API

API for Data Access Services (v0 - Archived)

Reference document for LSST data products:

- Data Products Definition Document (DPDD), LSE-163

General comments about URI structure

- Start with /<serviceType>. Supported service types: "meta", "db", "image"
  - all "meta" URIs are redirected to MetaServ
  - "db" URIs are redirected to different places (L1, Qserv) depending on database level
  - all "image" URIs are redirected to ImageCutout
- <serviceType> is followed by API version. Examples: meta/v3, /image/v0, db/v17. A new version will be assigned each time there is a breaking change. New additions to the API are considered a non-breaking change.
- Retrieving images
  - image kind + image Id uniquely identify an image.
  - Examples of supported image kinds: *bias*, *calexp*, *colorForEpo*, *deepCoadd*, *raw*, *template*,... etc.
  - By default, a returned image will contain all 3 planes: data, mask and variance. To select a subset, fetch one plane per request using "plane=<value>" parameter
- For URIs that are listing things, if the list is long, by default the first *<maxResultsPerPage>* items will be shown. To overwrite this, use "start=<x>;count=<y>", e.g. "start=2000;count=1000" to get elements 2000-2999.
- For all URIs one can use Accept request-header field to receive data in appropriate format. Formats we anticipate to support (default shown underlined, in bold):
  - for images: ***image/fits***, *application/hdf5, image/jpeg*
  - for metadata: *text/html, text/csv, text/tsv, application/fitsTable,* ***application/json***
  - for database query result: *text/csv, application/fitsTable, application/IPAC table format,* ***application/json****, text/csv, application/VOtable*
- Many requests can be run asynchronously (in background). These requests are marked with "**" next to "GET", which means that "GET" should be replaced with "POST". POST will return a resource id which can then be used to check the status and retrieve results.

General information about output:

- Get image or get image cutout will return url of the result. Result can be:
  - a status, e.g., "processing"
  - an error, e.g. "Image not found"
  - the requested full images, or cutout
- In case of URI that allows start/count parameters, return values will include
  - a flag indicating whether there are more results
  - a flag indicating whether the results are "stable" (e.g. if one selects results 0-1000, and then 1000-2000, for some tables, such as Level 1, the 0-1000 might be different that what was returned when we request 1000-2000). Hint: appropriete result sorting might alleviate this problem.

Unclassified:

- coadds should be addressable by either tract/patch (currently) unique identifer or by spatial region (with no unique identifer).There may need to be additional parameters like "filter" or "airmass".

Open questions, comments, concerns:

- It is not possible to retrieve all images meeting certain criteria regardless of image kind through a single query

Related pages/ticket(s):

- Query Types
- DM-1694 - Jira project doesn't exist or you don't have permission to view it.
- DM-1916 - Jira project doesn't exist or you don't have permission to view it.

- DM-2453 - Jira project doesn't exist or you don't have permission to view it.
- DM-1868 - Jira project doesn't exist or you don't have permission to view it.
- DM-3477 - Jira project doesn't exist or you don't have permission to view it.
- DM-3484 - Jira project doesn't exist or you don't have permission to view it.
- DM-3478 - Jira project doesn't exist or you don't have permission to view it.
- DM-3479 - Jira project doesn't exist or you don't have permission to view it.
- DM-3480 - Jira project doesn't exist or you don't have permission to view it.

#	API	Full Description	Optional Parameters	Returned JSON structure	Examples of Returned Result
	GET /	List services.		Array of strings	["db", "image", "meta"]
	Metadata Service (`metaserv`) API				
M1	GET /meta	List API versions for "meta".		Array of strings	["v0", "v1"]
M2	GET /meta/v0	List types served for v0 of "meta" API.		Array of strings	`{ "result": ['db'] }`
M3	GET /meta/v0/db	List levels of databases.		Array of strings	["dc", "L1", "L2", "L3", "dev"]

M4	GET /meta/v0/db/L1?containing=%Stripe82%	List databases available for a given level, containing substring "Stripe82"	• *start=0* • *count=1000* • *containing* (show only names containing a given substring / regexp)	Array of strings	for L1: ["live", "userDB"] for L2: ["DR1", "DR2"] for L3: ["joe_myDb", "bill_test1", "mike_scratch56"]
M5	GET /meta/v0/db/L3/joe_myDb	Retrieve information about L3 database "joe_myDb"		Array containing 2 dictionaries. Keys for 1st: • name • owner • connectionHost • connectionPort Keys for 2nd: • key-value pairs representing user annotations	[{"name":"joe_myDb", "owner": "joe", host: "lsst10", "port": "3360"}, {}]
M6	GET /meta/v0/db/L2/DC_W13_Stripe82/tables	List tables for L2 database "DC_W13_Stripe82"	• *containing* (show only names containing given keyword)	Array of strings	Example of results (truncated for formatting) `{ "results": [ [ "AvgForcedPhot" ], [ "AvgForcedPhotYearly" ], [ "DeepCoadd" ], [ "DeepCoadd_Metadata" ], [...] ] }`
M7	GET /meta/v0/db/L3/joe_myDb/tables/Object	Retrieve information about table "Object" in L3 database "joe_myDb"		Array of two dictionariers. Keys for 1st: • name • description Keys for 2nd: • key-value pairs representing user annotations	[{"name": "Object", "descr": "this is my object table"}, {}]

M8	GET
	
	/meta/v0/db/L2/DC_W13_Stripe82/tables/Science_Ccd_Exposure/schema
		Retrieve schema for table "Object" in database "Science_Ccd_Exposure".		String containing output from "SHOW CREATE TABLE"	Truncated for formatting:
					
					`{`
					`  "result": [`
					`  "Science_Ccd_Exposure",`
					`  "CREATE TABLE `Science_Ccd_Exposure` (\n`scienceCcdExposureId` bigint(20) NOT NULL,\n `run` int(11) NOT NULL,\n ... PRIMARY KEY (`scienceCcdExposureId`),\n ...) ENGINE=MyISAM DEFAULT CHARSET=latin1"`
					`  ]`
					`}`

M9	GET /meta/v0/image	List levels of images.		Array of strings	["DC", "L1", "L2", "L3", "dev"]
M10	GET /meta/v0/image/L1	List image collections available in a given <level>		Array of strings	["DR1", "DR2", "ktl/test20150202"]
M11	GET/meta/v0/image/L2/DR1	List image kinds available in a given collection		Array of strings	["raw", "fpCoadd", "deepCoadd", "diffIm", "template", "calExp"]
M12	GET /meta/v0/image/L2/DR1/coadd?start=200&count=100	List coadd images (200-300) for L2 DR1	• *start=0* • *count=1000* • *owner* • *createAfter* • *createBefore* • more TBD	Array of strings	["url/of/im1", "url/of/im2"]
M13	GET /meta/v0/image/L2/DR1/coadd/12345	Retrieve information about a coadd image identified by imageId = 12345		Dictionary. Keys: • url • owner • more TBD	{"url": "url/of/img", "owner": "tom"}
	Database Query (dbserv) API				
DB1	GET /db/v0/tap	<Nothing>			

DB2	POST** /db/v0/tap/sync?query=SELECT+id,ra,decl+FROM+myDb.Object+WHERE+flux=3.2	Run a given query on L2 DR1 database	• *sql*		2 rows from "select deepForcedSourceId,scienceCcdExposureId" would look like: `{ "result": { "metadata": { "elements": [ { "datatype": "long", "name": "deepForcedSourceId" }, { "datatype": "long", "name": "scienceCcdExposureId" } ] }, "table": { "data": [ [ 8404051561545729, 125230127 ],[ 8404051561545730, 125230127 ] ] } } }`
DB3	tbd, see ⚠ DM-3477 - Jira project doesn't exist or you don't have permission to view it.	Retrieve query type for a given query			
	Image Query (`imgserv`) API (see also Image Service and Image Cutout Details)				
I1	GET /image/v0/	<nothing>			

I2	GET /image/v0/654 /explain	Return cost estimate of asynchronous query identified by a resourceId (returned through "POST /image/...")		String (for now)	TBD
I3	GET /image/v0/654 /status	Retrieve status of asynchronous request identified by a given resourceId (returned through "POST /image/...")		Dictionary. Keys: • status • startTime • progress	[{"status": "running", "startTime: "2015/05/14 016:43:21", "progress": "34%"}]
I4	GET /image/v0/654 /results	Retrieve results of asynchronous request identified by a given resourceId (returned through "POST /image/...")		Array of strings	["/nfs/lsst/L3/jack/scratch/img1", "/nfs/lsst/L3/jack/scratch/img2", "/nfs /lsst/L3/jack/scratch/img3"]
I5	GET** /image/v0/L2 /DR7/coadd	Retrieve all coadd images for L2 DR7	• *start=0* • *count=1000* • *plane* (supported: data, mask, variance)	Array of strings	["/nfs/lsst/L2/coadds/coad001", "/nfs/lsst/L2/coadds/coad002", "/nfs/lsst /L2/coadds/coad003", "/nfs/lsst/L2/coadds/coad004"]
I6	GET** /image/v0/L2 /DR1/coadd/12345?plane=mask	Retrieve "mask" plane of a full "coadd" image from L2 DR1, identified by imageId = 12345	• *plane* (supported: data, mask, variance)	Image	
I7	GET /image/v0/L2/DR1 /coadd/12345? plane=data GET /image/v0/L2/DR1 /coadd/12345? plane=mask	Retrieve a multi-extension FITS file containing coadd identified by imageId = 12345, and the corresponding mask.	• *plane* (supported: data, mask, variance)	Image	
I8	GET** /image/v0/L2 /DR1/coadd/12345 /cutout? x=1&y=2&width=30&height=30	Retrieve a cutout of a "coadd" image identified by imageId = 12345. The cutout area: 30x30 pixels centered around (1,2)	• *plane* (supported: data, mask, variance)	Image	
I9	GET** /image/v0/L2 /DR1/calexp/12345 /cutout? x1=1&y1=1&x2=2&y2=2	Retrieve a cutout of an image identified by imageId. Corners of the cutout: (1,1), (2,2)	• *plane* (supported: data, mask, variance)	Image	
I10	GET** /image/v0/L2 /DR1/calexp/12345 /cutout? plane=data&ra=1&dec=1&deltaRa=2&deltaDec=2	Retrieve "data" plane of a cutout of an image identified by imageId centered around (ra,dec) = (1,1) with a box size 2x2 arcmin.	• *plane* (supported: data, mask, variance)	Image	
I11	GET /image/v0/L2/DR1 /calexp/12345/cutout? ra=1&dec=1&widthAng=10&heightAng=10	Retrieve a cutout of a "calexp" image identified by imageid=12345. The *height Ang* and *widthAng* are in arc seconds.	• *Natural 3-plane results*	Image	
I12	GET /image/v0/L2/DR1 /calexp/12345/cutout? ra=1&dec=1&widthPix=30&heightPix=30	Retrieve a cutout of a "calexp" image identified by imageid=12345. The *height Pix* and *widthPix* are in pixels.	• *Natural 3-plane results*	Image	