hpdr Documentation

Release 1.0.0

Mark Libucha

Apr 15, 2017

Contents

There's a lot a want to say, but it can't be in the toc, can it be?

CHAPTER 1

Why hpdr?

Partition date ranges are hard.

At its heart, hpdr peforms a single, seemingly simple task. It builds a Hive Query Language (HQL) time range condition out of partitions for year, month, day, etc. Something like this:

```
YYYY=2016 AND MM=05
```

That's how you have to specify "all of May 2016" when your partitions are YYYY, MM, and DD, instead of something easier to work with, like YYYYMMDD.

Seems simple enough, but it can quickly grow out of control.

```
(
    (YYYY=2015 AND MM=12 AND DD=28 AND HH>=18)
 OR (YYYY=2015 AND MM=12 AND DD>28)
 OR (YYYY=2016 AND MM=01 AND DD<28)
 OR (YYYY=2016 AND MM=01 AND DD=28 AND HH<18)
)
```

This also represents a one month span (2015 Dec 28, 6PM - 2016 Jan 28, 6PM), but it's more complicated to compose. And easier to get wrong.

In order to trust date ranges, they need to be auto-generated from a human-readable format.

Timezones make them harder.

Assuming the previous date range was for America/Los_Angeles, here it is converted to UTC.

```
(
    (YYYY=2015 AND MM=12 AND DD=29 AND HH>=02)
 OR (YYYY=2015 AND MM=12 AND DD>29)
 OR (YYYY=2016 AND MM=01 AND DD<29)
```

```
 OR (YYYY=2016 AND MM=01 AND DD=29 AND HH<02)
)
```

For Asia/Calcutta, it looks like this.

```
(
    (YYYY=2015 AND MM=12 AND DD=28 AND HH=12 AND MIN>=30)
 OR (YYYY=2015 AND MM=12 AND DD=28 AND HH>12)
 OR (YYYY=2015 AND MM=12 AND DD>28)
 OR (YYYY=2016 AND MM=01 AND DD<28)
 OR (YYYY=2016 AND MM=01 AND DD=28 AND HH<12)
 OR (YYYY=2016 AND MM=01 AND DD=28 AND HH=12 AND MIN<30)
)
```

You want to reuse your work.

When you write a complex date range by hand, it takes a while to get it right, and you're still not sure it is. A few months later, when you need to rerun your query over a new date range, you're going to have to redo all that work.

You want to save time and money.

Opting for an overly inclusive date range because it's easier to write is a waste of computing cycles and your time. As big data gets bigger, processing a few extra hours or days worth gets more and more expensive.

You're nice.

When you process more data than you need to, you're stealing resources from other people running their own jobs in the shared grid.

CHAPTER 2

Getting started

Installation

hpdr has been tested Python 2.7, and Python 3.5 and 3.6. It won't work with 2.6.

Install it with pip:

```
pip install hpdr
```

Simple API Usage

Two dates are required. They can be Python datetime objects, subclasses of datetime, or strings.

```
#!/usr/bin/python2.7

from datetime import datetime
from hpdr import api

begin, end = datetime(2016, 12, 1), '20170203'
rng = api.build(begin, end).partition_range

print rng.build_display(True)  # True gets pretty print
```

Prints:

```
(
    (YYYY=2016 AND MM>=12)
 OR (YYYY=2017 AND MM<02)
 OR (YYYY=2017 AND MM=02 AND DD<03)
)
```

With timezones

The datetime objects passed as the *begin* and *end* arguments must NOT have timezones associated with them, The timezone is assumed to be UTC unless you pass a different timezone with the *dzone* argument. The *dzone* timezone specifies the timezone the data is **stored** in, in Hive.

To specify the timezone the data is used in the **query**, use the *qzone* argument. If you don't specify *qzone*, UTC is used.

```
#!/usr/bin/python2.7

from datetime import datetime
from hpdr import api

begin, end = datetime(2016, 12, 1), '20170203'
rng = api.build(begin, end,
                dzone='America/Los_Angeles',
                qzone='Asia/Calcutta',
               ).partition_range

print rng.build_display(True)  # True gets pretty print
```

Prints:

```
(
    (YYYY=2016 AND MM=12 AND DD=01 AND HH=13 AND MIN>=30)
 OR (YYYY=2016 AND MM=12 AND DD=01 AND HH>13)
 OR (YYYY=2016 AND MM=12 AND DD>01)
 OR (YYYY=2017 AND MM<02)
 OR (YYYY=2017 AND MM=02 AND DD<03)
 OR (YYYY=2017 AND MM=02 AND DD=03 AND HH<13)
 OR (YYYY=2017 AND MM=02 AND DD=03 AND HH=13 AND MIN<30)
)
```

With your partition names

If your partition names are not *YYYY*, *MM*, *DD*, etc., which are the defaults for hpdr, you can pass your own names.

```
#!/usr/bin/python2.7

from hpdr import api

begin, end = '20161201', '20170215'
rng = api.build(begin, end, years='YEAR', months='MONTH', days='DAY').partition_range

print rng.build_display(True)
```

Prints:

```
(
 (    YEAR=2016 AND MONTH>=12)
  OR (YEAR=2017 AND MONTH<02)
  OR (YEAR=2017 AND MONTH=02 AND DAY<15)
 )
)
```

CHAPTER 3

Advanced features

Specifying slop

The data you're after is not always in the "right" partitions. For example, you may want all data for the month of May, but due to clock skew, or network delays, some of the data for May sits in the partition for the last hour of April or the first hour of June. This little bit of extra "slop" on both ends of the main logical time range can make specifying the partition range a lot harder. You can use the *slop* argument to handle it.

```
#!/usr/bin/python2.7

from hpdr import api

begin, end = '20160501', '20160601'
rng = api.build(begin, end, slop='1hours').partition_range

print rng.build_display(True)  # True gets pretty print
```

Prints:

```
(
 YYYY=2016 AND
 (
     (MM=04 AND DD=30 AND HH>=23)
  OR (MM=05)
  OR (MM=06 AND DD=01 AND HH<01)
 )
)
```

This may not seem all that useful until you consider using the hpdr steps feature, described next.

Using steps

Suppose you want to query all the data for a full year, but that's so much data that running a single query would take too long or hog too many resources. If the query can logically be broken down into multiple queries each covering a portion of the year, hpdr can handle the date ranges, including slop.

Here's how we can specify the ranges for 2016 in chunks of 60 days.

```
#!/usr/bin/python2.7

from hpdr import api

begin, end = '2016', '2017'
specs = api.build_with_steps(begin, end, slop='1hours', step='60days')
for spec in specs:
  print spec.partition_range.build_display(True)  # True gets pretty print
```

The query printed the first 60 days looks like this:

```
(
    (YYYY=2015 AND MM=12 AND DD=31 AND HH>=23)
 OR (YYYY=2016 AND MM<03)
 OR (YYYY=2016 AND MM=03 AND DD=01 AND HH<01)
)
```

And for the second, like this:

```
(
 YYYY=2016 AND
 (
     (MM=02 AND DD=29 AND HH>=23)
  OR (MM=03)
  OR (MM=04 AND DD<30)
  OR (MM=04 AND DD=30 AND HH<01)
 )
)
```

And so on.

But even that's not that useful without templating with HPDR_ variables.

CHAPTER 4

Examples

main.py

https://github.com/laboo/hpdr/blob/master/main.py

This simple script exposes all the functionality in hpdr. Here are a few examples:

```
> ./main.py -b 20160312 -e 20160412
  (YYYY=2016 AND ((MM=03 AND DD>=12) OR (MM=04 AND DD<12)))
```

With a timezone:

```
> ./main.py -b 20160312 -e 20160412 -q America/Los_Angeles -p
  (
  YYYY=2016 AND
   (
        (MM=03 AND DD=12 AND HH>=08)
    OR (MM=03 AND DD>12)
    OR (MM=04 AND DD<12)
    OR (MM=04 AND DD=12 AND HH<07)
   )
  )
```

The arguments:

```
>  ./main.py -h
usage: main.py [-h] -b BEGIN -e END [-t STEP] [-s SLOP] [-l LSLOP] [-r RSLOP]
               [-d DZONE] [-q--qzone Q__QZONE] [-p] [-v] [-f FILE]
               [--years YEARS] [--months MONTHS] [--days DAYS] [--hours HOURS]
               [--minutes MINUTES]
```

steps.py

https://github.com/laboo/hpdr/blob/master/steps.py

This script requires input and output file arguments. It substitutes range values the HPDR_ variables in the input query file and writes the result to the output file in steps.

CHAPTER 5

The payoff

Your query

Suppose you have to write a (greatly simplified) query with the following requirements:

- It runs every day
- It gathers data over the past 5 days from a single table
- It must accomodate dates specified the America/Los_Angeles, though the data is stored in UTC in Hive
- It must additionally fetch data from the partitions for the 1 hour just before and after the time range the query covers
- The *ts* column is a string data type representing unix time since the epoch in milliseconds

Middle of the month

Here's what your query might look like for the middle of May 2016:

```
SET BEGIN='2016-05-15'
SET END='2016-05-20'
SELECT * FROM my_table WHERE
ts > CAST(unix_timestamp(${hiveconf:BEGIN}, 'yyyy-MM-dd') as bigint) * 1000 AND
ts < CAST(unix_timestamp(${hiveconf:END}, 'yyyy-MM-dd') as bigint) * 1000 AND
(YYYY=2016 AND MM=05 AND ((DD=14 AND HH>=16) OR (DD>14 AND DD<19) OR (DD=19 AND HH
↪<18)))
```

At the end of the month, when you cross the May/June border, you'd have this:

```
SET BEGIN='2016-05-30'
SET END='2016-06-04'
SELECT * FROM my_table WHERE
ts > CAST(unix_timestamp(${hiveconf:BEGIN}, 'yyyy-MM-dd') as bigint) * 1000 AND
```

```
ts < CAST(unix_timestamp(${hiveconf:END}, 'yyyy-MM-dd') as bigint) * 1000 AND
(YYYY=2016 AND ((MM=05 AND DD=29 AND HH>=16) OR (MM=05 AND DD>29) OR (MM=06 AND DD
↪<03) OR (MM=06 AND DD=03 AND HH<18)))
```

Or, you can create a hpdr template. It's just a query with HPDR_ variables in it:

```
# q.hql
SELECT * FROM my_table WHERE
ts > ${HPDR_slop_begin_unixtime_ms} AND
ts < ${HPDR_slop_end_unixtime_ms} AND
${HPDR_range}
```

Then use it to create your query by using a hpdr date range. For example, with the main.py example:

```
> main.py -b 20160515 -e 20160520 --dzone America/Los_Angeles -s 1hours -f q.hql
-- tmp.hql
SELECT * FROM my_table WHERE
ts > 1463266800000 AND
ts < 1463706000000 AND
(YYYY=2016 AND MM=05 AND ((DD=14 AND HH>=16) OR (DD>14 AND DD<19) OR (DD=19 AND HH
↪<18)))
```

And:

```
> main.py -b 20160530 -e 20160604 --dzone America/Los_Angeles -s 1hours -f q.hql
-- tmp.hql
SELECT * FROM my_table WHERE
ts > 1464562800000 AND
ts < 1465002000000 AND
(YYYY=2016 AND ((MM=05 AND DD=29 AND HH>=16) OR (MM=05 AND DD>29) OR (MM=06 AND DD
↪<03) OR (MM=06 AND DD=03 AND HH<18)))
```

You can get a list of all the HPDR_ variables with the *-v* flag:

```
> main.py -b 20160530 -e 20160604 --dzone America/Los_Angeles -s 1hours -f /tmp/q.hql␣
↪-v
 -- tmp.hql
 SELECT * FROM my_table WHERE
 ts > 1464562800000 AND
 ts < 1465002000000 AND
 (YYYY=2016 AND ((MM=05 AND DD=29 AND HH>=16) OR (MM=05 AND DD>29) OR (MM=06 AND DD
↪<03) OR (MM=06 AND DD=03 AND HH<18)))
 ------------------------------------------------------------------------
 -- Parts of this query were auto-generated with hpdr (pip install hpdr)
 --
 --  /home/mlibucha/Envs/3hpdr/bin/python ../main.py -b 20160530 -e 20160604 --dzone␣
↪America/Los_Angeles -s 1hours -f /tmp/q.hql -v
 --
 --
 -- Input:
 ---------
 -- -- tmp.hql
 -- SELECT * FROM my_table WHERE
 -- ts > ${HPDR_slop_begin_unixtime_ms} AND
 -- ts < ${HPDR_slop_end_unixtime_ms} AND
 -- ${HPDR_range}
 ----------
 -- Output:
```

```
----------
-- -- tmp.hql
-- SELECT * FROM my_table WHERE
-- ts > 1464562800000 AND
-- ts < 1465002000000 AND
-- (YYYY=2016 AND ((MM=05 AND DD=29 AND HH>=16) OR (MM=05 AND DD>29) OR (MM=06 AND DD
↪<03) OR (MM=06 AND DD=03 AND HH<18)))
----------
--
-- This is a complete list of the available template variables and their values:
--
-- variable                      value
-- ----------------------------  -------------------
-- HPDR_dzone                    UTC
-- HPDR_qzone                    America/Los_Angeles
-- HPDR_begin_ts                 2016-05-29 17:00:00
-- HPDR_end_ts                   2016-06-03 17:00:00
-- HPDR_slop_begin_ts            2016-05-29 16:00:00
-- HPDR_slop_end_ts              2016-06-03 18:00:00
-- HPDR_begin_unixtime           1464566400
-- HPDR_begin_unixtime_ms        1464566400000
-- HPDR_begin_yyyymmdd           20160529
-- HPDR_begin_yyyy               2016
-- HPDR_begin_mm                 05
-- HPDR_begin_dd                 29
-- HPDR_begin_hh                 17
-- HPDR_begin_min                00
-- HPDR_begin_sec                00
-- HPDR_end_unixtime             1464998400
-- HPDR_end_unixtime_ms          1464998400000
-- HPDR_end_yyyymmdd             20160603
-- HPDR_end_yyyy                 2016
-- HPDR_end_mm                   06
-- HPDR_end_dd                   03
-- HPDR_end_hh                   17
-- HPDR_end_min                  00
-- HPDR_end_sec                  00
-- HPDR_slop_begin_unixtime      1464562800
-- HPDR_slop_begin_unixtime_ms   1464562800000
-- HPDR_slop_begin_yyyymmdd      20160529
-- HPDR_slop_begin_yyyy          2016
-- HPDR_slop_begin_mm            05
-- HPDR_slop_begin_dd            29
-- HPDR_slop_begin_hh            16
-- HPDR_slop_begin_min           00
-- HPDR_slop_begin_sec           00
-- HPDR_slop_end_unixtime        1465002000
-- HPDR_slop_end_unixtime_ms     1465002000000
-- HPDR_slop_end_yyyymmdd        20160603
-- HPDR_slop_end_yyyy            2016
-- HPDR_slop_end_mm              06
-- HPDR_slop_end_dd              03
-- HPDR_slop_end_hh              18
-- HPDR_slop_end_min             00
-- HPDR_slop_end_sec             00
--
-- Note that all values have been shifted to the query time zone (HPDR_qzone)
```

CHAPTER 6

Can you trust it?

How do you know it's right?

hpdr prints out some HQL when you call it with a couple of datetime objects, but how can you be sure what it prints out is accurate?

You could pretty print it and try to reason it out, but the whole purpose of hpdr is to eliminate that kind of tedious, error-prone approach.

Reversing the algorithm

hpdr is tested by comparing the number of seconds between the *begin* and *end* datetime objects with the number of seconds represented by each clause in the HQL output added together. Let's look at a simple example.

```
#!/usr/bin/python2.7

from datetime import datetime

begin = datetime(2016, 02, 02, 18)
end = datetime(2016, 05, 11, 3, 56)
print((end - begin).total_seconds())
```

This prints *8502960.0* (seconds).

When we have hpdr print out the range, we get:

```
(
 YYYY=2016 AND
 (
     (MM=02 AND DD=02 AND HH>=18)
  OR (MM=02 AND DD>02)
  OR (MM>02 AND MM<05)
  OR (MM=05 AND DD<11)
```

```
  OR (MM=05 AND DD=11 AND HH<03)
  OR (MM=05 AND DD=11 AND HH=03 AND MIN<56)
 )
)
```

We can calcuate how many seconds each clause in HQL query represents by starting at the earliest possible datetime for the begin and end times, and then triangulating the durations each HQL condition represents.

condition group	seconds	from (inclusive)	to (exclusive)
MM=02 DD=02 HH>=18	21600	2016-02-02 18:00	2016-02-03 00:00
MM=02 DD>02	2332800	2016-02-03 00:00	2016-03-01 00:00
MM>02 MM<05	5270400	2016-03-01 00:00	2016-05-01 00:00
MM=05 DD<11	864000	2016-05-01 00:00	2016-05-11 00:00
MM=05 DD=11 HH<03	10800	2016-05-11 00:00	2016-05-11 03:00
MM=05 DD=11 HH=03 MIN<56	3360	2016-05-11 03:00	2016-05-11 03:56
total	8502960		

If we further prohibit any empty condition groups – those which evaluate to 0 seconds – we can be fairly certain the results are correct.

CHAPTER 7

The algorithm

Details

Every date range breaks down in the same way. Here's an example range

```
----: YYYY-MM-DD HH:MM
From: 2017-02-15 12:30
To  : 2017-02-25 04:00
```

Parsing from left to right, the first part of the range is what's common between the two datetimes: *2017-02*. In the query these common parts are ANDed together using equals signs.

```
YYYY=2017 AND MM=02
```

The first unit to differ between the two datetimes is DD, with values *15* and *25*. This is referred to in the code as the "bridge". The bridge shows up in the query like this.

```
(DD>15 AND DD<25)
```

The other two parts of the query are the entrance to, and exit from, the bridge. Our bridge excludes the two days on the ends, the 15th and the 25th. So the entrance and exit parts must handle them. Here's the entrance.

```
(DD=15 AND HH=12 AND MIN>=30) OR (DD=15 AND HH>12)
```

And the exit.

```
(DD=25 AND HH<04)
```

All together, it looks like this.

```
(
YYYY=2017 AND MM=02 AND              -- [shared]
 (
     (DD=15 AND HH=12 AND MIN>=30)   -- [entrance]
```

```
  OR (DD=15 AND HH>12)              -- [entrance]
  OR (DD>15 AND DD<25)              -- [bridge]
  OR (DD=25 AND HH<04)              -- [exit]
 )
)
```

Requirements

Each date ranges hpdr outputs must be

- **Correct.** It represent the date range precisely in compilable HQL.
- **Readable.** It must display the range in human-readable formats.
- **Minimal.** It must be written in the fewest number of characters possible.

Correctness is a necessary condition for hpdr to be worth anything at all, but others are not.

Pretty printing helps hpdr users check the output visually, so they can verify its output.

The *minimal* requirement deserves a section of its own.

Minimal

Common sense minimal

There's an infinite number of bad ways to create any given date range. For example, the first 10 days of May 2015 could be written

```
YYYY=2015 AND MM=5 AND (DD=1 OR DD=2 OR DD=3 OR DD=4 OR DD=5 OR DD=6 OR DD=7 OR DD=8␣
↪OR DD=9 OR DD=10)
```

But this is better

```
YYYY=2015 AND MM=5 AND DD<11
```

because it's minimal.

Exceptional minimal

This should be avoided.

```
MM>=6 AND MM<7
```

because this is clearly better,

```
MM=6
```

even though the former mirrors the base case.

```
MM>=6 AND MM<10
```

Non-overlapping minimal

A hpdr date range can be correct, but can contain overlapping conditions. A stupid example is

```
YYYY=2016 OR (YYYY>=2010 AND YYYY <2017)  -- 2016 included twice
```

This is non-minimal and not allowed in hpdr. A suprising number of these were ferreted out by unit tests.

Questionable

I wrote hpdr to scratch an itch at work. I was composing, and was watching other people composing, these massively complex Hive date ranges. Strings turned into milliseconds truncated to seconds turned into Unix timestamps wrapped in timezone-shifting functions. They were unreadable and unmaintainable. I thought I would whip up a nice Python module that would fix it all.

But it turned out to be much harder than I thought. The code I've written to build a date range is pretty dense. The line of attack I settled on is indirect. But it was the best I could come up with.

Is there a simpler, recursive algorithm? I didn't see it.

CHAPTER 8

API Documentation

api.py

hpdr.api.**build**(*begin*, *end*, *dzone=u'UTC'*, *qzone=u'UTC'*, *slop=None*, *lslop=None*, *rslop=None*, *years=u'YYYY'*, *months=u'MM'*, *days=u'DD'*, *hours=u'HH'*, *minutes=u'MIN'*)

Build a specification for a date range.

Parameters

- **begin** (*str/datetime*) – begin date of range, a datetime or yyyy[mm[dd[mm[ss]]]] string
- **end** (*str/datetime*) – end date of range, a datetime or yyyy[mm[dd[mm[ss]]]] string
- **dzone** (*str*) – tzdata timezone data is stored in
- **qzone** (*str*) – tzdata timezone query dates and times are specified in
- **slop** (*str*) – duration to add to both ends of the partition range, specified as d+[years|months|days|hours|minutes], for example, 5hours
- **lslop** (*str*) – duration to add to the front end of the partition range, specified as d+[years|months|days|hours|minutes], for example, 5hours
- **rslop** (*str*) – duration to add to the back end of the partition range, specified as d+[years|months|days|hours|minutes], for example, 5hours
- **years** (*str*) – name for years partition
- **months** (*str*) – name for months partition
- **days** (*str*) – name for days partition
- **hours** (*str*) – name for hours partition
- **minutes** (*str*) – name for hours partition

Returns Object representing the date range

Return type *hpdr.models.Spec*

hpdr.api.**build_with_steps**(*begin*, *end*, *step=None*, *dzone=u'UTC'*, *qzone=u'UTC'*, *slop=None*, *lslop=None*, *rslop=None*, *years=u'YYYY'*, *months=u'MM'*, *days=u'DD'*, *hours=u'HH'*, *minutes=u'MIN'*)

Build a lists of specification for a date.

The specifications in the list are contiguous, chronological pieces of the list. Left slop followed by the begin-to-end range broken into parts of step size followed by right slop.

Parameters

- **begin** (*str/datetime*) – begin date of range, a datetime or yyyy[mm[dd[mm[ss]]]] string
- **end** (*str/datetime*) – end date of range, a datetime or yyyy[mm[dd[mm[ss]]]] string
- **step** (*str*) – duration to break individual Spec objects into, specified as d+[years|months|days|hours|minutes], for example, 5hours. If None, one Spec is returned.
- **dzone** (*str*) – tzdata timezone data is stored in
- **qzone** (*str*) – tzdata timezone query dates and times are specified in
- **slop** (*str*) – duration to add to both ends of the partition range, specified as d+[years|months|days|hours|minutes], for example, 5hours
- **lslop** (*str*) – duration to add to the front end of the partition range, specified as d+[years|months|days|hours|minutes], for example, 5hours
- **rslop** (*str*) – duration to add to the back end of the partition range, specified as d+[years|months|days|hours|minutes], for example, 5hours
- **years** (*str*) – name for years partition
- **months** (*str*) – name for months partition
- **days** (*str*) – name for days partition
- **hours** (*str*) – name for hours partition
- **minutes** (*str*) – name for hours partition

Returns

List representing the date range. For example,

build_with_steps(begin='20160901', end=20161001, step=10days, –slop=1hours)

returns a list of five Spec objects, representing these ranges:

(YYYY=2016 AND MM=08 AND DD=31 AND HH>=23) [left slop of 1 hour]

(YYYY=2016 AND MM=09 AND DD>=01 AND DD<11) [10 days]

(YYYY=2016 AND MM=09 AND DD>=11 AND DD<21) [10 days]

(YYYY=2016 AND MM=09 AND DD>=21) [10 days]

(YYYY=2016 AND MM=10 AND DD=01 AND HH=00) [right slop of 1 hour]

Return type A list of hpdr.model.Spec

models.py

class hpdr.models.**Range**(*ands*, *ors*)

A date range, abstractly represented by SQL conditions.

build_display(*pretty=False*)
Build a string for displaying the Range.

Create a string version of the Range in valid SQL syntax for a conditional clause.

class hpdr.models.**Spec**(*begin*, *end*, *dzone=u'UTC'*, *qzone=u'UTC'*, *slop=None*, *lslop=None*, *rslop=None*, *years=u'YYYY'*, *months=u'MM'*, *days=u'DD'*, *hours=u'HH'*, *minutes=u'MIN'*)
Object for representing a partition date range.

substitute(*query*, *verbose=False*, *pretty=False*)
Fills in the HPDR_ varibles with the values.

Parameters

- **query** (*string*) – a string (optionally) containing HPDR_ variables
- **verbose** (*bool*) – if True prints out lots of extra info as an SQL comment
- **pretty** (*bool*) – if True returns just HPDR_range_pretty variable

Returns query with HDPR_ variables substituted for, or HPDR_range_pretty value if pretty=True

Return type str

variables()
Return a map of all HPDR_ variables and their values defined for the range.

CHAPTER 9

Indices

- genindex
- modindex

Python Module Index

h

Index