# Integrating storage APIs into PMIx

Shane Snyder
PMIx ASC Q3 meeting
7/22/20

# Storage WG mission

Extend PMIx standard to support application, I/O middleware, and WLM/RM interaction with HPC storage hierarchies

- Discover available storage resources and learn about their characteristics/state
  - Query API
  - Event notification API


- Direct storage systems to accomplish some task, perhaps in coordination with other storage system layers
  - PMIx storage API


*Challenge*: Identifying suitable abstractions of key storage concepts as HPC storage ecosystem becomes more diverse

- Traditional file storage vs object storage (e.g., Lustre vs DAOS)

# Storage WG status

*Approach*: Add storage system support to existing PMIx APIs, allowing us to iterate and refine storage constructs, before moving on to a more storage-centric API

- Discover available storage resources and learn about their characteristics/state
  - Query API
  - Event notification API

  In progress, first reading @ Q4 meeting
- Direct storage systems to accomplish some task, perhaps in coordination with other storage system layers
  - PMIx storage API

  Not started

# PMIx query API

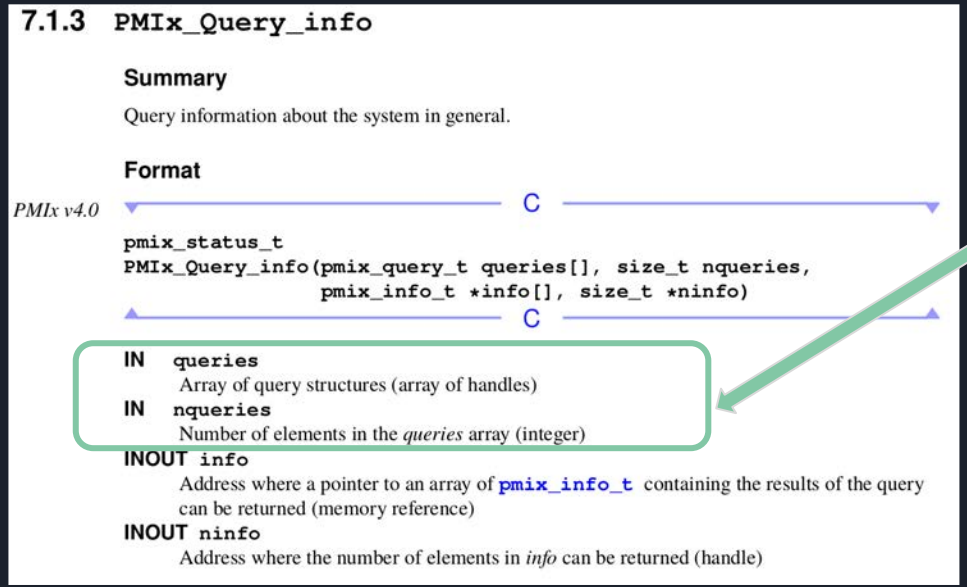
## Motivation

An ability to query capabilities, characteristics, and state of storage systems enables PMIx users to learn more about available storage hierarchies in a portable manner

- What storage resources are available?
- How do we access a specific storage resource (e.g., where is a file system mounted)?
- What are the capabilities of the storage device (e.g., peak bandwidth, persistence model, etc.)?
- What is the current state of the storage system (e.g., available capacity, file/object existence)?

# PMIx query API

## Overview



### 7.1.3 PMIx_Query_info

**Summary**

Query information about the system in general.

**Format**

*PMIx v4.0*

C

```
pmix_status_t
PMIx_Query_info(pmix_query_t queries[], size_t nqueries,
                pmix_info_t *info[], size_t *ninfo)
```

C

**IN** `queries`
Array of query structures (array of handles)

**IN** `nqueries`
Number of elements in the *queries* array (integer)

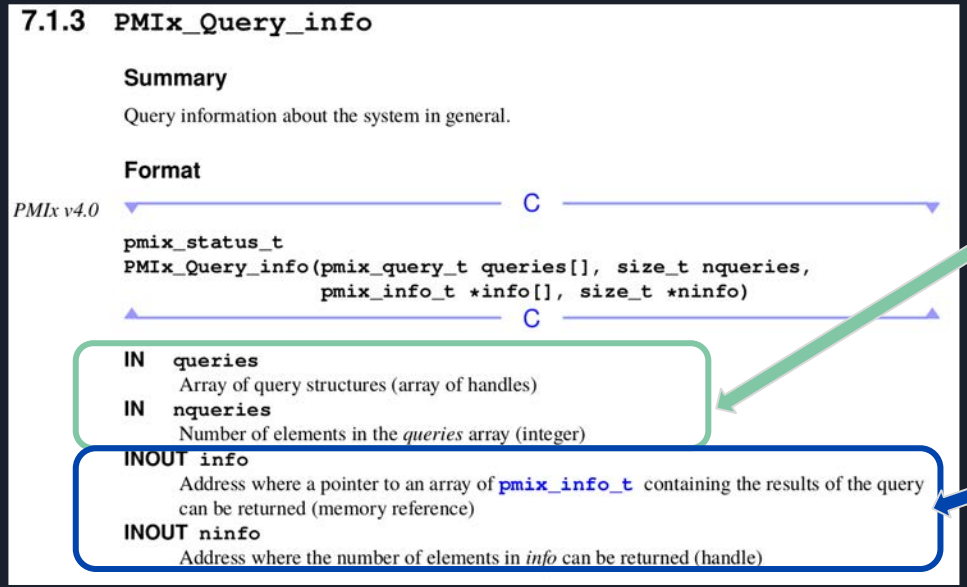
**INOUT** `info`
Address where a pointer to an array of `pmix_info_t` containing the results of the query can be returned (memory reference)

**INOUT** `ninfo`
Address where the number of elements in *info* can be returned (handle)

Input: `pmix_query_t` structure array describing set of queries to run

# PMIx query API

## Overview



### 7.1.3 PMIx_Query_info

**Summary**

Query information about the system in general.

**Format**

*PMIx v4.0*

C

```
pmix_status_t
PMIx_Query_info(pmix_query_t queries[], size_t nqueries,
                pmix_info_t *info[], size_t *ninfo)
```

C

**IN** `queries`
Array of query structures (array of handles)

**IN** `nqueries`
Number of elements in the *queries* array (integer)

**INOUT** `info`
Address where a pointer to an array of `pmix_info_t` containing the results of the query can be returned (memory reference)

**INOUT** `ninfo`
Address where the number of elements in *info* can be returned (handle)

Input: `pmix_query_t` structure array describing set of queries to run

Output: `pmix_info_t` structure array describing the results of the input queries

# PMIx query API

## Overview - query input

*PMIx v2.0*

```
typedef struct pmix_query {
    char **keys;
    pmix_info_t *qualifiers;
    size_t nqual;
} pmix_query_t;
```

*PMIx v1.0*

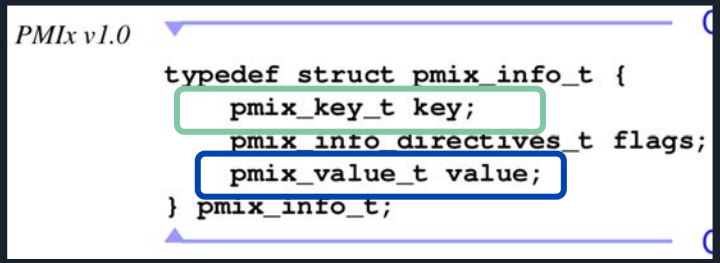
```
typedef struct pmix_info_t {
    pmix_key_t key;
    pmix_info_directives_t flags;
    pmix_value_t value;
} pmix_info_t;
```

A `pmix_query_t` describes input parameters of a query:

- Keys: attributes (`strings`) describing the types of queries to perform
  - `e.g.,PMIX_TIME_REMAINING`, to query job's remaining walltime
- Qualifiers: key-value constraints (`pmix_info_ts`) on the query(ies) to restrict scope of responses
  - `PMIX_PROC_ID=1234`, to query info for process 1234

# PMIx query API

## Overview - query output




A `pmix_info_t` describes the output of a query using a key-value type:

- Key: input attribute that was queried (`e.g.,PMIX_TIME_REMAINING`)
- Value: abstract value type allowing return of various types of data (strings, arrays, integers, etc.)

# PMIx query API

## Storage qualifiers (new)

Before considering storage system query attributes, it's helpful to consider what types of new qualifiers are required to describe storage requests:

| Qualifier | Data type | Description |
|---|---|---|
| `PMIX_STORAGE_ID` | char * | Qualifier to limit the query to a particular storage ID |
| `PMIX_STORAGE_PATH` | char * | Qualifier to limit the query to a particular storage path |
| `PMIX_STORAGE_TYPE*` | char * | Qualifier to limit the query to a particular storage type (e.g., lustre, DAOS, PFS, burst buffer, …) |

`PMIX_STORAGE_ID` is fundamental for storage queries, specifying a unique ID for the specific storage system to be queried

- E.g., 'lustre-fs0' and 'lustre-fs1' to distinguish between 2 available Lustre deployments
- Could be set manually be admins or generated by the PMIx server

# PMIx query API

## Storage qualifiers (new)

Before considering storage system query attributes, it's helpful to consider what types of new qualifiers are required to describe storage requests:

| Qualifier | Data type | Description |
|---|---|---|
| `PMIX_STORAGE_ID` | char * | Qualifier to limit the query to a particular storage ID |
| `PMIX_STORAGE_PATH` | char * | Qualifier to limit the query to a particular storage path |
| `PMIX_STORAGE_TYPE*` | char * | Qualifier to limit the query to a particular storage type (e.g., lustre, DAOS, PFS, burst buffer, …) |

`PMIX_STORAGE_PATH` qualifier is a convenience attribute for file systems to perform queries in terms of mount points or file paths, instead of PMIx storage IDs

- More natural for users who might have a file of interest and want to learn more about the storage system managing it

# PMIx query API

## Storage qualifiers (new)

Before considering storage system query attributes, it's helpful to consider what types of new qualifiers are required to describe storage requests:

| Qualifier | Data type | Description |
|---|---|---|
| `PMIX_STORAGE_ID` | char * | Qualifier to limit the query to a particular storage ID |
| `PMIX_STORAGE_PATH` | char * | Qualifier to limit the query to a particular storage path |
| `PMIX_STORAGE_TYPE*` | char * | Qualifier to limit the query to a particular storage type (e.g., lustre, DAOS, PFS, burst buffer, …) |

`PMIX_STORAGE_TYPE` qualifier is to indicate the type of underlying storage provided by the system

- Envisioned to orient users around what types of storage resources they have available (e.g., users might use node local NVM devices differently than a PFS)

# PMIx query API

## Storage qualifiers (new)

Before considering storage system query attributes, it's helpful to consider what types of new qualifiers are required to describe storage requests:

| Qualifier | Data type | Description |
|---|---|---|
| `PMIX_STORAGE_ID` | char * | Qualifier to limit the query to a particular storage ID |
| `PMIX_STORAGE_PATH` | char * | Qualifier to limit the query to a particular storage path |
| `PMIX_STORAGE_TYPE*` | char * | Qualifier to limit the query to a particular storage type (e.g., lustre, DAOS, PFS, burst buffer, …) |

We haven't been able to decide exactly what types of types makes sense to support for PMIx storage types:

- Specific things like "Lustre" and "DAOS"?
- Generic things like "PFS", "node-local", "SSD" ?.

# PMIx query API

## Storage qualifiers (existing)

Some existing PMIx qualifiers can likely be reused for storage queries:

| Qualifier | Data type | Description |
|---|---|---|
| PMIX_USERID | uid_t | Qualifier to limit the query to a particular user ID |
| PMIX_GRPID | gid_t | Qualifier to limit the query to a particular group ID |
| PMIX_HOSTNAME* | char * | Qualifier to limit the query to a particular storage host |
| PMIX_PROCID* | pmix_proc_t | Qualifier to limit the query to a particular storage process |

`PMIX_USERID` and `PMIX_GRPID` qualifiers are used to execute queries in terms of a specific users or groups (projects), rather than the entire system

- User and project quotas more relevant to users than system totals
- Sanity check with Lustre proof-of-concept for these qualifiers

# PMIx query API

## Storage qualifiers (existing)

Some existing PMIx qualifiers can likely be reused for storage queries:

| Qualifier | Data type | Description |
|---|---|---|
| PMIX_USERID | uid_t | Qualifier to limit the query to a particular user ID |
| PMIX_GRPID | gid_t | Qualifier to limit the query to a particular group ID |
| PMIX_HOSTNAME* | char * | Qualifier to limit the query to a particular storage host |
| PMIX_PROCID* | pmix_proc_t | Qualifier to limit the query to a particular storage process |

PMIX_HOSTNAME and PMIX_PROCID qualifiers are used to execute queries in terms of specific storage nodes or processes

- Measure storage capacity of storage server node X
- Measure bandwidth of storage server process Y

# PMIx query API

## Storage qualifiers (existing)

Some existing PMIx qualifiers can likely be reused for storage queries:

| Qualifier | Data type | Description |
|---|---|---|
| PMIX_USERID | uid_t | Qualifier to limit the query to a particular user ID |
| PMIX_GRPID | gid_t | Qualifier to limit the query to a particular group ID |
| PMIX_HOSTNAME* | char * | Qualifier to limit the query to a particular storage host |
| PMIX_PROCID* | pmix_proc_t | Qualifier to limit the query to a particular storage process |

But, are these the right qualifiers to use? Or do we need to add new ones?

- HOSTNAME is probably generic enough given it's just a string, but PROCID might not be appropriate for storage servers that are likely not running PMIx?

# PMIx query API

## Attributes

With suitable qualifiers defined, we can now enumerate new attribute keys needed to support storage system queries:

| Attribute | Value type | Description | Qualifiers |
|---|---|---|---|
| `PMIX_QUERY_STORAGE_LIST` | char * | Comma-delimited list of identifiers for all available storage systems (e.g, “gpfs-mirafs0, lus-thetafs0”) | `PMIX_STORAGE_TYPE` |

Applications can use this call to learn about available storage systems on a platform, and use subsequent queries to learn more about characteristics and capabilities of those systems.

`PMIX_STORAGE_TYPE` qualifier used to limit list to specific types of storage.

# PMIx query API

## Attributes

With suitable qualifiers defined, we can now enumerate new attribute keys needed to support storage system queries:

| Attribute | Value type | Description | Qualifiers |
|---|---|---|---|
| PMIX_STORAGE_CAPACITY_LIMIT | uint64_t | Overall capacity (in Megabytes[base2]) of specified storage system | PMIX_STORAGE_ID<br>PMIX_STORAGE_PATH<br>PMIX_STORAGE_TYPE<br>PMIX_USERID<br>PMIX_GRPID<br>PMIX_HOST<br>PMIX_PROCID |
| PMIX_STORAGE_CAPACITY_FREE | uint64_t | Used capacity (in Megabytes[base2]) of specified storage system | |

These queries require qualifiers to select specific storage systems (via identifier of path) or storage system types

# PMIx query API

## Attributes

With suitable qualifiers defined, we can now enumerate new attribute keys needed to support storage system queries:

| Attribute | Value type | Description | Qualifiers |
|---|---|---|---|
| PMIX_STORAGE_CAPACITY_LIMIT | uint64_t | Overall capacity (in Megabytes[base2]) of specified storage system | PMIX_STORAGE_ID<br>PMIX_STORAGE_PATH<br>PMIX_STORAGE_TYPE<br>PMIX_USERID<br>PMIX_GRPID<br>PMIX_HOST<br>PMIX_PROCID |
| PMIX_STORAGE_CAPACITY_FREE | uint64_t | Used capacity (in Megabytes[base2]) of specified storage system | |

Being able to qualify these queries with specific users or groups (projects) is necessary for shared HPC storage systems

# PMIx query API

## Attributes

With suitable qualifiers defined, we can now enumerate new attribute keys needed to support storage system queries:

| Attribute | Value type | Description | Qualifiers |
|---|---|---|---|
| PMIX_STORAGE_CAPACITY_LIMIT | uint64_t | Overall capacity (in Megabytes[base2]) of specified storage system | PMIX_STORAGE_ID<br>PMIX_STORAGE_PATH<br>PMIX_STORAGE_TYPE<br>PMIX_USERID<br>PMIX_GRPID<br>PMIX_HOST<br>PMIX_PROCID |
| PMIX_STORAGE_CAPACITY_FREE | uint64_t | Used capacity (in Megabytes[base2]) of specified storage system | |

Further qualifying these queries with specific storage servers or storage processes allows for learning more about storage systems at finer granularities (e.g., specific Lustre OSS or OST)

# PMIx query API

## Attributes

With suitable qualifiers defined, we can now enumerate new attribute keys needed to support storage system queries:

| Attribute | Value type | Description | Qualifiers |
|---|---|---|---|
| `PMIX_STORAGE_OBJECT_LIMIT` | uint64_t | Overall limit on number of objects (e.g., inodes) of specified storage system | `PMIX_STORAGE_ID`<br>`PMIX_STORAGE_PATH`<br>`PMIX_STORAGE_TYPE`<br>`PMIX_USERID`<br>`PMIX_GRPID`<br>`PMIX_HOST`<br>`PMIX_PROCID` |
| `PMIX_STORAGE_OBJECTS_FREE` | uint64_t | Number of used objects (e.g., inodes) of specified storage system | |

Standardize on objects, not files

Qualifiers same as previous capacity attributes

# PMIx query API

## Attributes

With suitable qualifiers defined, we can now enumerate new attribute keys needed to support storage system queries:

| Attribute | Value type | Description | Qualifiers |
|---|---|---|---|
| `PMIX_STORAGE_XFER_SIZE` | uint64_t | Optimal transfer size (in Kilobytes[base2]) of specified storage system | `PMIX_STORAGE_ID`<br>`PMIX_STORAGE_PATH`<br>`PMIX_STORAGE_TYPE` |

Motivated by file system block sizes, where optimal I/O transfer sizes are typically multiples of the block size used

Similar concepts likely exist in object stores.

# PMIx query API

## Attributes

Previous storage capacity, object, and transfer size attributes motivated by generalizing traditional `statfs()` syscalls

- Simplifies implementing query support for various POSIX file systems for storage system-wide queries
- Per-user and per-project quotas could be implemented by PMIx storage system plugins

```
struct statfs {
    __fsword_t f_type;    /* Type of filesystem (see below) */
    __fsword_t f_bsize;   /* Optimal transfer block size */
    fsblkcnt_t f_blocks;  /* Total data blocks in filesystem */
    fsblkcnt_t f_bfree;   /* Free blocks in filesystem */
    fsblkcnt_t f_bavail;  /* Free blocks available to
                             unprivileged user */
    fsfilcnt_t f_files;   /* Total file nodes in filesystem */
    fsfilcnt_t f_ffree;   /* Free file nodes in filesystem */
    fsid_t     f_fsid;    /* Filesystem ID */
    __fsword_t f_namelen; /* Maximum length of filenames */
    __fsword_t f_frsize;  /* Fragment size (since Linux 2.6) */
    __fsword_t f_flags;   /* Mount flags of filesystem
                             (since Linux 2.6.36) */
    __fsword_t f_spare[xxx];
                     /* Padding bytes reserved for future use */
};
```

# PMIx query API

## Attributes

Previous storage capacity, object, and transfer size attributes motivated by generalizing traditional `statfs()` syscalls

- Simplifies implementing query support for various POSIX file systems for storage system-wide queries
- Per-user and per-project quotas could be implemented by PMIx storage system plugins

We have already implemented this support for POSIX file systems, using a “common” file system storage plugin

Opens possibility of PMIx server autodiscovering available file systems (e.g., PFS, node-local storage, tmpfs) by iterating the currently mounted systems and providing generic queries for these systems

# PMIx query API

## Attributes

Previous storage capacity, object, and transfer size attributes motivated by generalizing traditional `statfs()` syscalls

- Simplifies implementing query support for various POSIX file systems for storage system-wide queries
- Per-user and per-project quotas could be implemented by PMIx storage system plugins

We are attempting to investigate this particular functionality using a Lustre PMIx storage plugin, as Lustre has internal capabilities for determining user/project quotas

# PMIx query API

## Attributes

With suitable qualifiers defined, we can now enumerate new attribute keys needed to support storage system queries:

| Attribute | Value type | Description | Qualifiers |
|---|---|---|---|
| `PMIX_STORAGE_BW_LIMIT` | float | Overall b/w limit (in Megabytes[base2]/sec) of specified storage system | `PMIX_STORAGE_ID`<br>`PMIX_STORAGE_PATH`<br>`PMIX_STORAGE_TYPE`<br>`PMIX_HOST`<br>`PMIX_PROCID` |
| `PMIX_STORAGE_BW` | float | Observed b/w (in Megabytes[base2]/sec) of specified storage system | |

Provide users with some expectations of peak and observed storage system bandwidth

PMIX_STORAGE_TYPE could potentially communicate this type of information, but that would require us to come up with pretty specific tags

# PMIx query API

## Attributes

With suitable qualifiers defined, we can now enumerate new attribute keys needed to support storage system queries:

| Attribute | Value type | Description | Qualifiers |
|---|---|---|---|
| `PMIX_STORAGE_ID` | char * | Storage ID corresponding to a given path | `PMIX_STORAGE_PATH`<br>`PMIX_STORAGE_TYPE` |
| `PMIX_STORAGE_PATH` | char * | Mount point corresponding to a specified storage ID | `PMIX_STORAGE_ID`<br>`PMIX_STORAGE_TYPE` |
| `PMIX_STORAGE_TYPE` | char * | Type of storage system given by given ID or path | `PMIX_STORAGE_ID`<br>`PMIX_STORAGE_PATH` |

Translate between previously defined qualifiers by using them as queriable attributes

# PMIx event notification API

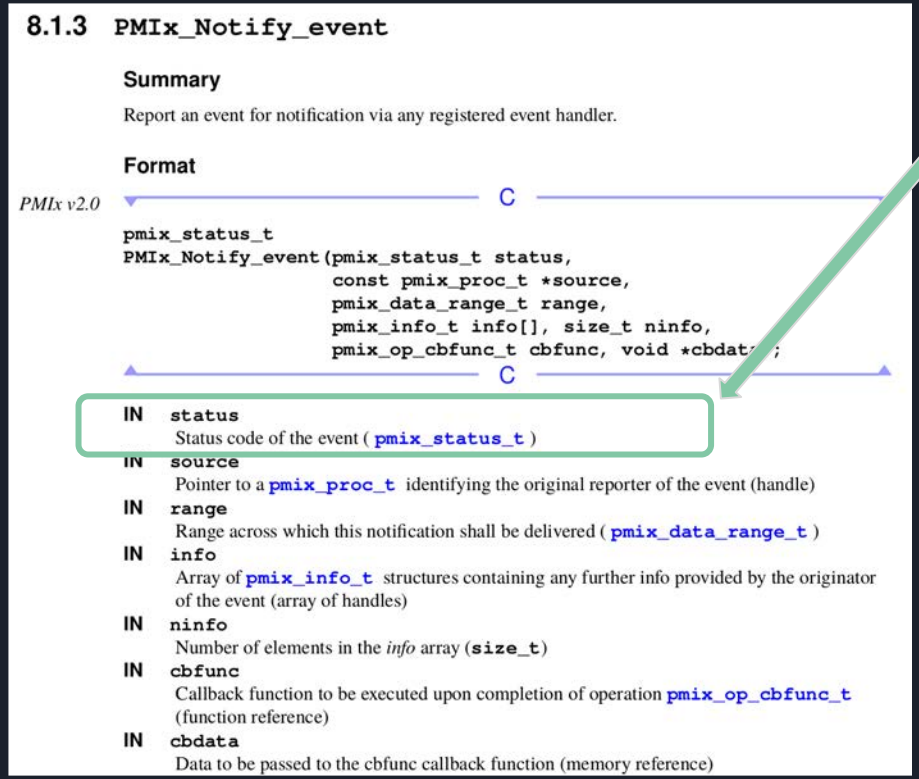
## Motivation

While queries are useful for learning more about the storage hierarchy on a given system, PMIx clients may also wish to respond to storage events of interest generated by storage-aware RMs

- Storage system failures (global failures, or node-specific or storage server-specific)
- Storage capacity getting low
- Storage servers idle/overloaded

# PMIx event notification API

## Overview



### 8.1.3 PMIx_Notify_event

**Summary**

Report an event for notification via any registered event handler.

**Format**

*PMIx v2.0*

C

```
pmix_status_t
PMIx_Notify_event(pmix_status_t status,
                  const pmix_proc_t *source,
                  pmix_data_range_t range,
                  pmix_info_t info[], size_t ninfo,
                  pmix_op_cbfunc_t cbfunc, void *cbdata);
```

C

**IN** **status**
Status code of the event ( **pmix_status_t** )

**IN** **source**
Pointer to a **pmix_proc_t** identifying the original reporter of the event (handle)

**IN** **range**
Range across which this notification shall be delivered ( **pmix_data_range_t** )

**IN** **info**
Array of **pmix_info_t** structures containing any further info provided by the originator of the event (array of handles)

**IN** **ninfo**
Number of elements in the *info* array (**size_t**)

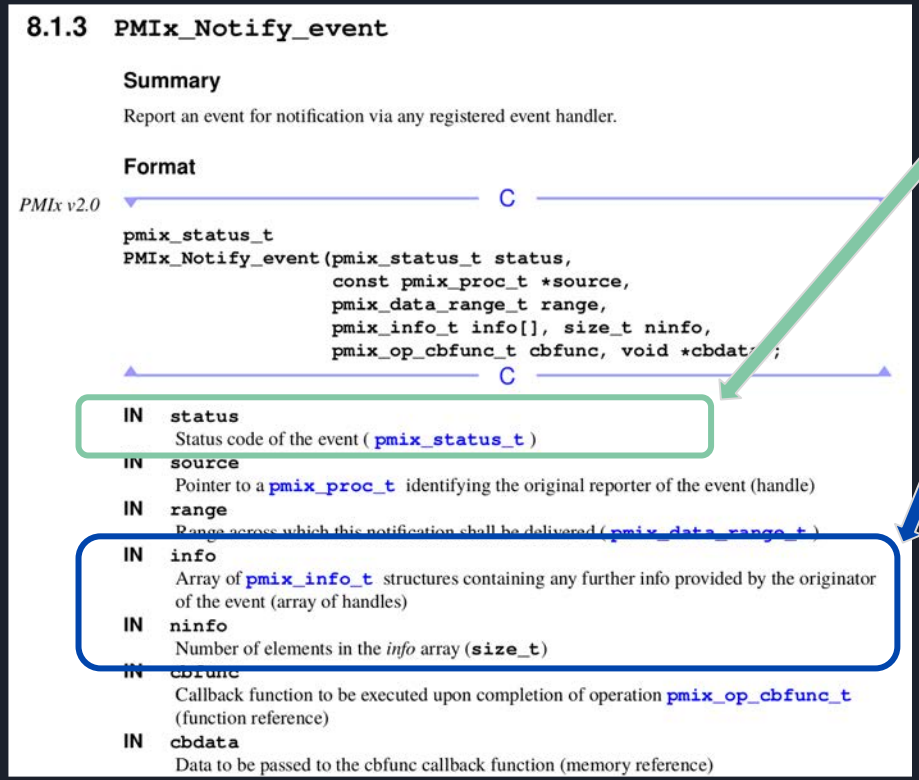
**IN** **cbfunc**
Callback function to be executed upon completion of operation **pmix_op_cbfunc_t** (function reference)

**IN** **cbdata**
Data to be passed to the cbfunc callback function (memory reference)

Input: `pmix_status_t` value indicating type of PMIx error event to report

# PMIx event notification API

## Overview



### 8.1.3 PMIx_Notify_event

**Summary**

Report an event for notification via any registered event handler.

**Format**

*PMIx v2.0*

C

```
pmix_status_t
PMIx_Notify_event(pmix_status_t status,
                  const pmix_proc_t *source,
                  pmix_data_range_t range,
                  pmix_info_t info[], size_t ninfo,
                  pmix_op_cbfunc_t cbfunc, void *cbdata);
```

C

**IN** **status**
Status code of the event ( **pmix_status_t** )

**IN** **source**
Pointer to a **pmix_proc_t** identifying the original reporter of the event (handle)

**IN** **range**
Range across which this notification shall be delivered ( **pmix_data_range_t** )

**IN** **info**
Array of **pmix_info_t** structures containing any further info provided by the originator of the event (array of handles)

**IN** **ninfo**
Number of elements in the *info* array (**size_t**)

**IN** **cbfunc**
Callback function to be executed upon completion of operation **pmix_op_cbfunc_t** (function reference)

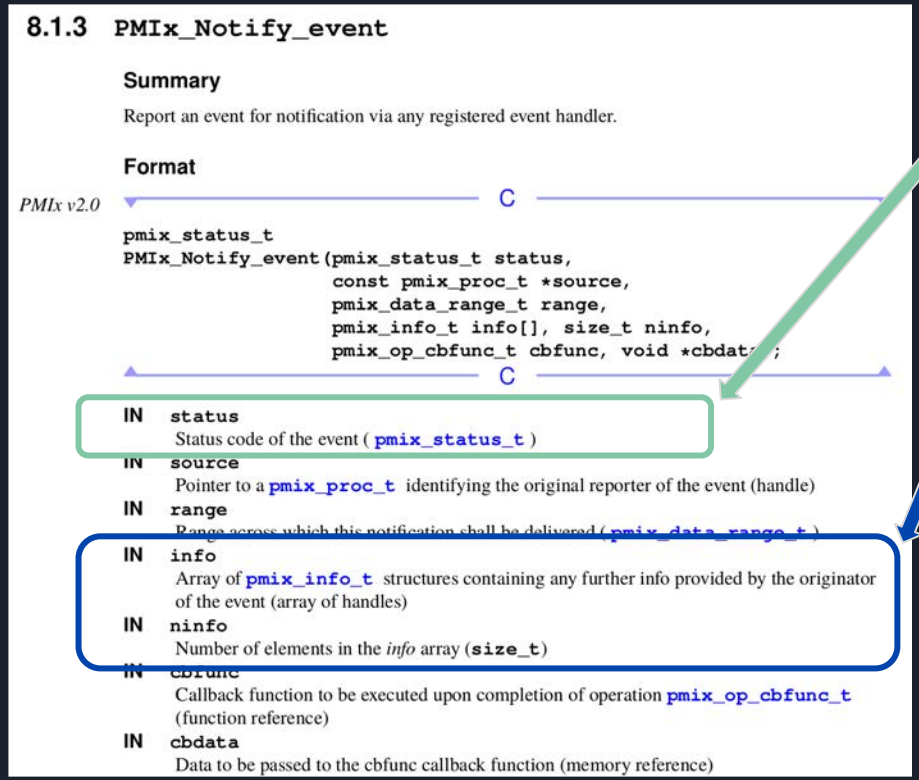
**IN** **cbdata**
Data to be passed to the cbfunc callback function (memory reference)

Input: `pmix_status_t` value indicating type of PMIx error event to report

Input: `pmix_info_t` structure array describing additional details (i.e., key-vals) of the event being reported

# PMIx event notification API

## Overview



### 8.1.3 PMIx_Notify_event

**Summary**

Report an event for notification via any registered event handler.

**Format**

*PMIx v2.0*

C

```
pmix_status_t
PMIx_Notify_event(pmix_status_t status,
                  const pmix_proc_t *source,
                  pmix_data_range_t range,
                  pmix_info_t info[], size_t ninfo,
                  pmix_op_cbfunc_t cbfunc, void *cbdata);
```

C

**IN** **status**
Status code of the event ( **pmix_status_t** )

**IN** **source**
Pointer to a **pmix_proc_t** identifying the original reporter of the event (handle)

**IN** **range**
Range across which this notification shall be delivered ( **pmix_data_range_t** )

**IN** **info**
Array of **pmix_info_t** structures containing any further info provided by the originator of the event (array of handles)

**IN** **ninfo**
Number of elements in the *info* array (**size_t**)

**IN** **cbfunc**
Callback function to be executed upon completion of operation **pmix_op_cbfunc_t** (function reference)

**IN** **cbdata**
Data to be passed to the cbfunc callback function (memory reference)

Input: `pmix_status_t` value indicating type of PMIx error event to report

Input: `pmix_info_t` structure array describing additional details (i.e., key-vals) of the event being reported

Approach: define new status codes for storage events of interest, and determine necessary attributes to describe event details

# PMIx event notification API

## Storage events

We are still investigating the storage events we want to support, but generally view the event notification API as a way of communicating critical information to users (storage errors, nearing storage capacity limit, etc.) rather than more mundane details of storage systems

At a glance, the qualifiers we have defined for storage queries can mostly be reused to help describe storage-related events:

- PMIX_STORAGE_ID to communicate the storage system generating the event
- PMIX_HOST, PMIX_PROCID for providing higher granularity information on the event (i.e., this storage server process just went down, or this storage target is running out of space)
- May also add new attributes to better describe storage events (i.e., an attribute describing storage capacity remaining if it is getting low)

# Next steps

Process for formally reading query & event notification work in Q4 meeting

- Create Issue for discussion
- Create PR to submit proposed changes
- Add straw poll and pass at least 2 weeks ahead of Q4 meeting

Start storage API work

- Need to try to loop in Cray (DataWarp) and potentially someone from OLCF (Spectral)

If you or anyone you know is interested in participating, we could definitely use more input:

- Every other Tuesday (next meeting August 4th), 4PM CST
- https://groups.google.com/forum/#!forum/pmix-forum-wg-storage for meeting invites, notes, and other updates

Questions? Comments?