This vignette gives users the summary information of API functions provided by UCSCXenaTools for UCSC Xena.

Before using API, user should know some concepts about Xena elements. Following description is copied from xenaPython __init__.py.

Data rows are associated with "sample" IDs.
Sample IDs are unique within a "cohort". A "dataset" is a particular assay of a cohort, e.g. gene expression.
Datasets have associated metadata, specifying their data type and cohort.
There are three primary data types: dense matrix (samples by probes), sparse (sample, position, variant), and segmented (sample, position, value).
Dense matrices can be genotypic or phenotypic. Phenotypic matrices have associated field metadata (descriptive names, codes, etc.). Genotypic matrices may have an associated probeMap, which maps probes to genomic locations. If a matrix has hugo probeMap, the probes themselves are gene names. Otherwise, a probeMap is used to map a gene location to a set of probes.

New features

A new series of functions (mostly) starting with fetch_ have been introduced to help fetch a small amount of data by Xena APIs.

Now three are available:

• fetch_dense_values(): fetches values from a dense matrix.
• fetch_dataset_samples(): fetches samples from a dataset
• fetch_dataset_identifiers(): fetches identifiers from a dataset.

They have similar arguments and all the details can be viewed by running ?fetch in R console after library(UCSCXenaTools).

API categories

API functions can be divided into two classes: lower API functions and higher API functions. They have following difference:

• The main difference between them is that the target of higher API functions is XenaHub object, which is a S4 class built in R. While the targets of lower API functions are Xena hub urls, cohort names or dataset names with character format. The XenaHub object can provide more uniform operation methods and can be used to download corresponding datasets quickly and easily (detail see another vignette).
• Lower API functions are not registered in package NAMESPACE, so user may not access them after library(UCSCXenaTools), user need to use UCSCXenaTools:::fun_name instead.
• Lower API functions have no help pages, so user cannot find any description about them in R, which means you cannot use ?fun_name to get help. However, API report part in this vignette shows all available API functions and their short description.
• Higher API functions are built on lower API functions, they return more meaningful and easy results for operation. Most of lower API functions return nested lists as results, user need to tidy them before using them in next step.

Lower API functions

Lower API functions also have 2 classes:

• one is generated from .xq files, function names all start with .p_. All .xq files are copied from xenaPython package, which is official Python API for Xena. These functions are dynamically created when UCSCXenaTools loaded. Their names are given as following:

```r
#> [1] "p_all_cohorts"
#> [2] "p_all_datasets"
#> [3] "p_all_datasets_n"
#> [4] "p_all_field_metadata"
#> [5] "p_cohort_samples"
#> [6] "p_cohort_summary"
#> [7] "p_dataset_fetch"
#> [8] "p_dataset_field"
#> [9] "p_dataset_field_examples"
#> [10] "p_dataset_field_n"
#> [12] "p_dataset_gene_probes_values"
#> [13] "p_dataset_list"
#> [14] "p_dataset_metadata"
#> [15] "p_dataset_probe_signature"
#> [16] "p_dataset_probe_values"
#> [17] "p_dataset_samples"
#> [18] "p_dataset_samples_ndense_matrix"
#> [19] "p_datasets_null_rows"
#> [20] "p_feature_list"
#> [21] "p_field_codes"
#> [22] "p_field_metadata"
#> [23] "p_gene_transcripts"
#> [24] "p_match_fields"
```
• the other one is created in package. The function names all start with ., are
given as following:

```r
> [1] ".host_cohorts"
> [2] ".cohort_datasets"
> [3] ".cohort_datasets_count"
> [4] ".cohort_samples_each"
> [5] ".cohort_samples_any"
> [6] ".cohort_samples_all"
> [7] ".dataset_samples_each"
> [8] ".dataset_samples_any"
> [9] ".dataset_samples_all"
```

I don't know how to write these query sentence for Xena Hubs. So here I
want to say thanks to authors of xenaPython and xenaR packages.
Of note, I don’t know test all functions generated from .xq files, most of them works. Sometimes functions return you errors or list() may caused by invalid format or bad network, you should try more times. If you make sure there are problems/errors in query procedure, you can check corresponding query variables:

```r
##> [1] ".xq_all_cohorts"
##> [2] ".xq_all_datasets"
##> [3] ".xq_all_datasets_n"
##> [4] ".xq_all_field_metadata"
##> [5] ".xq_cohort_samples"
##> [6] ".xq_cohort_summary"
##> [7] ".xq_dataset_fetch"
##> [8] ".xq_dataset_field"
##> [9] ".xq_dataset_field_examples"
##> [10] ".xq_dataset_field_n"
##> [12] ".xq_dataset_gene_probes_values"
##> [13] ".xq_dataset_list"
##> [14] ".xq_dataset_metadata"
##> [15] ".xq_dataset_probe_signature"
##> [16] ".xq_dataset_probe_values"
##> [17] ".xq_dataset_samples"
##> [18] ".xq_dataset_samples_ndense_matrix"
##> [19] ".xq_datasets_null_rows"
##> [20] ".xq_feature_list"
##> [21] ".xq_field_codes"
##> [22] ".xq_field_metadata"
##> [23] ".xq_gene_transcripts"
##> [24] ".xq_match_fields"
##> [25] ".xq_probe_count"
##> [26] ".xq_probemap_list"
##> [27] ".xq_ref_gene_exons"
##> [28] ".xq_ref_gene_position"
##> [29] ".xq_ref_gene_range"
##> [30] ".xq_segment_data_examples"
##> [31] ".xq_segmented_data_range"
##> [32] ".xq_sparse_data"
##> [33] ".xq_sparse_data_examples"
##> [34] ".xq_sparse_data_match_field"
##> [35] ".xq_sparse_data_match_field_slow"
##> [36] ".xq_sparse_data_match_partial_field"
##> [37] ".xq_sparse_data_range"
##> [38] ".xq_transcript_expression"
```
For example, you'd like to check `.p_all_cohorts` function, you can take a look at `.xq_all_cohorts` object.

```r
.xq_all_cohorts
#> 
#> [[1]] ";allCohorts\n(fn [exclude]\n  (map :cohort\n   (query\n    {:select [[#sql/call [:distinct #sql/call [:ifnull :cohort "(unassigned)"] ] :cohort]\n    :from [:dataset]\n    :where [:not [:in :type exclude]]])))
```

cat it may give you more easy-to-read format.

```r
cat(.xq_all_cohorts)
#> ;allCohorts
#> (fn [exclude]
#>  (map :cohort
#>   (query
#>    {:select [[#sql/call [:distinct #sql/call [:ifnull :cohort ",\"(unassigned)\"] ] :cohort]\n#>    :from [:dataset]
#>    :where [:not [:in :type exclude]]))))
```

### Use cases

Several use cases are modified from README of `xenaPython` package.

Load package firstly.

```r
library(UCSCXenaTools)
```

You can find out host id and dataset id from `https://xenabrowser.net/datapages/`, a more recommended way is use `XenaData` in `UCSCXenaTools`.

```r
head(XenaData) [, 1:5]
#> # A tibble: 6 x 5
#> #> XenaHosts XenaHostNames XenaCohorts
#> #> <chr> <chr> <chr>
#> 1 https://~ publicHub Acute lymph
#> 2 https://~ publicHub Acute lymph
#> 3 https://~ publicHub Acute lymph
#> 4 https://~ publicHub Breast Can
#> 5 https://~ publicHub Breast Can
#> 6 https://~ publicHub Breast Can
#> # ... with 2 more variables:
#> #> XenaDatasets <chr>, SampleCount <chr>
```

The host id is stored at `XenaHosts` column, and dataset id is stored at `XenaDatasets` column.

Of note, when you want to query single sample or gene with function starts with `.p_`, you must transform id of sample or gene into a list
Query four samples and three identifiers expression

```r
hub = "https://toil.xenahubs.net"
dataset = "tcga_RSEM_gene_tpm"
samples = c("TCGA-02-0047-01", "TCGA-02-0055-01",
            "TCGA-02-2483-01", "TCGA-02-2485-01")
probes = c("ENSG00000282740.1", "ENSG00000000005.5",
           "ENSG00000000419.12")
.p_dataset_probe_values(hub, dataset, samples,
                        probes)
```  

```r
#> [[1]]
#>    strand chromend  chromstart chrom
#> 1    -       16750589  16739938 chr1
#> 2    -       50935855  50934867 chr20
#> 3    +     100599895   100584802 chrX

#> [[2]]
#> [1,]  4.675  6.025  5.826  5.177
#> [2,] -3.171  4.165  5.574  6.171
#> [4,]  6.966  7.966  8.966  9.966
```

Query one probe. As mentioned above, one must transform id of probe or sample into a list when he wants to query only one sample/probe.

**Bad query:**

```r
.p_dataset_probe_values(hub, dataset, samples,
                        "ENSG000000282740.1")
```  

```
#> [[1]]
#> list()

#> [[2]]
#> [1,] NaN NaN NaN NaN
#> [2,] NaN NaN NaN NaN
#> [3,] NaN NaN NaN NaN
#> [4,] NaN NaN NaN NaN
#> [5,] NaN NaN NaN NaN
#> [6,] NaN NaN NaN NaN
#> [7,] NaN NaN NaN NaN
#> [8,] NaN NaN NaN NaN
#> [9,] NaN NaN NaN NaN
#> [10,] NaN NaN NaN NaN
#> [11,] NaN NaN NaN NaN
```
Good query:

```r
.p_dataset_probe_values(hub, dataset, samples,
    as.list("ENSG00000282740.1"))
```

Query four samples and three genes expression, when the dataset you want to query has a identifier-to-gene mapping

```r
genes = c("TP53", "RB1", "PIK3CA")
.p_dataset_gene_probe_avg(hub, dataset, samples, genes)
```

If the dataset does not have id-to-gene mapping, but the dataset used gene names as its identifier

In this situation, you can query gene expression like two ways above will not work.
hub = "https://toil.xenahubs.net"
dataset = "tcga_RSEM_Hugo_norm_count"
samples = c("TCGA-02-0047-01", "TCGA-02-0055-01",
            "TCGA-02-2483-01", "TCGA-02-2485-01")
probes = c("TP53", "RB1", "PIK3CA")

.p_dataset_probe_values(hub, dataset, samples, probes)

#> [[1]]
#>    strand chromend chromstart chrom
#> 1     +    48481986      48303751 chr13
#> 2     -     7687550      7661779 chr17
#> 3     +    179240093    179148114 chr3

#> [[2]]
#> [1,] 11.63 10.68 12.65 12.15
#> [2,] 12.04 10.93 11.59 11.41
#> [3,] 10.67 10.90 10.71 10.12

Find out the samples in a dataset

hub = "https://tcga.xenahubs.net"
dataset = "TCGA.BLCA.sampleMap/HiSeqV2"
.p_dataset_samples(hub, dataset, 10)

#> [1] "TCGA-BT-A20R-11" "TCGA-DK-AA6S-01"
#> [5] "TCGA-XF-A9T4-01" "TCGA-FD-A5C1-01"
#> [7] "TCGA-GU-A42Q-01" "TCGA-DK-A3IL-01"
#> [9] "TCGA-XF-A4NH-01" "TCGA-FT-A61P-01"

# obtain all samples
.p_dataset_samples(hub, dataset, NULL) %>% head()

#> [1] "TCGA-BT-A20R-11" "TCGA-DK-AA6S-01"
#> [5] "TCGA-XF-A9T4-01" "TCGA-FD-A5C1-01"

Higher API function samples() has more features. It can be used to do set
operation for samples in a host.

xe = XenaHub(cohorts = "Cancer Cell Line Encyclopedia (CCLE)")
# samples in each dataset, first host
x = samples(xe, by = "datasets", how = "each")[[1]]
lengths(x)  # data sets in ccle cohort on first (only) host
Find out the identifiers in a dataset

```r
hub = "https://tcga.xenahubs.net"
dataset = "TCGA.BLCA.sampleMap/HiSeqV2"
.p_dataset_field(hub, dataset) %>% head()
#> [1] "?|100130426" "?|100133144" "?|100134869"
#> [4] "?|10357" "?|10431" "?|136542"
```

Find out the number of identifiers in a dataset

```r
hub = "https://tcga.xenahubs.net"
dataset = "TCGA.BLCA.sampleMap/HiSeqV2"
.p_dataset_field_n(hub, dataset)
#> [1] 20531
```

**LICENSE**

GPL-3

Please note, code from XenaR package under Apache 2.0 license.