Kurl Quickstart

This section demonstrates how to load Kurl and perform basic sync and async requests against public and authenticated endpoints.

Setup

Ensure your cloud credentials are set as environment variables, for example:

```
export AWS_ACCESS_KEY_ID="… "
export AWS_SECRET_ACCESS_KEY="…"
export AWS_SESSION_TOKEN="… "
```

Basic usage

1. To make synchronous calls, use these details in your KDB-X session:

```
q).kurl:use`kx.kurl  // load in the module
q) info:`AccessKeyId`SecretAccessKey`Token!
(getenv`AWS_ACCESS_KEY_ID;getenv`AWS_SECRET_ACCESS_KEY;getenv`AWS_SESSION_TOKEN)
q).kurl.sync ("https://kx-mybucket.s3.us-east-
2.amazonaws.com/test_data/data.csv";`GET;::)  // sync calls fails as you have not
registered
403i
"<?xml version=\"1.0\" encoding=\"UTF-8\"?>\n<Error><Code>AccessDenied</Code>
<Message>Access Denied..
q).kurl.register (`aws_cred;"*amazonaws.com";"";info)  // register
q).kurl.sync ("https://kx-mybucket.s3.us-east-
2.amazonaws.com/test_data/data.csv";`GET;::)  // successful sync call
200i
"sym,price,size\nFDP,1.2,100\n"
```

2. To make asynchronous calls:

```
q) .kurl.async ("https://kx-mybucket.s3.us-east-
2.amazonaws.com/test_data/data.csv";`GET;``callback!(`;{.debug.x:x}))
.debug.x
200i
"sym,price,size\nFDP,1.2,100\n"
```

3. For public datasets (no authentication required):


```
.kurl:use`kx.kurl
.kurl.sync ("https://api.census.gov/data/2020/dec/pl?
get=NAME,P1_001N&for=state:*"; `GET; ::)
200i
"
[[\"NAME\",\"P1_001N\",\"state\"],\n[\"Pennsylvania\",\"13002700\",\"42\"],\n[\"Ca
lifornia\",\"395..
```

Kurl Examples

This section provides practical use cases across major cloud providers and APIs for the Kurl module in KDB-X.

Example 1 - Request data from Google BigQuery

Pre-requisite

Obtain credentials for your GCP account.

```
gcloud init
export GCP_TOKEN=$(gcloud auth print-access-token)
```

Request dataset

Use these details in your kdb-x session:

```
.kurl:use`kx.kurl
.kurl.register(`oauth2; "*.googleapis.com"; "";enlist[`access_token]!enlist
getenv`GCP_TOKEN)
.kurl.sync("https://bigquery.googleapis.com/bigquery/v2/projects/bigquery-public-
data/datasets";`GET; ::)
200i
"{\n  \"kind\": \"bigquery#datasetList\",\n  \"etag\":
\"IyFqxYKvltLbvNdsx9qlIQ==\",\n  \"nextPageT..
```

Example 2 - Push data to Azure

Pre-requisite

This example will not work against KX public buckets. Our public buckets are read only.

For a private bucket, use a Storage Account with the IAM role Storage Blob Data Owner

Upload non KDB-X data to Azure Storage

This example shows how to create non KDB-X files in cloud storage.

In the code snippet below replace `<account>` with the lowercase name of your storage account.

```
.kurl:use`kx.kurl
// Create a container
headers:enlist ["x-ms-version"]!enlist "2017-11-09";
opts:enlist[`headers]!enlist headers;
resp:.kurl.sync ("https://<account>.blob.core.windows.net/mycontainer?
restype=container"; `PUT; opts);

// Now, add a text file to the bucket
if[201 <> first resp; 'last resp];
opts[`headers]:("x-ms-version";"x-ms-blob-type";"Content-Type")!("2017-11-
09";"BlockBlob";"text/plain");
opts[`body]:"hello world";
resp:.kurl.sync ("https://<account>.blob.core.windows.net/mycontainer/hello.txt";
`PUT; opts);
if[201 <> first resp; 'last resp];
```

Uploading large files

For larger files, you may not be able to upload in one single request.

You may upload a file in sequential blocks by using the (`Put Block` and `Append Block` APIs) [https://docs.microsoft.com/en-us/rest/api/storageservices/understanding-block-blobs--append-blobs--and-page-blobs]

You may also upload a file in parallel using the same `Put Block` API with a BlockBlob.

Note: "azcopy for databases" Use `azcopy` to upload KDB-X databases. These examples demonstrate how to upload singular files easily.

=== "Append Block (single threaded)"

```
```q
filePath:`:/tmp/example.file

// Set fileSize and block sizes, produce ranges
fileSize:hcount filepath
blockSize:"j"$4e6 // 4Mb (Azure suppots 4Mib, 4Mb close enough)
ranges:"j"$fileSize&reverse each 1_,':[blockSize*til 1+ceiling fileSize%blockSize]

// First, create the empty blob
headers:("x-ms-version";"Content-Type";"x-ms-blob-type")!("2019-02-
02";"text/plain";"AppendBlob");
opts:`body`headers!("";headers);
resp:.kurl.sync
("https://<account>.blob.core.windows.net/mycontainer/myblob";`PUT; opts);
if[201i <> first resp; 'last resp];
```


```

```
// Read into a file at an offset, given a length, and upload that block.
uploadBlock:{[filePath;range]
    opts:`body`headers!(read1(filePath;range 0; range[1] - range 0);headers);
    .kurl.sync ("https://<account>.blob.core.windows.net/mycontainer/myblob?
comp=appendblock";`PUT; opts);
    if[201i <> first resp;'last resp];
    }

// Upload each block one after another
uploadBlock[filePath;] each ranges;
```

=== "Put Block (multi threaded)"

```
```q
filePath:`:/tmp/example.file

// Set fileSize and block sizes, produce ranges
fileSize:hcount filePath
blockSize:"j"$4e6 // 4Mb (Azure suppots 4Mib, 4Mb close enough)
ranges:"j"$fileSize&reverse each 1_,':[blockSize*til 1+ceiling fileSize%blockSize]

// Create a block blob
headers:("x-ms-version";"Content-Type";"x-ms-blob-type")!("2019-02-
02";"text/plain";"BlockBlob");
opts:`body`headers!("";headers);
resp:.kurl.sync
("https://<account>.blob.core.windows.net/mycontainer/myblob";`PUT; opts);
if[201i <> first resp; 'last resp];

// Generate block IDs, these need to be equal-length distinct strings
//  these strings must also be valid base64 + uri encoded, but are otherwise
arbitrary
blockIDs:{raze string x} each 0x0 vs/: til count ranges;

// Upload all the blocks in parallel
uploadBlock:{[filePath; range; blockID]
    opts:`body`headers!(read1 (filePath;range 0; range[1] - range 0);headers);
    resp:.kurl.sync ("https://<account>.blob.core.windows.net/mycontainer/myblob?
comp=block&blockid=",blockID;`PUT; opts);
    if[201i <> first resp;'last resp];
    }
.[uploadBlock[filePath;;];] peach flip (ranges;blockIDs);

// Now create the blob by putting all then block IDs together in order
blockListHead:("<?xml version=\"1.0\" encoding=\"utf-8\"?>";"<BlockList>")
blockListMid:"  <Latest>",/:blockIDs,\:"</Latest>";
blockListTail:enlist "</BlockList>";
headers:("x-ms-version";"Content-Type")#headers;
```


```

```
opts:`body`headers!("\n" sv blockListHead,blockListMid,blockListTail;headers);
resp:.kurl.sync ("https://<account>.blob.core.windows.net/mycontainer/myblob?
comp=blocklist";`PUT; opts);
if[201i <> first resp;'last resp];
```

Example 3 - Upload a folder to Amazon S3

This example shows how to create non KDB-X files in object storage. KDB-X objects can be accessed using the `objstor` module instead of kurl.

Pre-requisite

Replace `<bucket>` and `<region>` with the lowercase names of your S3 bucket and AWS Region.

Example

Create a folder of example empty files.

```
mkdir -p upload/
touch upload/hello1.txt
touch upload/hello2.txt
touch upload/hello3.txt
touch upload/hello3.txt
```

```
bucket:"https://<bucket>.s3.<region>.amazonaws.com/";

// Recursively list directories, returning only files
getFiles:{$[11h=type d:key x;raze .z.s each` sv/:x,/:d;d]};
// Stringify and drop colons
files:reverse getFiles `:upload;

// Synchronously upload files one after another.
// @param bucket {string} HTTPS URL for bucket, must end in trailing slash
// @param hfile {symbol} Local file path
uploadFile:{[bucket;hfile]
    filename:1 _ string hfile;
    resp:.kurl.sync (bucket,filename; `PUT; ``file!(::;hfile));
    // if not HTTP 200 OK, or 201 Created, its failed
    if[not first[resp] in 200 201; 'last resp];
    }[bucket;];

uploadFile each files;
```

Example 4 - Get and put files into S3-Compatible storage

This example shows how to create and retrieve non KDB-X files from object storage. KDB-X objects should be accessed using the `objstor` module instead of kurl.

With s3-compatible storage set up, such as MinIO, kurl may be used to read files from storage.

Setting up MinIO would have you export an endpoint, similar to:

```
export KX_S3_ENDPOINT=http://127.0.0.1:9000"
```

See below an example statement showing how one would read a CSV from a bucket hosted on s3:

```
r: .kurl.sync ("http://127.0.0.1:9000/myBucket/hello.csv"; `GET; `service`region!
("s3";"us-east-1"));
if[first resp <> 200; 'last resp];

// Print the CSV file
\c 200 200
show last r;
```

To upload a CSV file from in-memory variable:

```
// Upload in-memory text
myCSV:"\n" sv csv 0: ([] x:til 50; y: 50?`8);
r: .kurl.sync ("http://127.0.0.1:9000/myBucket/world.csv"; `PUT;
`body`service`region!(myCSV; "s3";"us-east-1"));
if[not first[resp] in 200 201; 'last resp];
```

To upload a CSV file from local disk:

```
r: .kurl.sync ("http://127.0.0.1:9000/myBucket/world.csv"; `PUT;
`file`service`region!(`:path/to/myCSV.csv; "s3";"us-east-1"));
if[not first[resp] in 200 201; 'last resp];
```

Example 5 - Create a namespace in AWS Cloud Map

Use AWS Cloud Map to create an HTTP namespace and wait for it to come online.

```
// Create a new namespace, this returns an operation ID
body:.j.j enlist[`Name]!enlist "kxceNamespace"

headers:("Content-Type";"x-amz-target")!
  ("application/x-amz-json-1.1";"Route53AutoNaming_v20170314.CreateHttpNamespace")

URL:"https://servicediscovery.us-east-2.amazonaws.com"
```



```
resp:.kurl.sync (URL;`POST;`headers`body!(headers;body))
if[first resp <> 200; 'last resp]

checkOperation:{[resp]
  if[200 <> first resp; ' last resp];
  j:.j.k last resp;
  if["SUCCESS" ~ j . `Operation`Status; show "Namespace created"; .test.done:1b];
}

getOperation:{[url;id]
  show "Calling getOperation to see if namespace has been created...";
  headers:("Content-Type";"x-amz-target")!
    ("application/x-amz-json-1.1";"Route53AutoNaming_v20170314.GetOperation");
  opts:`headers`body`callback!(headers;id; checkOperation);
  .kurl.async (url;`POST;opts)
  }[URL;]

// The response of creating a namespace returns an input to GetOperation
.test.id:last resp
.test.done:0b
.z.ts:{ if[.test.done; system "t 0"; :(::)]; getOperation .test.id; }
\t 500
```

Start a kdb-x session and load `service.q`: This will check, ever 500ms, to see if the namespace has been created.

```
q)\l service.q
```

KURL reference

This section provides detailed technical reference for the Kurl module APIs.

Registration

By registering with Kurl, requests made against a given domain wildcard will use cached authentication information returned from the registration call.

There are a number of different ways that the module allows you to register. The `.kurl.init` function listed below will cycle through these options for you.

If you want to explicitly call the underlying APIs, these are also documented below.

Registration credentials

Note: "Pre-requisite" When registering with cloud vendors you must first obtain an access key, secret and token for your cloud provider. These can be defined in a config file or as environment variables. See examples for each below:

=== "Amazon Web Services"

```bash
export AWS_ACCESS_KEY_ID=".."
export AWS_SECRET_ACCESS_KEY=".."
export AWS_SESSION_TOKEN=".."
```

These values could also be stored in a credentials file, typically this is ~/.aws/credentials.

```bash
cat ~/.aws/credentials
[default]
aws_access_key_id=".."
aws_secret_access_key=".."
aws_session_token=".."
```

=== "Microsoft Azure"

```bash
export AZURE_STORAGE_ACCOUNT="..."
export AZURE_STORAGE_SHARED_KEY="..."
```

=== "Google Cloud Platform"

```bash
gcloud init
export GCP_TOKEN=$(gcloud auth print-access-token)
```

.kurl.init

This API accepts a list of vendors or none to register against.

Input Parameters

Name	Type	Description	Required	Default
vendors	list/symbols	The cloud vendor to connect to, options include - aws, azr, gcp, none or leave empty to register with 3 cloud vendors	No	all

Example


```
.kurl:use`kurl
.kurl.init`aws // register with AWS
.kurl.init[]  // register with AWS, AZR, GCP
```

.kurl.register

This API accepts a 4 item list. The elements in this list as defined below:

Input Parameters

Name	Type	Description	Required	Default
type	symbol	one of aws_cred, aws_sts, oauth2, oauth2_jwt, azure, basic	Yes	None
domain	str	wildcard for domain	Yes	None
tenant	str/symbol	session name, username, ID, etc., meaningful to your application	Yes	""
authInfo	list/dict	authentication info specific to the type	Yes	None

Example

Registering using environment variables:

```
info:`AccessKeyId`SecretAccessKey`Token!
(getenv`AWS_ACCESS_KEY_ID;getenv`AWS_SECRET_ACCESS_KEY;getenv`AWS_SESSION_TOKEN)
.kurl.register (`aws_cred;"*amazonaws.com";"";info)
```

AWS Registration

.kurl.aws.registerByCredentialsFile

This API accepts a credentials file.

Input Parameters

Name	Type	Description	Required	Default
file	symbol	credentials file location	Yes	None

Example

```
.kurl.aws.registerByCredentialsFile hsym`$getenv[`HOME],"/.aws/credentials"
```



```
setenv[`AWS_SHARED_CREDENTIALS_FILE] "/tmp/credentials"
.kurl.aws.registerByCredentialsFile hsym`$getenv`AWS_SHARED_CREDENTIALS_FILE
```

.kurl.deregister

This API accepts a 2 item list. The elements in this list as defined below:

Input Parameters

Name	Type	Description	Required	Default
domain	str	wildcard for domain	Yes	None
tenant	str/symbol	session name, username, ID, etc., meaningful to your application	Yes	""

Example

```
.kurl.deregister("*amazonaws.com";"")
```

Requests

Note: "Request examples" In the examples below it assumes that a CSV file is present in a S3 bucket which you can access.

.kurl.sync

Input Parameters

Name	Type	Description	Required	Default
url	str	URL to target	Yes	None
method	str	GET or POST request	Yes	None
options	dict	Can contain the keys:	Yes	""

Example

```
.kurl.sync ("https://kx-bucket.s3.us-east-
2.amazonaws.com/test_data/data.csv";`GET;::)
200i
"sym,price,size\nFDP,1.2,100\n"

// example with no registration required
.kurl.sync ("https://api.census.gov/data/2020/dec/pl?
get=NAME,P1_001N&for=state:*"; `GET; ::)
```


.kurl.async

Example

```
.kurl.async ("https://kx-bucket.s3.us-east-
2.amazonaws.com/test_data/data.csv";`GET;``callback!(`;{.debug.x:x}))
.debug.x
200i
"sym,price,size\nFDP,1.2,100\n"
```

