DOME
A rest-inspired engine for DPM

August 25, 2016

The DPM (Disk Pool Manager) project is the most widely deployed solution for storage of large data repositories on Grid sites, and is completing the most important upgrade in its history, with the aim of bringing important new features, performance and easier long term maintainability. Work has been done to make the so-called "legacy stack" optional, and substitute it with an advanced implementation that is based on the fastCGI and RESTful technologies. Beside the obvious gain in making optional several legacy components that are difficult to maintain, this step brings important features together with performance enhancements. Among the most important features we can cite the simplification of the configuration, the possibility of working in a totally SRM-free mode, the implementation of quotas, free/used space on directories, and the implementation of volatile pools that can pull files from external sources, which can be used to deploy simple data caches.

Moreover, the communication with the new core, called DOME (Disk Operations Management Engine) now happens through secure HTTPS channels through an extensively documented, industry-compliant protocol.

Contents

1 Dome

DOME (Disk operations Management Engine) is a robust, high performance server that manages the operations of a DPM cluster. DOME is built on the FastCGI technology, and uses HTTP and JSON to communicate with clients. This initiative aims at augmenting the Disk Pool Manager (DPM) system so that its core coordination functions and inter-cluster communication paths are implemented through open components, and following contemporary development approaches headed to performance, scalability and maintainability. Among our goals we cite:

- Making optional all the so-called legacy components that are provided by the *lcg-dm* code tree, namely *libshift*, *rfiod*, *dpm(daemon)*, *dpnsdaemon*, *CSec* and others.
- provide a software infrastructure where adding new coordination features is easier than with *lcg-dm*
- provide full support for asynchronous calculation of file checksums of multiple types
- provide support for checking the consistence of replicas through their checksums
- provide structure, hooks and callouts that allow the usage of DPM as a fast and large *file cache*
- having a unified configuration file that is readable and synthetic, as opposed to the *lcg-dm* approach of having several configuration files here and there, all with differently over-simplified syntax rules (or no syntax at all, e.g. */etc/NSCONFIG*)

The DOME component has the shape of a *fastCGI* daemon, and has to be triggered by the Apache instances running in the DPM head node and in all the DPM disk servers. A configuration option defines whether it is running as head node or disk server.

For simplicity of expression, in this document we may refer to these modalities as two different components, named *DOMEhead* and *DOMEdisk*. In practice, these refer

to the same component which has been given a different command-line flag to enable/disable a different command set, implemented in the same software skeleton.

DOME is primarily a service provider for the *dmlite* framework, through the dmlite plugin called DOMEAdapter.

1.1 DOME: Main features

DOME has two modalities: headnode and disknode, which respectively represent evolutions of the *dpm* daemon and of *rfiod*, together with *libshift* and *Csec*. The functionalities are roughly as follows:

- headnode: general coordination function

  spreads load (PUT, GET, checksums) towards the available disk nodes

  keeps an in memory status of the DPM disk/pool topology with disk sizes and free space

  keeps an in memory status of the ongoing asynchronous checksum calculations

  keeps an in memory status of the ongoing asynchronous file callouts

  queues and dispatches to disk nodes the requests for asynchronous checksum calculations that have to be delayed for load balancing reasons

  queues and dispatches to disk nodes the requests for asynchronous file callouts that have to be delayed for load balancing reasons

- disknode: local disk and space-related services

  Allows to *stat* individual physical files and directories

  Allows to *statfs* filesystems to get used and free space

  Allows the local submission of checksum calculations

  Allows the local submission of file callouts

The historical tunnelling feature provided by the *rfio* infrastructure (and used by gridftp in some boundary scenarios) is implemented by DOMEAdapter directly on the top of HTTP, hence namely it does not use the DOME server.

The main difference from the legacy components is that DOME does not apply authorization again for individual user file access, as this task is already accomplished by the dmlite frontends. DOME only checks that the sender of a request is authorized to send requests, in a way that is similar to the libshift "root mode". DOME applies strong, industry standard authentication protocols to this task.
Authentication in DOME is zero-config for the regular cases (one head node and multiple disk servers), and flexible enough to add arbitrary identities that will be allowed to send commands to it.

DOME is protocol-agnostic. Its concepts of logical file name and physical file name are not linked to a particular data/metadata transfer protocol. DOME manages paths and filenames, not URLs. URLs can be constructed by the DPM frontends starting from the pfn or lfn information given by DOME.

1.1.1 From spacetokens to quota tokens

Historically, DPM does space accounting through a set of individual named space reservations, kept in the DB in the head node, and associated to pools.
Semantically, space reservations are named reservations of a part of the space of a disk pool. Requests to write a replica specify a pool that has to host the replica, hence ultimately the replica will be subject to the space reservations.

One of the weakest points in this schema is that the writer has to know technical details of the destination storage, to be able to write and be properly accounted for.

Another historical weak point is that calculating good numbers for the space occupancies can be a challenging exercise, especially if the structure of the pools has been modified in the years.
The development direction of DOME is to evolve this mechanism towards *subdirectory-based space accounting*, instead than pool-based.

In subdirectory-based space accounting, every subdirectory at less than N levels from the root is kept updated with the total size of the replicas of files that reside in that directory subtree.

This *subdirectory size* together with the information on free/used space in the pools associated to these subdirectory tree can then be used to compute the needed space

occupancy numbers.

DOME uses the records describing spacetokens that are kept in the head node DB, with minimal modification. Their meaning is slightly changed, into semantically representing a quota on one and only one directory subtree. From this point on, we will refer to them as *quotatokens*, whose behavior is similar to that of an old spacetoken associated to a directory.

A *quotatoken* attached to a directory subtree **overrides others that may be attached to its parents**.
If a directory content (counting all the replicas) exceeds the quota spacified by the quotatoken that influences it, then new PUT requests on that dir will be denied.

As a summary, the meaning of a quotatoken specifying a quota of N on poolX, associated to directory "/dir1" is *give poolX as space for hosting the files that will be written into dir1. Do not allow more than N terabytes to be hosted there.*

1.1.2 Pools and filesystems

A pool is a logical group of mount points in individual disk servers that are used to store replicas.

DOME uses the same concept of Pool than the historical DPM, hence the "Pool management" functionalities of the legacy lcg-dm components continue to work mostly as they were.

A Pool can be associated to a directory subtree by creating suitable quotatokens that describe the individual associations.

A Pool assigned to a subtree acts as a sort of "replication domain". Replicas of files belonging to that subtree are stored in filesystems belonging to this pool. Multiple replicas are spread through different file systems.

We would like to emphasize that only one pool can be assigned to the same directory subtree (e.g. /dpm/cern/ch/home/dteam/scratch). Writes into this subtree will be space-balanced between all the filesystems composing the pool assigned to it. A pool instead can be associated to any number of quotatokens (and hence, directories).

1.1.3 Open checksumming

DOME supports requests for checksums of arbitrary kind. It can:

- return the corresponding checksum that is stored in the name space
- choose an appropriate replica of the file and tell to the disk node managing it to calculate its checksum
- force the recalculation of the checksum and store it into the name space

The checksum calculation request may be queued in the head node, in memory. The architecture is designed to be self-healing in the case the checksum calculations do not end correctly, or some machines are restarted.

1.2 Tech

The architecture of DOME is expandable, through the usage of an open, human-readable protocol (JSON) and through proper design.

1.2.1 Architecture

Deployment diagram adapter_rest take Eric's picture and comments

1.2.2 Security

DOME by default accepts requests from the disk servers of the cluster it manages. This mechanism is based on secure HTTPS handshakes.
Additionally, the configuration file can specify criteria to accept requests that come to DOME. These criteria have the form of a list of allowed DNs (taken from X509 certificates).

The typical configuration uses HTTPS in the frontend configuration, to enforce the usage of a valid certificate.

1.2.3 Checksum queuer

DOME internally queues and schedules checksum calculation requests in the head node.
No more than N checksums will be run per disk mount
No more than L checksums will be run per disk server
No more than M checksums will be run in total

Checksum requests are queued in memory and dispatched to suitable disk nodes that become available with respect to the mentioned criteria. The disk nodes instances constantly update the head node about the running checksums, hance there is no need for persistence, and the system will self-heal on restarts of the head node. When finished calculating a checksum, a disk node will notify the head node and pass the result (or failure).
Eventually in the future memcached can be used for queue synchronization purposes. This evenience would require more development effort, and would have the advantage of making the DOME service able to scale horizontally. So far, we have no evidence that the DOME service needs that.

1.2.4 File pulls queuer

DOME internally on the head node queues and schedules requests for file pulls from external locations No more than N pulls will be run per disk mount
No more than L pulls will be run per disk server
No more than M pulls will be run in total

The pull itself is implemented as a simple callout in the disk server, that can invoke any file movement mechanism, from dd to create an empty file to a simple copy to uber-complex multi hop FTS xfers. The pull callout in the disk server is complemented by a stat callout, which is able to stat an external system for the presence of an offline file.
This mechanism should be polished enough to support the construction of simple file caches, without necessarily needing external, complex components. Invoking FTS instead than *dd* or *davix-get* must be an option.

Pull requests are queued in memory and dispatched to disk nodes that match the request and become available. Please note that stat requests are not queued. Please

also note that the DOME API has no stat primitive.
The disk nodes instances constantly update the head node on the running callouts, hance there is no need for persistence, and the system will self-heal on restarts of the head node. When finished pulling a file, a disk node will notify the head node and pass the result (or failure).

Eventually memcached can be used for queue synchronization purposes, only if it turns out that even in SOME cases the code is not totally preventing the spawning of new processes (which has never to happen!!!). This evenience would require more development effort, and would have the advantage of making the dpm service able to scale horizontally. So far, we have no evidence that the dpm service needs to scale horizontally the head node.

1.2.5 Only one process

The fastcgi app named DOME has only have one process and multiple internal thread pools. This simplifies a lot the development.

NB: lcg-dm contains generic utilities too, sql stuff, DB upgrade scripts, metapackages etc. These things should be moved somewhere else, possibly a place that refers to a part of the project that is not optional and that has low maintenance needs.

1.3 Application programming interface

Historically DPM implements low level functionality that is used by frontends to coordinate their activities of exposing data access protocols to clients.
In some cases, the historical DPM API has been also exposed to clients/users, eventually through a Storage Resource Manager (SRM) server.

DOME is not supposed to be used by remote clients and users. Users interact with DPM through a suitable frontend (e.g. gridFTP, xrootd, Apache) that relies on the services of dmlite and DOME in the background.

1.3.1 DPM historical primitives

The goal of this section is to present a quick list of the historical API of the DPM daemon, for subsequent reference. For details about the various calls, the reader is

encouraged to refer to the respective manpages or to the code.

- DPM_ABORTFILES:
- DPM_ABORTREQ:
- DPM_ADDFS:
- DPM_ADDPOOL:
- DPM_COPY: copy from surl to surl. Bound to rfio.
- DPM_DELREPLICA:
- DPM_EXTENDLIFE: unclear, not manpage documented
- DPM_GET:
- DPM_GETPOOLFS: mgmt, already in dmlite
- DPM_GETPOOLS: mgmt, already in dmlite
- DPM_GETPROTO: unclear, not manpage documented
- DPM_GETREQID: explicit async way to queue requests. Never used AFAIK ?
- DPM_GETREQSUM: unclear, not manpage documented
- DPM_GETSPACEMD: get spacetoken info. Unclear why it's bulk request. Some of the fields are unnecessarily complex or with involuted definitions.
- DPM_GETSPACETKN: unclear, not manpage documented
- DPM_GETSTSCOPY: unclear, not manpage documented. Seems related to the COPY command. Never used AFAIK ?
- DPM_GETSTSGET: unclear, not manpage documented. Seems related to the status of GET command through SRM
- DPM_GETSTSPUT: Not manpage documented. Polling mechanism to accommodate writes to disks or tapes.
- DPM_INCREQCTR: unclear, not manpage documented. Never used AFAIK ?
- DPM_MODFS: mgmt, already in dmlite
- DPM_MODPOOL: mgmt, already in dmlite

- DPM_PING: the best!
- DPM_PUT: main functionality that we miss
- DPM_PUTX: do we really need to make it a bulk request ? Maybe yes if we define hooks and callouts
- DPM_PUTDONE: sob
- DPM_RLSSPACE: unclear, not manpage documented
- DPM_RELFILES: documented as "release a set of files" . not clear if it makes sense
- DPM_RSVSPACE: unclear, not manpage documented
- DPM_RM: mgmt, already in dmlite
- DPM_RMFS: mgmt, already in dmlite
- DPM_RMPOOL: mgmt, already in dmlite
- DPM_SHUTDOWN: OK, we get it
- DPM_UPDSPACE: unclear, not manpage documented
- DPM_UPDFILSTS: unclear, not manpage documented
- DPM_ACCESSR: checks the existence or the accessibility of the file replica according to the dpm. The name server entry for the replica is taken into account and that of the associated pool and, if relevant, the status of an ongoing put request. The physical file name pfn is checked according to the bit pattern in amode

NOTES:

- chksum calculation/mgmt is inflexible and supports only one value per file
- NOTE: the PUT polling mechanism is among the main responsibles for the latency of writes into DPM (almost 1sec per write, in the optimal case)
- NOTE: the PUT polling mechanism is responsible of a much higher load on the head node, as multiple polling sycles have to be performed for every request
- The DPM daemon uses rfio as generic subsystem for inter-cluster communication and data sharing

1.3.2 RFIO historical primitives

These are used in the adapter, mainly for GridFTP tunnelling purposes:

- rfio_lseek
- rfio_parse
- rfio_open
- rfio_close
- rfio_write
- rfio_read
- rfio_flush
- rfio_stat64

These are used in the DPM daemon, mainly for metadata and disk stats:

- rfio_errno
- rfio_serror
- rfio_stat
- rfio_mkdir
- rfio_chown
- rfio_stat64
- rfio_allowed
- rfio_statfs64
- rfio_rcp (used to replicate files...)

1.4 Command sets of DOME

Goals:

- keep the system architecture, databases, format of the physical file names
- coherent support for multiple types of checksums
- substitute dpmd, libshift and rfio, in favor of HTTP and REST
- simplify the semantics of the commands with respect to the SRM-ish one

Each command is encoded as an HTTPS request that is inspired by the RESTful paradigm, where only the command name is URL-encoded, and every other dpm-specific parameter is encoded in a JSON snippet supplied as BODY of the request.

A legitimate response can be a 202-Accepted, with means that the client should retry the same request later to get the final result or another 202 response.

Each client request is implemented on a simplified client API based on davix. Each command is ALSO implemented on a command line tool that has the same name, the same parameters and prints the output in a pretty and readable way.

1.4.1 Common header fields

remoteclientdn

The DN of the original client that submitted the request. Typically, a Grid user with X509 credentials.

remoteclienthost

The hostname of the original client that submitted the request.

1.4.2 Head node operation

dome_put

initiates a replica upload. The client is given a location where to write the replica (redirection)

Command: `POST /dome/command/dome_put`
Request header:
no particular fields in the header
Params:

- lfn: logical file name of the entry
- additionalreplica: true—yes—1 specify to upload one more replica to an lfn that already has
- pool: suggested pool where to write (optional)
- host: suggested host where to write to particular filesys (optional)
- fs: filesystem prefix where to write the new file (optional). If specified, then host becomes mandatory. DOME will compute the remaining part of the full physical filename.

Returns 200 if OK. Other HTTP codes for the corresponding errors.
Response body:

- pool: chosen pool
- host: chosen host:port
- pfn: physical filename to be used

dome_putdone

Notifies that the upload of a replica finished successfully. It also can carry a checksum type/value that may have been calculated during the upload.

Workflow:
The notification from the data access frontend (e.g. GridFTP) always goes to the DOME instance that is running in the **disk node**. This means that generally the notification will be sent to *localhost*.
The DOME in the disk node doublechecks the existence of the file, the correctness of the path and the correctness of the file size that the frontend presumes.
If the local checks are passed, the request is forwarded automatically to the instance of DOME running in the head node, which will take the new replica into account.
Command: `POST /dome/command/dome_putdone`
Request header:

no particular fields in the header

Params:

- pfn: Physical filename
- size: size of the file
- checksumtype: Type of checksum (optional)
- checksum: Checksum value (optional)

Returns 200 if OK. Other HTTP codes for the corresponding errors.

dome_delreplica

Removes a replica, both from the logical name space and physically from the disks. Valid only in head nodes.
Command: `POST /dome/command/dome_delreplica`

Request header:
`cmd=dome_delreplica`

Params:

- server: server hosting the replica
- pfn: absolute path to the physical file or directory. The prefix must match an existing filesystem.

Returns:

-

dome_getspaceinfo

Returns total and free space information for all the pools and filesystems at once (the list is supposed to be in memory all the time)
Command: `GET /dome/command/dome_getspaceinfo`
Request header:
no particular fields in the header
Params: no parameters

Returned information:
two sequences, names **fsinfo** and **poolinfo**.
Fsinfo describes each known file system. File systems are grouped by server.
Poolinfo describes each pool, listing the filesystems that compose it.

JSON example:


```
{
    "fsinfo": {
        "fab-dpm-dev0.cern.ch": {
            "\/testfsdata": {
                "poolname": "fabpool",
                "fsstatus": "0",
                "freespace": "0",
                "physicalsize": "0"
            },
            "\/yukyuk": {
                "poolname": "fabpool",
                "fsstatus": "0",
                "freespace": "0",
                "physicalsize": "0"
            }
        },
        "pcitsdcfab.cern.ch": {
            "\/tmp": {
                "poolname": "fabpool",
                "fsstatus": "0",
                "freespace": "194393481216",
                "physicalsize": "228677218304"
            }
        }
    },
    "poolinfo": {
        "fabpool": {
            "poolstatus": "0",
            "freespace": "194393481216",
            "physicalsize": "228677218304",
            "fsinfo": {
                "fab-dpm-dev0.cern.ch": {
```

```
                "\/testfsdata": {
                    "fsstatus": "0",
                    "freespace": "0",
                    "physicalsize": "0"
                },
                "\/yukyuk": {
                    "fsstatus": "0",
                    "freespace": "0",
                    "physicalsize": "0"
                }
            },
            "pcitsdcfab.cern.ch": {
                "\/tmp": {
                    "fsstatus": "0",
                    "freespace": "194393481216",
                    "physicalsize": "228677218304"
                }
            }
        }
    }
}
```


dome_getquotatoken

Gets a quota token, using the lfn as a key. The lfn must be an existing directory. It also returns the space that is still available for each of the quota tokens listed.

Command:

```
GET /dome/command/dome_getquotatoken
```

Request header:
no particular fields in the header

Params:

- path: absolute logical path to query for quotatokens
- getsubdirs: if true, the output will include quotatokens that refer to the parent directories of the query

- getparentdirs: if true, the output will include quotatokens that refer to the subdirectories of the query

Returns: 200 if OK. Other HTTP codes for the corresponding errors.

- a sequence of :

  path: the absolute logical path a quota token is referring to

  quotatkname: Human readable name for the quota

  quotatkpoolname: The pool that serves this quotatoken

  quotatktotspace: The max number of bytes that anyone will be allowed to write into this path if there is enough free space in the pool.

  pooltotspace: total space on the assigned pool

  pathusedspace: how much space is occupied by files in that path

  pathfreespace: how much data one could still write into that path

dome_setquotatoken

Modifies or creates a quota token, using the path prefix and the poolname as a key. The path prefix must be an existing directory, the poolname should be the name of an existing pool.

Command: `POST /dome/command/dome_setquotatoken`

Request header:
no particular fields in the header

Params:

- path: the absolute logical path a quota token is referring to
- poolname: the pool that will host the replicas that are written into paths associated to this quotatoken
- quotaspace: the maximum number of bytes that the subtree rooted at path can acquire through write operations
- description: a human readable description, e.g. ATLASSCRATCH
- uniqueid (optional): normally an uuid that is internally used to assign replicas to this quotatoken (for backwards compatibility with the DPM daemon)

Returns: 200 if OK. Other HTTP codes for the corresponding errors. If the quota being set exceeds the size of the directory subtree it refers to, DOME will set the quota anyway, and give a warning in the body of the response. The result will be that noone will be able to write in that subtree until a sufficient number of files is removed.

A file write is accounted in the directory tree that contains the logical file name. A quotatoken must be assigned to one of the parent directories to tell DOME which pool the physical write should be directed to. At the same time, the quotatoken will also set a limit (quota) to the maximum number of bytes that can be written into the directory it's assigned to.

dome_delquotatoken

Deletes a quota token, using the path prefix and a poolname as a key. The path prefix must be an existing directory.

Command: `POST /dome/command/dome_delquotatoken`

Request header:
no particular fields in the header

Params:

- path: the absolute logical path a quota token is referring to
- poolname: the pool that will host the replicas that are written into paths associated to this quotatoken

Returns: 200 if OK. Other HTTP codes for the corresponding errors.

dome_getdirspaces

Computes used/free space for a path. The path must be an existing directory.

Command: `GET /dome/command/dome_getdirspaces`

Request header:
no particular fields in the header

Params:

- path: the absolute logical path to query for space

Returns: 200 if OK. Other HTTP codes for the corresponding errors.

- quotatotspace: the total space that the quota allows
- quotafreespace: how much free space the quota still has
- poolfreespace: how much physical disk space space is available in the related pool
- usedspace: how much space this directory is using
- quotatoken: the name of the quotatoken that affects the available space
- poolname: the name of the pool that affects the available space

dome_chksum

Checks, calculates or recalculates the checksum of files/replicas.
Command: `GET /dome/command/dome_chksum`

Request header:
no praticular fields in the header

Params:

- lfn: logical file name of the entry to query for the checksum
- checksum-type: Kind of checksum that is requested (e.g. adler32, MD5, etc...)
- pfn: Physical filename as it appears in the db (optional)
- force-recalc: true—false—yes—no—0—1 (optional)

Returns:

- status: whether enqueued or being calculated
- Checksum (optional if still being calculated)
- PfnChecksum (optional)

When checksum calculation was enqueued, the function additionally returns:

- queue-size: the current size of the checksum queue
- server: if enqueued, the server the calculation was delegated to
- pfn: the pfn of the file on the server

Behavior with the ForceRecalc flag unset

If the `ForceRecalc` flag is **not** set, then DOME will check the namespace for an already stored checksum of type X. If it's found in the namespace then it's returned in the body with a return code 200 'Ok'.

If a pfn is provided, then DOME will return the private checksum of that replica **and the one of the lfn**. A client will be able to compare them.

If the requested checksum is not found, then DOME will:

- if checksum of type X is already being calculated for the given resource or one of its replicas, return 202 'pending'.
- if checksum of type X is not being calculated for the given resource or one of its replicas, enqueue the request for calculating it asynchronously, and return 202 'pending'

Behavior with the ForceRecalc flag set

If the `ForceRecalc` flag is set, then DOME will unconditionally recalculate one checksum, using a random replica or the one that is specified in the Pfn parameter. If a Pfn is not specified, then the result of the calculation will be set into the metadata associated to the lfn.
A client sending this request with the `ForceRecalc` flag **set**, and getting a 202 'pending' response will have to retry the request with the `ForceRecalc` flag **unset** in order to get the result.
When the calculation task finishes, the database is

dome_chksumstatus

A disk node that has calculated a checksum (or failed) will invoke this function to store it and notify the head node that it has finished. This is also used as a sort of

heartbeat to notify the head node about checksum calculations that are pending or running.

Command: `POST /dome/command/dome_chksumstatus`

Request header:
`cmd=dome_chksumstatus`

Params:

- lfn: logical file name of the entry to submit a checksum to
- checksum-type: Kind of checksum that was requested (e.g. adler32, MD5, etc...)
- force-recalc: tells if the original request was for a forced recalculation
- checksum: value of the computed checksum (optional if still being calculated)
- update-lfn-checksum: true — false. Tells the head node whether it also needs to update the lfn checksum. (optional if still being calculated)
- pfn: Physical filename that was used to calculate it
- status: Pending—Done—Aborted
- reason: Free string describing errors or similar (for logging)

Returns: 200, unconditionally No response body.

dome_get

Returns a pfn that can be used to read a file.

If the given lfn belongs to a directory subtree that is associated to pools marked as "V" (volatile), then the file puller callout may be invoked if the file is absent AND is not being already pulled.
The result may be 'pending' if the file is being pulled.

Command: `GET /dome/command/dome_get`

Request header:
no particular fields in the the header

Returns: Code: 200 or corresponding HTTP code corresponding to the errors

- lfn: logical file name of the entry
- server: name of the server that hosts the file
- pfn: full physical filename of the file
- filesystem: filesystem that is hosting the file

dome_pullstatus

Notifies the status of a pending file pull, including its end. Until the end it must be invoked to send a sort of progress report or heartbeat. This notification is usually sent by a disk server to the head node.

Command: `POST /dome/command/pullstatus`

Request header:
no particular fields in the header

Params:

- host: server name
- pfn: physical filename
- server: server the pfn refers to
- lfn: Logical filename
- status: pending—done—aborted
- reason: Free string describing errors or similar (for logging)
- checksum-type: optional checksum type
- checksum: optional file checksum that the remote file puller may have calculated on the fly

Returns: Code: 200 if the notification's fields are correct

dome_statpool

Gets total and free space information for one pool

```
GET /dome/command/dome_statpool
```

Request header:
no particular fields in the header

Params:

- **poolname**: pool name to stat

Returns: 200 if the notification's fields are correct

- **poolstatus**: the status of the pool. 0 means active.
- **physicalsize**: the total space physically available for this pool
- **freespace**: the free space
- all the server and mountpoints that this pool contains

  for each server:mountpoint, the total space, and the free space, and the status of the mountpoint (0 means active)

JSON example:

```
{
    "poolinfo": {
        "fabpool": {
            "poolstatus": "0",
            "freespace": "194394103808",
            "physicalsize": "228677218304",
            "fsinfo": {
                "fab-dpm-dev0.cern.ch": {
                    "\/testfsdata": {
                        "fsstatus": "0",
                        "freespace": "0",
                        "physicalsize": "0"
                    },
                    "\/yukyuk": {
                        "fsstatus": "0",
                        "freespace": "0",
```

```
                    "physicalsize": "0"
                }
            },
            "pcitsdcfab.cern.ch": {
                "\/tmp": {
                    "fsstatus": "0",
                    "freespace": "194394103808",
                    "physicalsize": "228677218304"
                }
            }
        }
    }
}
```


dome_addpool

Adds a pool, or modifies an existing one with the same name. Valid only in head nodes.
Command: `POST /dome/command/dome_addpool`

Request header:
no particular fields in the header

Params:

- poolname: the name of the pool
- pool_defsize: the default file size for new files, when their size is not known yet
- pool_stype: the space type for the pool. P$\bar{\text{P}}$ermanent, V$\bar{\text{V}}$olatile

dome_rmpool

Remove from DPM control all the filesystems that are related to the same pool. Valid only in head nodes.
Command: `POST /dome/command/dome_rmpool`

Request header:
no particular fields in the header

Params:

- poolname: the name of the pool to remove

dome_addfstopool

Adds a filesystem for DPM to manage. Valid only in head nodes.
Command: `POST /dome/command/dome_addfstopool`

Request header:
no particular fields in the header

Params:

- server: server hosting the replica
- fs: absolute path to the physical file or directory. The prefix must match an existing filesystem that is mounted in the indicated disk server.
- poolname: name of the DPM pool this filesystem will be used for
- status: the status of the filesystem. Allowed values are:
  - 0: Active
  - 1: Disabled
  - 2: ReadOnly

dome_rmfs

Removes a filesystem, no matter to which pool it was attached. Valid only in head nodes.
Command: `POST /dome/command/dome_rmfs`
Request header:
no particular fields in the header

Params:

- server: server hosting the replica
- fs: absolute path to the mount point that hosts data for this fs. This must match an existing filesystem.

dome_modifyfs

Modifies the information that defines a DPM file system. Valid only in head nodes.
Command: `POST /dome/command/dome_modifyfs`
Request header:
no particular fields in the header

Params:

- poolname
- server: server hosting the replica
- fs: absolute path to the mount point that hosts data for this fs
- status: 0=active, 1=disabled, 2=readonly
- pool_defsize: the default space (bytes) checked for a new file to be written. Default is 3GB. This number will affect all the filesystems belonging to the same pool.
- pool_stype: the space type of the pool this filesystem belongs to. P=permanent, V=volatile. Default is P. This value will affect all the filesystems belonging to the same pool.

dome_getstatinfo

Returns stat information about a logical or physical file managed by DPM. If a lfn is provided and the associated pools allow file pulling, then the external stat hook will be invoked (if provided) to produce the stat information by contacting an external system.

Command: `GET /dome/command/dome_getstatinfo`

Request header:
no particular fields in the header

Params:

- lfn: a logical path to return information about. Can be omitted if a physical file name is provided.
- server: the server part of a physical file name. Can be omitted if a logical file name is provided.

- pfn: the path/filename part of a physical file name. Can be omitted if a logical file name is provided.
- rfn: a physical file name in the rfio syntax. Can be omitted if a logical file name is provided.

Returns: Code: 200 or pending

- fileid: private DPM core information
- parentfileid: private DPM core information
- size: file size
- mode: unix flags
- isdir: tells if it's a directory

dome_getreplicainfo

Returns replica information about a replica, identified by its RFN.
Command: `GET /dome/command/dome_getreplicainfo`

Request header:
no particular fields in the header

Params:

- rfn: a physical file name in the rfio syntax.

Returns: Code: 200 or pending

- replicaid
- fileid
- nbaccesses
- atime
- ptime
- ltime
- status

- type
- pool
- server
- filesystem
- rfn
- setname
- xattrs: the replica's extended attributes in JSON format

dome_addfstopool

Adds a filesystem to a pool. This implicitely creates a pool with the given name if it did not already exist.

Command:
`POST /dome/command/dome_addfstopool`

Request header:
no particular fields in the header

Params:

- poolname: a logical path to return information about. Can be omitted if a physical file name is provided.
- server: the server part of a physical file name. Can be omitted if a logical file name is provided.
- fs: the path/filename part of a physical file name. Can be omitted if a logical file name is provided.
- pool_defsize: the default size for a file whose space must be allocated without knowing its size
- pool_stype: "V" for a volatile pool that can pull files using the file pull hooks. "P" for a standard, permanent pool.

dome_getdir

Lists the content of a logical directory Command: `GET /dome/command/dome_getdir`

Request header:
no particular fields in the header

Params:

- lfn: a logical path to return information about. Can be omitted if a physical file name is provided.
- server: the server part of a physical file name. Can be omitted if a logical file name is provided.
- pfn: the path/filename part of a physical file name. Can be omitted if a logical file name is provided.
- rfn: a physical file name in the rfio syntax. Can be omitted if a logical file name is provided.

Returns: Code: 200 or pending

- fileid: private DPM core information
- parentfileid: private DPM core information
- size: file size
- mode: unix flags
- isdir: tells if it's a directory

dome_getuser

Get information about a user
Command: `GET /dome/command/dome_getuser`

Request header:
no particular fields in the header

Params:

- username: the username to extract information about

Returns: Code: 200 or error

- uid: the user's uid
- banned: whether the user is banned or not

dome_updatexattr

Update the extended attributes associated with a file.
Command: `GET /dome/command/dome_updatexattr`

Request header:
no particular fields in the header

Params:

- lfn: the lfn
- xattr: the serialized extended attributes

Returns: Code: 200 or error

dome_getidmap

Get id mapping for a user
Command: `GET /dome/command/dome_getidmap`

Request header:
no particular fields in the header

Params:

- username: the username to extract information about
- groupnames: array of strings to translate into gids, can be empty

Returns: Code: 200 or error

- uid: the user's uid
- banned: whether the user is banned or not
- groups: mapping of groupnames to gids

Example request:

```
{
  "username" : "/DC=ch/DC=cern/OU=Organic
      Units/OU=Users/CN=gbitzes/CN=749194/CN=Georgios Bitzes",
  "groupnames" : []
}
```

Example response:

```
{
    "uid": "3",
    "banned": "0",
    "groups":
    {
        "dteam":
        {
            "gid": "104",
            "banned": "0"
        }
    }
}
```

1.4.3 Disk node operation

The purpose of DOME being executed in the disk node is to give the rfio functionalities that are not given by WebDAV/HTTP, and to control the checksum calculations.

dome_makespace

Clears space on a specified volatile filesystem and VO. Used when a pending file pull is unable to start due to insufficient space.

```
POST /dome/command/dome_makespace
```

Request header:

```
cmd=dome_makespace
```

Params:

- fs: the filesystem on the local disk server to clear space from
- vo: the VO to clear space from
- size: the minimum amount of bytes which should be cleared

dome_dochksum

Immediately start an external process that calculates the checksum and returns it (or error). Upon return (or error), the the disk node invokes dome_checksumstatus in the head node with the result.

The DOME disk node is responsible for keeping the head node informed of the checksums being calculated at regular intervals, through te dome_checksumstatus command.

Command: `POST /dome/pathfile/command/dome_docheck sum`

Request header:
no particular fields in the header

Params:

- checksum-type: Kind of checksum that is requested (e.g. adler32, MD5, etc...)
- update-lfn-checksum: tells if this request also needs to update the lfn checksum. To be passed later on to the head node.
- pfn: Physical filename

Returns: 200 or various errors if the calculation process cannot be started.

dome_pull

Invokes the file puller callout. At the end of the pull, the dome_pulldone is invoked towards the head node. The DOME disk node is responsible for keeping the head node informed of the file pulls being performed at regular intervals, through te dome_pulldone command.

Command: `GET /dome/command/dome_pull`

Request header:
no particular fields in the header

Params:

- pfn: Physical filename
- lfn: Logical filename
- checksum-type: a suggestion about the kind of checksum that the external file puller may want to calculate
- neededspace: an estimation of the space that needs to be free to pull a file

Returns: 200 or various errors if the pull process cannot be started.

dome_pfnrm

Removes a physical file or directory from the disks. Valid only in disk nodes.
Command: `POST /dome/command/dome_pfnrm`

Request header:
no particular fields in the header

Params:

- pfn: absolute path to the physical file or directory. The prefix must match an existing filesystem.

Returns:

-

dome_statpfn

Return information about a physical file or directory. Valid only in disk nodes.
Command: `POST /dome/command/dome_statpfn`
Request header:
no particular fields in the header

Params:

- pfn: absolute path to the physical file or directory. The prefix must match an existing filesystem that is mounted in the indicated disk server.
- matchfs: ensure that the pfn given above is part of an existing filesystem. (default: true)

Returns:

- size: the size of the file
- mode: the unix mode bits of the file
- isdir: tells if it's a file or directory

2 Configuration

Here we list all the directives and parameters of DOME for both disk and head modalities. This chapter will become the full configuration reference.

2.1 Command-line parameters

Coming soon

2.2 Configuration file: Structure and location

The path and filename of the main configuration file is specified as a command-line parameter in the command that starts DOME. A common choice for the configuration file is::

```
/etc/dome.conf
```

The main configuration file may contain an INCLUDE directive, in order to allow a setup that contains multiple partial configuration files into a directory like:

```
/etc/dome.conf.d/
```

At the time of writing this document, the low complexity of the configuration file does not necessarily impose such a structure.

2.3 Configuration file: Directives and parameters

The parameters are subdivided nito three sets, respectively global parameters, parameters that are honoured only by a head node, parameters that are recognized only by a disk server.

2.3.1 Configuration file: Common directives for head nodes and disk nodes

INCLUDE

Interpret as configuration files all the files that are contained in the given directory. Only absolute paths are accepted.

Syntax:

```
INCLUDE: <path>
```

`<path>` is a directory containing DOME configuration files.

The configuration files are loaded and processed by the Ugr configuration subsystem in ascending alphabetic order. It's a good idea to create file names that start with a number, representing their loading priority.

Example:
Load all the configuration files that are contained in `/etc/ugr.conf.d/`
`INCLUDE: /etc/dpm.conf.d/`

glb.debug

Sets the DOME log verbosity.

Syntax:

```
glb.debug: <level>
```

`<level>` is the desired debug level, from 0 to 10.

NOTE: DOME internally uses `syslog` to log its activity, in the `user` class. We refer to the `syslog` documentation for the platform in use in order to configure its behavior, e.g. to output the log to a logfile.

NOTE: a debug level higher than 1 severely affects the performance of DOME. Never set it to a value higher than 1 in a production server.

glb.debug.components[]

Allows selecting the internal components that produce log lines. By default all the internal components produce log activity. The presence of this directive in the configuration file allows only the named components to produce log lines.

Every `glb.debug.components[]` line that appears in the configuration is considered as an individual component to enable log production for.

Syntax:

```
glb.debug.components[]: log_component
```

Example:

```
glb.debug.components[]: queue.CKSUM
glb.debug.components[]: queue.PULLER
```

glb.myhostname

Tell Dome the hostname of the machine. The hostname of the machine must match the hostname that describes its file systems, as described in the database table dpm_db.dpm_pool .

In the cases where the network configuration has several interfaces and/or multiple hostnames, we can provide this information to the running Dome instance, to avoid ambiguity.

Syntax:

```
glb.myhostname <full hostname as used to specify filesystems>
```

Example:

```
glb.myhostname domedisk457.cern.ch
```

glb.fcgi.listenport

If Dome is run in external server mode it must specify a TCP port number to bind to. Please note that a fastcgi server like Dome, when run as external server must be run using some custom init script. Please refer to the Fastcgi documentation for more information on these details.

If the given port number is 0, or the directive is asent then Dome assumes that the lifetime of this daemon will be managed by Apache.

Default value: 0

head.db.host

The host where the MySQL service is available for the dpm_db database.

head.db.user

The username to be used to connect to MySQL.

head.db.password

The password to be used to connect to MySQL.

head.db.port

The port number of the MySQL service.
Default value: 0

head.db.poolsz

The size of the internal pool of MySQL clients.
Default value: 128

glb.restclient.conn_timeout

When contacting other servers with http/rest, use the provided timeout value.

glb.restclient.ops_timeout

When contacting other servers with http/rest, use the provided timeout value.

glb.restclient.ssl_check

If false, the ssl certificate authority check is not enforced.

glb.restclient.ca_path

Path containing the certificate authority files

glb.restclient.cli_certificate

When contacting other servers with http/rest, use the provided identity/certificate. Normally the host certificate or a service certificate.

glb.restclient.cli_private_key

When contacting other servers with http/rest, use the provided identity/certificate. Normally the host certificate or a service certificate.

glb.reloadfsquotas

Dome will automatically refresh its knowledge of quota tokens and file systems. This parameter is the number of seconds between two consecutive refreshes.
Default value: 60

glb.reloadusersgroups

Dome will automatically refresh its knowledge of users and groups. This parameter is the number of seconds between two consecutive refreshes.
Default value: 60

glb.role

Configures this instance as a head node or a disk node, respectively. Syntax:
`glb.role: head|disk`
Default value: head

glb.auth.authorizeDN[]

Array containing DNs that are authorized to send commands. Commands sent by different senders will not be accepted.
Used in a head node, this list includes all the DNs that are used by disk nodes to communicate with the head node. Among the possibilities, all the disk nodes can use the same certificate, in this case only one entry has to be put here.
Used in a disk node, this list contains the DN that is used by the head node to communicate with the disk server.

head.put.minfreespace_mb

Specifies the minimum free space in megabytes for a PUT requests to consider a filesystem for writing into. Default: 4096 [which corresponds to 4 gigabytes]

glb.dmlite.configfile

The full path to a DMLite configuration file. To configure DMLite, we refer the reader to the DMLite documentation.

Example:

```
glb.dmlite.configfile: /etc/dmlite-DOME.conf
```

glb.dmlite.poolsize

The number of dmlite instances that are pooled to give internal dmlite services. This number is likely to be in the order of 50-100 in the head node, and the order of 2-10 in the disk servers.

Example:

```
glb.dmlite.poolsize: 10
```

glb.workers

The number of worker threads that execute the requests.
Default value: 300

2.3.2 Specific to head nodes

head.chksumstatus.heartbeattimeout

Maximum time, in seconds, that an entry about a checksum that is being calculated will stay in memory.
Checksums that have been dispatched and that do not send the heartbeat will be internally treated as failed for unknown reasons.
Default: 60

head.checksum.maxtotal

Maximum number of file callouts that can be run concurrently in this DPM instance. Additional requests will be queued until the condition is met.
Default: 10

head.checksum.maxpernode

Maximum number of checksums that can be assigned to a single disk node. Additional requests will be queued until the condition is met.
Default: 2

head.checksum.qtmout

Maximum time an entry in the internal file pulls queue will remain alive if not referenced either by clients or by servers while they process it.
Default: 180 secs

head.filepulls.maxtotal

Maximum number of file callouts that can be run concurrently in this DPM instance. Additional requests will be queued until the condition is met.
Default: 10

head.filepulls.maxpernode

Maximum number of checksums that can be assigned to a single disk node. Additional requests will be queued until the condition is met.
Default: 2

head.filepulls.qtmout

Maximum time an entry in the internal file pulls queue will remain alive if not referenced either by clients or by servers while they process it.
Default: 180 secs

head.gridmapfile

Absolute path to the gridmap file. The default value is /etc/lcgdm-gridmap

head.filepuller.stathook

Absolute path to an executable that can produce stat information for a logical file name by contacting an external system. This exacutable will be invoked passing the LFN as the only parameter.
The stat information has to be produced as a text line in the standard output, prefixed by the string `>>>>>` Example: `glb.filepuller.stathook: /usr/bin/externalstat.py`
Example output:

```
>>>>> Size:898945
```

DOME does not provide any such executable.

head.filepuller.stathooktimeout

Timeout in seconds for an external stathook call.
Default: 60

2.3.3 Specific to disk nodes

disk.headnode.domeurl

Url prefix for the DOME service in the headnode. This is used to contact the head node.

Internally, this URL is also used in disk nodes to determine the hostname of the head node, and authorize its attempts to connect.

Example:

```
disk.headnode.DOMEurl: https://dpmhead-rc.cern.ch/DOME
```

disk.cksummgr.heartbeatperiod

Number of seconds between notifications to the head node about the status of the checksum calculations that are being calculated.
Default: 10

disk.filepuller.pullhook

Absolute path to an executable that can ppull a file identified by a logical file name by contacting an external system. This exacutable will be invoked passing the LFN as the first parameter, and the PFN as second parameter.

Example:

```
glb.filepuller.pullhook: /usr/bin/externalpull.py
```

DOME does not provide any such executable.

3 subsystems and development tasks

This section will not be part of the official documentation. It's here for organization purposes.

- server

  server skeleton

  db pools (borrowed from dmlite)

  logger (borrowed from dmlite)

  config subsystem copied from ugr

  one class containing the internal status. Same ticker-alive objects approach like ugr. A singleton approach seems reasonable and will reduce the dev effort.

  one class describing the status of pools and filesystems, able to gather it through dmlite, refreshing every N seconds

  one class describing a request. Is boost::propertytree fine or over-engineered project poison ?

  checksum queuer and manager for running chacksums This class will need some unit tests, working locally with no setup

  file pull queuer, to queue callout requests and manage the running ones This class will need some unit tests, working locally with no setup

  The tech description of the chksum and filepull queuers is very similar. I am not convinced yet that it can be factorized into one class.

  One class that manages a set of spawned commandlines, regularly checks if they are alive and sends the pending notifications to the head node This class will need some unit tests, working locally with no setup

if DOME needs to talk to the db and do queries, it will do it directly, eventually using the same components used by dmlite-mysql

Question: do we still need to keep in the DB the log of all the PUTs ?

- c++ client class

  Maximum simplicity, just a set of calls wrapping requests and responses

  Call signature must be synchronous. Result is always the error code.

  New eventual purely async calls must be an easy addition if needed one day

  Used in DOME_adapter, avoid things that are not dmlite-friendly

  Used inside dopmrest to communicate between head and disknodes

  Uses davix to communicate

  Investigate on advantage of libs to build/parse json. There are many, I like boost property tree. The advantagte over a 20 lines implementation is not clear to me.

- DOME_adapter, which uses the client class

  implements the calls that are already there, using the current adapter as reference

- command line interface, as a set of executables that also constitute a big part of the tests!!

- scheduled tests that use the command line to do operations towards one of the testbeds

NOTE: investigate on possibility to add a list primitive that works on the physical namespace. An alternative could be to do it through Apache/webdav without passing for DOME.