HCP version availability.

**0.9.2-29 2015-05-20**

- Fixed proper handling of http.client.CannotSendRequest in *Connection.request()*

**0.9.2-28 2015-05-20**

- If a socket.timeout is raised in *hcpsdk.Connection.read()*, re-raise it as *hcpsdk.HcpsdkTimeoutError*.

**0.9.2-27 2015-05-19**

- *hcpsdk.Connection.request()* is now aware of ResponseNotReady being raised during *http.client.HTTPConnection.getresponse()* and retries appropriately.

**0.9.2-26 2015-05-19**

- Corrected the behaviour of the 'all' parameter in hcpsdk.namespace.listaccessiblens()

**0.9.2-25 2015-05-13**

- One more debug message right after getResponse()

**0.9.2-24 2015-05-13**

- Added output of service_time2 to debug messages

**0.9.2-23 2015-05-13**

- Output of service times in debug messages set to 17 digits

**0.9.2-22 2015-05-13**

- Output of service times in debug messages are more precise, now

**0.9.2-21 2015-03-28**

- Tuned the exception handling in *hcpsdk.request()*
- Fixed/added testcases

**0.9.2-20 2015-03-26**

- Fixed/added testcases

**0.9.2-19 2015-03-26**

- *hcpsdk.Connection.request()*: changed behavior for the cases where we receive one of ConnectionAbortedError, http.client.ResponseNotReady, TimeoutError and socket.timeout. We now refresh the cached IP addresses and setup a new connection.

**0.9.2-18 2015-03-25**

- *hcpsdk.Connection.request()* accidentally quoted blanks in an URL as '+', which is not valid for HCP. Replaced *urllib.parse.quote_plus()* by *urllib.parse.quote()*.

**0.9.2-17 2015-03-24**

- *hcpsdk.Connection.request()* is now aware of timeouts that occur during *http.client.HTTPConnection.getresponse()* and retries appropriately.

**0.9.2-16 2015-03-22**

- *hcpsdk.Connection.close()* now checks if the underlying connection is really open before trying to close it.

**0.9.2-15 2015-03-22**

- *hcpsdk.Connection.request()* excluded '+' from being urlencoded in params.

**0.9.2-14 2015-03-20**

- *hcpsdk.Connection.POST()* now allows to add a body to the request.

**0.9.2-13 2015-03-16**

- Changed some unnecessary logging.error calls to logging.debug

**0.9.2-12 2015-03-16**

- Now raising HcpsdkReplicaInitError id a *hcpsdk.Target* is initialized with a replica HCP (not yet implemented).
- Improved error handling in *hcpsdk.Connection.request()*.
- *hcpsdk.Target()* will now raise *ips.IpsError* if DNS name resolution fails.

**0.9.2-11 2015-03-12**

- Fixed an issue in *hcpsdk.Connection.__str__()* where a false attribute was referenced.

**0.9.2-10 2015-03-11**

- Fixed an issue in *hcpsdk.Connection.request()* that led to situations where a failed connection wasn't recovered correctly.

**0.9.2-9 2015-03-09**

- Added missing import of subpackage pathbuilder into hcpsdk.__init__.py

**0.9.2-8 2015-03-09**

- Fixed:

  as *socket.getaddrinfo()* seems to double the resolved IP addresses under some circumstances, added a check to make sure we don't have duplicates in the result of *hcpsdk.ips.query()*

**0.9.2-7 2015-03-09**

- Fixed dependency handling, again...

**0.9.2-6 2015-03-08**

- Now handling *ConnectionAbortedError* properly in hcpsdk.Connection() by closing and re-opening the connection on the same target IP address

**0.9.2-5 2015-03-07**

- Fixed __all__ in several modules, some typos in comments

**0.9.2-4 2015-03-06**

- Added the missing param keyword argument to hcpsdk.Connection.PUT()

**0.9.2-3 2015-03-06**

- Fixed a missing import in hcpsdk.__init__.py that led to an unrecoverable error when running on Python 3.4.3

**0.9.2-1 2015-03-01**

- hcpsdk.Connection.request() now logs exception information and stack trace if a catched exception is re-raised as an *hcpsdk.[..]Error*. This will get visible only if the application has initialized the logging subsystem.

**0.9.1-8 2015-02-27**

- Fixed line width in documentation (.rst files) to match limitations for pdf generation

**0.9.1-7 2015-02-27**

- Fixed pip distribution fixed to allow auto-install of dependencies when running 'pip install hcpsdk'

**0.9.1-6 2015-02-18**

- Added automatic retires for hcpsdk.Connection.request() in case of a timeout or connection abort,
- A DummyAuthorization class for use with the Default Namespace,
- An appendiy on the difference when working with the Default Namespace.

CHAPTER 17

# 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*)

**Default Tenant** The legacy Tenant, containing the *Default Namespace*, only. Seldomly used in our days, deprecated. (*default.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[10] 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.

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

# Python Module Index

## h

# Index

## Symbols



## A



## C



## D



## F



## G



## H

## S



## T



## V