Process Management Interface for Exascale (PMIx) Standard

Version 2.1 (Draft)

December 2018

This document describes the Process Management Interface for Exascale (PMIx) Standard, version 2.1 (Draft).

Comments: Please provide comments on the PMIx Standard by filing issues on the document repository https://github.com/pmix/pmix-standard/issues or by sending them to the PMIx Community mailing list at https://groups.google.com/forum/#!forum/pmix. Comments should include the version of the PMIx standard you are commenting about, and the page, section, and line numbers that you are referencing. Please note that messages sent to the mailing list from an unsubscribed e-mail address will be ignored.

Contents

CHAPTER 1

Introduction

The Process Management Interface (PMI) has been used for quite some time as a means of exchanging wireup information needed for inter-process communication. Two versions (PMI-1 and PMI-2) have been released as part of the MPICH effort, with PMI-2 demonstrating better scaling properties than its PMI-1 predecessor. However, two significant challenges face the High Performance Computing (HPC) community as it continues to move towards machines capable of exaflop and higher performance levels:

- the physical scale of the machines, and the corresponding number of total processes they support, is expected to reach levels approaching 1 million processes executing across 100 thousand nodes. Prior methods for initiating applications relied on exchanging communication endpoint information between the processes, either directly or in some form of hierarchical collective operation. Regardless of the specific mechanism employed, the exchange across such large applications would consume considerable time, with estimates running in excess of 5-10 minutes; and
- whether it be hybrid applications that combine OpenMP threading operations with MPI, or application-steered workflow computations, the HPC community is experiencing an unprecedented wave of new approaches for computing at exascale levels. One common thread across the proposed methods is an increasing need for orchestration between the application and the system management software stack (SMS) comprising the scheduler (a.k.a. the workload manager (WLM)), the resource manager (RM), global file system, fabric, and other subsystems. The lack of available support for application-to-SMS integration has forced researchers to develop "virtual" environments that hide the SMS behind a customized abstraction layer, but this results in considerable duplication of effort and a lack of portability.

Process Management Interface - Exascale (PMIx) represents an attempt to resolve these questions by providing an extended version of the PMI definitions specifically designed to support clusters up to exascale and larger sizes. The overall objective of the project is not to branch the existing definitions – in fact, PMIx fully supports both of the existing PMI-1 and PMI-2 Application Programming Interfaces (APIs) – but rather to:

a) add flexibility to the existing APIs by adding an array of key-value “attribute” pairs to each API signature that allows implementers to customize the behavior of the API as future needs emerge without having to alter or create new variants of it;
b) add new APIs that provide extended capabilities such as asynchronous event notification plus dynamic resource allocation and management;

c) establish a collaboration between SMS subsystem providers including resource manager, fabric, file system, and programming library developers to define integration points between the various subsystems as well as agreed upon definitions for associated APIs, attribute names, and data types;
d) form a standards-like body for the definitions; and
e) provide a reference implementation of the PMIx standard.

Complete information about the PMIx standard and affiliated projects can be found at the PMIx web site: https://pmix.org

1.1 Charter

The charter of the PMIx community is to:

- Define a set of agnostic APIs (not affiliated with any specific programming model or code base) to support interactions between application processes and the SMS.
- Develop an open source (non-copy-left licensed) standalone "reference" library implementation to facilitate adoption of the PMIx standard.
- Retain transparent backward compatibility with the existing PMI-1 and PMI-2 definitions, any future PMI releases, and across all PMIx versions.
- Support the "Instant On" initiative for rapid startup of applications at exascale and beyond.
- Work with the HPC community to define and implement new APIs that support evolving programming model requirements for application interactions with the SMS.

Participation in the PMIx community is open to anyone, and not restricted to only code contributors to the reference implementation.

1.2 PMIx Standard Overview

The PMIx Standard defines and describes the interface developed by the PMIx Reference Implementation (PRI). Much of this document is specific to the PMIx Reference Implementation (PRI)'s design and implementation. Specifically the standard describes the functionality provided by the PRI, and what the PRI requires of the clients and resource managers (RMs) that use it's interface.

1.2.1 Who should use the standard?

The PMIx Standard informs PMIx clients and RMs of the syntax and semantics of the PMIx APIs.

PMIx clients (e.g., tools, Message Passing Environment (MPE) libraries) can use this standard to understand the set of attributes provided by various APIs of the PRI and their intended behavior. Additional information about the rationale for the selection of specific interfaces and attributes is also provided.

PMIx-enabled RMs can use this standard to understand the expected behavior required of them when they support various interfaces/attributes. In addition, optional features and suggestions on behavior are also included in the discussion to help guide RM design and implementation.

1.2.2 What is defined in the standard?

The PMIx Standard defines and describes the interface developed by the PMIx Reference Implementation (PRI). It defines the set of attributes that the PRI supports; the set of attributes that are required of a RM to support, for a given interface; and the set of optional attributes that an RM may choose to support, for a given interface.

1.2.3 What is *not* defined in the standard?

No standards body can require an implementer to support something in their standard, and PMIx is no different in that regard. While an implementer of the PMIx library itself must at least include the standard PMIx headers and instantiate each function, they are free to return "not supported" for any function they choose not to implement.

This also applies to the host environments. Resource managers and other system management stack components retain the right to decide on support of a particular function. The PMIx community continues to look at ways to assist SMS implementers in their decisions by highlighting functions that are critical to basic application execution (e.g., **`PMIx_Get`**), while leaving flexibility for tailoring a vendor's software for their target market segment.

One area where this can become more complicated is regarding the attributes that provide information to the client process and/or control the behavior of a PMIx standard API. For example, the **`PMIX_TIMEOUT`** attribute can be used to specify the time (in seconds) before the requested operation should time out. The intent of this attribute is to allow the client to avoid "hanging" in a request that takes longer than the client wishes to wait, or may never return (e.g., a **`PMIx_Fence`** that a blocked participant never enters).

If an application (for example) truly relies on the **`PMIX_TIMEOUT`** attribute in a call to **`PMIx_Fence`** , it should set the required flag in the **`pmix_info_t`** for that attribute. This informs the library and its SMS host that it must return an immediate error if this attribute is not supported. By not setting the flag, the library and SMS host are allowed to treat the attribute as optional, ignoring it if support is not available.

It is therefore critical that users and application implementers:

a) consider whether or not a given attribute is required, marking it accordingly; and
b) check the return status on all PMIx function calls to ensure support was present and that the request was accepted. Note that for non-blocking APIs, a return of **PMIX_SUCCESS** only indicates that the request had no obvious errors and is being processed – the eventual callback will return the status of the requested operation itself.

While a PMIx library implementer, or an SMS component server, may choose to support a particular PMIx API, they are not required to support every attribute that might apply to it. This would pose a significant barrier to entry for an implementer as there can be a broad range of applicable attributes to a given API, at least some of which may rarely be used. The PMIx community is attempting to help differentiate the attributes by indicating those that are generally used (and therefore, of higher importance to support) vs those that a "complete implementation" would support.

Note that an environment that does not include support for a particular attribute/API pair is not "incomplete" or of lower quality than one that does include that support. Vendors must decide where to invest their time based on the needs of their target markets, and it is perfectly reasonable for them to perform cost/benefit decisions when considering what functions and attributes to support.

The flip side of that statement is also true: Users who find that their current vendor does not support a function or attribute they require may raise that concern with their vendor and request that the implementation be expanded. Alternatively, users may wish to utilize the PMIx-based Reference RunTime Environment (PRRTE) as a "shim" between their application and the host environment as it might provide the desired support until the vendor can respond. Finally, in the extreme, one can exploit the portability of PMIx-based applications to change vendors.

1.2.4 General Guidance for PMIx Users and Implementors

The PMIx Standard defines the behavior of the PMIx Reference Implementation (PRI). A complete system harnessing the PMIx interface requires an agreement between the PMIx client, be it a tool or library, and the PMIx-enabled RM. The PRI acts as an intermediary between these two entities by providing a standard API for the exchange of requests and responses. The degree to which the PMIx client and the PMIx-enabled RM may interact needs to be defined by those developer communities. The PMIx standard can be used to define the specifics of this interaction.

PMIx clients (e.g., tools, MPE libraries) may find that they depend only on a small subset of interfaces and attributes to work correctly. PMIx clients are strongly advised to define a document itemizing the PMIx interfaces and associated attributes that are required for correct operation, and are optional but recommended for full functionality. The PMIx standard cannot define this list for all given PMIx clients, but such a list is valuable to RMs desiring to support these clients.

PMIx-enabled RMs may choose to implement a subset of the PMIx standard and/or define attributes beyond those defined herein. PMIx-enabled RMs are strongly advised to define a document itemizing the PMIx interfaces and associated attributes they support, with any annotations about behavior limitations. The PMIx standard cannot define this list for all given PMIx-enabled RMs, but such a list is valuable to PMIx clients desiring to support a broad range of PMIx-enabled RMs.

1.3 PMIx Architecture Overview

This section presents a brief overview of the PMIx Architecture [1]. Note that this is a conceptual model solely used to help guide the standards process — it does not represent a design requirement on any PMIx implementation. Instead, the model is used by the PMIx community as a sounding board for evaluating proposed interfaces and avoid unintentionally imposing constraints on implementers. Built into the model are two guiding principles also reflected in the standard. First, PMIx operates in the mode of a *messenger*, and not a *doer* — i.e., the role of PMIx is to provide communication between the various participants, relaying requests and returning responses. The intent of the standard is not to suggest that PMIx itself actually perform any of the defined operations — this is left to the various SMS elements and/or the application. Any exceptions to that intent are left to the discretion of the particular implementation.

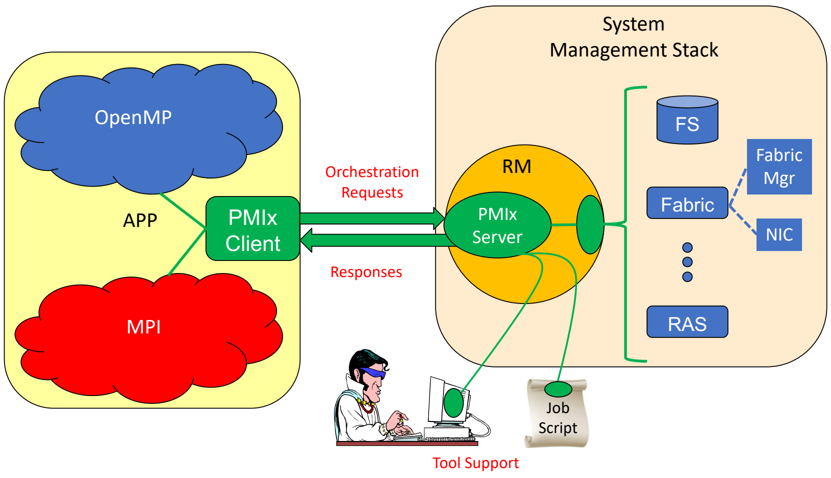

Figure 1.1.: PMIx-SMS Interactions

Thus, as the diagram in Fig. 1.1 shows, the application is built against a PMIx client library that contains the client-side APIs, attribute definitions, and communication support for interacting with the local PMIx server. Intra-process cross-library interactions are supported at the client level to avoid unnecessary burdens on the server. Orchestration requests are sent to the local PMIx server, which subsequently passes them to the host SMS (here represented by an RM daemon) using the

PMIx server callback functions the host SMS registered during PMIx_server_init. The host SMS can indicate its lack of support for any operation by simply providing a *NULL* for the associated callback function, or can create a function entry that returns *not supported* when called.

The conceptual model places the burden of fulfilling the request on the host SMS. This includes performing any inter-node communications, or interacting with other SMS elements. Thus, a client request for a network traffic report does not go directly from the client to the Fabric Manager (FM), but instead is relayed to the PMIx server, and then passed to the host SMS for execution. This architecture reflects the second principle underlying the standard — namely, that connectivity is to be minimized by channeling all application interactions with the SMS through the local PMIx server.

Recognizing the burden this places on SMS vendors, the PMIx community has included interfaces by which the host can request support from local SMS elements. Once the SMS has transferred the request to an appropriate location, a PMIx server interface can be used to pass the request between SMS subsystems. For example, a request for network traffic statistics can utilize the PMIx networking abstractions to retrieve the information from the FM. This reduces the portability and interoperability issues between the individual subsystems by transferring the burden of defining the interoperable interfaces from the SMS subsystems to the PMIx community, which continues to work with those providers to develop the necessary support.

Tools, whether standalone or embedded in job scripts, are an exception to the communication rule and can connect to any PMIx server providing they are given adequate rendezvous information. The PMIx conceptual model views the collection of PMIx servers as a cloud-like conglomerate — i.e., orchestration and information requests can be given to any server regardless of location. However, tools frequently execute on locations that may not house an operating PMIx server — e.g., a users notebook computer. Thus, tools need the ability to remotely connect to the PMIx server "cloud".

The scope of the PMIx standard therefore spans the range of these interactions, between client-and-SMS and between SMS subsystems. Note again that this does not impose a requirement on any given PMIx implementation to cover the entire range — implementers are free to return *not supported* from any PMIx function.

1.3.1 The PMIx Reference Implementation (PRI)

The PMIx community has committed to providing a complete, reference implementation of each version of the standard. Note that the definition of the PMIx Standard is not contingent upon use of the PMIx Reference Implementation (PRI) — any implementation that supports the defined APIs is a PMIx Standard compliant implementation. The PRI is provided solely for the following purposes:

- Validation of the standard.
  No proposed change and/or extension to the PMIx standard is accepted without an accompanying prototype implementation in the PRI. This ensures that the proposal has undergone at least some minimal level of scrutiny and testing before being considered.

- Ease of adoption.
  The PRI is designed to be particularly easy for resource managers (and the SMS in general) to adopt, thus facilitating a rapid uptake into that community for application portability. Both client and server PMIx libraries are included, along with examples of client usage and server-side integration. A list of supported environments and versions is maintained on the PMIx web site https://pmix.org/support/faq/what-apis-are-supported-on-my-rm/

The PRI does provide some internal implementations that lie outside the scope of the PMIx standard. This includes several convenience macros as well as support for consolidating collectives for optimization purposes (e.g., the PMIx server aggregates all local **`PMIx_Fence`** calls before passing them to the SMS for global execution). In a few additional cases, the PMIx community (in partnership with the SMS subsystem providers) have determined that a base level of support for a given operation can best be portably provided by including it in the PRI.

Instructions for downloading, and installing the PRI are available on the community's web site https://pmix.org/code/getting-the-reference-implementation/.The PRI targets support for the Linux operating system. A reasonable effort is made to support all major, modern Linux distributions; however, validation is limited to the most recent 2-3 releases of RedHat Enterprise Linux (RHEL), Fedora, CentOS, and SUSE Linux Enterprise Server (SLES). In addition, development support is maintained for Mac OSX. Production support for vendor-specific operating systems is included as provided by the vendor.

1.3.2 The PMIx Reference RunTime Environment (PRRTE)

The PMIx community has also released PRRTE — i.e., a runtime environment containing the reference implementation and capable of operating within a host SMS. PRRTE provides an easy way of exploring PMIx capabilities and testing PMIx-based applications outside of a PMIx-enabled environment by providing a "shim" between the application and the host environment that includes full support for the PRI. The intent of PRRTE is not to replace any existing production environment, but rather to enable developers to work on systems that do not yet feature a PMIx-enabled host SMS or one that lacks a PMIx feature of interest. Instructions for downloading, installing, and using PRRTE are available on the community's web site https://pmix.org/code/getting-the-pmix-reference-server/

1.4 Organization of this document

The remainder of this document is structured as follows:

- Introduction and Overview in Chapter 1 on page 1
- Terms and Conventions in Chapter 2 on page 11
- Data Structures and Types in Chapter 3 on page 16

- PMIx Initialization and Finalization in Chapter 4 on page 86
- Key/Value Management in Chapter 5 on page 99
- Process Management in Chapter 6 on page 129
- Job Management in Chapter 7 on page 148
- Event Notification in Chapter 8 on page 166
- Data Packing and Unpacking in Chapter 9 on page 175
- PMIx Server Specific Interfaces in Chapter 10 on page 185

1.5 Version 1.0: June 12, 2015

The PMIx version 1.0 *ad hoc* standard was defined in the PMIx Reference Implementation (PRI) header files as part of the PRI v1.0.0 release prior to the creation of the formal PMIx 2.0 standard. Below are a summary listing of the interfaces defined in the 1.0 headers.

- Client APIs
  - PMIx_Init, **PMIx_Initialized** , **PMIx_Abort** , **PMIx_Finalize**
  - **PMIx_Put** , **PMIx_Commit** ,
  - **PMIx_Fence** , **PMIx_Fence_nb**
  - **PMIx_Get** , **PMIx_Get_nb**
  - **PMIx_Publish** , **PMIx_Publish_nb**
  - **PMIx_Lookup** , **PMIx_Lookup**
  - **PMIx_Unpublish** , **PMIx_Unpublish_nb**
  - **PMIx_Spawn** , **PMIx_Spawn_nb**
  - **PMIx_Connect** , **PMIx_Connect_nb**
  - **PMIx_Disconnect** , **PMIx_Disconnect_nb**
  - **PMIx_Resolve_nodes** , **PMIx_Resolve_peers**
- Server APIs
  - **PMIx_server_init** , **PMIx_server_finalize**
  - **PMIx_generate_regex** , **PMIx_generate_ppn**
  - **PMIx_server_register_nspace** , **PMIx_server_deregister_nspace**
  - **PMIx_server_register_client** , **PMIx_server_deregister_client**

  - **PMIx_server_setup_fork** , **PMIx_server_dmodex_request**
- Common APIs
  - **PMIx_Get_version** , **PMIx_Store_internal** , **PMIx_Error_string**
  - **PMIx_Register_errhandler**, **PMIx_Deregister_errhandler**, **PMIx_Notify_error**

The **PMIx_Init** API was subsequently modified in the PRI release v1.1.0.

1.6 Version 2.0: Sept. 2018

The following APIs were introduced in v2.0 of the PMIx Standard:

- Client APIs
  - **PMIx_Query_info_nb** , **PMIx_Log_nb**
  - **PMIx_Allocation_request_nb** , **PMIx_Job_control_nb** , **PMIx_Process_monitor_nb** , **PMIx_Heartbeat**
- Server APIs
  - **PMIx_server_setup_application** , **PMIx_server_setup_local_support**
- Tool APIs
  - **PMIx_tool_init** , **PMIx_tool_finalize**
- Common APIs
  - **PMIx_Register_event_handler** , **PMIx_Deregister_event_handler**
  - **PMIx_Notify_event**
  - **PMIx_Proc_state_string** , **PMIx_Scope_string**
  - **PMIx_Persistence_string** , **PMIx_Data_range_string**
  - **PMIx_Info_directives_string** , **PMIx_Data_type_string**
  - **PMIx_Alloc_directive_string**
  - **PMIx_Data_pack** , **PMIx_Data_unpack** , **PMIx_Data_copy**
  - **PMIx_Data_print** , **PMIx_Data_copy_payload**

The **PMIx_Init** API was modified in v2.0 of the standard from its *ad hoc* v1.0 signature to include passing of a **pmix_info_t** array for flexibility and "future-proofing" of the API. In addition, the **PMIx_Notify_error**, **PMIx_Register_errhandler**, and **PMIx_Deregister_errhandler** APIs were replaced.

1.7 Version 2.1: Dec. 2018

The v2.1 update includes clarifications and corrections, plus addition of examples:

- Clarify description of `PMIx_Connect` and `PMIx_Disconnect` APIs.
- Explain that values for the `PMIX_COLLECTIVE_ALGO` are environment-dependent
- Identify the namespace/rank values required for retrieving attribute-associated information using the `PMIx_Get` API
- Provide definitions for `session` , `job` , `application` , and other terms used throughout the document
- Clarify definitions of `PMIX_UNIV_SIZE` versus `PMIX_JOB_SIZE`
- Clarify server module function return values
- Provide examples of the use of `PMIx_Get` for retrieval of information
- Clarify the use of `PMIx_Get` versus `PMIx_Query_info_nb`
- Clarify return values for non-blocking APIs and emphasize that callback functions must not be invoked prior to return from the API
- Provide detailed example for construction of the `PMIx_server_register_nspace` input information array
- Define information levels (e.g., `session` vs `job`) and associated attributes for both storing and retrieving values
- Clarify roles of PMIx server library and host environment for collective operations
- Clarify definition of `PMIX_UNIV_SIZE`

CHAPTER 2

PMIx Terms and Conventions

The PMIx Standard has adopted the widespread use of key-value *attributes* to add flexibility to the functionality expressed in the existing APIs. Accordingly, the community has chosen to require that the definition of each standard API include the passing of an array of attributes. These provide a means of customizing the behavior of the API as future needs emerge without having to alter or create new variants of it. In addition, attributes provide a mechanism by which researchers can easily explore new approaches to a given operation without having to modify the API itself.

The PMIx community has further adopted a policy that modification of existing released APIs will only be permitted under extreme circumstances. In its effort to avoid introduction of any such backward incompatibility, the community has avoided the definitions of large numbers of APIs that each focus on a narrow scope of functionality, and instead relied on the definition of fewer generic APIs that include arrays of directives for "tuning" the function's behavior. Thus, modifications to the PMIx standard increasingly consist of the definition of new attributes along with a description of the APIs to which they relate and the expected behavior when used with those APIs.

One area where this can become more complicated relates to the attributes that provide directives to the client process and/or control the behavior of a PMIx standard API. For example, the **`PMIX_TIMEOUT`** attribute can be used to specify the time (in seconds) before the requested operation should time out. The intent of this attribute is to allow the client to avoid hanging in a request that takes longer than the client wishes to wait, or may never return (e.g., a **`PMIx_Fence`** that a blocked participant never enters).

If an application truly relies on the **`PMIX_TIMEOUT`** attribute in a call to **`PMIx_Fence`**, it should set the *required* flag in the **`pmix_info_t`** for that attribute. This informs the library and its SMS host that it must return an immediate error if this attribute is not supported. By not setting the flag, the library and SMS host are allowed to treat the attribute as optional, silently ignoring it if support is not available.

Advice to users

It is critical that users and application developers consider whether or not a given attribute is required (marking it accordingly) and always check the return status on all PMIx function calls to ensure support was present and that the request was accepted. Note that for non-blocking APIs, a return of **`PMIX_SUCCESS`** only indicates that the request had no obvious errors and is being processed. The eventual callback will return the status of the requested operation itself.

While a PMIx library implementer, or an SMS component server, may choose to support a particular PMIx API, they are not required to support every attribute that might apply to it. This would pose a significant barrier to entry for an implementer as there can be a broad range of applicable attributes to a given API, at least some of which may rarely be used in a specific market area. The PMIx community is attempting to help differentiate the attributes by indicating in the standard those that are generally used (and therefore, of higher importance to support) versus those that a "complete implementation" would support.

In addition, the document refers to the following entities and process stages when describing use-cases or operations involving PMIx:

- *session* refers to an allocated set of resources assigned to a particular user by the system WLM. Historically, HPC sessions have consisted of a static allocation of resources - i.e., a block of resources are assigned to a user in response to a specific request and managed as a unified collection. However, this is changing in response to the growing use of dynamic programming models that require on-the-fly allocation and release of system resources. Accordingly, the term *session* in this document refers to the current block of assigned resources and is a potentially dynamic entity.
- *slot* refers to an allocated entry for a process. WLMs frequently allocate entire nodes to a *session*, but can also be configured to define the maximum number of processes that can simultaneously be executed on each node. This often corresponds to the number of hardware Processing Units (PUs) (typically cores, but can also be defined as hardware threads) on the node. However, the correlation between hardware PUs and slot allocations strictly depends upon system configuration.
- *job* refers to a set of one or more *applications* executed as a single invocation by the user within a session. For example, "*mpiexec -n 1 app1 : -n 2 app2*" is considered a single Multiple Program Multiple Data (MPMD) job containing two applications.
- *namespace* refers to a character string value assigned by the RM to a *job*. All *applications* executed as part of that *job* share the same *namespace*. The *namespace* assigned to each *job* must be unique within the scope of the governing RM.
- *application* refers to a single executable (binary, script, etc.) member of a *job*. Applications consist of one or more *processes*, either operating independently or in parallel at any given time during their execution.
- *rank* refers to the numerical location (starting from zero) of a process within the defined scope. Thus, global rank is the rank of a process within its *job*, while *application rank* is the rank of that process within its *application*.
- *workflow* refers to an orchestrated execution plan frequently spanning multiple *jobs* carried out under the control of a *workflow manager* process. An example workflow might first execute a computational job to generate the flow of liquid through a complex cavity, followed by a visualization job that takes the output of the first job as its input to produce an image output.

- *resource manager* is used in a generic sense to represent the system that will host the PMIx server library. This could be a vendor's RM, a programming library's RunTime Environment (RTE), or some other agent.
- *host environment* is used interchangeably with *resource manager* to refer to the process hosting the PMIx server library.

This document borrows freely from other standards (most notably from the Message Passing Interface (MPI) and OpenMP standards) in its use of notation and conventions in an attempt to reduce confusion. The following sections provide an overview of the conventions used throughout the PMIx Standard document.

2.1 Notational Conventions

Some sections of this document describe programming language specific examples or APIs. Text that applies only to programs for which the base language is C is shown as follows:

C

C specific text...

```
int foo = 42;
```

C

Some text is for information only, and is not part of the normative specification. These take several forms, described in their examples below:

Note: General text...

Rationale

Throughout this document, the rationale for the design choices made in the interface specification is set off in this section. Some readers may wish to skip these sections, while readers interested in interface design may want to read them carefully.

Advice to users

Throughout this document, material aimed at users and that illustrates usage is set off in this section. Some readers may wish to skip these sections, while readers interested in programming with the PMIx API may want to read them carefully.

Advice to PMIx library implementers

Throughout this document, material that is primarily commentary to PMIx library implementers is set off in this section. Some readers may wish to skip these sections, while readers interested in PMIx implementations may want to read them carefully.

Advice to PMIx server hosts

Throughout this document, material that is primarily commentary aimed at host environments (e.g., RMs and RTEs) providing support for the PMIx server library is set off in this section. Some readers may wish to skip these sections, while readers interested in integrating PMIx servers into their environment may want to read them carefully.

2.2 Semantics

The following terms will be taken to mean:

- *shall* and *will* indicate that the specified behavior is *required* of all conforming implementations
- *should* and *may* indicate behaviors that a quality implementation would include, but are not required of all conforming implementations

2.3 Naming Conventions

The PMIx standard has adopted the following conventions:

- PMIx constants and attributes are prefixed with **`PMIX`**.
- Structures and type definitions are prefixed with **`pmix`**.
- Underscores are used to separate words in a function or variable name.
- Lowercase letters are used in PMIx client APIs except for the PMIx prefix (noted below) and the first letter of the word following it. For example, **`PMIx_Get_version`** .
- PMIx server and tool APIs are all lower case letters following the prefix - e.g., **`PMIx_server_register_nspace`** .
- The **`PMIx_`** prefix is used to denote functions.
- The **`pmix_`** prefix is used to denote function pointer and type definitions.

Users should not use the **`PMIX`**, **`PMIx`**, or **`pmix`** prefixes in their applications or libraries so as to avoid symbol conflicts with current and later versions of the PMIx standard and implementations such as the PRI.

2.4 Procedure Conventions

While the current PMIx Reference Implementation (PRI) is solely based on the C programming language, it is not the intent of the PMIx Standard to preclude the use of other languages. Accordingly, the procedure specifications in the PMIx Standard are written in a language-independent syntax with the arguments marked as IN, OUT, or INOUT. The meanings of these are:

- IN: The call may use the input value but does not update the argument from the perspective of the caller at any time during the calls execution,
- OUT: The call may update the argument but does not use its input value
- INOUT: The call may both use and update the argument.

2.5 Standard vs Reference Implementation

The *PMIx Standard* is implementation independent. The *PMIx Reference Implementation* (PRI) is one implementation of the Standard and the PMIx community strives to ensure that it fully implements the Standard. Given its role as the community's testbed and its widespread use, this document cites the attributes supported by the PRI for each API where relevant by marking them in red. This is not meant to imply nor confer any special role to the PRI with respect to the Standard itself, but instead to provide a convenience to users of the Standard and PRI.

Similarly, the *PMIx Reference RunTime Environment* (PRRTE) is provided by the community to enable users operating in non-PMIx environments to develop and execute PMIx-enabled applications and tools. Attributes supported by the PRRTE are marked in green.

CHAPTER 3

Data Structures and Types

This chapter defines PMIx standard data structures, types, and constants. These apply to all consumers of the PMIx interface. Where necessary for clarification, the description of, for example, an attribute may be copied from this chapter into a section where it is used.

A PMIx implementation may define additional attributes beyond those specified in this document.

Advice to PMIx library implementers

Structures, types, and macros in the PMIx Standard are defined in terms of the C-programming language. Implementers wishing to support other languages should provide the equivalent definitions in a language-appropriate manner.

If a PMIx implementation chooses to define additional attributes they should avoid using the **PMIX** prefix in their name or starting the attribute string with a *pmix* prefix. This helps the end user distinguish between what is defined by the PMIx standard and what is specific to that PMIx implementation, and avoids potential conflicts with attributes defined by the standard.

3.1 Constants

PMIx defines a few values that are used throughout the standard to set the size of fixed arrays or as a means of identifying values with special meaning. The community makes every attempt to minimize the number of such definitions. The constants defined in this section may be used before calling any PMIx library initialization routine. Additional constants associated with specific data structures or types are defined in the section describing that data structure or type.

PMIX_MAX_NSLEN Maximum namespace string length as an integer.

Advice to PMIx library implementers

PMIX_MAX_NSLEN should have a minimum value of 63 characters. Namespace arrays in PMIx defined structures must reserve a space of size **PMIX_MAX_NSLEN** +1 to allow room for the **NULL** terminator

PMIX_MAX_KEYLEN Maximum key string length as an integer.

Advice to PMIx library implementers

PMIX_MAX_KEYLEN should have a minimum value of 63 characters. Key arrays in PMIx defined structures must reserve a space of size **PMIX_MAX_KEYLEN** +1 to allow room for the **NULL** terminator

3.1.1 Error Constants

The **pmix_status_t** structure is an **int** type for return status.

The tables shown in this section define the possible values for **pmix_status_t** . PMIx errors are required to always be negative, with 0 reserved for **PMIX_SUCCESS** .

A PMIx implementation must define all of the constants defined in this section, even if they will never return the specific value to the caller.

Advice to users

Other than **PMIX_SUCCESS** (which is required to be zero), the actual value of any PMIx error constant is left to the PMIx library implementer. Thus, users are advised to always refer to constant by name, and not a specific implementation's value, for portability between implementations and compatibility across library versions.

3.1.1.1 PMIx v1 Error Constants

The following list contains those constants defined in the PMIx v1 standard. Those values in the list that were deprecated in later standards are denoted as such. PMIx errors are always negative, with **0** reserved for success.

PMIX_SUCCESS	Success
PMIX_ERROR	General Error
PMIX_ERR_SILENT	Silent error
PMIX_ERR_DEBUGGER_RELEASE	Error in debugger release
PMIX_ERR_PROC_RESTART	Fault tolerance: Error in process restart
PMIX_ERR_PROC_CHECKPOINT	Fault tolerance: Error in process checkpoint
PMIX_ERR_PROC_MIGRATE	Fault tolerance: Error in process migration
PMIX_ERR_PROC_ABORTED	Process was aborted
PMIX_ERR_PROC_REQUESTED_ABORT	Process is already requested to abort
PMIX_ERR_PROC_ABORTING	Process is being aborted
PMIX_ERR_SERVER_FAILED_REQUEST	Failed to connect to the server
PMIX_EXISTS	Requested operation would overwrite an existing value
PMIX_ERR_INVALID_CRED	Invalid security credentials
PMIX_ERR_HANDSHAKE_FAILED	Connection handshake failed
PMIX_ERR_READY_FOR_HANDSHAKE	Ready for handshake
PMIX_ERR_WOULD_BLOCK	Operation would block
PMIX_ERR_UNKNOWN_DATA_TYPE	Unknown data type
PMIX_ERR_PROC_ENTRY_NOT_FOUND	Process not found
PMIX_ERR_TYPE_MISMATCH	Invalid type
PMIX_ERR_UNPACK_INADEQUATE_SPACE	Inadequate space to unpack data
PMIX_ERR_UNPACK_FAILURE	Unpack failed
PMIX_ERR_PACK_FAILURE	Pack failed
PMIX_ERR_PACK_MISMATCH	Pack mismatch
PMIX_ERR_NO_PERMISSIONS	No permissions
PMIX_ERR_TIMEOUT	Timeout expired
PMIX_ERR_UNREACH	Unreachable
PMIX_ERR_IN_ERRNO	Error defined in **errno**
PMIX_ERR_BAD_PARAM	Bad parameter
PMIX_ERR_RESOURCE_BUSY	Resource busy
PMIX_ERR_OUT_OF_RESOURCE	Resource exhausted
PMIX_ERR_DATA_VALUE_NOT_FOUND	Data value not found
PMIX_ERR_INIT	Error during initialization
PMIX_ERR_NOMEM	Out of memory
PMIX_ERR_INVALID_ARG	Invalid argument
PMIX_ERR_INVALID_KEY	Invalid key
PMIX_ERR_INVALID_KEY_LENGTH	Invalid key length
PMIX_ERR_INVALID_VAL	Invalid value

PMIX_ERR_INVALID_VAL_LENGTH Invalid value length
PMIX_ERR_INVALID_LENGTH Invalid argument length
PMIX_ERR_INVALID_NUM_ARGS Invalid number of arguments
PMIX_ERR_INVALID_ARGS Invalid arguments
PMIX_ERR_INVALID_NUM_PARSED Invalid number parsed
PMIX_ERR_INVALID_KEYVALP Invalid key/value pair
PMIX_ERR_INVALID_SIZE Invalid size
PMIX_ERR_INVALID_NAMESPACE Invalid namespace
PMIX_ERR_SERVER_NOT_AVAIL Server is not available
PMIX_ERR_NOT_FOUND Not found
PMIX_ERR_NOT_SUPPORTED Not supported
PMIX_ERR_NOT_IMPLEMENTED Not implemented
PMIX_ERR_COMM_FAILURE Communication failure
PMIX_ERR_UNPACK_READ_PAST_END_OF_BUFFER Unpacking past the end of the buffer provided

3.1.1.2 PMIx v2 Error Constants

The following list contains constants added in the PMIx v2 standard.

PMIX_ERR_LOST_CONNECTION_TO_SERVER Lost connection to server
PMIX_ERR_LOST_PEER_CONNECTION Lost connection to peer
PMIX_ERR_LOST_CONNECTION_TO_CLIENT Lost connection to client
PMIX_QUERY_PARTIAL_SUCCESS Query partial success (used by query system)
PMIX_NOTIFY_ALLOC_COMPLETE Notify that allocation is complete
PMIX_JCTRL_CHECKPOINT Job control: Monitored by PMIx client to trigger checkpoint operation
PMIX_JCTRL_CHECKPOINT_COMPLETE Job control: Sent by PMIx client and monitored by PMIx server to notify that requested checkpoint operation has completed.
PMIX_JCTRL_PREEMPT_ALERT Job control: Monitored by PMIx client to detect an RM intending to preempt the job.
PMIX_MONITOR_HEARTBEAT_ALERT Job monitoring: Heartbeat alert
PMIX_MONITOR_FILE_ALERT Job monitoring: File alert
PMIX_PROC_TERMINATED Process terminated - can be either normal or abnormal termination
PMIX_ERR_INVALID_TERMINATION Process terminated without calling **PMIx_Finalize**, or was a member of an assemblage formed via **PMIx_Connect** and terminated or called **PMIx_Finalize** without first calling **PMIx_Disconnect** (or its non-blocking form) from that assemblage.

The following list contains operational error constants introduced in the v2 standard.

PMIX_ERR_EVENT_REGISTRATION Error in event registration
PMIX_ERR_JOB_TERMINATED Error job terminated
PMIX_ERR_UPDATE_ENDPOINTS Error updating endpoints

PMIX_MODEL_DECLARED Model declared

PMIX_GDS_ACTION_COMPLETE The global data storage (GDS) action has completed

PMIX_ERR_INVALID_OPERATION The requested operation is supported by the implementation and host environment, but fails to meet a requirement (e.g., requesting to *disconnect* from processes without first *connecting* to them).

The following list contains system error constants introduced in the v2 standard.

PMIX_ERR_NODE_DOWN Node down

PMIX_ERR_NODE_OFFLINE Node is marked as offline

The following list contains event handler error constants introduced in the v2 standard.

PMIX_EVENT_NO_ACTION_TAKEN Event handler: No action taken

PMIX_EVENT_PARTIAL_ACTION_TAKEN Event handler: Partial action taken

PMIX_EVENT_ACTION_DEFERRED Event handler: Action deferred

PMIX_EVENT_ACTION_COMPLETE Event handler: Action complete

3.1.1.3 User-Defined Error Constants

PMIx establishes an error code boundary for constants defined in the PMIx standard. Negative values larger than this (and any positive values greater than zero) are guaranteed not to conflict with PMIx values.

PMIX_EXTERNAL_ERR_BASE A starting point for user-level defined error constants. Negative values lower than this are guaranteed not to conflict with PMIx values. Definitions should always be based on the **PMIX_EXTERNAL_ERR_BASE** constant and *not* a specific value as the value of the constant may change.

3.2 Data Types

This section defines various data types used by the PMIx APIs.

3.2.1 Key Structure

The **pmix_key_t** structure is a statically defined character array of length **PMIX_MAX_KEYLEN** +1, thus supporting keys of maximum length **PMIX_MAX_KEYLEN** while preserving space for a mandatory **NULL** terminator.

PMIx v2.0 C

```
typedef char pmix_key_t[PMIX_MAX_KEYLEN+1];
```

C

Characters in the key must be standard alphanumeric values supported by common utilities such as *strcmp*.

Advice to users

References to keys in PMIx v1 rwere defined simply as an array of characters of size **PMIX_MAX_KEYLEN+1**. The **pmix_key_t** type definition was introduced in version 2 of the standard. The two definitions are code-compatible and thus do not represent a break in backward compatibility.

Passing a **pmix_key_t** value to the standard *sizeof* utility can result in compiler warnings of incorrect returned value. Users are advised to avoid using *sizeof(pmix_key_t)* and instead rely on the **PMIX_MAX_KEYLEN** constant.

3.2.2 Namespace Structure

The **pmix_nspace_t** structure is a statically defined character array of length **PMIX_MAX_NSLEN** +1, thus supporting namespaces of maximum length **PMIX_MAX_NSLEN** while preserving space for a mandatory **NULL** terminator.

PMIx v2.0

C

```
typedef char pmix_nspace_t[PMIX_MAX_NSLEN+1];
```

C

Characters in the namespace must be standard alphanumeric values supported by common utilities such as *strcmp*.

Advice to users

References to namespace values in PMIx v1 rwere defined simply as an array of characters of size **PMIX_MAX_NSLEN+1**. The **pmix_nspace_t** type definition was introduced in version 2 of the standard. The two definitions are code-compatible and thus do not represent a break in backward compatibility.

Passing a **pmix_nspace_t** value to the standard *sizeof* utility can result in compiler warnings of incorrect returned value. Users are advised to avoid using *sizeof(pmix_nspace_t)* and instead rely on the **PMIX_MAX_NSLEN** constant.

3.2.3 Rank Structure

The **pmix_rank_t** structure is a **uint32_t** type for rank values.

PMIx v1.0

C

```c
typedef uint32_t pmix_rank_t;
```

C

The following constants can be used to set a variable of the type **pmix_rank_t** . All definitions were introduced in version 1 of the standard unless otherwise marked. Valid rank values start at zero.

PMIX_RANK_UNDEF A value to request job-level data where the information itself is not associated with any specific rank, or when passing a **pmix_proc_t** identifier to an operation that only references the namespace field of that structure.

PMIX_RANK_WILDCARD A value to indicate that the user wants the data for the given key from every rank that posted that key.

PMIx v2.0 **PMIX_RANK_LOCAL_NODE** Special rank value used to define groups of ranks for use in collectives. This constant defines the group of all ranks on a local node.

3.2.4 Process Structure

The **pmix_proc_t** structure is used to identify a single process in the PMIx universe. It contains a reference to the namespace and the **pmix_rank_t** within that namespace.

PMIx v1.0

C

```c
typedef struct pmix_proc {
    pmix_nspace_t nspace;
    pmix_rank_t rank;
} pmix_proc_t;
```

C

3.2.5 Process structure support macros

The following macros are provided to support the **pmix_proc_t** structure.

3.2.5.1 Initialize the `pmix_proc_t` structure

Initialize the `pmix_proc_t` fields

PMIx v1.0

C

```c
PMIX_PROC_CONSTRUCT(m)
```

C

IN `m`
Pointer to the structure to be initialized (pointer to `pmix_proc_t`)

3.2.5.2 Destruct the `pmix_proc_t` structure

Clear the `pmix_proc_t` fields

PMIx v1.0

C

```c
PMIX_PROC_DESTRUCT(m)
```

C

IN `m`
Pointer to the structure to be destructed (pointer to `pmix_proc_t`)

This macro performs the identical operations as `PMIX_PROC_CONSTRUCT` , but is provided for symmetry in user code.

3.2.5.3 Create a `pmix_proc_t` array

Allocate and initialize an array of `pmix_proc_t` structures

PMIx v1.0

C

```c
PMIX_PROC_CREATE(m, n)
```

C

INOUT `m`
Address where the pointer to the array of `pmix_proc_t` structures shall be stored (handle)

IN `n`
Number of structures to be allocated (`size_t`)

3.2.5.4 Free a pmix_proc_t array

Release an array of pmix_proc_t structures

PMIx v1.0

C

```
PMIX_PROC_FREE(m, n)
```

C

IN `m`
Pointer to the array of pmix_proc_t structures (handle)

IN `n`
Number of structures in the array (`size_t`)

3.2.5.5 Load a pmix_proc_t structure

Load values into a pmix_proc_t

PMIx v2.0

C

```
PMIX_PROC_LOAD(m, n, r)
```

C

IN `m`
Pointer to the structure to be loaded (pointer to pmix_proc_t)

IN `n`
Namespace to be loaded (pmix_nspace_t)

IN `r`
Rank to be assigned (pmix_rank_t)

3.2.6 Process State Structure

PMIx v2.0 The pmix_proc_state_t structure is a `uint8_t` type for process state values. The following constants can be used to set a variable of the type pmix_proc_state_t . All values were originally defined in version 2 of the standard unless otherwise marked.

Advice to users

The fine-grained nature of the following constants may exceed the ability of an RM to provide updated process state values during the process lifetime. This is particularly true of states in the launch process, and for short-lived processes.

PMIX_PROC_STATE_UNDEF Undefined process state

PMIX_PROC_STATE_PREPPED Process is ready to be launched

PMIX_PROC_STATE_LAUNCH_UNDERWAY Process launch is underway

PMIX_PROC_STATE_RESTART Process is ready for restart

PMIX_PROC_STATE_TERMINATE Process is marked for termination

PMIX_PROC_STATE_RUNNING Process has been locally **fork**'ed by the RM

PMIX_PROC_STATE_CONNECTED Process has connected to PMIx server

PMIX_PROC_STATE_UNTERMINATED Define a "boundary" between this constant and **PMIX_PROC_STATE_CONNECTED** so users can easily and quickly determine if a process is still running or not. Any value less than this constant means that the process has not terminated.

PMIX_PROC_STATE_TERMINATED Process has terminated and is no longer running

PMIX_PROC_STATE_ERROR Define a boundary so users can easily and quickly determine if a process abnormally terminated. Any value above this constant means that the process has terminated abnormally.

PMIX_PROC_STATE_KILLED_BY_CMD Process was killed by a command

PMIX_PROC_STATE_ABORTED Process was aborted by a call to **PMIx_Abort**

PMIX_PROC_STATE_FAILED_TO_START Process failed to start

PMIX_PROC_STATE_ABORTED_BY_SIG Process aborted by a signal

PMIX_PROC_STATE_TERM_WO_SYNC Process exited without calling **PMIx_Finalize**

PMIX_PROC_STATE_COMM_FAILED Process communication has failed

PMIX_PROC_STATE_CALLED_ABORT Process called **PMIx_Abort**

PMIX_PROC_STATE_MIGRATING Process failed and is waiting for resources before restarting

PMIX_PROC_STATE_CANNOT_RESTART Process failed and cannot be restarted

PMIX_PROC_STATE_TERM_NON_ZERO Process exited with a non-zero status

PMIX_PROC_STATE_FAILED_TO_LAUNCH Unable to launch process

3.2.7 Process Information Structure

The **pmix_proc_info_t** structure defines a set of information about a specific process including it's name, location, and state.

PMIx v2.0

C

```c
typedef struct pmix_proc_info {
    /** Process structure */
    pmix_proc_t proc;
    /** Hostname where process resides */
    char *hostname;
    /** Name of the executable */
    char *executable_name;
    /** Process ID on the host */
    pid_t pid;
    /** Exit code of the process. Default: 0 */
    int exit_code;
    /** Current state of the process */
    pmix_proc_state_t state;
} pmix_proc_info_t;
```

C

3.2.8 Process Information Structure support macros

The following macros are provided to support the **pmix_proc_info_t** structure.

3.2.8.1 Initialize the pmix_proc_info_t structure

Initialize the **pmix_proc_info_t** fields

PMIx v2.0

C

```c
PMIX_PROC_INFO_CONSTRUCT(m)
```

C

IN **m**
Pointer to the structure to be initialized (pointer to **pmix_proc_info_t**)

3.2.8.2 Destruct the pmix_proc_info_t structure

Destruct the **pmix_proc_info_t** fields

PMIx v2.0

C

```c
PMIX_PROC_INFO_DESTRUCT(m)
```

C

IN **m**
Pointer to the structure to be destructed (pointer to **pmix_proc_info_t**)

3.2.8.3 Create a pmix_proc_info_t array

Allocate and initialize a **pmix_proc_info_t** array

PMIx v2.0

C

```
PMIX_PROC_INFO_CREATE(m, n)
```

C

INOUT m
: Address where the pointer to the array of **pmix_proc_info_t** structures shall be stored (handle)

IN n
: Number of structures to be allocated (**size_t**)

3.2.8.4 Free a pmix_proc_info_t array

Release an array of **pmix_proc_info_t** structures

PMIx v2.0

C

```
PMIX_PROC_INFO_FREE(m, n)
```

C

IN m
: Pointer to the array of **pmix_proc_info_t** structures (handle)

IN n
: Number of structures in the array (**size_t**)

3.2.9 Scope of Put Data

PMIx v1.0 The **pmix_scope_t** structure is a **uint8_t** type that defines the scope for data passed to **PMIx_Put**. The following constants can be used to set a variable of the type **pmix_scope_t**. All definitions were introduced in version 1 of the standard unless otherwise marked.

Specific implementations may support different scope values, but all implementations must support at least **PMIX_GLOBAL**. If a scope value is not supported, then the **PMIx_Put** call must return **PMIX_ERR_NOT_SUPPORTED**.

PMIX_SCOPE_UNDEF Undefined scope

PMIX_LOCAL The data is intended only for other application processes on the same node. Data marked in this way will not be included in data packages sent to remote requestors — i.e., it is only available to processes on the local node.

PMIX_REMOTE The data is intended solely for applications processes on remote nodes. Data marked in this way will not be shared with other processes on the same node — i.e., it is only available to processes on remote nodes.

PMIX_GLOBAL The data is to be shared with all other requesting processes, regardless of location.

PMIx v2.0 **PMIX_INTERNAL** The data is intended solely for this process and is not shared with other processes.

3.2.10 Range of Published Data

PMIx v1.0 The **pmix_data_range_t** structure is a **uint8_t** type that defines a range for data *published* via functions other than **PMIx_Put** - e.g., the **PMIx_Publish** API. The following constants can be used to set a variable of the type **pmix_data_range_t** . Several values were initially defined in version 1 of the standard but subsequently renamed and other values added in version 2. Thus, all values shown below are as they were defined in version 2 except where noted.

PMIX_RANGE_UNDEF Undefined range
PMIX_RANGE_RM Data is intended for the host resource manager.
PMIX_RANGE_LOCAL Data is only available to processes on the local node.
PMIX_RANGE_NAMESPACE Data is only available to processes in the same namespace.
PMIX_RANGE_SESSION Data is only available to all processes in the session.
PMIX_RANGE_GLOBAL Data is available to all processes.
PMIX_RANGE_CUSTOM Range is specified in the **pmix_info_t** associated with this call.
PMIX_RANGE_PROC_LOCAL Data is only available to this process.

Advice to users

The names of the **pmix_data_range_t** values changed between version 1 and version 2 of the standard, thereby breaking backward compatibility

3.2.11 Data Persistence Structure

PMIx v1.0 The **pmix_persistence_t** structure is a **uint8_t** type that defines the policy for data published by clients via the **PMIx_Publish** API. The following constants can be used to set a variable of the type **pmix_persistence_t** . All definitions were introduced in version 1 of the standard unless otherwise marked.

PMIX_PERSIST_INDEF Retain data until specifically deleted.
PMIX_PERSIST_FIRST_READ Retain data until the first access, then the data is deleted.
PMIX_PERSIST_PROC Retain data until the publishing process terminates.
PMIX_PERSIST_APP Retain data until the application terminates.
PMIX_PERSIST_SESSION Retain data until the session/allocation terminates.

3.2.12 Value Structure

The **pmix_value_t** structure is used to represent the value passed to **PMIx_Put** and retrieved by **PMIx_Get** , as well as many of the other PMIx functions.

A collection of values may be specified under a single key by passing a **pmix_value_t** containing an array of type **pmix_data_array_t** , with each array element containing its own object. All members shown below were introduced in version 1 of the standard unless otherwise marked.

PMIx v1.0 C

```
typedef struct pmix_value {
    pmix_data_type_t type;
    union {
        bool flag;
        uint8_t byte;
        char *string;
        size_t size;
        pid_t pid;
        int integer;
        int8_t int8;
        int16_t int16;
        int32_t int32;
        int64_t int64;
        unsigned int uint;
        uint8_t uint8;
        uint16_t uint16;
        uint32_t uint32;
        uint64_t uint64;
        float fval;
        double dval;
        struct timeval tv;
        time_t time;                    // version 2.0
        pmix_status_t status;           // version 2.0
        pmix_rank_t rank;               // version 2.0
        pmix_proc_t *proc;              // version 2.0
        pmix_byte_object_t bo;
        pmix_persistence_t persist;     // version 2.0
        pmix_scope_t scope;             // version 2.0
        pmix_data_range_t range;        // version 2.0
        pmix_proc_state_t state;        // version 2.0
        pmix_proc_info_t *pinfo;        // version 2.0
        pmix_data_array_t *darray;      // version 2.0
        void *ptr;                      // version 2.0
```

```
            pmix_alloc_directive_t adir;    // version 2.0
            /**** DEPRECATED in PMIx 2 ****/
            pmix_info_array_t *array;
            /****************************/
        } data;
    } pmix_value_t;
```

C

3.2.13 Value structure support macros

The following macros are provided to support the **pmix_value_t** structure.

3.2.13.1 Initialize the pmix_value_t structure

Initialize the **pmix_value_t** fields

PMIx v1.0

C

```
PMIX_VALUE_CONSTRUCT(m)
```

C

IN **m**
Pointer to the structure to be initialized (pointer to **pmix_value_t**)

3.2.13.2 Destruct the pmix_value_t structure

Destruct the **pmix_value_t** fields

PMIx v1.0

C

```
PMIX_VALUE_DESTRUCT(m)
```

C

IN **m**
Pointer to the structure to be destructed (pointer to **pmix_value_t**)

3.2.13.3 Create a pmix_value_t array

Allocate and initialize an array of **pmix_value_t** structures

PMIx v1.0

C

```
PMIX_VALUE_CREATE(m, n)
```

C

INOUT **m**
Address where the pointer to the array of **pmix_value_t** structures shall be stored (handle)

IN **n**
Number of structures to be allocated (**size_t**)

3.2.13.4 Free a `pmix_value_t` array

Release an array of `pmix_value_t` structures

PMIx v1.0

C

```
PMIX_VALUE_FREE(m, n)
```

C

IN `m`
Pointer to the array of `pmix_value_t` structures (handle)

IN `n`
Number of structures in the array (`size_t`)

3.2.14 Load a `pmix_value_t` structure

Summary

Load data into a `pmix_value_t` structure.

PMIx v2.0

C

```
PMIX_VALUE_LOAD(v, d, t);
```

C

IN `v`
The `pmix_value_t` into which the data is to be loaded (pointer to `pmix_value_t`)

IN `d`
Pointer to the data value to be loaded (handle)

IN `t`
Type of the provided data value (`pmix_data_type_t`)

Description

This macro simplifies the loading of data into a `pmix_value_t` by correctly assigning values to the structure's fields.

Advice to users

The data will be copied into the `pmix_value_t` - thus, any data stored in the source value can be modified or free'd without affecting the copied data once the macro has completed.

3.2.14.1 Transfer data between `pmix_value_t` structures

Summary

Transfer the data value between two `pmix_value_t` structures.

PMIx v2.0 C

```c
PMIX_VALUE_XFER(r, d, s);
```

C

OUT `r`
Status code indicating success or failure of the transfer (`pmix_status_t`)

IN `d`
Pointer to the `pmix_value_t` destination (handle)

IN `s`
Pointer to the `pmix_value_t` source (handle)

Description

This macro simplifies the transfer of data between two `pmix_value_t` structures, ensuring that all fields are properly copied.

Advice to users

The data will be copied into the destination `pmix_value_t` - thus, any data stored in the source value can be modified or free'd without affecting the copied data once the macro has completed.

3.2.15 Info and Info Array Structures

The `pmix_info_t` structure defines a key/value pair with associated directive. All fields were defined in version 1.0 unless otherwise marked.

PMIx v1.0 C

```c
typedef struct pmix_info_t {
    pmix_key_t key;
    pmix_info_directives_t flags;    // version 2.0
    pmix_value_t value;
} pmix_info_t;
```

C

The `pmix_info_array` structure defines an array of `pmix_info_t` structures.

Note: The **pmix_info_array** structure has been deprecated and will be removed in future versions of the PMIx Standard.

PMIx v1.0

```c
typedef struct pmix_info_array {
    size_t size;
    pmix_info_t *array;
} pmix_info_array_t;
```

3.2.16 Info structure support macros

The following macros are provided to support the **pmix_info_t** structure.

3.2.16.1 Initialize the pmix_info_t structure

Initialize the **pmix_info_t** fields

PMIx v1.0

```c
PMIX_INFO_CONSTRUCT(m)
```

IN **m**
Pointer to the structure to be initialized (pointer to **pmix_info_t**)

3.2.16.2 Destruct the pmix_info_t structure

Destruct the **pmix_info_t** fields

PMIx v1.0

```c
PMIX_INFO_DESTRUCT(m)
```

IN **m**
Pointer to the structure to be destructed (pointer to **pmix_info_t**)

3.2.16.3 Create a pmix_info_t array

Allocate and initialize an array of **pmix_info_t** structures

PMIx v1.0

C

```
PMIX_INFO_CREATE(m, n)
```

C

INOUT m
    Address where the pointer to the array of **pmix_info_t** structures shall be stored (handle)

IN n
    Number of structures to be allocated (**size_t**)

3.2.16.4 Free a pmix_info_t array

Release an array of **pmix_info_t** structures

PMIx v1.0

C

```
PMIX_INFO_FREE(m, n)
```

C

IN m
    Pointer to the array of **pmix_info_t** structures (handle)

IN n
    Number of structures in the array (**size_t**)

3.2.16.5 Load key and value data into a pmix_info_t

PMIx v1.0

C

```
PMIX_INFO_LOAD(v, k, d, t);
```

C

IN v
    Pointer to the **pmix_info_t** into which the key and data are to be loaded (pointer to **pmix_info_t**)

IN k
    String key to be loaded - must be less than or equal to **PMIX_MAX_KEYLEN** in length (handle)

IN d
    Pointer to the data value to be loaded (handle)

IN t
    Type of the provided data value (**pmix_data_type_t**)

This macro simplifies the loading of key and data into a **pmix_info_t** by correctly assigning values to the structure's fields.

Advice to users

Both key and data will be copied into the **pmix_info_t** - thus, the key and any data stored in the source value can be modified or free'd without affecting the copied data once the macro has completed.

3.2.16.6 Copy data between `pmix_info_t` structures

Copy all data (including key, value, and directives) between two **pmix_info_t** structures.

PMIx v2.0

C

```
PMIX_INFO_XFER(d, s);
```

C

IN **d**
  Pointer to the destination **pmix_info_t** (pointer to **pmix_info_t**)

IN **s**
  Pointer to the source **pmix_info_t** (pointer to **pmix_info_t**)

This macro simplifies the transfer of data between two **pmix_info_t** structures.

Advice to users

All data (including key, value, and directives) will be copied into the destination **pmix_info_t** - thus, the source **pmix_info_t** may be free'd without affecting the copied data once the macro has completed.

3.2.16.7 Test a boolean `pmix_info_t`

A special macro for checking if a boolean **pmix_info_t** is **true**

PMIx v2.0

C

```
PMIX_INFO_TRUE(m)
```

C

IN **m**
  Pointer to a **pmix_info_t** structure (handle)

A **pmix_info_t** structure is considered to be of type **PMIX_BOOL** and value **true** if:

- the structure reports a type of **PMIX_UNDEF** , or
- the structure reports a type of **PMIX_BOOL** and the data flag is **true**

3.2.17 Info Type Directives

PMIx v2.0 The **pmix_info_directives_t** structure is a **uint32_t** type that defines the behavior of command directives via **pmix_info_t** arrays. By default, the values in the **pmix_info_t** array passed to a PMIx are *optional*.

Advice to users

A PMIx implementation or PMIx-enabled RM may ignore any **pmix_info_t** value passed to a PMIx API if it is not explicitly marked as **PMIX_INFO_REQD** . This is because the values specified default to optional, meaning they can be ignored. This may lead to unexpected behavior if the user is relying on the behavior specified by the **pmix_info_t** value. If the user relies on the behavior defined by the **pmix_info_t** then they must set the **PMIX_INFO_REQD** flag using the **PMIX_INFO_REQUIRED** macro.

Advice to PMIx library implementers

The top 16-bits of the **pmix_info_directives_t** are reserved for internal use by PMIx library implementers - the PMIx standard will *not* specify their intent, leaving them for customized use by implementers. Implementers are advised to use the provided **PMIX_INFO_IS_REQUIRED** macro for testing this flag, and must return **PMIX_ERR_NOT_SUPPORTED** as soon as possible to the caller if the required behavior is not supported.

The following constants were introduced in version 2.0 (unless otherwise marked) and can be used to set a variable of the type **pmix_info_directives_t** .

PMIX_INFO_REQD The behavior defined in the **pmix_info_t** array is required, and not optional. This is a bit-mask value.

Advice to PMIx server hosts

Host environments are advised to use the provided **PMIX_INFO_IS_REQUIRED** macro for testing this flag and must return **PMIX_ERR_NOT_SUPPORTED** as soon as possible to the caller if the required behavior is not supported.

3.2.18 Info Directive support macros

The following macros are provided to support the setting and testing of **pmix_info_t** directives.

3.2.18.1 Mark an info structure as required

Summary

Set the **PMIX_INFO_REQD** flag in a **pmix_info_t** structure.

PMIx v2.0

C

```c
PMIX_INFO_REQUIRED(info);
```

C

IN `info`
Pointer to the **pmix_info_t** (pointer to **pmix_info_t**)

This macro simplifies the setting of the **PMIX_INFO_REQD** flag in **pmix_info_t** structures.

3.2.18.2 Test an info structure for *required* directive

Summary

Test the **PMIX_INFO_REQD** flag in a **pmix_info_t** structure, returning **true** if the flag is set.

PMIx v2.0

C

```c
PMIX_INFO_IS_REQUIRED(info);
```

C

IN `info`
Pointer to the **pmix_info_t** (pointer to **pmix_info_t**)

This macro simplifies the testing of the required flag in **pmix_info_t** structures.

3.2.19 Job Allocation Directives

PMIx v2.0 The **pmix_alloc_directive_t** structure is a **uint8_t** type that defines the behavior of allocation requests. The following constants can be used to set a variable of the type **pmix_alloc_directive_t** . All definitions were introduced in version 2 of the standard unless otherwise marked.

PMIX_ALLOC_NEW A new allocation is being requested. The resulting allocation will be disjoint (i.e., not connected in a job sense) from the requesting allocation.

PMIX_ALLOC_EXTEND Extend the existing allocation, either in time or as additional resources.

PMIX_ALLOC_RELEASE Release part of the existing allocation. Attributes in the accompanying **pmix_info_t** array may be used to specify permanent release of the identified resources, or "lending" of those resources for some period of time.

PMIX_ALLOC_REAQUIRE Reacquire resources that were previously "lent" back to the scheduler.

PMIX_ALLOC_EXTERNAL A value boundary above which implementers are free to define their own directive values.

3.2.20 Lookup Returned Data Structure

The **pmix_pdata_t** structure is used by **PMIx_Lookup** to describe the data being accessed.

PMIx v1.0 C

```c
typedef struct pmix_pdata {
    pmix_proc_t proc;
    pmix_key_t key;
    pmix_value_t value;
} pmix_pdata_t;
```

C

3.2.21 Lookup data structure support macros

The following macros are provided to support the **pmix_pdata_t** structure.

3.2.21.1 Initialize the pmix_pdata_t structure

Initialize the **pmix_pdata_t** fields

PMIx v1.0 C

```c
PMIX_PDATA_CONSTRUCT(m)
```

C

IN **m**
Pointer to the structure to be initialized (pointer to **pmix_pdata_t**)

3.2.21.2 Destruct the pmix_pdata_t structure

Destruct the **pmix_pdata_t** fields

PMIx v1.0 C

```c
PMIX_PDATA_DESTRUCT(m)
```

C

IN **m**
Pointer to the structure to be destructed (pointer to **pmix_pdata_t**)

3.2.21.3 Create a `pmix_pdata_t` array

Allocate and initialize an array of `pmix_pdata_t` structures

PMIx v1.0

C

```c
PMIX_PDATA_CREATE(m, n)
```

C

INOUT `m`
: Address where the pointer to the array of `pmix_pdata_t` structures shall be stored (handle)

IN `n`
: Number of structures to be allocated (`size_t`)

3.2.21.4 Free a `pmix_pdata_t` array

Release an array of `pmix_pdata_t` structures

PMIx v1.0

C

```c
PMIX_PDATA_FREE(m, n)
```

C

IN `m`
: Pointer to the array of `pmix_pdata_t` structures (handle)

IN `n`
: Number of structures in the array (`size_t`)

3.2.21.5 Load a lookup data structure

Summary

Load key, process identifier, and data value into a `pmix_pdata_t` structure.

PMIx v1.0

C

```c
PMIX_PDATA_LOAD(m, p, k, d, t);
```

C

IN `m`
: Pointer to the `pmix_pdata_t` structure into which the key and data are to be loaded (pointer to `pmix_pdata_t`)

IN `p`
: Pointer to the `pmix_proc_t` structure containing the identifier of the process being referenced (pointer to `pmix_proc_t`)

IN `k`
: String key to be loaded - must be less than or equal to `PMIX_MAX_KEYLEN` in length (handle)

IN `d`
    Pointer to the data value to be loaded (handle)
IN `t`
    Type of the provided data value (`pmix_data_type_t`)

This macro simplifies the loading of key, process identifier, and data into a `pmix_proc_t` by correctly assigning values to the structure's fields.

Advice to users

Key, process identifier, and data will all be copied into the `pmix_pdata_t` - thus, the source information can be modified or free'd without affecting the copied data once the macro has completed.

3.2.21.6 Transfer a lookup data structure

Summary

Transfer key, process identifier, and data value between two `pmix_pdata_t` structures.

PMIx v2.0

C

```
PMIX_PDATA_XFER(d, s);
```

C

IN `d`
    Pointer to the destination `pmix_pdata_t` (pointer to `pmix_pdata_t`)
IN `s`
    Pointer to the source `pmix_pdata_t` (pointer to `pmix_pdata_t`)

This macro simplifies the transfer of key and data between two `pmix_pdata_t` structures.

Advice to users

Key, process identifier, and data will all be copied into the destination `pmix_pdata_t` - thus, the source `pmix_pdata_t` may free'd without affecting the copied data once the macro has completed.

3.2.22 Application Structure

The **pmix_app_t** structure describes the application context for the **PMIx_Spawn** and **PMIx_Spawn_nb** operations.

PMIx v1.0

C

```
typedef struct pmix_app {
    /** Executable */
    char *cmd;
    /** Argument set, NULL terminated */
    char **argv;
    /** Environment set, NULL terminated */
    char **env;
    /** Current working directory */
    char *cwd;
    /** Maximum processes with this profile */
    int maxprocs;
    /** Array of info keys describing this application*/
    pmix_info_t *info;
    /** Number of info keys in 'info' array */
    size_t ninfo;
} pmix_app_t;
```

C

3.2.23 App structure support macros

The following macros are provided to support the **pmix_app_t** structure.

3.2.23.1 Initialize the pmix_app_t structure

Initialize the **pmix_app_t** fields

PMIx v1.0

C

```
PMIX_APP_CONSTRUCT(m)
```

C

IN **m**
Pointer to the structure to be initialized (pointer to **pmix_app_t**)

3.2.23.2 Destruct the pmix_app_t structure

Destruct the pmix_app_t fields

PMIx v1.0

C

```
PMIX_APP_DESTRUCT(m)
```

C

IN `m`
: Pointer to the structure to be destructed (pointer to pmix_app_t)

3.2.23.3 Create a pmix_app_t array

Allocate and initialize an array of pmix_app_t structures

PMIx v1.0

C

```
PMIX_APP_CREATE(m, n)
```

C

INOUT `m`
: Address where the pointer to the array of pmix_app_t structures shall be stored (handle)

IN `n`
: Number of structures to be allocated (**size_t**)

3.2.23.4 Free a pmix_app_t array

Release an array of pmix_app_t structures

PMIx v1.0

C

```
PMIX_APP_FREE(m, n)
```

C

IN `m`
: Pointer to the array of pmix_app_t structures (handle)

IN `n`
: Number of structures in the array (**size_t**)

3.2.24 Query Structure

The **pmix_query_t** structure is used by **PMIx_Query_info_nb** to describe a single query operation.

PMIx v2.0

C

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

C

3.2.25 Query structure support macros

The following macros are provided to support the **pmix_query_t** structure.

3.2.25.1 Initialize the pmix_query_t structure

Initialize the **pmix_query_t** fields

PMIx v2.0

C

```
PMIX_QUERY_CONSTRUCT(m)
```

C

IN **m**
Pointer to the structure to be initialized (pointer to **pmix_query_t**)

3.2.25.2 Destruct the pmix_query_t structure

Destruct the **pmix_query_t** fields

PMIx v2.0

C

```
PMIX_QUERY_DESTRUCT(m)
```

C

IN **m**
Pointer to the structure to be destructed (pointer to **pmix_query_t**)

3.2.25.3 Create a `pmix_query_t` array

Allocate and initialize an array of `pmix_query_t` structures

PMIx v2.0

C

```
PMIX_QUERY_CREATE(m, n)
```

C

INOUT `m`
Address where the pointer to the array of `pmix_query_t` structures shall be stored (handle)

IN `n`
Number of structures to be allocated (`size_t`)

3.2.25.4 Free a `pmix_query_t` array

Release an array of `pmix_query_t` structures

PMIx v2.0

C

```
PMIX_QUERY_FREE(m, n)
```

C

IN `m`
Pointer to the array of `pmix_query_t` structures (handle)

IN `n`
Number of structures in the array (`size_t`)

3.2.26 Modex Structure

The `pmix_modex_data_t` structure describes the business card exchange (BCX) information.

Note: This structure and its supporting macros have been deprecated and will be removed in future versions of the PMIx Standard.

PMIx v1.0

C

```
typedef struct pmix_modex_data {
    pmix_nspace_t nspace;
    int rank;
    uint8_t *blob;
    size_t size;
} pmix_modex_data_t;
```

C

3.2.27 Modex data structure support macros

The following macros are provided to support the **pmix_modex_data_t** structure.

3.2.27.1 Initialize the pmix_modex_data_t structure

Initialize the **pmix_modex_data_t** fields

PMIx v1.0

C

```
PMIX_MODEX_CONSTRUCT(m)
```

C

IN **m**
Pointer to the structure to be initialized (pointer to **pmix_modex_data_t**)

3.2.27.2 Destruct the pmix_modex_data_t structure

Destruct the **pmix_modex_data_t** fields

PMIx v1.0

C

```
PMIX_MODEX_DESTRUCT(m)
```

C

IN **m**
Pointer to the structure to be destructed (pointer to **pmix_modex_data_t**)

3.2.27.3 Create a pmix_modex_data_t array

Allocate and initialize an array of **pmix_modex_data_t** structures

PMIx v1.0

C

```
PMIX_MODEX_CREATE(m, n)
```

C

INOUT **m**
Address where the pointer to the array of **pmix_modex_data_t** structures shall be stored (handle)

IN **n**
Number of structures to be allocated (**size_t**)

3.2.27.4 Free a `pmix_modex_data_t` array

Release an array of **pmix_modex_data_t** structures

PMIx v1.0

C

```c
PMIX_MODEX_FREE(m, n)
```

C

IN **m**
Pointer to the array of **pmix_modex_data_t** structures (handle)

IN **n**
Number of structures in the array (**size_t**)

3.3 Data Packing/Unpacking Types and Structures

This section defines types and structures used to pack and unpack data passed through the PMIx API.

3.3.1 Byte Object Type

The **pmix_byte_object_t** structure describes a raw byte sequence.

PMIx v1.0

C

```c
typedef struct pmix_byte_object {
    char *bytes;
    size_t size;
} pmix_byte_object_t;
```

C

3.3.2 Byte object support macros

The following macros support the **pmix_byte_object_t** structure.

3.3.2.1 Initialize the `pmix_byte_object_t` structure

Initialize the **pmix_byte_object_t** fields

PMIx v2.0

C

```c
PMIX_BYTE_OBJECT_CONSTRUCT(m)
```

C

IN **m**
Pointer to the structure to be initialized (pointer to **pmix_byte_object_t**)

3.3.2.2 Destruct the pmix_byte_object_t structure

Clear the pmix_byte_object_t fields

PMIx v2.0

C

```
PMIX_BYTE_OBJECT_DESTRUCT(m)
```

C

IN m
Pointer to the structure to be destructed (pointer to pmix_byte_object_t)

3.3.2.3 Create a pmix_byte_object_t structure

Allocate and intitialize an array of pmix_byte_object_t structures

PMIx v2.0

C

```
PMIX_BYTE_OBJECT_CREATE(m, n)
```

C

INOUT m
Address where the pointer to the array of pmix_byte_object_t structures shall be stored (handle)

IN n
Number of structures to be allocated (size_t)

3.3.2.4 Free a pmix_byte_object_t array

Release an array of pmix_byte_object_t structures

PMIx v2.0

C

```
PMIX_BYTE_OBJECT_FREE(m, n)
```

C

IN m
Pointer to the array of pmix_byte_object_t structures (handle)

IN n
Number of structures in the array (size_t)

3.3.2.5 Load a pmix_byte_object_t structure

Load values into a pmix_byte_object_t

PMIx v2.0

```c
PMIX_BYTE_OBJECT_LOAD(b, d, s)
```

IN b
Pointer to the structure to be loaded (pointer to pmix_byte_object_t)

IN d
Pointer to the data to be loaded (char*)

IN s
Number of bytes in the data array (size_t)

3.3.3 Data Buffer Type

The pmix_data_buffer_t structure describes a data buffer used for packing and unpacking.

PMIx v2.0

```c
typedef struct pmix_data_buffer {
    /** Start of my memory */
    char *base_ptr;
    /** Where the next data will be packed to (within the allocated
        memory starting at base_ptr) */
    char *pack_ptr;
    /** Where the next data will be unpacked from (within the
        allocated memory starting as base_ptr) */
    char *unpack_ptr;
    /** Number of bytes allocated (starting at base_ptr) */
    size_t bytes_allocated;
    /** Number of bytes used by the buffer (i.e., amount of data --
        including overhead -- packed in the buffer) */
    size_t bytes_used;
} pmix_data_buffer_t;
```

3.3.4 Data buffer support macros

The following macros support the pmix_data_buffer_t structure.

3.3.4.1 Initialize the `pmix_data_buffer_t` structure

Initialize the `pmix_data_buffer_t` fields

PMIx v2.0 C

```c
PMIX_DATA_BUFFER_CONSTRUCT(m)
```

C

IN `m`
Pointer to the structure to be initialized (pointer to `pmix_data_buffer_t`)

3.3.4.2 Destruct the `pmix_data_buffer_t` structure

Clear the `pmix_data_buffer_t` fields

PMIx v2.0 C

```c
PMIX_DATA_BUFFER_DESTRUCT(m)
```

C

IN `m`
Pointer to the structure to be destructed (pointer to `pmix_data_buffer_t`)

3.3.4.3 Create a `pmix_data_buffer_t` structure

Allocate and intitialize a `pmix_data_buffer_t` structure

PMIx v2.0 C

```c
PMIX_DATA_BUFFER_CREATE(m)
```

C

INOUT `m`
Address where the pointer to the `pmix_data_buffer_t` structure shall be stored (handle)

3.3.4.4 Free a `pmix_data_buffer_t`

Release a `pmix_data_buffer_t` structure

PMIx v2.0 C

```c
PMIX_DATA_BUFFER_RELEASE(m)
```

C

IN `m`
Pointer to the `pmix_data_buffer_t` structure to be released (handle)

3.3.5 Data Array Structure

The **pmix_data_array_t** structure defines an array data structure.

PMIx v2.0

C

```
typedef struct pmix_data_array {
    pmix_data_type_t type;
    size_t size;
    void *array;
} pmix_data_array_t;
```

C

3.3.6 Generalized Data Types Used for Packing/Unpacking

The **pmix_data_type_t** structure is a **uint16_t** type for identifying the data type for packing/unpacking purposes.

Advice to PMIx library implementers

The following constants can be used to set a variable of the type **pmix_data_type_t** . Data types in the PMIx Standard are defined in terms of the C-programming language. Implementers wishing to support other languages should provide the equivalent definitions in a language-appropriate manner. Additionally, a PMIx implementation may choose to add additional types.

3.3.6.1 PMIx v1 Data Types

The following types were introduced in version 1 of the PMIx Standard.

PMIX_UNDEF	Undefined
PMIX_BOOL	Boolean (converted to/from native **true/false**) (**bool**)
PMIX_BYTE	A byte of data (**uint8_t**)
PMIX_STRING	**NULL** terminated string (**char***)
PMIX_SIZE	Size **size_t**
PMIX_PID	Operating process identifier (PID) (**pid_t**)
PMIX_INT	Integer (**int**)
PMIX_INT8	8-byte integer (**int8_t**)
PMIX_INT16	16-byte integer (**int16_t**)
PMIX_INT32	32-byte integer (**int32_t**)
PMIX_INT64	64-byte integer (**int64_t**)
PMIX_UINT	Unsigned integer (**unsigned int**)
PMIX_UINT8	Unsigned 8-byte integer (**uint8_t**)

PMIX_UINT16 Unsigned 16-byte integer (**uint16_t**)
PMIX_UINT32 Unsigned 32-byte integer (**uint32_t**)
PMIX_UINT64 Unsigned 64-byte integer (**uint64_t**)
PMIX_FLOAT Float (**float**)
PMIX_DOUBLE Double (**double**)
PMIX_TIMEVAL Time value (**struct timeval**)
PMIX_TIME Time (**time_t**)
PMIX_VALUE Value (**pmix_value_t**)
PMIX_PROC Process (**pmix_proc_t**)
PMIX_APP Application context
PMIX_INFO Info object
PMIX_PDATA Pointer to data
PMIX_BUFFER Buffer
PMIX_BYTE_OBJECT Byte object (**pmix_byte_object_t**)
PMIX_KVAL Key/value pair
PMIX_MODEX (Deprecated in PMIx 2.0) Modex
PMIX_PERSIST Persistance (**pmix_persistence_t**)
PMIX_INFO_ARRAY (Deprecated in PMIx 2.0) Info array

3.3.6.2 PMIx v2 Data Types

The following types were introduced in version 2 of the PMIx Standard.

PMIX_STATUS Status (**pmix_status_t**)
PMIX_POINTER Pointer (**void***)
PMIX_SCOPE Scope (**pmix_scope_t**)
PMIX_DATA_RANGE Data range (**pmix_data_range_t**)
PMIX_COMMAND Command
PMIX_INFO_DIRECTIVES Info directives
PMIX_DATA_TYPE Data type
PMIX_PROC_STATE Process state (**pmix_proc_state_t**)
PMIX_PROC_INFO Process info (**pmix_proc_info_t**)
PMIX_DATA_ARRAY Data array (**pmix_data_array_t**)
PMIX_PROC_RANK Process rank (**pmix_rank_t**)
PMIX_QUERY Query
PMIX_COMPRESSED_STRING Compressed string (with zlib)
PMIX_ALLOC_DIRECTIVE Allocation directive (**pmix_alloc_directive_t**)
PMIX_DATA_TYPE_MAX A boundary for implementers above which they can add their own data types.

3.4 Reserved attributes

The PMIx standard defines a relatively small set of APIs and the caller may customize the behavior of the API by passing one or more attributes to that API. Additionally, attributes may be keys passed to `PMIx_Get` calls to access the specified values from the system.

Each attribute is represented by a *key* string, and a type for the associated *value*. This section defines a set of **reserved** keys which are prefixed with `pmix.` to designate them as PMIx standard reserved keys. All definitions were introduced in version 1 of the standard unless otherwise marked.

Applications or associated libraries (e.g., MPI) may choose to define additional attributes. The attributes defined in this section are of the system and job as opposed to the attributes that the application (or associated libraries) might choose to expose. Due to this extensibility the `PMIx_Get` API will return `PMIX_ERR_NOT_FOUND` if the provided *key* cannot be found.

Attributes added in this version of the standard are shown in ***magenta*** to distinguish them from those defined in prior versions, which are shown in ***black***. Deprecated attributes are shown in ***green*** and will be removed in future versions of the standard.

`PMIX_ATTR_UNDEF` **`NULL`** (**`NULL`**)
Constant representing an undefined attribute.

3.4.1 Initialization attributes

These attributes are defined to assist the caller with initialization by passing them into the appropriate initialization API - thus, they are not typically accessed via the `PMIx_Get` API.

`PMIX_EVENT_BASE` **`"pmix.evbase"`** (**`struct event_base *`**)
Pointer to libevent[1] **`event_base`** to use in place of the internal progress thread.

`PMIX_SERVER_TOOL_SUPPORT` **`"pmix.srvr.tool"`** (**`bool`**)
The host RM wants to declare itself as willing to accept tool connection requests.

`PMIX_SERVER_REMOTE_CONNECTIONS` **`"pmix.srvr.remote"`** (**`bool`**)
Allow connections from remote tools. Forces the PMIx server to not exclusively use loopback device.

`PMIX_SERVER_SYSTEM_SUPPORT` **`"pmix.srvr.sys"`** (**`bool`**)
The host RM wants to declare itself as being the local system server for PMIx connection requests.

`PMIX_SERVER_TMPDIR` **`"pmix.srvr.tmpdir"`** (**`char*`**)
Top-level temporary directory for all *client* processes connected to this server, and where the PMIx server will place its *tool* rendezvous point and contact information.

`PMIX_SYSTEM_TMPDIR` **`"pmix.sys.tmpdir"`** (**`char*`**)
Temporary directory for this system, and where a PMIx server that declares itself to be a system-level server will place a *tool* rendezvous point and contact information.

[1] http://libevent.org/

PMIX_REGISTER_NODATA **"pmix.reg.nodata"** (**bool**)
Registration is for the namespace only. Do not copy job data.

PMIX_SERVER_ENABLE_MONITORING **"pmix.srv.monitor"** (**bool**)
Enable PMIx internal monitoring by the PMIx server.

PMIX_SERVER_NSPACE **"pmix.srv.nspace"** (**char***)
Name of the namespace to use for this PMIx server.

PMIX_SERVER_RANK **"pmix.srv.rank"** (**pmix_rank_t**)
Rank of this PMIx server

3.4.2 Tool-related attributes

These attributes are defined to assist PMIx-enabled tools to connect with the PMIx server by passing them into the **PMIx_tool_init** API - thus, they are not typically accessed via the **PMIx_Get** API.

PMIX_TOOL_NSPACE **"pmix.tool.nspace"** (**char***)
Name of the namespace to use for this tool.

PMIX_TOOL_RANK **"pmix.tool.rank"** (**uint32_t**)
Rank of this tool.

PMIX_SERVER_PIDINFO **"pmix.srvr.pidinfo"** (**pid_t**)
PID of the target PMIx server for a tool.

PMIX_CONNECT_TO_SYSTEM **"pmix.cnct.sys"** (**bool**)
The requestor requires that a connection be made only to a local, system-level PMIx server.

PMIX_CONNECT_SYSTEM_FIRST **"pmix.cnct.sys.first"** (**bool**)
Preferentially, look for a system-level PMIx server first.

PMIX_SERVER_URI **"pmix.srvr.uri"** (**char***)
uniform resource identifier (URI) of the PMIx server to be contacted.

PMIX_SERVER_HOSTNAME **"pmix.srvr.host"** (**char***)
Host where target PMIx server is located.

PMIX_CONNECT_MAX_RETRIES **"pmix.tool.mretries"** (**uint32_t**)
Maximum number of times to try to connect to PMIx server.

PMIX_CONNECT_RETRY_DELAY **"pmix.tool.retry"** (**uint32_t**)
Time in seconds between connection attempts to a PMIx server.

PMIX_TOOL_DO_NOT_CONNECT **"pmix.tool.nocon"** (**bool**)
The tool wants to use internal PMIx support, but does not want to connect to a PMIx server.

3.4.3 Identification attributes

These attributes are defined to identify a process and it's associated PMIx-enabled library. They are not typically accessed via the **PMIx_Get** API, and thus are not associated with a particular rank.

PMIX_USERID **"pmix.euid"** (**uint32_t**)
Effective user id.

PMIX_GRPID "pmix.egid" (uint32_t)
Effective group id.

PMIX_DSTPATH "pmix.dstpath" (char*)
Path to shared memory data storage (dstore) files.

PMIX_VERSION_INFO "pmix.version" (char*)
PMIx version of contractor.

PMIX_PROGRAMMING_MODEL "pmix.pgm.model" (char*)
Programming model being initialized (e.g., "MPI" or "OpenMP")

PMIX_MODEL_LIBRARY_NAME "pmix.mdl.name" (char*)
Programming model implementation ID (e.g., "OpenMPI" or "MPICH")

PMIX_MODEL_LIBRARY_VERSION "pmix.mld.vrs" (char*)
Programming model version string (e.g., "2.1.1")

PMIX_THREADING_MODEL "pmix.threads" (char*)
Threading model used (e.g., "pthreads")

PMIX_REQUESTOR_IS_TOOL "pmix.req.tool" (bool)
The requesting process is a PMIx tool.

PMIX_REQUESTOR_IS_CLIENT "pmix.req.client" (bool)
The requesting process is a PMIx client.

3.4.4 UNIX socket rendezvous socket attributes

These attributes are used to describe a UNIX socket for rendezvous with the local RM by passing them into the relevant initialization API - thus, they are not typically accessed via the **PMIx_Get** API.

PMIX_USOCK_DISABLE "pmix.usock.disable" (bool)
Disable legacy UNIX socket (usock) support

PMIX_SOCKET_MODE "pmix.sockmode" (uint32_t)
POSIX *mode_t* (9 bits valid)

PMIX_SINGLE_LISTENER "pmix.sing.listnr" (bool)
Use only one rendezvous socket, letting priorities and/or environment parameters select the active transport.

3.4.5 TCP connection attributes

These attributes are used to describe a TCP socket for rendezvous with the local RM by passing them into the relevant initialization API - thus, they are not typically accessed via the **PMIx_Get** API.

PMIX_TCP_REPORT_URI "pmix.tcp.repuri" (char*)
If provided, directs that the TCP URI be reported and indicates the desired method of reporting: `'-'` for stdout, `'+'` for stderr, or filename.

PMIX_TCP_URI "pmix.tcp.uri" (char*)

The URI of the PMIx server to connect to, or a file name containing it in the form of **`file:<name of file containing it>`**.

`PMIX_TCP_IF_INCLUDE` **`"pmix.tcp.ifinclude"`** (**`char*`**)
: Comma-delimited list of devices and/or Classless Inter-Domain Routing (CIDR) notation to include when establishing the TCP connection.

`PMIX_TCP_IF_EXCLUDE` **`"pmix.tcp.ifexclude"`** (**`char*`**)
: Comma-delimited list of devices and/or CIDR notation to exclude when establishing the TCP connection.

`PMIX_TCP_IPV4_PORT` **`"pmix.tcp.ipv4"`** (**`int`**)
: The IPv4 port to be used.

`PMIX_TCP_IPV6_PORT` **`"pmix.tcp.ipv6"`** (**`int`**)
: The IPv6 port to be used.

`PMIX_TCP_DISABLE_IPV4` **`"pmix.tcp.disipv4"`** (**`bool`**)
: Set to **`true`** to disable IPv4 family of addresses.

`PMIX_TCP_DISABLE_IPV6` **`"pmix.tcp.disipv6"`** (**`bool`**)
: Set to **`true`** to disable IPv6 family of addresses.

3.4.6 Global Data Storage (GDS) attributes

These attributes are used to define the behavior of the GDS used to manage key/value pairs by passing them into the relevant initialization API - thus, they are not typically accessed via the **`PMIx_Get`** API.

`PMIX_GDS_MODULE` **`"pmix.gds.mod"`** (**`char*`**)
: Comma-delimited string of desired modules.

3.4.7 General process-level attributes

These attributes are used to define process attributes and are referenced by their process rank.

`PMIX_CPUSET` **`"pmix.cpuset"`** (**`char*`**)
: hwloc[2] bitmap to be applied to the process upon launch.

`PMIX_CREDENTIAL` **`"pmix.cred"`** (**`char*`**)
: Security credential assigned to the process.

`PMIX_SPAWNED` **`"pmix.spawned"`** (**`bool`**)
: **`true`** if this process resulted from a call to **`PMIx_Spawn`** .

`PMIX_ARCH` **`"pmix.arch"`** (**`uint32_t`**)
: Architecture flag.

[2] https://www.open-mpi.org/projects/hwloc/

3.4.8 Scratch directory attributes

These attributes are used to define an application scratch directory and are referenced using the **PMIX_RANK_WILDCARD** rank.

PMIX_TMPDIR "pmix.tmpdir" (char*)
Full path to the top-level temporary directory assigned to the session.

PMIX_NSDIR "pmix.nsdir" (char*)
Full path to the temporary directory assigned to the namespace, under **PMIX_TMPDIR** .

PMIX_PROCDIR "pmix.pdir" (char*)
Full path to the subdirectory under **PMIX_NSDIR** assigned to the process.

PMIX_TDIR_RMCLEAN "pmix.tdir.rmclean" (bool)
Resource Manager will clean session directories

3.4.9 Relative Rank Descriptive Attributes

These attributes are used to describe information about relative ranks as assigned by the RM, and thus are referenced using the process rank except where noted.

PMIX_PROCID "pmix.procid" (pmix_proc_t)
Process identifier

PMIX_NSPACE "pmix.nspace" (char*)
Namespace of the job.

PMIX_JOBID "pmix.jobid" (char*)
Job identifier assigned by the scheduler.

PMIX_APPNUM "pmix.appnum" (uint32_t)
Application number within the job.

PMIX_RANK "pmix.rank" (pmix_rank_t)
Process rank within the job.

PMIX_GLOBAL_RANK "pmix.grank" (pmix_rank_t)
Process rank spanning across all jobs in this session.

PMIX_APP_RANK "pmix.apprank" (pmix_rank_t)
Process rank within this application.

PMIX_NPROC_OFFSET "pmix.offset" (pmix_rank_t)
Starting global rank of this job - referenced using **PMIX_RANK_WILDCARD** .

PMIX_LOCAL_RANK "pmix.lrank" (uint16_t)
Local rank on this node within this job.

PMIX_NODE_RANK "pmix.nrank" (uint16_t)
Process rank on this node spanning all jobs.

PMIX_LOCALLDR "pmix.lldr" (pmix_rank_t)
Lowest rank on this node within this job - referenced using **PMIX_RANK_WILDCARD** .

PMIX_APPLDR "pmix.aldr" (pmix_rank_t)
Lowest rank in this application within this job - referenced using **PMIX_RANK_WILDCARD** .

PMIX_PROC_PID "pmix.ppid" (pid_t)

PID of specified process.

PMIX_SESSION_ID "pmix.session.id" (uint32_t)
Session identifier - referenced using **PMIX_RANK_WILDCARD** .

PMIX_NODE_LIST "pmix.nlist" (char*)
Comma-delimited list of nodes running processes for the specified namespace - referenced using **PMIX_RANK_WILDCARD** .

PMIX_ALLOCATED_NODELIST "pmix.alist" (char*)
Comma-delimited list of all nodes in this allocation regardless of whether or not they currently host processes - referenced using **PMIX_RANK_WILDCARD** .

PMIX_HOSTNAME "pmix.hname" (char*)
Name of the host where the specified process is running.

PMIX_NODEID "pmix.nodeid" (uint32_t)
Node identifier where the specified process is located, expressed as the node's index (beginning at zero) in the array resulting from expansion of the **PMIX_NODE_MAP** regular expression for the **job**

PMIX_LOCAL_PEERS "pmix.lpeers" (char*)
Comma-delimited list of ranks on this node within the specified namespace - referenced using **PMIX_RANK_WILDCARD** .

PMIX_LOCAL_PROCS "pmix.lprocs" (pmix_proc_t array)
Array of **pmix_proc_t** of all processes on the specified node - referenced using **PMIX_RANK_WILDCARD** .

PMIX_LOCAL_CPUSETS "pmix.lcpus" (char*)
Colon-delimited cpusets of local peers within the specified namespace - referenced using **PMIX_RANK_WILDCARD** .

PMIX_PROC_URI "pmix.puri" (char*)
URI containing contact information for a given process.

PMIX_LOCALITY "pmix.loc" (uint16_t)
Relative locality of the specified process to the requestor.

PMIX_PARENT_ID "pmix.parent" (pmix_proc_t)
Process identifier of the parent process of the calling process.

3.4.10 Information retrieval attributes

The following attributes are used to specify the level of information (e.g., **session** , **job** , or **application**) being requested where ambiguity may exist - see 5.1.5 for examples of their use.

PMIX_SESSION_INFO "pmix.ssn.info" (bool)
Return information about the specified session. If information about a session other than the one containing the requesting process is desired, then the attribute array must contain a **PMIX_SESSION_ID** attribute identifying the desired target.

PMIX_JOB_INFO "pmix.job.info" (bool)

Return information about the specified job or namespace. If information about a job or namespace other than the one containing the requesting process is desired, then the attribute array must contain a **PMIX_JOBID** or **PMIX_NSPACE** attribute identifying the desired target. Similarly, if information is requested about a job or namespace in a session other than the one containing the requesting process, then an attribute identifying the target session must be provided.

PMIX_APP_INFO "pmix.app.info" (bool)
Return information about the specified application. If information about an application other than the one containing the requesting process is desired, then the attribute array must contain a **PMIX_APPNUM** attribute identifying the desired target. Similarly, if information is requested about an application in a job or session other than the one containing the requesting process, then attributes identifying the target job and/or session must be provided.

PMIX_NODE_INFO "pmix.node.info" (bool)
Return information about the specified node. If information about a node other than the one containing the requesting process is desired, then the attribute array must contain either the **PMIX_NODEID** or **PMIX_HOSTNAME** attribute identifying the desired target.

3.4.11 Information storage attributes

The following attributes are used to assemble information by its level (e.g., **session** , **job** , or **application**) for storage where ambiguity may exist - see 10.1.3.1 for examples of their use.

PMIX_SESSION_INFO_ARRAY "pmix.ssn.arr" (pmix_data_array_t)
Provide an array of **pmix_info_t** containing session-level information. The **PMIX_SESSION_ID** attribute is *required* to be included in the array.

PMIX_JOB_INFO_ARRAY "pmix.job.arr" (pmix_data_array_t)
Provide an array of **pmix_info_t** containing job-level information. Information is registered one job (aka namespace) at a time via the **PMIx_server_register_nspace** API. Thus, there is no requirement that the array contain either the **PMIX_NSPACE** or **PMIX_JOBID** attributes, though either or both of them *may* be included.

PMIX_APP_INFO_ARRAY "pmix.app.arr" (pmix_data_array_t)
Provide an array of **pmix_info_t** containing app-level information. The **PMIX_NSPACE** or **PMIX_JOBID** attributes of the **job** containing the appplication, plus its **PMIX_APPNUM** attribute, are *required* to be included in the array.

PMIX_NODE_INFO_ARRAY "pmix.node.arr" (pmix_data_array_t)
Provide an array of **pmix_info_t** containing node-level information. At a minimum, either the **PMIX_NODEID** or **PMIX_HOSTNAME** attribute is *required* to be included in the array, though both *may* be included.

3.4.12 Size information attributes

These attributes are used to describe the size of various dimensions of the PMIx universe - all are referenced using **PMIX_RANK_WILDCARD** .

PMIX_UNIV_SIZE "pmix.univ.size" (uint32_t)

Number of allocated slots in a session - each slot may or may not be occupied by an executing process. Note that this attribute is the equivalent to the combination of **PMIX_SESSION_INFO_ARRAY** with the **PMIX_NUM_SLOTS** entry in the array - it is included in the Standard for historical reasons.

PMIX_JOB_SIZE "pmix.job.size" (uint32_t)
Total number of processes in this job across all contained applications

PMIX_JOB_NUM_APPS "pmix.job.napps" (uint32_t)
Number of applications in this job.

PMIX_APP_SIZE "pmix.app.size" (uint32_t)
Number of processes in this application.

PMIX_LOCAL_SIZE "pmix.local.size" (uint32_t)
Number of processes in this job on this node.

PMIX_NODE_SIZE "pmix.node.size" (uint32_t)
Number of processes across all jobs on this node.

PMIX_MAX_PROCS "pmix.max.size" (uint32_t)
Maximum number of processes for this job.

PMIX_NUM_NODES "pmix.num.nodes" (uint32_t)
Number of nodes in this session or namespace.

PMIX_NUM_SLOTS "pmix.num.slots" (uint32_t)
Number of slots allocated to the session, namespace, or application.

3.4.13 Memory information attributes

These attributes are used to describe memory available and used in the system - all are referenced using **PMIX_RANK_WILDCARD** .

PMIX_AVAIL_PHYS_MEMORY "pmix.pmem" (uint64_t)
Total available physical memory on this node.

PMIX_DAEMON_MEMORY "pmix.dmn.mem" (float)
Megabytes of memory currently used by the RM daemon.

PMIX_CLIENT_AVG_MEMORY "pmix.cl.mem.avg" (float)
Average Megabytes of memory used by client processes.

3.4.14 Topology information attributes

These attributes are used to describe topology information in the PMIx universe - all are referenced using **PMIX_RANK_WILDCARD** except where noted.

PMIX_NET_TOPO "pmix.ntopo" (char*)
eXtensible Markup Language (XML) representation of the network topology.

PMIX_LOCAL_TOPO "pmix.ltopo" (char*)
XML representation of local node topology.

PMIX_NODE_LIST "pmix.nlist" (char*)

Comma-delimited list of nodes running processes for this job.

PMIX_TOPOLOGY "pmix.topo" (hwloc_topology_t)
Pointer to the PMIx client's internal hwloc topology object.

PMIX_TOPOLOGY_SIGNATURE "pmix.toposig" (char*)
Topology signature string.

PMIX_LOCALITY_STRING "pmix.locstr" (char*)
String describing a process's bound location - referenced using the process's rank. The string is of the form:
NM%s:SK%s:L3%s:L2%s:L1%s:CR%s:HT%s
Where the **%s** is replaced with an integer index or inclusive range for hwloc. **NM** identifies the numa node(s). **SK** identifies the socket(s). **L3** identifies the L3 cache(s). **L2** identifies the L2 cache(s). **L1** identifies the L1 cache(s). **CR** identifies the cores(s). **HT** identifies the hardware thread(s). If your architecture does not have the specified hardware designation then it can be omitted from the signature.
For example: **NM0:SK0:L30-4:L20-4:L10-4:CR0-4:HT0-39**.
This means numa node **0**, socket **0**, L3 caches **0,1,2,3,4**, L2 caches **0-4**, L1 caches **0-4**, cores **0,1,2,3,4**, and hardware threads **0-39**.

PMIX_HWLOC_SHMEM_ADDR "pmix.hwlocaddr" (size_t)
Address of the hwloc shared memory segment.

PMIX_HWLOC_SHMEM_SIZE "pmix.hwlocsize" (size_t)
Size of the hwloc shared memory segment.

PMIX_HWLOC_SHMEM_FILE "pmix.hwlocfile" (char*)
Path to the hwloc shared memory file.

PMIX_HWLOC_XML_V1 "pmix.hwlocxml1" (char*)
XML representation of local topology using hwloc's v1.x format.

PMIX_HWLOC_XML_V2 "pmix.hwlocxml2" (char*)
XML representation of local topology using hwloc's v2.x format.

3.4.15 Request-related attributes

These attributes are used to influence the behavior of various PMIx operations - they do not represent values accessed using the **PMIx_Get** API.

PMIX_COLLECT_DATA "pmix.collect" (bool)
Collect data and return it at the end of the operation.

PMIX_TIMEOUT "pmix.timeout" (int)
Time in seconds before the specified operation should time out (*0* indicating infinite) in error. The timeout parameter can help avoid "hangs" due to programming errors that prevent the target process from ever exposing its data.

PMIX_IMMEDIATE "pmix.immediate" (bool)
Specified operation should immediately return an error from the PMIx server if the requested data cannot be found - do not request it from the host RM.

PMIX_WAIT "pmix.wait" (int)

Caller requests that the PMIx server wait until at least the specified number of values are found (*0* indicates all and is the default).

PMIX_COLLECTIVE_ALGO "pmix.calgo" (char*)
Comma-delimited list of algorithms to use for the collective operation. PMIx does not impose any requirements on a host environment's collective algorithms. Thus, the acceptable values for this attribute will be environment-dependent - users are encouraged to check their host environment for supported values.

PMIX_COLLECTIVE_ALGO_REQD "pmix.calreqd" (bool)
If **true**, indicates that the requested choice of algorithm is mandatory.

PMIX_NOTIFY_COMPLETION "pmix.notecomp" (bool)
Notify the parent process upon termination of child job.

PMIX_RANGE "pmix.range" (pmix_data_range_t)
Value for calls to publish/lookup/unpublish or for monitoring event notifications.

PMIX_PERSISTENCE "pmix.persist" (pmix_persistence_t)
Value for calls to **PMIx_Publish** .

PMIX_DATA_SCOPE "pmix.scope" (pmix_scope_t)
Scope of the data to be found in a **PMIx_Get** call.

PMIX_OPTIONAL "pmix.optional" (bool)
Look only in the client's local data store for the requested value - do not request data from the PMIx server if not found.

PMIX_EMBED_BARRIER "pmix.embed.barrier" (bool)
Execute a blocking fence operation before executing the specified operation. For example, **PMIx_Finalize** does not include an internal barrier operation by default. This attribute would direct **PMIx_Finalize** to execute a barrier as part of the finalize operation.

PMIX_JOB_TERM_STATUS "pmix.job.term.status" (pmix_status_t)
Status to be returned upon job termination.

PMIX_PROC_STATE_STATUS "pmix.proc.state" (pmix_proc_state_t)
Process state

3.4.16 Server-to-PMIx library attributes

Attributes used by the host environment to pass data to its PMIx server library. The data will then be parsed and provided to the local PMIx clients. These attributes are all referenced using **PMIX_RANK_WILDCARD** except where noted.

PMIX_REGISTER_NODATA "pmix.reg.nodata" (bool)
Registration is for this namespace only, do not copy job data - this attribute is not accessed using the **PMIx_Get**

PMIX_PROC_DATA "pmix.pdata" (pmix_data_array_t)
Array of process data. Starts with rank, then contains more data.

PMIX_NODE_MAP "pmix.nmap" (char*)
Regular expression of nodes - see 10.1.3.1 for an explanation of its generation.

PMIX_PROC_MAP "pmix.pmap" (char*)

Regular expression describing processes on each node - see 10.1.3.1 for an explanation of its generation.

PMIX_ANL_MAP "pmix.anlmap" (char*)
Process mapping in Argonne National Laboratory's PMI-1/PMI-2 notation.

PMIX_APP_MAP_TYPE "pmix.apmap.type" (char*)
Type of mapping used to layout the application (e.g., **cyclic**).

PMIX_APP_MAP_REGEX "pmix.apmap.regex" (char*)
Regular expression describing the result of the process mapping.

3.4.17 Server-to-Client attributes

Attributes used internally to communicate data from the PMIx server to the PMIx client - they do not represent values accessed using the **PMIx_Get** API.

PMIX_PROC_BLOB "pmix.pblob" (pmix_byte_object_t)
Packed blob of process data.

PMIX_MAP_BLOB "pmix.mblob" (pmix_byte_object_t)
Packed blob of process location.

3.4.18 Event handler registration and notification attributes

Attributes to support event registration and notification - they are values passed to the event registration and notification APIs and therefore are not accessed using the **PMIx_Get** API.

Advice to users

The event handler subsystem defined in the PMIx *ad hoc* version 1 Standard was completely overhauled in version 2 to resolve design flaws. Deprecated attributes shown below were therefore removed in the version 2 Standard.

PMIX_ERROR_NAME "pmix.errname" (pmix_status_t)
Specific error to be notified

PMIX_ERROR_GROUP_COMM "pmix.errgroup.comm" (bool)
Set true to get comm errors notification

PMIX_ERROR_GROUP_ABORT "pmix.errgroup.abort" (bool)
Set true to get abort errors notification

PMIX_ERROR_GROUP_MIGRATE "pmix.errgroup.migrate" (bool)
Set true to get migrate errors notification

PMIX_ERROR_GROUP_RESOURCE "pmix.errgroup.resource" (bool)
Set true to get resource errors notification

PMIX_ERROR_GROUP_SPAWN "pmix.errgroup.spawn" (bool)
Set true to get spawn errors notification

PMIX_ERROR_GROUP_NODE "pmix.errgroup.node" (bool)

Set true to get node status notification

PMIX_ERROR_GROUP_LOCAL **"pmix.errgroup.local"** (**bool**)
Set true to get local errors notification

PMIX_ERROR_GROUP_GENERAL **"pmix.errgroup.gen"** (**bool**)
Set true to get notified of generic errors

PMIX_ERROR_HANDLER_ID **"pmix.errhandler.id"** (**int**)
Errhandler reference id of notification being reported

PMIX_EVENT_HDLR_NAME **"pmix.evname"** (**char***)
String name identifying this handler.

PMIX_EVENT_HDLR_FIRST **"pmix.evfirst"** (**bool**)
Invoke this event handler before any other handlers.

PMIX_EVENT_HDLR_LAST **"pmix.evlast"** (**bool**)
Invoke this event handler after all other handlers have been called.

PMIX_EVENT_HDLR_FIRST_IN_CATEGORY **"pmix.evfirstcat"** (**bool**)
Invoke this event handler before any other handlers in this category.

PMIX_EVENT_HDLR_LAST_IN_CATEGORY **"pmix.evlastcat"** (**bool**)
Invoke this event handler after all other handlers in this category have been called.

PMIX_EVENT_HDLR_BEFORE **"pmix.evbefore"** (**char***)
Put this event handler immediately before the one specified in the **(char*)** value.

PMIX_EVENT_HDLR_AFTER **"pmix.evafter"** (**char***)
Put this event handler immediately after the one specified in the **(char*)** value.

PMIX_EVENT_HDLR_PREPEND **"pmix.evprepend"** (**bool**)
Prepend this handler to the precedence list within its category.

PMIX_EVENT_HDLR_APPEND **"pmix.evappend"** (**bool**)
Append this handler to the precedence list within its category.

PMIX_EVENT_CUSTOM_RANGE **"pmix.evrange"** (**pmix_data_array_t***)
Array of **pmix_proc_t** defining range of event notification.

PMIX_EVENT_AFFECTED_PROC **"pmix.evproc"** (**pmix_proc_t**)
The single process that was affected.

PMIX_EVENT_AFFECTED_PROCS **"pmix.evaffected"** (**pmix_data_array_t***)
Array of **pmix_proc_t** defining affected processes.

PMIX_EVENT_NON_DEFAULT **"pmix.evnondef"** (**bool**)
Event is not to be delivered to default event handlers.

PMIX_EVENT_RETURN_OBJECT **"pmix.evobject"** (**void ***)
Object to be returned whenever the registered callback function **cbfunc** is invoked. The object will *only* be returned to the process that registered it.

PMIX_EVENT_DO_NOT_CACHE **"pmix.evnocache"** (**bool**)
Instruct the PMIx server not to cache the event.

PMIX_EVENT_SILENT_TERMINATION **"pmix.evsilentterm"** (**bool**)
Do not generate an event when this job normally terminates.

3.4.19 Fault tolerance attributes

Attributes to support fault tolerance behaviors - they are values passed to the event notification API and therefore are not accessed using the **PMIx_Get** API.

PMIX_EVENT_TERMINATE_SESSION **"pmix.evterm.sess"** (**bool**)
The RM intends to terminate this session.
PMIX_EVENT_TERMINATE_JOB **"pmix.evterm.job"** (**bool**)
The RM intends to terminate this job.
PMIX_EVENT_TERMINATE_NODE **"pmix.evterm.node"** (**bool**)
The RM intends to terminate all processes on this node.
PMIX_EVENT_TERMINATE_PROC **"pmix.evterm.proc"** (**bool**)
The RM intends to terminate just this process.
PMIX_EVENT_ACTION_TIMEOUT **"pmix.evtimeout"** (**int**)
The time in seconds before the RM will execute error response.
PMIX_EVENT_NO_TERMINATION **"pmix.evnoterm"** (**bool**)
Indicates that the handler has satisfactorily handled the event and believes termination of the application is not required.
PMIX_EVENT_WANT_TERMINATION **"pmix.evterm"** (**bool**)
Indicates that the handler has determined that the application should be terminated

3.4.20 Spawn attributes

Attributes used to describe **PMIx_Spawn** behavior - they are values passed to the **PMIx_Spawn** API and therefore are not accessed using the **PMIx_Get** API when used in that context. However, some of the attributes defined in this section can be provided by the host environment for other purposes - e.g., the environment might provide the **PMIX_MAPPER** attribute in the job-related information so that an application can use **PMIx_Get** to discover the layout algorithm used for determining process locations. Multi-use attributes and their respective access reference rank are denoted below.

PMIX_PERSONALITY **"pmix.pers"** (**char***)
Name of personality to use.
PMIX_HOST **"pmix.host"** (**char***)
Comma-delimited list of hosts to use for spawned processes.
PMIX_HOSTFILE **"pmix.hostfile"** (**char***)
Hostfile to use for spawned processes.
PMIX_ADD_HOST **"pmix.addhost"** (**char***)
Comma-delimited list of hosts to add to the allocation.
PMIX_ADD_HOSTFILE **"pmix.addhostfile"** (**char***)
Hostfile listing hosts to add to existing allocation.
PMIX_PREFIX **"pmix.prefix"** (**char***)
Prefix to use for starting spawned processes.
PMIX_WDIR **"pmix.wdir"** (**char***)

Working directory for spawned processes.

PMIX_MAPPER "pmix.mapper" (char*)
Mapping mechanism to use for placing spawned processes - when accessed using **PMIx_Get** , use the **PMIX_RANK_WILDCARD** value for the rank to discover the mapping mechanism used for the provided namespace.

PMIX_DISPLAY_MAP "pmix.dispmap" (bool)
Display process mapping upon spawn.

PMIX_PPR "pmix.ppr" (char*)
Number of processes to spawn on each identified resource.

PMIX_MAPBY "pmix.mapby" (char*)
Process mapping policy - when accessed using **PMIx_Get** , use the **PMIX_RANK_WILDCARD** value for the rank to discover the mapping policy used for the provided namespace

PMIX_RANKBY "pmix.rankby" (char*)
Process ranking policy - when accessed using **PMIx_Get** , use the **PMIX_RANK_WILDCARD** value for the rank to discover the ranking algorithm used for the provided namespace

PMIX_BINDTO "pmix.bindto" (char*)
Process binding policy - when accessed using **PMIx_Get** , use the **PMIX_RANK_WILDCARD** value for the rank to discover the binding policy used for the provided namespace

PMIX_PRELOAD_BIN "pmix.preloadbin" (bool)
Preload binaries onto nodes.

PMIX_PRELOAD_FILES "pmix.preloadfiles" (char*)
Comma-delimited list of files to pre-position on nodes.

PMIX_NON_PMI "pmix.nonpmi" (bool)
Spawned processes will not call **PMIx_Init** .

PMIX_STDIN_TGT "pmix.stdin" (uint32_t)
Spawned process rank that is to receive **stdin**.

PMIX_FWD_STDIN "pmix.fwd.stdin" (bool)
Forward this process's **stdin** to the designated process.

PMIX_FWD_STDOUT "pmix.fwd.stdout" (bool)
Forward **stdout** from spawned processes to this process.

PMIX_FWD_STDERR "pmix.fwd.stderr" (bool)
Forward **stderr** from spawned processes to this process.

PMIX_DEBUGGER_DAEMONS "pmix.debugger" (bool)
Spawned application consists of debugger daemons.

PMIX_COSPAWN_APP "pmix.cospawn" (bool)
Designated application is to be spawned as a disconnected job. Meaning that it is not part of the "comm_world" of the parent process.

PMIX_SET_SESSION_CWD "pmix.ssncwd" (bool)

Set the application's current working directory to the session working directory assigned by the RM - when accessed using **PMIx_Get** , use the **PMIX_RANK_WILDCARD** value for the rank to discover the session working directory assigned to the provided namespace

PMIX_TAG_OUTPUT **"pmix.tagout"** (**bool**)
Tag application output with the identity of the source process.

PMIX_TIMESTAMP_OUTPUT **"pmix.tsout"** (**bool**)
Timestamp output from applications.

PMIX_MERGE_STDERR_STDOUT **"pmix.mergeerrout"** (**bool**)
Merge **stdout** and **stderr** streams from application processes.

PMIX_OUTPUT_TO_FILE **"pmix.outfile"** (**char***)
Output application output to the specified file.

PMIX_INDEX_ARGV **"pmix.indxargv"** (**bool**)
Mark the **argv** with the rank of the process.

PMIX_CPUS_PER_PROC **"pmix.cpuperproc"** (**uint32_t**)
Number of cpus to assign to each rank - when accessed using **PMIx_Get** , use the **PMIX_RANK_WILDCARD** value for the rank to discover the cpus/process assigned to the provided namespace

PMIX_NO_PROCS_ON_HEAD **"pmix.nolocal"** (**bool**)
Do not place processes on the head node.

PMIX_NO_OVERSUBSCRIBE **"pmix.noover"** (**bool**)
Do not oversubscribe the cpus.

PMIX_REPORT_BINDINGS **"pmix.repbind"** (**bool**)
Report bindings of the individual processes.

PMIX_CPU_LIST **"pmix.cpulist"** (**char***)
List of cpus to use for this job - when accessed using **PMIx_Get** , use the **PMIX_RANK_WILDCARD** value for the rank to discover the cpu list used for the provided namespace

PMIX_JOB_RECOVERABLE **"pmix.recover"** (**bool**)
Application supports recoverable operations.

PMIX_JOB_CONTINUOUS **"pmix.continuous"** (**bool**)
Application is continuous, all failed processes should be immediately restarted.

PMIX_MAX_RESTARTS **"pmix.maxrestarts"** (**uint32_t**)
Maximum number of times to restart a job - when accessed using **PMIx_Get** , use the **PMIX_RANK_WILDCARD** value for the rank to discover the max restarts for the provided namespace

3.4.21 Query attributes

Attributes used to describe **PMIx_Query_info_nb** behavior - these are values passed to the **PMIx_Query_info_nb** API and therefore are not passed to the **PMIx_Get** API.

PMIX_QUERY_REFRESH_CACHE **"pmix.qry.rfsh"** (**bool**)
Retrieve updated information from server.

PMIX_QUERY_NAMESPACES **"pmix.qry.ns"** (**char***)
Request a comma-delimited list of active namespaces.

PMIX_QUERY_JOB_STATUS **"pmix.qry.jst"** (**pmix_status_t**)
Status of a specified, currently executing job.

PMIX_QUERY_QUEUE_LIST **"pmix.qry.qlst"** (**char***)
Request a comma-delimited list of scheduler queues.

PMIX_QUERY_QUEUE_STATUS **"pmix.qry.qst"** (**TBD**)
Status of a specified scheduler queue.

PMIX_QUERY_PROC_TABLE **"pmix.qry.ptable"** (**char***)
Input namespace of the job whose information is being requested returns (**pmix_data_array_t**) an array of **pmix_proc_info_t** .

PMIX_QUERY_LOCAL_PROC_TABLE **"pmix.qry.lptable"** (**char***)
Input namespace of the job whose information is being requested returns (**pmix_data_array_t**) an array of **pmix_proc_info_t** for processes in job on same node.

PMIX_QUERY_LOCAL_ONLY **"pmix.qry.local"** (**bool**)
Constrain the query to local information only.

PMIX_QUERY_AUTHORIZATIONS **"pmix.qry.auths"** (**bool**)
Return operations the PMIx tool is authorized to perform.

PMIX_QUERY_SPAWN_SUPPORT **"pmix.qry.spawn"** (**bool**)
Return a comma-delimited list of supported spawn attributes.

PMIX_QUERY_DEBUG_SUPPORT **"pmix.qry.debug"** (**bool**)
Return a comma-delimited list of supported debug attributes.

PMIX_QUERY_MEMORY_USAGE **"pmix.qry.mem"** (**bool**)
Return information on memory usage for the processes indicated in the qualifiers.

PMIX_QUERY_REPORT_AVG **"pmix.qry.avg"** (**bool**)
Report average values.

PMIX_QUERY_REPORT_MINMAX **"pmix.qry.minmax"** (**bool**)
Report minimum and maximum values.

PMIX_QUERY_ALLOC_STATUS **"pmix.query.alloc"** (**char***)
String identifier of the allocation whose status is being requested.

PMIX_TIME_REMAINING **"pmix.time.remaining"** (**char***)
Query number of seconds (**uint32_t**) remaining in allocation for the specified namespace.

3.4.22 Log attributes

Attributes used to describe **PMIx_Log_nb** behavior - these are values passed to the **PMIx_Log_nb** API and therefore are not accessed using the **PMIx_Get** API.

PMIX_LOG_STDERR **"pmix.log.stderr"** (**char***)
Log string to **stderr**.

PMIX_LOG_STDOUT **"pmix.log.stdout"** (**char***)
Log string to **stdout**.

PMIX_LOG_SYSLOG **"pmix.log.syslog"** (**char***)
Log data to syslog. Defaults to **ERROR** priority.
PMIX_LOG_MSG **"pmix.log.msg"** (**pmix_byte_object_t**)
Message blob to be sent somewhere.
PMIX_LOG_EMAIL **"pmix.log.email"** (**pmix_data_array_t**)
Log via email based on **pmix_info_t** containing directives.
PMIX_LOG_EMAIL_ADDR **"pmix.log.emaddr"** (**char***)
Comma-delimited list of email addresses that are to receive the message.
PMIX_LOG_EMAIL_SUBJECT **"pmix.log.emsub"** (**char***)
Subject line for email.
PMIX_LOG_EMAIL_MSG **"pmix.log.emmsg"** (**char***)
Message to be included in email.

3.4.23 Debugger attributes

Attributes used to assist debuggers - these are values that can be passed to the **PMIx_Spawn** or **PMIx_Init** APIs. Some may be accessed using the **PMIx_Get** API with the **PMIX_RANK_WILDCARD** rank.

======= **PMIX_DEBUG_STOP_ON_EXEC** **"pmix.dbg.exec"** (**bool**)
Passed to **PMIx_Spawn** to indicate that the specified application is being spawned under debugger, and that the launcher is to pause the resulting application processes on first instruction for debugger attach.
PMIX_DEBUG_STOP_IN_INIT **"pmix.dbg.init"** (**bool**)
Passed to **PMIx_Spawn** to indicate that the specified application is being spawned under debugger, and that the PMIx client library is to pause the resulting application processes during **PMIx_Init** until debugger attach and release.
PMIX_DEBUG_WAIT_FOR_NOTIFY **"pmix.dbg.notify"** (**bool**)
Passed to **PMIx_Spawn** to indicate that the specified application is being spawned under debugger, and that the resulting application processes are to pause at some application-determined location until debugger attach and release.
PMIX_DEBUG_JOB **"pmix.dbg.job"** (**char***)
Namespace of the job to be debugged - provided to the debugger upon launch.
PMIX_DEBUG_WAITING_FOR_NOTIFY **"pmix.dbg.waiting"** (**bool**)
Job to be debugged is waiting for a release - this is not a value accessed using the **PMIx_Get** API.

3.4.24 Resource manager attributes

Attributes used to describe the RM - these are values assigned by the host environment and accessed using the **PMIx_Get** API. The value of the provided namespace is unimportant but should be given as the namespace of the requesting process and a rank of **PMIX_RANK_WILDCARD** used to indicate that the information will be found with the job-level information.

PMIX_RM_NAME **"pmix.rm.name"** (**char***)

String name of the RM.

PMIX_RM_VERSION **"pmix.rm.version"** (**char***)
RM version string.

3.4.25 Environment variable attributes

Attributes used to adjust environment variables - these are values passed to the **PMIx_Spawn** API and are not accessed using the **PMIx_Get** API.

PMIX_SET_ENVAR **"pmix.set.envar"** (**char***)
String "**key=value**" value shall be put into the environment.

PMIX_UNSET_ENVAR **"pmix.unset.envar"** (**char***)
Unset the environment variable specified in the string.

3.4.26 Job Allocation attributes

Attributes used to describe the job allocation - these are values passed to the **PMIx_Allocation_request_nb** API and are not accessed using the **PMIx_Get** API

PMIX_ALLOC_ID **"pmix.alloc.id"** (**char***)
Provide a string identifier for this allocation request which can later be used to query status of the request.

PMIX_ALLOC_NUM_NODES **"pmix.alloc.nnodes"** (**uint64_t**)
The number of nodes.

PMIX_ALLOC_NODE_LIST **"pmix.alloc.nlist"** (**char***)
Regular expression of the specific nodes.

PMIX_ALLOC_NUM_CPUS **"pmix.alloc.ncpus"** (**uint64_t**)
Number of cpus.

PMIX_ALLOC_NUM_CPU_LIST **"pmix.alloc.ncpulist"** (**char***)
Regular expression of the number of cpus for each node.

PMIX_ALLOC_CPU_LIST **"pmix.alloc.cpulist"** (**char***)
Regular expression of the specific cpus indicating the cpus involved.

PMIX_ALLOC_MEM_SIZE **"pmix.alloc.msize"** (**float**)
Number of Megabytes.

PMIX_ALLOC_NETWORK **"pmix.alloc.net"** (**array**)
Array of **pmix_info_t** describing requested network resources. If not given as part of an **pmix_info_t** struct that identifies the involved nodes, then the description will be applied across all nodes in the requestor's allocation.

PMIX_ALLOC_NETWORK_ID **"pmix.alloc.netid"** (**char***)
Name of the network.

PMIX_ALLOC_BANDWIDTH **"pmix.alloc.bw"** (**float**)
Mbits/sec.

PMIX_ALLOC_NETWORK_QOS **"pmix.alloc.netqos"** (**char***)

Quality of service level.

PMIX_ALLOC_TIME **"pmix.alloc.time"** (**uint32_t**)
Time in seconds.

3.4.27 Job control attributes

Attributes used to request control operations on an executing application - these are values passed to the **PMIx_Job_control_nb** API and are not accessed using the **PMIx_Get** API.

PMIX_JOB_CTRL_ID **"pmix.jctrl.id"** (**char***)
Provide a string identifier for this request.

PMIX_JOB_CTRL_PAUSE **"pmix.jctrl.pause"** (**bool**)
Pause the specified processes.

PMIX_JOB_CTRL_RESUME **"pmix.jctrl.resume"** (**bool**)
Resume ("un-pause") the specified processes.

PMIX_JOB_CTRL_CANCEL **"pmix.jctrl.cancel"** (**char***)
Cancel the specified request (**NULL** implies cancel all requests from this requestor).

PMIX_JOB_CTRL_KILL **"pmix.jctrl.kill"** (**bool**)
Forcibly terminate the specified processes and cleanup.

PMIX_JOB_CTRL_RESTART **"pmix.jctrl.restart"** (**char***)
Restart the specified processes using the given checkpoint ID.

PMIX_JOB_CTRL_CHECKPOINT **"pmix.jctrl.ckpt"** (**char***)
Checkpoint the specified processes and assign the given ID to it.

PMIX_JOB_CTRL_CHECKPOINT_EVENT **"pmix.jctrl.ckptev"** (**bool**)
Use event notification to trigger a process checkpoint.

PMIX_JOB_CTRL_CHECKPOINT_SIGNAL **"pmix.jctrl.ckptsig"** (**int**)
Use the given signal to trigger a process checkpoint.

PMIX_JOB_CTRL_CHECKPOINT_TIMEOUT **"pmix.jctrl.ckptsig"** (**int**)
Time in seconds to wait for a checkpoint to complete.

PMIX_JOB_CTRL_CHECKPOINT_METHOD
"pmix.jctrl.ckmethod" (**pmix_data_array_t**)
Array of **pmix_info_t** declaring each method and value supported by this application.

PMIX_JOB_CTRL_SIGNAL **"pmix.jctrl.sig"** (**int**)
Send given signal to specified processes.

PMIX_JOB_CTRL_PROVISION **"pmix.jctrl.pvn"** (**char***)
Regular expression identifying nodes that are to be provisioned.

PMIX_JOB_CTRL_PROVISION_IMAGE **"pmix.jctrl.pvnimg"** (**char***)
Name of the image that is to be provisioned.

PMIX_JOB_CTRL_PREEMPTIBLE **"pmix.jctrl.preempt"** (**bool**)
Indicate that the job can be pre-empted.

PMIX_JOB_CTRL_TERMINATE **"pmix.jctrl.term"** (**bool**)
Politely terminate the specified processes.

3.4.28 Monitoring attributes

Attributes used to control monitoring of an executing application- these are values passed to the **PMIx_Process_monitor_nb** API and are not accessed using the **PMIx_Get** API.

PMIX_MONITOR_ID **"pmix.monitor.id"** (**char***)
: Provide a string identifier for this request.

PMIX_MONITOR_CANCEL **"pmix.monitor.cancel"** (**char***)
: Identifier to be canceled (**NULL** means cancel all monitoring for this process).

PMIX_MONITOR_APP_CONTROL **"pmix.monitor.appctrl"** (**bool**)
: The application desires to control the response to a monitoring event.

PMIX_MONITOR_HEARTBEAT **"pmix.monitor.mbeat"** (**void**)
: Register to have the PMIx server monitor the requestor for heartbeats.

PMIX_SEND_HEARTBEAT **"pmix.monitor.beat"** (**void**)
: Send heartbeat to local PMIx server.

PMIX_MONITOR_HEARTBEAT_TIME **"pmix.monitor.btime"** (**uint32_t**)
: Time in seconds before declaring heartbeat missed.

PMIX_MONITOR_HEARTBEAT_DROPS **"pmix.monitor.bdrop"** (**uint32_t**)
: Number of heartbeats that can be missed before generating the event.

PMIX_MONITOR_FILE **"pmix.monitor.fmon"** (**char***)
: Register to monitor file for signs of life.

PMIX_MONITOR_FILE_SIZE **"pmix.monitor.fsize"** (**bool**)
: Monitor size of given file is growing to determine if the application is running.

PMIX_MONITOR_FILE_ACCESS **"pmix.monitor.faccess"** (**char***)
: Monitor time since last access of given file to determine if the application is running.

PMIX_MONITOR_FILE_MODIFY **"pmix.monitor.fmod"** (**char***)
: Monitor time since last modified of given file to determine if the application is running.

PMIX_MONITOR_FILE_CHECK_TIME **"pmix.monitor.ftime"** (**uint32_t**)
: Time in seconds between checking the file.

PMIX_MONITOR_FILE_DROPS **"pmix.monitor.fdrop"** (**uint32_t**)
: Number of file checks that can be missed before generating the event.

3.5 Callback Functions

PMIx provides blocking and nonblocking versions of most APIs. In the nonblocking versions, a callback is activated upon completion of the the operation. This section describes many of those callbacks.

3.5.1 Release Callback Function

Summary

The **pmix_release_cbfunc_t** is used by the **pmix_modex_cbfunc_t** and **pmix_info_cbfunc_t** operations to indicate that the callback data may be reclaimed/freed by the caller.

Format

PMIx v1.0

C

```
typedef void (*pmix_release_cbfunc_t)
    (void *cbdata)
```

C

INOUT `cbdata`
Callback data passed to original API call (memory reference)

Description

Since the data is "owned" by the host server, provide a callback function to notify the host server that we are done with the data so it can be released.

3.5.2 Modex Callback Function

Summary

The **pmix_modex_cbfunc_t** is used by the **pmix_server_fencenb_fn_t** and **pmix_server_dmodex_req_fn_t** PMIx server operations to return modex BCX data.

PMIx v1.0

C

```
typedef void (*pmix_modex_cbfunc_t)
    (pmix_status_t status,
     const char *data, size_t ndata,
     void *cbdata,
     pmix_release_cbfunc_t release_fn,
     void *release_cbdata)
```

C

IN `status`
Status associated with the operation (handle)

IN `data`
Data to be passed (pointer)

IN **`ndata`**
size of the data (**`size_t`**)

IN **`cbdata`**
Callback data passed to original API call (memory reference)

IN **`release_fn`**
Callback for releasing *data* (function pointer)

IN **`release_cbdata`**
Pointer to be passed to *release_fn* (memory reference)

Description

A callback function that is solely used by PMIx servers, and not clients, to return modex BCX data in response to "fence" and "get" operations. The returned blob contains the data collected from each server participating in the operation.

3.5.3 Spawn Callback Function

Summary

The **`pmix_spawn_cbfunc_t`** is used on the PMIx client side by **`PMIx_Spawn_nb`** and on the PMIx server side by **`pmix_server_spawn_fn_t`** .

PMIx v1.0

C

```
typedef void (*pmix_spawn_cbfunc_t)
    (pmix_status_t status,
     pmix_nspace_t nspace, void *cbdata);
```

C

IN **`status`**
Status associated with the operation (handle)

IN **`nspace`**
Namespace string (**`pmix_nspace_t`**)

IN **`cbdata`**
Callback data passed to original API call (memory reference)

Description

The callback will be executed upon launch of the specified applications in **`PMIx_Spawn_nb`** , or upon failure to launch any of them.

The *status* of the callback will indicate whether or not the spawn succeeded. The *nspace* of the spawned processes will be returned, along with any provided callback data. Note that the returned *nspace* value will not be protected by the PRI upon return from the callback function, so the receiver must copy it if it needs to be retained.

3.5.4 Op Callback Function

Summary

The **pmix_op_cbfunc_t** is used by operations that simply return a status.

PMIx v1.0

C

```
typedef void (*pmix_op_cbfunc_t)
    (pmix_status_t status, void *cbdata);
```

C

IN **status**
Status associated with the operation (handle)

IN **cbdata**
Callback data passed to original API call (memory reference)

Description

Used by a wide range of PMIx API's including **PMIx_Fence_nb** , **pmix_server_client_connected_fn_t** , **PMIx_server_register_nspace** . This callback function is used to return a status to an often nonblocking operation.

3.5.5 Lookup Callback Function

Summary

The **pmix_lookup_cbfunc_t** is used by **PMIx_Lookup_nb** to return data.

PMIx v1.0

C

```
typedef void (*pmix_lookup_cbfunc_t)
    (pmix_status_t status,
     pmix_pdata_t data[], size_t ndata,
     void *cbdata);
```

C

IN **status**
Status associated with the operation (handle)

IN **data**
Array of data returned (**pmix_pdata_t**)

IN **ndata**
Number of elements in the *data* array (**size_t**)

IN **cbdata**
Callback data passed to original API call (memory reference)

Description

A callback function for calls to **PMIx_Lookup_nb** The function will be called upon completion of the command with the *status* indicating the success or failure of the request. Any retrieved data will be returned in an array of **pmix_pdata_t** structs. The namespace and rank of the process that provided each data element is also returned.

Note that these structures will be released upon return from the callback function, so the receiver must copy/protect the data prior to returning if it needs to be retained.

3.5.6 Value Callback Function

Summary

The **pmix_value_cbfunc_t** is used by **PMIx_Get_nb** to return data.

PMIx v1.0

C

```
typedef void (*pmix_value_cbfunc_t)
       (pmix_status_t status,
        pmix_value_t *kv, void *cbdata);
```

C

IN **status**
Status associated with the operation (handle)

IN **kv**
Key/value pair representing the data (**pmix_value_t**)

IN **cbdata**
Callback data passed to original API call (memory reference)

Description

A callback function for calls to **PMIx_Get_nb** . The *status* indicates if the requested data was found or not. A pointer to the **pmix_value_t** structure containing the found data is returned. The pointer will be **NULL** if the requested data was not found.

3.5.7 Info Callback Function

Summary

The **pmix_info_cbfunc_t** is a general information callback used by various APIs.

PMIx v2.0

C

```c
typedef void (*pmix_info_cbfunc_t)
    (pmix_status_t status,
     pmix_info_t info[], size_t ninfo,
     void *cbdata,
     pmix_release_cbfunc_t release_fn,
     void *release_cbdata);
```

C

IN `status`
Status associated with the operation (`pmix_status_t`)

IN `info`
Array of `pmix_info_t` returned by the operation (pointer)

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

IN `cbdata`
Callback data passed to original API call (memory reference)

IN `release_fn`
Function to be called when done with the *info* data (function pointer)

IN `release_cbdata`
Callback data to be passed to *release_fn* (memory reference)

Description

The *status* indicates if requested data was found or not. An array of `pmix_info_t` will contain the key/value pairs.

3.5.8 Event Handler Registration Callback Function

The `pmix_evhdlr_reg_cbfunc_t` callback function.

Advice to users

The PMIx *ad hoc* v1.0 Standard defined an error handler registration callback function with a compatible signature, but with a different type definition function name (pmix_errhandler_reg_cbfunc_t). It was removed from the v2.0 Standard and is not included in this document to avoid confusion.

PMIx v2.0

C

```c
typedef void (*pmix_evhdlr_reg_cbfunc_t)
    (pmix_status_t status,
     size_t evhdlr_ref,
     void *cbdata)
```

C

IN **`status`**
Status indicates if the request was successful or not (**`pmix_status_t`**)

IN **`evhdlr_ref`**
Reference assigned to the event handler by PMIx — this reference * must be used to deregister the err handler (**`size_t`**)

IN **`cbdata`**
Callback data passed to original API call (memory reference)

Description

Define a callback function for calls to **`PMIx_Register_event_handler`**

3.5.9 Notification Handler Completion Callback Function

Summary

The **`pmix_event_notification_cbfunc_fn_t`** is called by event handlers to indicate completion of their operations.

PMIx v2.0

C

```c
typedef void (*pmix_event_notification_cbfunc_fn_t)
    (pmix_status_t status,
     pmix_info_t *results, size_t nresults,
     pmix_op_cbfunc_t cbfunc, void *thiscbdata,
     void *notification_cbdata);
```

C

IN **`status`**
Status returned by the event handler's operation (**`pmix_status_t`**)

IN **`results`**
Results from this event handler's operation on the event (**`pmix_info_t`**)

IN **`nresults`**
Number of elements in the results array (**`size_t`**)

IN **`cbfunc`**
`pmix_op_cbfunc_t` function to be executed when PMIx completes processing the callback (function reference)

IN `thiscbdata`
Callback data that was passed in to the handler (memory reference)

IN `cbdata`
Callback data to be returned when PMIx executes cbfunc (memory reference)

Description

Define a callback by which an event handler can notify the PMIx library that it has completed its response to the notification. The handler is *required* to execute this callback so the library can determine if additional handlers need to be called. The handler shall return **`PMIX_ERR_EVENT_COMPLETE`** if no further action is required. The return status of each event handler and any returned **`pmix_info_t`** structures will be added to the *results* array of **`pmix_info_t`** passed to any subsequent event handlers to help guide their operation.

If non-NULL, the provided callback function will be called to allow the event handler to release the provided info array and execute any other required cleanup operations.

3.5.10 Notification Function

Summary

The **`pmix_notification_fn_t`** is called by PMIx to deliver notification of an event.

Advice to users

The PMIx *ad hoc* v1.0 Standard defined an error notification function with an identical name, but different signature than the v2.0 Standard described below. The *ad hoc* v1.0 version was removed from the v2.0 Standard is not included in this document to avoid confusion.

PMIx v2.0 C

```
typedef void (*pmix_notification_fn_t)
    (size_t evhdlr_registration_id,
     pmix_status_t status,
     const pmix_proc_t *source,
     pmix_info_t info[], size_t ninfo,
     pmix_info_t results[], size_t nresults,
     pmix_event_notification_cbfunc_fn_t cbfunc,
     void *cbdata);
```

C

IN `evhdlr_registration_id`
Registration number of the handler being called (`size_t`)

IN `status`
Status associated with the operation (`pmix_status_t`)

IN `source`
Identifier of the process that generated the event (`pmix_proc_t`). If the source is the SMS, then the nspace will be empty and the rank will be PMIX_RANK_UNDEF

IN `info`
Information describing the event (`pmix_info_t`). This argument will be NULL if no additional information was provided by the event generator.

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

IN `results`
Aggregated results from prior event handlers servicing this event (`pmix_info_t`). This argument will be **`NULL`** if this is the first handler servicing the event, or if no prior handlers provided results.

IN `nresults`
Number of elements in the results array (`size_t`)

IN `cbfunc`
`pmix_event_notification_cbfunc_fn_t` callback function to be executed upon completion of the handler's operation and prior to handler return (function reference).

IN `cbdata`
Callback data to be passed to cbfunc (memory reference)

Description

Note that different RMs may provide differing levels of support for event notification to application processes. Thus, the *info* array may be **`NULL`** or may contain detailed information of the event. It is the responsibility of the application to parse any provided info array for defined key-values if it so desires.

Advice to users

Possible uses of the *info* array include:

- for the host RM to alert the process as to planned actions, such as aborting the session, in response to the reported event
- provide a timeout for alternative action to occur, such as for the application to request an alternate response to the event

For example, the RM might alert the application to the failure of a node that resulted in termination of several processes, and indicate that the overall session will be aborted unless the application requests an alternative behavior in the next 5 seconds. The application then has time to respond with a checkpoint request, or a request to recover from the failure by obtaining replacement nodes and restarting from some earlier checkpoint.

Support for these options is left to the discretion of the host RM. Info keys are included in the common definitions above but may be augmented by environment vendors.

Advice to PMIx server hosts

On the server side, the notification function is used to inform the PMIx server library's host of a detected event in the PMIx server library. Events generated by PMIx clients are communicated to the PMIx server library, but will be relayed to the host via the **`pmix_server_notify_event_fn_t`** function pointer, if provided.

3.5.11 Server Setup Application Callback Function

The **`PMIx_server_setup_application`** callback function.

Summary

Provide a function by which the resource manager can receive application-specific environmental variables and other setup data prior to launch of an application.

Format

PMIx v2.0

C

```c
typedef void (*pmix_setup_application_cbfunc_t)(
                         pmix_status_t status,
                         pmix_info_t info[], size_t ninfo,
                         void *provided_cbdata,
                         pmix_op_cbfunc_t cbfunc, void *cbdata)
```

C

IN `status`
returned status of the request (`pmix_status_t`)

IN `info`
Array of info structures (array of handles)

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

IN `provided_cbdata`
Data originally passed to call to `PMIx_server_setup_application` (memory reference)

IN `cbfunc`
`pmix_op_cbfunc_t` function to be called when processing completed (function reference)

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

Description

Define a function to be called by the PMIx server library for return of application-specific setup data in response to a request from the host RM. The returned *info* array is owned by the PMIx server library and will be free'd when the provided *cbfunc* is called.

3.5.12 Server Direct Modex Response Callback Function

The `PMIx_server_dmodex_request` callback function.

Summary

Provide a function by which the local PMIx server library can return connection and other data posted by local application processes to the host resource manager.

Format

PMIx v1.0

C

```c
typedef void (*pmix_dmodex_response_fn_t)(pmix_status_t status,
                                char *data, size_t sz,
                                void *cbdata);
```

C

IN `status`
Returned status of the request (**`pmix_status_t`**)

IN `data`
Pointer to a data "blob" containing the requested information (handle)

IN `sz`
Number of bytes in the *data* blob (integer)

IN `cbdata`
Data passed into the initial call to **`PMIx_server_dmodex_request`** (memory reference)

Description

Define a function to be called by the PMIx server library for return of information posted by a local application process (via **`PMIx_Put`** with subsequent **`PMIx_Commit`**) in response to a request from the host RM. The returned *data* blob is owned by the PMIx server library and will be free'd upon return from the function.

3.5.13 `pmix_connection_cbfunc_t`

Summary

Callback function for incoming connection request from a local client

Format

PMIx v1.0

C

```c
typedef void (*pmix_connection_cbfunc_t)(
                                    int incoming_sd, void *cbdata)
```

C

IN `incoming_sd`
(integer)

IN `cbdata`
(memory reference)

Description

Callback function for incoming connection requests from local clients - only used by host environments that wish to directly handle socket connection requests.

3.5.14 `pmix_tool_connection_cbfunc_t`

Summary

Callback function for incoming tool connections.

Format

PMIx v2.0

C

```c
typedef void (*pmix_tool_connection_cbfunc_t)(
                                  pmix_status_t status,
                                  pmix_proc_t *proc, void *cbdata)
```

C

IN `status`
`pmix_status_t` value (handle)

IN `proc`
`pmix_proc_t` structure containing the identifier assigned to the tool (handle)

IN `cbdata`
Data to be passed (memory reference)

Description

Callback function for incoming tool connections. The host environment shall provide a namespace/rank identifier for the connecting tool.

Advice to PMIx server hosts

It is assumed that **`rank=0`** will be the normal assignment, but allow for the future possibility of a parallel set of tools connecting, and thus each process requiring a unique rank.

3.5.15 Constant String Functions

Provide a string representation for several types of values. Note that the provided string is statically defined and must NOT be **`free`**'d.

Summary

String representation of a **pmix_status_t** .

PMIx v1.0 C

```c
const char*
PMIx_Error_string(pmix_status_t status);
```

C

Summary

String representation of a **pmix_proc_state_t** .

PMIx v2.0 C

```c
const char*
PMIx_Proc_state_string(pmix_proc_state_t state);
```

C

Summary

String representation of a **pmix_scope_t** .

PMIx v2.0 C

```c
const char*
PMIx_Scope_string(pmix_scope_t scope);
```

C

Summary

String representation of a **pmix_persistence_t** .

PMIx v2.0 C

```c
const char*
PMIx_Persistence_string(pmix_persistence_t persist);
```

C

Summary

String representation of a **pmix_data_range_t** .

PMIx v2.0 C

```c
const char*
PMIx_Data_range_string(pmix_data_range_t range);
```

C

Summary

String representation of a **pmix_info_directives_t** .

PMIx v2.0

C
```c
const char*
PMIx_Info_directives_string(pmix_info_directives_t directives);
```

Summary

String representation of a **pmix_data_type_t** .

PMIx v2.0

C
```c
const char*
PMIx_Data_type_string(pmix_data_type_t type);
```

Summary

String representation of a **pmix_alloc_directive_t** .

PMIx v2.0

C
```c
const char*
PMIx_Alloc_directive_string(pmix_alloc_directive_t directive);
```

CHAPTER 4

Initialization and Finalization

The PMIx library is required to be initialized and finalized around the usage of most of the APIs. The APIs that may be used outside of the initialized and finalized region are noted. All other APIs must be used inside this region.

There are three sets of initialization and finalization functions depending upon the role of the process in the PMIx universe. Each of these functional sets are described in this chapter. Note that a process can only call *one* of the init/finalize functional pairs - e.g., a process that calls the client initialization function cannot also call the tool or server initialization functions, and must call the corresponding client finalize.

Advice to users

Processes that initialize as a server or tool automatically are given access to all client APIs. Server initialization includes setting up the infrastructure to support local clients - thus, it necessarily includes overhead and an increased memory footprint. Tool initialization automatically searches for a server to which it can connect — if declared as a *launcher*, the PMIx library sets up the required "hooks" for other tools (e.g., debuggers) to attach to it.

4.1 Query

The API defined in this section can be used by any PMIx process, regardless of their role in the PMIx universe.

4.1.1 PMIx_Initialized

Format

PMIx v1.0

C

```
int PMIx_Initialized(void)
```

C

A value of **1** (true) will be returned if the PMIx library has been initialized, and **0** (false) otherwise.

Rationale

The return value is an integer for historical reasons as that was the signature of prior PMI libraries.

Description

Check to see if the PMIx library has been initialized using any of the init functions: **PMIx_Init** , **PMIx_server_init** , or **PMIx_tool_init** .

4.1.2 PMIx_Get_version

Summary

Get the PMIx version information.

Format

PMIx v1.0

C

```
const char* PMIx_Get_version(void)
```

C

Description

Get the PMIx version string. Note that the provided string is statically defined and must *not* be free'd.

4.2 Client Initialization and Finalization

Initialization and finalization routines for PMIx clients.

Advice to users

The PMIx *ad hoc* v1.0 Standard defined the **PMIx_Init** function, but modified the function signature in the v1.2 version. The *ad hoc* v1.0 version is not included in this document to avoid confusion.

4.2.1 PMIx_Init

Summary

Initialize the PMIx client library

Format

PMIx v1.2

C

```
pmix_status_t
PMIx_Init(pmix_proc_t *proc,
          pmix_info_t info[], size_t ninfo)
```

C

INOUT `proc`
  proc structure (handle)

IN `info`
  Array of **`pmix_info_t`** structures (array of handles)

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

Returns **`PMIX_SUCCESS`** or a negative value corresponding to a PMIx error constant.

Optional Attributes

The following attributes are optional for implementers of PMIx libraries:

`PMIX_USOCK_DISABLE` **`"pmix.usock.disable"`** (**`bool`**)
  Disable legacy UNIX socket (usock) support If the library supports Unix socket connections, this attribute may be supported for disabling it.

`PMIX_SOCKET_MODE` **`"pmix.sockmode"`** (**`uint32_t`**)
  POSIX *mode_t* (9 bits valid) If the library supports socket connections, this attribute may be supported for setting the socket mode.

`PMIX_SINGLE_LISTENER` **`"pmix.sing.listnr"`** (**`bool`**)
  Use only one rendezvous socket, letting priorities and/or environment parameters select the active transport. If the library supports multiple methods for clients to connect to servers, this attribute may be supported for disabling all but one of them.

`PMIX_TCP_REPORT_URI` **`"pmix.tcp.repuri"`** (**`char*`**)
  If provided, directs that the TCP URI be reported and indicates the desired method of reporting: **`'-'`** for stdout, **`'+'`** for stderr, or filename. If the library supports TCP socket connections, this attribute may be supported for reporting the URI.

`PMIX_TCP_IF_INCLUDE` **`"pmix.tcp.ifinclude"`** (**`char*`**)
  Comma-delimited list of devices and/or CIDR notation to include when establishing the TCP connection. If the library supports TCP socket connections, this attribute may be supported for specifying the interfaces to be used.

`PMIX_TCP_IF_EXCLUDE` **`"pmix.tcp.ifexclude"`** (**`char*`**)
  Comma-delimited list of devices and/or CIDR notation to exclude when establishing the TCP connection. If the library supports TCP socket connections, this attribute may be supported for specifying the interfaces that are *not* to be used.

PMIX_TCP_IPV4_PORT **"pmix.tcp.ipv4"** (**int**)
The IPv4 port to be used. If the library supports IPV4 connections, this attribute may be supported for specifying the port to be used.

PMIX_TCP_IPV6_PORT **"pmix.tcp.ipv6"** (**int**)
The IPv6 port to be used. If the library supports IPV6 connections, this attribute may be supported for specifying the port to be used.

PMIX_TCP_DISABLE_IPV4 **"pmix.tcp.disipv4"** (**bool**)
Set to **true** to disable IPv4 family of addresses. If the library supports IPV4 connections, this attribute may be supported for disabling it.

PMIX_TCP_DISABLE_IPV6 **"pmix.tcp.disipv6"** (**bool**)
Set to **true** to disable IPv6 family of addresses. If the library supports IPV6 connections, this attribute may be supported for disabling it.

PMIX_EVENT_BASE **"pmix.evbase"** (**struct event_base ***)
Pointer to libevent[1] **event_base** to use in place of the internal progress thread.

PMIX_GDS_MODULE **"pmix.gds.mod"** (**char***)
Comma-delimited string of desired modules. This attribute is specific to the PRI and controls only the selection of GDS module for internal use by the process. Module selection for interacting with the server is performed dynamically during the connection process.

Description

Initialize the PMIx client, returning the process identifier assigned to this client's application in the provided **pmix_proc_t** struct. Passing a value of **NULL** for this parameter is allowed if the user wishes solely to initialize the PMIx system and does not require return of the identifier at that time.

When called, the PMIx client shall check for the required connection information of the local PMIx server and establish the connection. If the information is not found, or the server connection fails, then an appropriate error constant shall be returned.

If successful, the function shall return **PMIX_SUCCESS** and fill the *proc* structure (if provided) with the server-assigned namespace and rank of the process within the application. In addition, all startup information provided by the resource manager shall be made available to the client process via subsequent calls to **PMIx_Get** .

The PMIx client library shall be reference counted, and so multiple calls to **PMIx_Init** are allowed by the standard. Thus, one way for an application process to obtain its namespace and rank is to simply call **PMIx_Init** with a non-NULL *proc* parameter. Note that each call to **PMIx_Init** must be balanced with a call to **PMIx_Finalize** to maintain the reference count.

[1] http://libevent.org/

Each call to **PMIx_Init** may contain an array of **pmix_info_t** structures passing directives to the PMIx client library as per the above attributes.

Multiple calls to **PMIx_Init** shall not include conflicting directives. The **PMIx_Init** function will return an error when directives that conflict with prior directives are encountered.

4.2.2 PMIx_Finalize

Summary

Finalize the PMIx client library.

Format

PMIx v1.0

C

```
pmix_status_t
PMIx_Finalize(const pmix_info_t info[], size_t ninfo)
```

C

IN **info**
Array of **pmix_info_t** structures (array of handles)
IN **ninfo**
Number of element in the *info* array (**size_t**)

Returns **PMIX_SUCCESS** or a negative value corresponding to a PMIx error constant.

Optional Attributes

The following attributes are optional for implementers of PMIx libraries:

PMIX_EMBED_BARRIER **"pmix.embed.barrier"** (**bool**)
Execute a blocking fence operation before executing the specified operation. For example, **PMIx_Finalize** does not include an internal barrier operation by default. This attribute would direct **PMIx_Finalize** to execute a barrier as part of the finalize operation.

Description

Decrement the PMIx client library reference count. When the reference count reaches zero, the library will finalize the PMIx client, closing the connection with the local PMIx server and releasing all internally allocated memory.

4.3 Tool Initialization and Finalization

Initialization and finalization routines for PMIx tools.

4.3.1 PMIx_tool_init

Summary

Initialize the PMIx library for operating as a tool.

Format

PMIx v2.0

C

```
pmix_status_t
PMIx_tool_init(pmix_proc_t *proc,
               pmix_info_t info[], size_t ninfo)
```

C

INOUT proc
  pmix_proc_t structure (handle)

IN info
  Array of pmix_info_t structures (array of handles)

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

Returns PMIX_SUCCESS or a negative value corresponding to a PMIx error constant.

Required Attributes

The following attributes are required to be supported by all PMIx libraries:

PMIX_TOOL_NSPACE "pmix.tool.nspace" (char*)
  Name of the namespace to use for this tool.

PMIX_TOOL_RANK "pmix.tool.rank" (uint32_t)
  Rank of this tool.

PMIX_TOOL_DO_NOT_CONNECT "pmix.tool.nocon" (bool)
  The tool wants to use internal PMIx support, but does not want to connect to a PMIx server.

PMIX_SERVER_URI "pmix.srvr.uri" (char*)
  URI of the PMIx server to be contacted.

Optional Attributes

The following attributes are optional for implementers of PMIx libraries:

PMIX_CONNECT_TO_SYSTEM **"pmix.cnct.sys"** (**bool**)
The requestor requires that a connection be made only to a local, system-level PMIx server.

PMIX_CONNECT_SYSTEM_FIRST **"pmix.cnct.sys.first"** (**bool**)
Preferentially, look for a system-level PMIx server first.

PMIX_SERVER_PIDINFO **"pmix.srvr.pidinfo"** (**pid_t**)
PID of the target PMIx server for a tool.

PMIX_TCP_URI **"pmix.tcp.uri"** (**char***)
The URI of the PMIx server to connect to, or a file name containing it in the form of **file:<name of file containing it>**.

PMIX_CONNECT_RETRY_DELAY **"pmix.tool.retry"** (**uint32_t**)
Time in seconds between connection attempts to a PMIx server.

PMIX_CONNECT_MAX_RETRIES **"pmix.tool.mretries"** (**uint32_t**)
Maximum number of times to try to connect to PMIx server.

PMIX_SOCKET_MODE **"pmix.sockmode"** (**uint32_t**)
POSIX *mode_t* (9 bits valid) If the library supports socket connections, this attribute may be supported for setting the socket mode.

PMIX_TCP_REPORT_URI **"pmix.tcp.repuri"** (**char***)
If provided, directs that the TCP URI be reported and indicates the desired method of reporting: **'-'** for stdout, **'+'** for stderr, or filename. If the library supports TCP socket connections, this attribute may be supported for reporting the URI.

PMIX_TCP_IF_INCLUDE **"pmix.tcp.ifinclude"** (**char***)
Comma-delimited list of devices and/or CIDR notation to include when establishing the TCP connection. If the library supports TCP socket connections, this attribute may be supported for specifying the interfaces to be used.

PMIX_TCP_IF_EXCLUDE **"pmix.tcp.ifexclude"** (**char***)
Comma-delimited list of devices and/or CIDR notation to exclude when establishing the TCP connection. If the library supports TCP socket connections, this attribute may be supported for specifying the interfaces that are *not* to be used.

PMIX_TCP_IPV4_PORT **"pmix.tcp.ipv4"** (**int**)
The IPv4 port to be used. If the library supports IPV4 connections, this attribute may be supported for specifying the port to be used.

PMIX_TCP_IPV6_PORT **"pmix.tcp.ipv6"** (**int**)
The IPv6 port to be used. If the library supports IPV6 connections, this attribute may be supported for specifying the port to be used.

PMIX_TCP_DISABLE_IPV4 **"pmix.tcp.disipv4"** (**bool**)
Set to **true** to disable IPv4 family of addresses. If the library supports IPV4 connections, this attribute may be supported for disabling it.

PMIX_TCP_DISABLE_IPV6 **"pmix.tcp.disipv6"** (**bool**)
Set to **true** to disable IPv6 family of addresses. If the library supports IPV6 connections, this attribute may be supported for disabling it.

PMIX_EVENT_BASE **"pmix.evbase"** (**struct event_base ***)
Pointer to libevent[2] **event_base** to use in place of the internal progress thread.

PMIX_GDS_MODULE **"pmix.gds.mod"** (**char***)
Comma-delimited string of desired modules. This attribute is specific to the PRI and controls only the selection of GDS module for internal use by the process. Module selection for interacting with the server is performed dynamically during the connection process.

Description

Initialize the PMIx tool, returning the process identifier assigned to this tool in the provided **pmix_proc_t** struct. The *info* array is used to pass user requests pertaining to the init and subsequent operations. Passing a **NULL** value for the array pointer is supported if no directives are desired.

If called with the **PMIX_TOOL_DO_NOT_CONNECT** attribute, the PMIx tool library will fully initialize but not attempt to connect to a PMIx server. The tool can connect to a server at a later point in time, if desired. In all other cases, the PMIx tool library will attempt to connect to according to the following precedence chain:

- if **PMIX_SERVER_URI** or **PMIX_TCP_URI** is given, then connection will be attempted to the server at the specified URI. Note that it is an error for both of these attributes to be specified. **PMIX_SERVER_URI** is the preferred method as it is more generalized — **PMIX_TCP_URI** is provided for those cases where the user specifically wants to use a TCP transport for the connection and wants to error out if it isn't available or cannot succeed. The PMIx library will return an error if connection fails — it will not proceed to check for other connection options as the user specified a particular one to use

- if **PMIX_SERVER_PIDINFO** was provided, then the tool will search under the directory provided by the PMIX_SERVER_TMPDIR environmental variable for a rendezvous file created by the process corresponding to that PID. The PMIx library will return an error if the rendezvous file cannot be found, or the connection is refused by the server

[2]http://libevent.org/

- if **PMIX_CONNECT_TO_SYSTEM** is given, then the tool will search for a system-level rendezvous file created by a PMIx server in the directory specified by the PMIX_SYSTEM_TMPDIR environmental variable. If found, then the tool will attempt to connect to it. An error is returned if the rendezvous file cannot be found or the connection is refused.
- if **PMIX_CONNECT_SYSTEM_FIRST** is given, then the tool will search for a system-level rendezvous file created by a PMIx server in the directory specified by the PMIX_SYSTEM_TMPDIR environmental variable. If found, then the tool will attempt to connect to it. In this case, no error will be returned if the rendezvous file is not found or connection is refused — the PMIx library will silently continue to the next option
- by default, the tool will search the directory tree under the directory provided by the PMIX_SERVER_TMPDIR environmental variable for rendezvous files of PMIx servers, attempting to connect to each it finds until one accepts the connection. If no rendezvous files are found, or all contacted servers refuse connection, then the PMIx library will return an error.

If successful, the function will return **PMIX_SUCCESS** and will fill the provided structure (if provided) with the server-assigned namespace and rank of the tool. Note that each connection attempt in the above precedence chain will retry (with delay between each retry) a number of times according to the values of the corresponding attributes. Default is no retries.

Note that the PMIx tool library is referenced counted, and so multiple calls to **PMIx_tool_init** are allowed. Thus, one way to obtain the namespace and rank of the process is to simply call **PMIx_tool_init** with a non-NULL parameter.

4.3.2 PMIx_tool_finalize

Summary

Finalize the PMIx library for a tool connection.

Format

PMIx v2.0

C

```
pmix_status_t
PMIx_tool_finalize(void)
```

C

Returns **PMIX_SUCCESS** or a negative value corresponding to a PMIx error constant.

Description

Finalize the PMIx tool library, closing the connection to the server. An error code will be returned if, for some reason, the connection cannot be cleanly terminated — in this case, the connection is dropped.

4.4 Server Initialization and Finalization

Initialization and finalization routines for PMIx servers.

4.4.1 `PMIx_server_init`

Summary

Initialize the PMIx server.

Format

PMIx v1.0

C

```c
pmix_status_t
PMIx_server_init(pmix_server_module_t *module,
                 pmix_info_t info[], size_t ninfo)
```

C

INOUT `module`
  `pmix_server_module_t` structure (handle)

IN `info`
  Array of `pmix_info_t` structures (array of handles)

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

Returns `PMIX_SUCCESS` or a negative value corresponding to a PMIx error constant.

Required Attributes

The following attributes are required to be supported by all PMIx libraries:

`PMIX_SERVER_NSPACE` `"pmix.srv.nspace"` (`char*`)
  Name of the namespace to use for this PMIx server.

`PMIX_SERVER_RANK` `"pmix.srv.rank"` (`pmix_rank_t`)
  Rank of this PMIx server

`PMIX_SERVER_TMPDIR` `"pmix.srvr.tmpdir"` (`char*`)
  Top-level temporary directory for all *client* processes connected to this server, and where the PMIx server will place its *tool* rendezvous point and contact information.

`PMIX_SYSTEM_TMPDIR` `"pmix.sys.tmpdir"` (`char*`)
  Temporary directory for this system, and where a PMIx server that declares itself to be a system-level server will place a *tool* rendezvous point and contact information.

`PMIX_SERVER_TOOL_SUPPORT` `"pmix.srvr.tool"` (`bool`)

The host RM wants to declare itself as willing to accept tool connection requests.

PMIX_SERVER_SYSTEM_SUPPORT **"pmix.srvr.sys"** (**bool**)
The host RM wants to declare itself as being the local system server for PMIx connection requests.

Optional Attributes

The following attributes are optional for implementers of PMIx libraries:

PMIX_USOCK_DISABLE **"pmix.usock.disable"** (**bool**)
Disable legacy UNIX socket (usock) support If the library supports Unix socket connections, this attribute may be supported for disabling it.

PMIX_SOCKET_MODE **"pmix.sockmode"** (**uint32_t**)
POSIX *mode_t* (9 bits valid) If the library supports socket connections, this attribute may be supported for setting the socket mode.

PMIX_TCP_REPORT_URI **"pmix.tcp.repuri"** (**char***)
If provided, directs that the TCP URI be reported and indicates the desired method of reporting: **'-'** for stdout, **'+'** for stderr, or filename. If the library supports TCP socket connections, this attribute may be supported for reporting the URI.

PMIX_TCP_IF_INCLUDE **"pmix.tcp.ifinclude"** (**char***)
Comma-delimited list of devices and/or CIDR notation to include when establishing the TCP connection. If the library supports TCP socket connections, this attribute may be supported for specifying the interfaces to be used.

PMIX_TCP_IF_EXCLUDE **"pmix.tcp.ifexclude"** (**char***)
Comma-delimited list of devices and/or CIDR notation to exclude when establishing the TCP connection. If the library supports TCP socket connections, this attribute may be supported for specifying the interfaces that are *not* to be used.

PMIX_TCP_IPV4_PORT **"pmix.tcp.ipv4"** (**int**)
The IPv4 port to be used. If the library supports IPV4 connections, this attribute may be supported for specifying the port to be used.

PMIX_TCP_IPV6_PORT **"pmix.tcp.ipv6"** (**int**)
The IPv6 port to be used. If the library supports IPV6 connections, this attribute may be supported for specifying the port to be used.

PMIX_TCP_DISABLE_IPV4 **"pmix.tcp.disipv4"** (**bool**)
Set to **true** to disable IPv4 family of addresses. If the library supports IPV4 connections, this attribute may be supported for disabling it.

PMIX_TCP_DISABLE_IPV6 **"pmix.tcp.disipv6"** (**bool**)
Set to **true** to disable IPv6 family of addresses. If the library supports IPV6 connections, this attribute may be supported for disabling it.

PMIX_SERVER_REMOTE_CONNECTIONS **"pmix.srvr.remote"** (**bool**)
Allow connections from remote tools. Forces the PMIx server to not exclusively use loopback device. If the library supports connections from remote tools, this attribute may be supported for enabling or disabling it.

PMIX_EVENT_BASE **"pmix.evbase"** (**struct event_base ***)
Pointer to libevent[3] **event_base** to use in place of the internal progress thread.

PMIX_GDS_MODULE **"pmix.gds.mod"** (**char***)
Comma-delimited string of desired modules. This attribute is specific to the PRI and controls only the selection of GDS module for internal use by the process. Module selection for interacting with the server is performed dynamically during the connection process.

Description

Initialize the PMIx server support library, and provide a pointer to a **pmix_server_module_t** structure containing the caller's callback functions. The array of **pmix_info_t** structs is used to pass additional info that may be required by the server when initializing. For example, it may include the **PMIX_SERVER_TOOL_SUPPORT** key, thereby indicating that the daemon is willing to accept connection requests from tools.

Advice to PMIx server hosts

Providing a value of **NULL** for the *module* argument is permitted, as is passing an empty *module* structure. Doing so indicates that the host environment will not provide support for multi-node operations such as **PMIx_Fence**, but does intend to support local clients access to information.

4.4.2 PMIx_server_finalize

Summary

Finalize the PMIx server library.

Format

PMIx v1.0

C

```
pmix_status_t
PMIx_server_finalize(void)
```

C

Returns **PMIX_SUCCESS** or a negative value corresponding to a PMIx error constant.

[3] http://libevent.org/

Description

Finalize the PMIx server support library, terminating all connections to attached tools and any local clients. All memory usage is released.

CHAPTER 5

Key/Value Management

Management of key-value pairs in PMIx is a distributed responsibility. While the stated objective of the PMIx community is to eliminate collective operations, it is recognized that the traditional method of publishing/exchanging data must be supported until that objective can be met. This method relies on processes to discover and publish their local information which is collected by the local PMIx server library. Global exchange of the published information is then executed via a collective operation performed by the host SMS servers.

Keys are required to be unique within a specific level of informarion as defined in 3.4.10. For example, a value for **PMIX_NUM_NODES** can be specified for each of the **session**, **job**, and **application** levels. However, subsequently specifying another value for that attribute in the **session** level will overwrite the prior value.

5.1 Setting and Accessing Key/Value Pairs

5.1.1 PMIx_Put

Summary

Push a key/value pair into the client's namespace.

Format

PMIx v1.0

C

```c
pmix_status_t
PMIx_Put(pmix_scope_t scope,
         const pmix_key_t key,
         pmix_value_t *val)
```

C

IN **scope**
Distribution scope of the provided value (handle)

IN **key**
key (**pmix_key_t**)

IN **value**
Reference to a **pmix_value_t** structure (handle)

Returns **PMIX_SUCCESS** or a negative value corresponding to a PMIx error constant.

Description

Push a value into the client's namespace. The client's PMIx library will cache the information locally until **PMIx_Commit** is called.

The provided *scope* is passed to the local PMIx server, which will distribute the data to other processes according to the provided scope. The **pmix_scope_t** values are defined in Section 3.2.9 on page 27. Specific implementations may support different scope values, but all implementations must support at least **PMIX_GLOBAL**.

The **pmix_value_t** structure supports both string and binary values. PMIx implementations will support heterogeneous environments by properly converting binary values between host architectures, and will copy the provided *value* into internal memory.

Advice to PMIx library implementers

The PMIx server library will properly pack/unpack data to accommodate heterogeneous environments. The host SMS is not involved in this action. The *value* argument must be copied - the caller is free to release it following return from the function.

Advice to users

The value is copied by the PMIx client library. Thus, the application is free to release and/or modify the value once the call to **PMIx_Put** has completed.

Note that keys starting with a string of "**pmix**" are exclusively reserved for the PMIx standard and must not be used in calls to **PMIx_Put** . Thus, applications should never use a defined "PMIX_" attribute as the key in a call to **PMIx_Put** .

5.1.2 PMIx_Get

Summary

Retrieve a key/value pair from the client's namespace.

Format

PMIx v1.0

C

```c
pmix_status_t
PMIx_Get(const pmix_proc_t *proc, const pmix_key_t key,
         const pmix_info_t info[], size_t ninfo,
         pmix_value_t **val)
```

C

IN `proc`
process reference (handle)

IN `key`
key to retrieve (`pmix_key_t`)

IN `info`
Array of info structures (array of handles)

IN `ninfo`
Number of element in the *info* array (integer)

OUT `val`
value (handle)

Returns `PMIX_SUCCESS` or a negative value corresponding to a PMIx error constant.

Required Attributes

The following attributes are required to be supported by all PMIx libraries:

`PMIX_OPTIONAL` `"pmix.optional"` (`bool`)
Look only in the client's local data store for the requested value - do not request data from the PMIx server if not found.

`PMIX_IMMEDIATE` `"pmix.immediate"` (`bool`)
Specified operation should immediately return an error from the PMIx server if the requested data cannot be found - do not request it from the host RM.

`PMIX_DATA_SCOPE` `"pmix.scope"` (`pmix_scope_t`)
Scope of the data to be found in a `PMIx_Get` call.

`PMIX_SESSION_INFO` `"pmix.ssn.info"` (`bool`)
Return information about the specified session. If information about a session other than the one containing the requesting process is desired, then the attribute array must contain a `PMIX_SESSION_ID` attribute identifying the desired target.

`PMIX_JOB_INFO` `"pmix.job.info"` (`bool`)

Return information about the specified job or namespace. If information about a job or namespace other than the one containing the requesting process is desired, then the attribute array must contain a **PMIX_JOBID** or **PMIX_NSPACE** attribute identifying the desired target. Similarly, if information is requested about a job or namespace in a session other than the one containing the requesting process, then an attribute identifying the target session must be provided.

PMIX_APP_INFO **"pmix.app.info"** (**bool**)
Return information about the specified application. If information about an application other than the one containing the requesting process is desired, then the attribute array must contain a **PMIX_APPNUM** attribute identifying the desired target. Similarly, if information is requested about an application in a job or session other than the one containing the requesting process, then attributes identifying the target job and/or session must be provided.

PMIX_NODE_INFO **"pmix.node.info"** (**bool**)
Return information about the specified node. If information about a node other than the one containing the requesting process is desired, then the attribute array must contain either the **PMIX_NODEID** or **PMIX_HOSTNAME** attribute identifying the desired target.

Optional Attributes

The following attributes are optional for host environments:

PMIX_TIMEOUT **"pmix.timeout"** (**int**)
Time in seconds before the specified operation should time out (*0* indicating infinite) in error. The timeout parameter can help avoid "hangs" due to programming errors that prevent the target process from ever exposing its data.

Advice to PMIx library implementers

We recommend that implementation of the **PMIX_TIMEOUT** attribute be left to the host environment due to race condition considerations between delivery of the data by the host environment versus internal timeout in the PMIx server library. Implementers that choose to support **PMIX_TIMEOUT** directly in the PMIx server library must take care to resolve the race condition and should avoid passing **PMIX_TIMEOUT** to the host environment so that multiple competing timeouts are not created.

Description

Retrieve information for the specified *key* as published by the process identified in the given **pmix_proc_t** , returning a pointer to the value in the given address.

This is a blocking operation - the caller will block until either the specified data becomes available from the specified rank in the *proc* structure or the operation times out should the **PMIX_TIMEOUT** attribute have been given. The caller is responsible for freeing all memory associated with the returned *value* when no longer required.

The *info* array is used to pass user requests regarding the get operation.

Advice to users

Information provided by the PMIx server at time of process start is accessed by providing the namespace of the job with the rank set to **PMIX_RANK_WILDCARD** . The list of data referenced in this way is maintained on the PMIx web site at https://pmix.org/support/faq/wildcard-rank-access/ but includes items such as the number of processes in the namespace (**PMIX_JOB_SIZE**), total available slots in the allocation (**PMIX_UNIV_SIZE**), and the number of nodes in the allocation (**PMIX_NUM_NODES**).

Data posted by a process via **PMIx_Put** needs to be retrieved by specifying the rank of the posting process. All other information is retrievable using a rank of **PMIX_RANK_WILDCARD** when the information being retrieved refers to something non-rank specific (e.g., number of processes on a node, number of processes in a job), and using the rank of the relevant process when requesting information that is rank-specific (e.g., the URI of the process, or the node upon which it is executing). Each subsection of Section 3.4 indicates the appropriate rank value for referencing the defined attribute.

5.1.3 PMIx_Get_nb

Summary

Nonblocking **PMIx_Get** operation.

Format

PMIx v1.0

C

```
pmix_status_t
PMIx_Get_nb(const pmix_proc_t *proc, const char key[],
            const pmix_info_t info[], size_t ninfo,
            pmix_value_cbfunc_t cbfunc, void *cbdata)
```

C

IN `proc`
process reference (handle)

IN `key`
key to retrieve (string)

IN `info`
Array of info structures (array of handles)

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

IN `cbfunc`
Callback function (function reference)

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

Function returns either:

- `PMIX_SUCCESS` indicating that the request has been accepted for processing and the provided callback function will be executed upon completion of the operation
- a non-zero PMIx error constant indicating a reason for the request to have been rejected. In this case, the provided callback function will *not* be executed

If executed, the status returned in the provided callback function will be one of the following constants:

- `PMIX_SUCCESS` The requested data has been returned
- `PMIX_ERR_NOT_FOUND` The requested data was not available
- a non-zero PMIx error constant indicating a reason for the request's failure

Required Attributes

The following attributes are required to be supported by all PMIx libraries:

`PMIX_OPTIONAL` `"pmix.optional"` (`bool`)
Look only in the client's local data store for the requested value - do not request data from the PMIx server if not found.

`PMIX_IMMEDIATE` `"pmix.immediate"` (`bool`)

Specified operation should immediately return an error from the PMIx server if the requested data cannot be found - do not request it from the host RM.

`PMIX_DATA_SCOPE` **`"pmix.scope"`** (**`pmix_scope_t`**)
Scope of the data to be found in a **`PMIx_Get`** call.

`PMIX_SESSION_INFO` **`"pmix.ssn.info"`** (**`bool`**)
Return information about the specified session. If information about a session other than the one containing the requesting process is desired, then the attribute array must contain a **`PMIX_SESSION_ID`** attribute identifying the desired target.

`PMIX_JOB_INFO` **`"pmix.job.info"`** (**`bool`**)
Return information about the specified job or namespace. If information about a job or namespace other than the one containing the requesting process is desired, then the attribute array must contain a **`PMIX_JOBID`** or **`PMIX_NSPACE`** attribute identifying the desired target. Similarly, if information is requested about a job or namespace in a session other than the one containing the requesting process, then an attribute identifying the target session must be provided.

`PMIX_APP_INFO` **`"pmix.app.info"`** (**`bool`**)
Return information about the specified application. If information about an application other than the one containing the requesting process is desired, then the attribute array must contain a **`PMIX_APPNUM`** attribute identifying the desired target. Similarly, if information is requested about an application in a job or session other than the one containing the requesting process, then attributes identifying the target job and/or session must be provided.

`PMIX_NODE_INFO` **`"pmix.node.info"`** (**`bool`**)
Return information about the specified node. If information about a node other than the one containing the requesting process is desired, then the attribute array must contain either the **`PMIX_NODEID`** or **`PMIX_HOSTNAME`** attribute identifying the desired target.

Optional Attributes

The following attributes are optional for host environments that support this operation:

`PMIX_TIMEOUT` **`"pmix.timeout"`** (**`int`**)
Time in seconds before the specified operation should time out (*0* indicating infinite) in error. The timeout parameter can help avoid “hangs” due to programming errors that prevent the target process from ever exposing its data.

Advice to PMIx library implementers

We recommend that implementation of the **PMIX_TIMEOUT** attribute be left to the host environment due to race condition considerations between delivery of the data by the host environment versus internal timeout in the PMIx server library. Implementers that choose to support **PMIX_TIMEOUT** directly in the PMIx server library must take care to resolve the race condition and should avoid passing **PMIX_TIMEOUT** to the host environment so that multiple competing timeouts are not created.

Description

The callback function will be executed once the specified data becomes available from the identified process and retrieved by the local server. The *info* array is used as described by the **PMIx_Get** routine.

Advice to users

Information provided by the PMIx server at time of process start is accessed by providing the namespace of the job with the rank set to **PMIX_RANK_WILDCARD** . The list of data referenced in this way is maintained on the PMIx web site at https://pmix.org/support/faq/wildcard-rank-access/ but includes items such as the number of processes in the namespace (**PMIX_JOB_SIZE**), total available slots in the allocation (**PMIX_UNIV_SIZE**), and the number of nodes in the allocation (**PMIX_NUM_NODES**).

In general, only data posted by a process via **PMIx_Put** needs to be retrieved by specifying the rank of the posting process. All other information is retrievable using a rank of **PMIX_RANK_WILDCARD** . See 3.4.10 for an explanation regarding use of the *level* attributes.

5.1.4 `PMIx_Store_internal`

Summary

Store some data locally for retrieval by other areas of the proc.

Format

PMIx v1.0

C
```c
pmix_status_t
PMIx_Store_internal(const pmix_proc_t *proc,
                    const pmix_key_t key,
                    pmix_value_t *val);
```
C

IN `proc`
process reference (handle)

IN `key`
key to retrieve (string)

IN `val`
Value to store (handle)

Returns **`PMIX_SUCCESS`** or a negative value corresponding to a PMIx error constant.

Description

Store some data locally for retrieval by other areas of the proc. This is data that has only internal scope - it will never be “pushed” externally.

5.1.5 Accessing information: examples

This section provides examples illustrating methods for accessing information at various levels. The intent of the examples is not to provide comprehensive coding guidance, but rather to illustrate how **`PMIx_Get`** can be used to obtain information on a **`session`** , **`job`** , **`application`** , process, and node.

5.1.5.1 Session-level information

The **`PMIx_Get`** API does not include an argument for specifying the **`session`** associated with the information being requested. Information regarding the session containing the requestor can be obtained by the following methods:

- for session-level attributes (e.g., **`PMIX_UNIV_SIZE`**), specifying the requestor’s namespace and a rank of **`PMIX_RANK_WILDCARD`** ; or
- for non-specific attributes (e.g., **`PMIX_NUM_NODES`**), including the **`PMIX_SESSION_INFO`** attribute to indicate that the session-level information for that attribute is being requested

Example requests are shown below:

C

```c
pmix_info_t info;
pmix_value_t *value;
pmix_status_t rc;
pmix_proc_t myproc, wildcard;

/* initialize the client library */
PMIx_Init(&myproc, NULL, 0);

/* get the #slots in our session */
PMIX_PROC_LOAD(&wildcard, myproc.nspace, PMIX_RANK_WILDCARD);
rc = PMIx_Get(&wildcard, PMIX_UNIV_SIZE, NULL, 0, &value);

/* get the #nodes in our session */
PMIX_INFO_LOAD(&info, PMIX_SESSION_INFO, NULL, PMIX_BOOL);
rc = PMIx_Get(&wildcard, PMIX_NUM_NODES, &info, 1, &value);
```

C

Information regarding a different session can be requested by either specifying the namespace and a rank of **PMIX_RANK_WILDCARD** for a process in the target session, or adding the **PMIX_SESSION_ID** attribute identifying the target session. In the latter case, the *proc* argument to **PMIx_Get** will be ignored:

C

```c
pmix_info_t info[2];
pmix_value_t *value;
pmix_status_t rc;
pmix_proc_t myproc;
uint32_t sid;

/* initialize the client library */
PMIx_Init(&myproc, NULL, 0);

/* get the #nodes in a different session */
sid = 12345;
PMIX_INFO_LOAD(&info[0], PMIX_SESSION_INFO, NULL, PMIX_BOOL);
PMIX_INFO_LOAD(&info[1], PMIX_SESSION_ID, &sid, PMIX_UINT32);
rc = PMIx_Get(&myproc, PMIX_NUM_NODES, info, 2, &value);
```

C

5.1.5.2 Job-level information

Information regarding a job can be obtained by the following methods:

- for job-level attributes (e.g., **PMIX_JOB_SIZE** or **PMIX_JOB_NUM_APPS**), specifying the namespace of the job and a rank of **PMIX_RANK_WILDCARD** for the *proc* argument to **PMIx_Get**; or
- for non-specific attributes (e.g., **PMIX_NUM_NODES**), including the **PMIX_JOB_INFO** attribute to indicate that the job-level information for that attribute is being requested

Example requests are shown below:

C

```c
pmix_info_t info;
pmix_value_t *value;
pmix_status_t rc;
pmix_proc_t myproc, wildcard;

/* initialize the client library */
PMIx_Init(&myproc, NULL, 0);

/* get the #apps in our job */
PMIX_PROC_LOAD(&wildcard, myproc.nspace, PMIX_RANK_WILDCARD);
rc = PMIx_Get(&wildcard, PMIX_JOB_NUM_APPS, NULL, 0, &value);

/* get the #nodes in our job */
PMIX_INFO_LOAD(&info, PMIX_JOB_INFO, NULL, PMIX_BOOL);
rc = PMIx_Get(&wildcard, PMIX_NUM_NODES, &info, 1, &value);
```

C

5.1.5.3 Application-level information

Information regarding an application can be obtained by the following methods:

- for application-level attributes (e.g., **PMIX_APP_SIZE**), specifying the namespace and rank of a process within that application;
- for application-level attributes (e.g., **PMIX_APP_SIZE**), including the **PMIX_APPNUM** attribute specifying the application whose information is being requested. In this case, the namespace field of the *proc* argument is used to reference the **job** containing the application - the **rank** field is ignored;
- or application-level attributes (e.g., **PMIX_APP_SIZE**), including the **PMIX_APPNUM** and **PMIX_NSPACE** or **PMIX_JOBID** attributes specifying the job/application whose information is being requested. In this case, the *proc* argument is ignored;

- for non-specific attributes (e.g., **PMIX_NUM_NODES**), including the **PMIX_APP_INFO** attribute to indicate that the application-level information for that attribute is being requested

Example requests are shown below:

C

```c
pmix_info_t info;
pmix_value_t *value;
pmix_status_t rc;
pmix_proc_t myproc, otherproc;
uint32_t appsize, appnum;

/* initialize the client library */
PMIx_Init(&myproc, NULL, 0);

/* get the #processes in our application */
rc = PMIx_Get(&myproc, PMIX_APP_SIZE, NULL, 0, &value);
appsize = value->data.uint32;

/* get the #nodes in an application containing "otherproc".
 * Note that the rank of a process in the other application
 * must be obtained first - a simple method is shown here */

/* assume for this example that we are in the first application
 * and we want the #nodes in the second application - use the
 * rank of the first process in that application, remembering
 * that ranks start at zero */
PMIX_PROC_LOAD(&otherproc, myproc.nspace, appsize);

PMIX_INFO_LOAD(&info, PMIX_APP_INFO, NULL, PMIX_BOOL);
rc = PMIx_Get(&otherproc, PMIX_NUM_NODES, &info, 1, &value);

/* alternatively, we can directly ask for the #nodes in
 * the second application in our job, again remembering that
 * application numbers start with zero */
appnum = 1;
PMIX_INFO_LOAD(&appinfo[0], PMIX_APP_INFO, NULL, PMIX_BOOL);
PMIX_INFO_LOAD(&appinfo[1], PMIX_APPNUM, &appnum, PMIX_UINT32);
rc = PMIx_Get(&myproc, PMIX_NUM_NODES, appinfo, 2, &value);
```

C

5.1.5.4 Process-level information

Process-level information is accessed by providing the namespace and rank of the target process. In the absence of any directive as to the level of information being requested, the PMIx library will always return the process-level value.

5.1.5.5 Node-level information

Information regarding a node within the system can be obtained by the following methods:

- for node-level attributes (e.g., **PMIX_NODE_SIZE**), specifying the namespace and rank of a process executing on the target node;
- for node-level attributes (e.g., **PMIX_NODE_SIZE**), including the **PMIX_NODEID** or **PMIX_HOSTNAME** attribute specifying the node whose information is being requested. In this case, the *proc* argument's values are ignored; or
- for non-specific attributes (e.g., **PMIX_NUM_SLOTS**), including the **PMIX_NODE_INFO** attribute to indicate that the node-level information for that attribute is being requested

Example requests are shown below:

C

```c
pmix_info_t info[2];
pmix_value_t *value;
pmix_status_t rc;
pmix_proc_t myproc, otherproc;
uint32_t nodeid;

/* initialize the client library */
PMIx_Init(&myproc, NULL, 0);

/* get the #procs on our node */
rc = PMIx_Get(&myproc, PMIX_NODE_SIZE, NULL, 0, &value);

/* get the #slots on another node */
PMIX_INFO_LOAD(&info[0], PMIX_NODE_INFO, NULL, PMIX_BOOL);
PMIX_INFO_LOAD(&info[1], PMIX_HOSTNAME, "remotehost", PMIX_STRING);
rc = PMIx_Get(&myproc, PMIX_NUM_SLOTS, info, 2, &value);

```

C

Advice to users

An explanation of the use of **PMIx_Get** versus **PMIx_Query_info_nb** is provided in 7.1.3.1.

5.2 Exchanging Key/Value Pairs

The APIs defined in this section push key/value pairs from the client to the local PMIx server, and circulate the data between PMIx servers for subsequent retrieval by the local clients.

5.2.1 `PMIx_Commit`

Summary

Push all previously **`PMIx_Put`** values to the local PMIx server.

Format

PMIx v1.0

C

```
pmix_status_t PMIx_Commit(void)
```

C

Returns **`PMIX_SUCCESS`** or a negative value corresponding to a PMIx error constant.

Description

This is an asynchronous operation. The PRI will immediately return to the caller while the data is transmitted to the local server in the background.

Advice to users

The local PMIx server will cache the information locally - i.e., the committed data will not be circulated during **`PMIx_Commit`** . Availability of the data upon completion of **`PMIx_Commit`** is therefore implementation-dependent.

5.2.2 `PMIx_Fence`

Summary

Execute a blocking barrier across the processes identified in the specified array, collecting information posted via **`PMIx_Put`** as directed.

Format

PMIx v1.0

C

```
pmix_status_t
PMIx_Fence(const pmix_proc_t procs[], size_t nprocs,
           const pmix_info_t info[], size_t ninfo)
```

C

IN `procs`
Array of `pmix_proc_t` structures (array of handles)

IN `nprocs`
Number of element in the *procs* array (integer)

IN `info`
Array of info structures (array of handles)

IN `ninfo`
Number of element in the *info* array (integer)

Returns `PMIX_SUCCESS` or a negative value corresponding to a PMIx error constant.

Required Attributes

The following attributes are required to be supported by all PMIx libraries:

`PMIX_COLLECT_DATA` `"pmix.collect"` (`bool`)
Collect data and return it at the end of the operation.

Optional Attributes

The following attributes are optional for host environments:

`PMIX_TIMEOUT` `"pmix.timeout"` (`int`)
Time in seconds before the specified operation should time out (*0* indicating infinite) in error. The timeout parameter can help avoid “hangs” due to programming errors that prevent the target process from ever exposing its data.

`PMIX_COLLECTIVE_ALGO` `"pmix.calgo"` (`char*`)
Comma-delimited list of algorithms to use for the collective operation. PMIx does not impose any requirements on a host environment’s collective algorithms. Thus, the acceptable values for this attribute will be environment-dependent - users are encouraged to check their host environment for supported values.

`PMIX_COLLECTIVE_ALGO_REQD` `"pmix.calreqd"` (`bool`)
If `true`, indicates that the requested choice of algorithm is mandatory.

Advice to PMIx library implementers

We recommend that implementation of the **PMIX_TIMEOUT** attribute be left to the host environment due to race condition considerations between completion of the operation versus internal timeout in the PMIx server library. Implementers that choose to support **PMIX_TIMEOUT** directly in the PMIx server library must take care to resolve the race condition and should avoid passing **PMIX_TIMEOUT** to the host environment so that multiple competing timeouts are not created.

Description

Passing a **NULL** pointer as the *procs* parameter indicates that the fence is to span all processes in the client's namespace. Each provided **pmix_proc_t** struct can pass **PMIX_RANK_WILDCARD** to indicate that all processes in the given namespace are participating.

The *info* array is used to pass user requests regarding the fence operation.

Note that for scalability reasons, the default behavior for **PMIx_Fence** is to *not* collect the data.

Advice to PMIx library implementers

PMIx_Fence and its non-blocking form are both *collective* operations. Accordingly, the PMIx server library is required to aggregate participation by local clients, passing the request to the host environment once all local participants have executed the API.

Advice to PMIx server hosts

The host will receive a single call for each collective operation. It is the responsibility of the host to identify the nodes containing participating processes, execute the collective across all participating nodes, and notify the local PMIx server library upon completion of the global collective.

5.2.3 PMIx_Fence_nb

Summary

Execute a nonblocking **PMIx_Fence** across the processes identified in the specified array of processes, collecting information posted via **PMIx_Put** as directed.

Format

PMIx v1.0

C

```
pmix_status_t
PMIx_Fence_nb(const pmix_proc_t procs[], size_t nprocs,
              const pmix_info_t info[], size_t ninfo,
              pmix_op_cbfunc_t cbfunc, void *cbdata)
```

C

IN `procs`
  Array of `pmix_proc_t` structures (array of handles)

IN `nprocs`
  Number of element in the *procs* array (integer)

IN `info`
  Array of info structures (array of handles)

IN `ninfo`
  Number of element in the *info* array (integer)

IN `cbfunc`
  Callback function (function reference)

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

Returns one of the following:

- `PMIX_SUCCESS` , indicating that the request is being processed by the host environment - result will be returned in the provided *cbfunc*. Note that the library *must not* invoke the callback function prior to returning from the API.
- a PMIx error constant indicating an error in the input - the *cbfunc* will *not* be called

Required Attributes

The following attributes are required to be supported by all PMIx libraries:

`PMIX_COLLECT_DATA` `"pmix.collect"` (`bool`)
  Collect data and return it at the end of the operation.

Optional Attributes

The following attributes are optional for host environments that support this operation:

PMIX_TIMEOUT **"pmix.timeout"** (**int**)
: Time in seconds before the specified operation should time out (*0* indicating infinite) in error. The timeout parameter can help avoid "hangs" due to programming errors that prevent the target process from ever exposing its data.

PMIX_COLLECTIVE_ALGO **"pmix.calgo"** (**char***)
: Comma-delimited list of algorithms to use for the collective operation. PMIx does not impose any requirements on a host environment's collective algorithms. Thus, the acceptable values for this attribute will be environment-dependent - users are encouraged to check their host environment for supported values.

PMIX_COLLECTIVE_ALGO_REQD **"pmix.calreqd"** (**bool**)
: If **true**, indicates that the requested choice of algorithm is mandatory.

Advice to PMIx library implementers

We recommend that implementation of the **PMIX_TIMEOUT** attribute be left to the host environment due to race condition considerations between completion of the operation versus internal timeout in the PMIx server library. Implementers that choose to support **PMIX_TIMEOUT** directly in the PMIx server library must take care to resolve the race condition and should avoid passing **PMIX_TIMEOUT** to the host environment so that multiple competing timeouts are not created.

Description

Nonblocking **PMIx_Fence** routine. Note that the function will return an error if a **NULL** callback function is given.

Note that for scalability reasons, the default behavior for **PMIx_Fence_nb** is to *not* collect the data.

See the **PMIx_Fence** description for further details.

5.3 Publish and Lookup Data

The APIs defined in this section publish data from one client that can be later exchanged and looked up by another client.

Advice to PMIx library implementers

PMIx libraries that support any of the functions in this section are required to support *all* of them.

Advice to PMIx server hosts

Host environments that support any of the functions in this section are required to support *all* of them.

5.3.1 `PMIx_Publish`

Summary

Publish data for later access via **`PMIx_Lookup`** .

Format

PMIx v1.0

C

```
pmix_status_t
PMIx_Publish(const pmix_info_t info[], size_t ninfo)
```

C

IN `info`
Array of info structures (array of handles)

IN `ninfo`
Number of element in the *info* array (integer)

Returns `PMIX_SUCCESS` or a negative value corresponding to a PMIx error constant.

Required Attributes

PMIx libraries are not required to directly support any attributes for this function. However, any provided attributes must be passed to the host SMS daemon for processing, and the PMIx library is *required* to add the `PMIX_USERID` and the `PMIX_GRPID` attributes of the client process that published the info.

Optional Attributes

The following attributes are optional for host environments that support this operation:

`PMIX_TIMEOUT` `"pmix.timeout"` (`int`)
Time in seconds before the specified operation should time out (*0* indicating infinite) in error. The timeout parameter can help avoid "hangs" due to programming errors that prevent the target process from ever exposing its data.

`PMIX_RANGE` `"pmix.range"` (`pmix_data_range_t`)
Value for calls to publish/lookup/unpublish or for monitoring event notifications.

`PMIX_PERSISTENCE` `"pmix.persist"` (`pmix_persistence_t`)
Value for calls to `PMIx_Publish` .

Advice to PMIx library implementers

We recommend that implementation of the `PMIX_TIMEOUT` attribute be left to the host environment due to race condition considerations between completion of the operation versus internal timeout in the PMIx server library. Implementers that choose to support `PMIX_TIMEOUT` directly in the PMIx server library must take care to resolve the race condition and should avoid passing `PMIX_TIMEOUT` to the host environment so that multiple competing timeouts are not created.

Description

Publish the data in the *info* array for subsequent lookup. By default, the data will be published into the **PMIX_SESSION** range and with **PMIX_PERSIST_APP** persistence. Changes to those values, and any additional directives, can be included in the **pmix_info_t** array. Attempts to access the data by processes outside of the provided data range will be rejected. The persistence parameter instructs the server as to how long the data is to be retained.

The blocking form will block until the server confirms that the data has been sent to the PMIx server and that it has obtained confirmation from its host SMS daemon that the data is ready to be looked up. Data is copied into the backing key-value data store, and therefore the *info* array can be released upon return from the blocking function call.

Advice to users

Duplicate keys within the specified data range may lead to unexpected behavior depending on host RM implementation of the backing key-value store.

Advice to PMIx library implementers

Implementations should, to the best of their ability, detect duplicate keys and protect the user from unexpected behavior - preferably returning an error. This version of the standard does not define a specific error code to be returned, so the implementation must make it clear to the user what to expect in this scenario. One suggestion is to define an RM specific error code beyond the **PMIX_EXTERNAL_ERR_BASE** boundary. Future versions of the standard will clarify that a specific PMIx error be returned when conflicting values are published for a given key, and will provide attributes to allow modified behaviors such as overwrite.

5.3.2 `PMIx_Publish_nb`

Summary

Nonblocking **PMIx_Publish** routine.

Format

PMIx v1.0

C

```c
pmix_status_t
PMIx_Publish_nb(const pmix_info_t info[], size_t ninfo,
                pmix_op_cbfunc_t cbfunc, void *cbdata)
```

C

IN `info`
Array of info structures (array of handles)

IN `ninfo`
Number of element in the *info* array (integer)

IN `cbfunc`
Callback function `pmix_op_cbfunc_t` (function reference)

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

Returns one of the following:

- `PMIX_SUCCESS` , indicating that the request is being processed by the host environment - result will be returned in the provided *cbfunc*. Note that the library *must not* invoke the callback function prior to returning from the API.
- `PMIX_OPERATION_SUCCEEDED` , indicating that the request was immediately processed and returned *success* - the *cbfunc* will *not* be called
- a PMIx error constant indicating either an error in the input or that the request was immediately processed and failed - the *cbfunc* will *not* be called

Required Attributes

PMIx libraries are not required to directly support any attributes for this function. However, any provided attributes must be passed to the host SMS daemon for processing, and the PMIx library is *required* to add the `PMIX_USERID` and the `PMIX_GRPID` attributes of the client process that published the info.

Optional Attributes

The following attributes are optional for host environments that support this operation:

PMIX_TIMEOUT **"pmix.timeout"** (**int**)
Time in seconds before the specified operation should time out (*0* indicating infinite) in error. The timeout parameter can help avoid "hangs" due to programming errors that prevent the target process from ever exposing its data.

PMIX_RANGE **"pmix.range"** (**pmix_data_range_t**)
Value for calls to publish/lookup/unpublish or for monitoring event notifications.

PMIX_PERSISTENCE **"pmix.persist"** (**pmix_persistence_t**)
Value for calls to **PMIx_Publish** .

Advice to PMIx library implementers

We recommend that implementation of the **PMIX_TIMEOUT** attribute be left to the host environment due to race condition considerations between completion of the operation versus internal timeout in the PMIx server library. Implementers that choose to support **PMIX_TIMEOUT** directly in the PMIx server library must take care to resolve the race condition and should avoid passing **PMIX_TIMEOUT** to the host environment so that multiple competing timeouts are not created.

Description

Nonblocking **PMIx_Publish** routine. The non-blocking form will return immediately, executing the callback when the PMIx server receives confirmation from its host SMS daemon.

Note that the function will return an error if a **NULL** callback function is given, and that the *info* array must be maintained until the callback is provided.

5.3.3 PMIx_Lookup

Summary

Lookup information published by this or another process with **PMIx_Publish** or **PMIx_Publish_nb** .

Format

PMIx v1.0

C

```c
pmix_status_t
PMIx_Lookup(pmix_pdata_t data[], size_t ndata,
            const pmix_info_t info[], size_t ninfo)
```

C

INOUT `data`
: Array of publishable data structures (array of handles)

IN `ndata`
: Number of elements in the *data* array (integer)

IN `info`
: Array of info structures (array of handles)

IN `ninfo`
: Number of elements in the *info* array (integer)

Returns **`PMIX_SUCCESS`** or a negative value corresponding to a PMIx error constant.

Required Attributes

PMIx libraries are not required to directly support any attributes for this function. However, any provided attributes must be passed to the host SMS daemon for processing, and the PMIx library is *required* to add the **`PMIX_USERID`** and the **`PMIX_GRPID`** attributes of the client process that is requesting the info.

Optional Attributes

The following attributes are optional for host environments that support this operation:

`PMIX_TIMEOUT` **`"pmix.timeout"`** (**`int`**)
: Time in seconds before the specified operation should time out (*0* indicating infinite) in error. The timeout parameter can help avoid "hangs" due to programming errors that prevent the target process from ever exposing its data.

`PMIX_RANGE` **`"pmix.range"`** (**`pmix_data_range_t`**)
: Value for calls to publish/lookup/unpublish or for monitoring event notifications.

`PMIX_WAIT` **`"pmix.wait"`** (**`int`**)
: Caller requests that the PMIx server wait until at least the specified number of values are found (*0* indicates all and is the default).

Advice to PMIx library implementers

We recommend that implementation of the **PMIX_TIMEOUT** attribute be left to the host environment due to race condition considerations between completion of the operation versus internal timeout in the PMIx server library. Implementers that choose to support **PMIX_TIMEOUT** directly in the PMIx server library must take care to resolve the race condition and should avoid passing **PMIX_TIMEOUT** to the host environment so that multiple competing timeouts are not created.

Description

Lookup information published by this or another process. By default, the search will be conducted across the **PMIX_SESSION** range. Changes to the range, and any additional directives, can be provided in the **pmix_info_t** array.

Note that the search is also constrained to only data published by the current user (i.e., the search will not return data published by an application being executed by another user). There currently is no option to override this behavior - such an option may become available later via an appropriate **pmix_info_t** directive.

The *data* parameter consists of an array of **pmix_pdata_t** struct with the keys specifying the requested information. Data will be returned for each key in the associated *value* struct. Any key that cannot be found will return with a data type of **PMIX_UNDEF** . The function will return **PMIX_SUCCESS** if *any* values can be found, so the caller must check each data element to ensure it was returned.

The proc field in each **pmix_pdata_t** struct will contain the namespace/rank of the process that published the data.

Advice to users

Although this is a blocking function, it will *not* wait by default for the requested data to be published. Instead, it will block for the time required by the server to lookup its current data and return any found items. Thus, the caller is responsible for ensuring that data is published prior to executing a lookup, using **PMIX_WAIT** to instruct the server to wait for the data to be published, or for retrying until the requested data is found.

5.3.4 PMIx_Lookup_nb

Summary

Nonblocking version of **PMIx_Lookup** .

Format

PMIx v1.0

C

```c
pmix_status_t
PMIx_Lookup_nb(char **keys,
               const pmix_info_t info[], size_t ninfo,
               pmix_lookup_cbfunc_t cbfunc, void *cbdata)
```

C

IN `keys`
Array to be provided to the callback (array of strings)

IN `info`
Array of info structures (array of handles)

IN `ninfo`
Number of element in the *info* array (integer)

IN `cbfunc`
Callback function (handle)

IN `cbdata`
Callback data to be provided to the callback function (pointer)

Returns one of the following:

- `PMIX_SUCCESS`, indicating that the request is being processed by the host environment - result will be returned in the provided *cbfunc*. Note that the library *must not* invoke the callback function prior to returning from the API.
- a PMIx error constant indicating an error in the input - the *cbfunc* will *not* be called

Required Attributes

PMIx libraries are not required to directly support any attributes for this function. However, any provided attributes must be passed to the host SMS daemon for processing, and the PMIx library is *required* to add the `PMIX_USERID` and the `PMIX_GRPID` attributes of the client process that is requesting the info.

Optional Attributes

The following attributes are optional for host environments that support this operation:

`PMIX_TIMEOUT` `"pmix.timeout"` (`int`)
Time in seconds before the specified operation should time out (*0* indicating infinite) in error. The timeout parameter can help avoid "hangs" due to programming errors that prevent the target process from ever exposing its data.

`PMIX_RANGE` `"pmix.range"` (`pmix_data_range_t`)
Value for calls to publish/lookup/unpublish or for monitoring event notifications.

`PMIX_WAIT` `"pmix.wait"` (`int`)

Caller requests that the PMIx server wait until at least the specified number of values are found (*0* indicates all and is the default).

Advice to PMIx library implementers

We recommend that implementation of the **PMIX_TIMEOUT** attribute be left to the host environment due to race condition considerations between completion of the operation versus internal timeout in the PMIx server library. Implementers that choose to support **PMIX_TIMEOUT** directly in the PMIx server library must take care to resolve the race condition and should avoid passing **PMIX_TIMEOUT** to the host environment so that multiple competing timeouts are not created.

Description

Non-blocking form of the **PMIx_Lookup** function. Data for the provided NULL-terminated *keys* array will be returned in the provided callback function. As with **PMIx_Lookup** , the default behavior is to *not* wait for data to be published. The *info* array can be used to modify the behavior as previously described by **PMIx_Lookup** . Both the *info* and *keys* arrays must be maintained until the callback is provided.

5.3.5 PMIx_Unpublish

Summary

Unpublish data posted by this process using the given keys.

Format

PMIx v1.0

C

```c
pmix_status_t
PMIx_Unpublish(char **keys,
               const pmix_info_t info[], size_t ninfo)
```

C

IN **`info`**
Array of info structures (array of handles)

IN **`ninfo`**
Number of element in the *info* array (integer)

Returns **`PMIX_SUCCESS`** or a negative value corresponding to a PMIx error constant.

Required Attributes

PMIx libraries are not required to directly support any attributes for this function. However, any provided attributes must be passed to the host SMS daemon for processing, and the PMIx library is *required* to add the **`PMIX_USERID`** and the **`PMIX_GRPID`** attributes of the client process that is requesting the operation.

Optional Attributes

The following attributes are optional for host environments that support this operation:

`PMIX_TIMEOUT` **`"pmix.timeout"`** (**`int`**)
Time in seconds before the specified operation should time out (*0* indicating infinite) in error. The timeout parameter can help avoid "hangs" due to programming errors that prevent the target process from ever exposing its data.

`PMIX_RANGE` **`"pmix.range"`** (**`pmix_data_range_t`**)
Value for calls to publish/lookup/unpublish or for monitoring event notifications.

Advice to PMIx library implementers

We recommend that implementation of the **`PMIX_TIMEOUT`** attribute be left to the host environment due to race condition considerations between completion of the operation versus internal timeout in the PMIx server library. Implementers that choose to support **`PMIX_TIMEOUT`** directly in the PMIx server library must take care to resolve the race condition and should avoid passing **`PMIX_TIMEOUT`** to the host environment so that multiple competing timeouts are not created.

Description

Unpublish data posted by this process using the given *keys*. The function will block until the data has been removed by the server (i.e., it is safe to publish that key again). A value of **NULL** for the *keys* parameter instructs the server to remove *all* data published by this process.

By default, the range is assumed to be **PMIX_SESSION** . Changes to the range, and any additional directives, can be provided in the *info* array.

5.3.6 PMIx_Unpublish_nb

Summary

Nonblocking version of **PMIx_Unpublish** .

Format

PMIx v1.0

C

```
pmix_status_t
PMIx_Unpublish_nb(char **keys,
                  const pmix_info_t info[], size_t ninfo,
                  pmix_op_cbfunc_t cbfunc, void *cbdata)
```

C

IN **keys**
(array of strings)

IN **info**
Array of info structures (array of handles)

IN **ninfo**
Number of element in the *info* array (integer)

IN **cbfunc**
Callback function **pmix_op_cbfunc_t** (function reference)

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

Returns one of the following:

- **PMIX_SUCCESS** , indicating that the request is being processed by the host environment - result will be returned in the provided *cbfunc*. Note that the library *must not* invoke the callback function prior to returning from the API.
- **PMIX_OPERATION_SUCCEEDED** , indicating that the request was immediately processed and returned *success* - the *cbfunc* will *not* be called
- a PMIx error constant indicating either an error in the input or that the request was immediately processed and failed - the *cbfunc* will *not* be called

Required Attributes

PMIx libraries are not required to directly support any attributes for this function. However, any provided attributes must be passed to the host SMS daemon for processing, and the PMIx library is *required* to add the **PMIX_USERID** and the **PMIX_GRPID** attributes of the client process that is requesting the operation.

Optional Attributes

The following attributes are optional for host environments that support this operation:

PMIX_TIMEOUT **"pmix.timeout"** (**int**)
Time in seconds before the specified operation should time out (*0* indicating infinite) in error. The timeout parameter can help avoid "hangs" due to programming errors that prevent the target process from ever exposing its data.

PMIX_RANGE **"pmix.range"** (**pmix_data_range_t**)
Value for calls to publish/lookup/unpublish or for monitoring event notifications.

Advice to PMIx library implementers

We recommend that implementation of the **PMIX_TIMEOUT** attribute be left to the host environment due to race condition considerations between completion of the operation versus internal timeout in the PMIx server library. Implementers that choose to support **PMIX_TIMEOUT** directly in the PMIx server library must take care to resolve the race condition and should avoid passing **PMIX_TIMEOUT** to the host environment so that multiple competing timeouts are not created.

Description

Non-blocking form of the **PMIx_Unpublish** function. The callback function will be executed once the server confirms removal of the specified data. The *info* array must be maintained until the callback is provided.

CHAPTER 6

Process Management

This chapter defines functionality used by clients to create and destroy/abort processes in the PMIx universe.

6.1 Abort

PMIx provides a dedicated API by which an application can request that specified processes be aborted by the system.

6.1.1 `PMIx_Abort`

Summary

Abort the specified processes

Format

PMIx v1.0

C

```c
pmix_status_t
PMIx_Abort(int status, const char msg[],
           pmix_proc_t procs[], size_t nprocs)
```

C

IN `status`
Error code to return to invoking environment (integer)

IN `msg`
String message to be returned to user (string)

IN `procs`
Array of `pmix_proc_t` structures (array of handles)

IN `nprocs`
Number of elements in the *procs* array (integer)

Returns `PMIX_SUCCESS` or a negative value corresponding to a PMIx error constant.

Description

Request that the host resource manager print the provided message and abort the provided array of *procs*. A Unix or POSIX environment should handle the provided status as a return error code from the main program that launched the application. A **`NULL`** for the *procs* array indicates that all processes in the caller's namespace are to be aborted, including itself. Passing a **`NULL`** *msg* parameter is allowed.

Advice to users

The response to this request is somewhat dependent on the specific resource manager and its configuration (e.g., some resource managers will not abort the application if the provided status is zero unless specifically configured to do so, and some cannot abort subsets of processes in an application), and thus lies outside the control of PMIx itself. However, the PMIx client library shall inform the RM of the request that the specified *procs* be aborted, regardless of the value of the provided status.

Note that race conditions caused by multiple processes calling **`PMIx_Abort`** are left to the server implementation to resolve with regard to which status is returned and what messages (if any) are printed.

6.2 Process Creation

The **`PMIx_Spawn`** commands spawn new processes and/or applications in the PMIx universe. This may include requests to extend the existing resource allocation or obtain a new one, depending upon provided and supported attributes.

6.2.1 `PMIx_Spawn`

Summary

Spawn a new job.

Format

PMIx v1.0

C

```c
pmix_status_t
PMIx_Spawn(const pmix_info_t job_info[], size_t ninfo,
           const pmix_app_t apps[], size_t napps,
           char nspace[])
```

C

IN `job_info`
Array of info structures (array of handles)

IN `ninfo`
Number of elements in the *job_info* array (integer)

IN `apps`
Array of `pmix_app_t` structures (array of handles)

IN `napps`
Number of elements in the *apps* array (integer)

OUT `nspace`
Namespace of the new job (string)

Returns `PMIX_SUCCESS` or a negative value corresponding to a PMIx error constant.

Required Attributes

PMIx libraries are not required to directly support any attributes for this function. However, any provided attributes must be passed to the host SMS daemon for processing, and the PMIx library is required to add the following attributes to those provided before passing the request to the host:

`PMIX_SPAWNED` `"pmix.spawned"` (`bool`)
`true` if this process resulted from a call to `PMIx_Spawn`.

`PMIX_PARENT_ID` `"pmix.parent"` (`pmix_proc_t`)
Process identifier of the parent process of the calling process.

`PMIX_REQUESTOR_IS_CLIENT` `"pmix.req.client"` (`bool`)
The requesting process is a PMIx client.

`PMIX_REQUESTOR_IS_TOOL` `"pmix.req.tool"` (`bool`)
The requesting process is a PMIx tool.

Host environments that implement support for `PMIx_Spawn` are required to pass the `PMIX_SPAWNED` and `PMIX_PARENT_ID` attributes to all PMIx servers launching new child processes so those values can be returned to clients upon connection to the PMIx server. In addition, they are required to support the following attributes when present in either the *job_info* or the *info* array of an element of the *apps* array:

`PMIX_WDIR` `"pmix.wdir"` (`char*`)
Working directory for spawned processes.

PMIX_SET_SESSION_CWD **"pmix.ssncwd"** (**bool**)
Set the application's current working directory to the session working directory assigned by the RM - when accessed using **PMIx_Get** , use the **PMIX_RANK_WILDCARD** value for the rank to discover the session working directory assigned to the provided namespace

PMIX_PREFIX **"pmix.prefix"** (**char***)
Prefix to use for starting spawned processes.

PMIX_HOST **"pmix.host"** (**char***)
Comma-delimited list of hosts to use for spawned processes.

PMIX_HOSTFILE **"pmix.hostfile"** (**char***)
Hostfile to use for spawned processes.

Optional Attributes

The following attributes are optional for host environments that support this operation:

PMIX_ADD_HOSTFILE **"pmix.addhostfile"** (**char***)
Hostfile listing hosts to add to existing allocation.

PMIX_ADD_HOST **"pmix.addhost"** (**char***)
Comma-delimited list of hosts to add to the allocation.

PMIX_PRELOAD_BIN **"pmix.preloadbin"** (**bool**)
Preload binaries onto nodes.

PMIX_PRELOAD_FILES **"pmix.preloadfiles"** (**char***)
Comma-delimited list of files to pre-position on nodes.

PMIX_PERSONALITY **"pmix.pers"** (**char***)
Name of personality to use.

PMIX_MAPPER **"pmix.mapper"** (**char***)
Mapping mechanism to use for placing spawned processes - when accessed using **PMIx_Get** , use the **PMIX_RANK_WILDCARD** value for the rank to discover the mapping mechanism used for the provided namespace.

PMIX_DISPLAY_MAP **"pmix.dispmap"** (**bool**)
Display process mapping upon spawn.

PMIX_PPR **"pmix.ppr"** (**char***)
Number of processes to spawn on each identified resource.

PMIX_MAPBY **"pmix.mapby"** (**char***)
Process mapping policy - when accessed using **PMIx_Get** , use the **PMIX_RANK_WILDCARD** value for the rank to discover the mapping policy used for the provided namespace

PMIX_RANKBY **"pmix.rankby"** (**char***)

Process ranking policy - when accessed using **PMIx_Get** , use the **PMIX_RANK_WILDCARD** value for the rank to discover the ranking algorithm used for the provided namespace

PMIX_BINDTO **"pmix.bindto"** (**char***)
Process binding policy - when accessed using **PMIx_Get** , use the **PMIX_RANK_WILDCARD** value for the rank to discover the binding policy used for the provided namespace

PMIX_NON_PMI **"pmix.nonpmi"** (**bool**)
Spawned processes will not call **PMIx_Init** .

PMIX_STDIN_TGT **"pmix.stdin"** (**uint32_t**)
Spawned process rank that is to receive **stdin**.

PMIX_FWD_STDIN **"pmix.fwd.stdin"** (**bool**)
Forward this process's **stdin** to the designated process.

PMIX_FWD_STDOUT **"pmix.fwd.stdout"** (**bool**)
Forward **stdout** from spawned processes to this process.

PMIX_FWD_STDERR **"pmix.fwd.stderr"** (**bool**)
Forward **stderr** from spawned processes to this process.

PMIX_DEBUGGER_DAEMONS **"pmix.debugger"** (**bool**)
Spawned application consists of debugger daemons.

PMIX_TAG_OUTPUT **"pmix.tagout"** (**bool**)
Tag application output with the identity of the source process.

PMIX_TIMESTAMP_OUTPUT **"pmix.tsout"** (**bool**)
Timestamp output from applications.

PMIX_MERGE_STDERR_STDOUT **"pmix.mergeerrout"** (**bool**)
Merge **stdout** and **stderr** streams from application processes.

PMIX_OUTPUT_TO_FILE **"pmix.outfile"** (**char***)
Output application output to the specified file.

PMIX_INDEX_ARGV **"pmix.indxargv"** (**bool**)
Mark the **argv** with the rank of the process.

PMIX_CPUS_PER_PROC **"pmix.cpuperproc"** (**uint32_t**)
Number of cpus to assign to each rank - when accessed using **PMIx_Get** , use the **PMIX_RANK_WILDCARD** value for the rank to discover the cpus/process assigned to the provided namespace

PMIX_NO_PROCS_ON_HEAD **"pmix.nolocal"** (**bool**)
Do not place processes on the head node.

PMIX_NO_OVERSUBSCRIBE **"pmix.noover"** (**bool**)

Do not oversubscribe the cpus.

PMIX_REPORT_BINDINGS **"pmix.repbind"** (**bool**)
Report bindings of the individual processes.

PMIX_CPU_LIST **"pmix.cpulist"** (**char***)
List of cpus to use for this job - when accessed using **PMIx_Get** , use the **PMIX_RANK_WILDCARD** value for the rank to discover the cpu list used for the provided namespace

PMIX_JOB_RECOVERABLE **"pmix.recover"** (**bool**)
Application supports recoverable operations.

PMIX_JOB_CONTINUOUS **"pmix.continuous"** (**bool**)
Application is continuous, all failed processes should be immediately restarted.

PMIX_MAX_RESTARTS **"pmix.maxrestarts"** (**uint32_t**)
Maximum number of times to restart a job - when accessed using **PMIx_Get** , use the **PMIX_RANK_WILDCARD** value for the rank to discover the max restarts for the provided namespace

Description

Spawn a new job. The assigned namespace of the spawned applications is returned in the *nspace* parameter. A **NULL** value in that location indicates that the caller doesn't wish to have the namespace returned. The *nspace* array must be at least of size one more than **PMIX_MAX_NSLEN** .

By default, the spawned processes will be PMIx "connected" to the parent process upon successful launch (see **PMIx_Connect** description for details). Note that this only means that (a) the parent process will be given a copy of the new job's information so it can query job-level info without incurring any communication penalties, (b) newly spawned child processes will receive a copy of the parent processes job-level info, and (c) both the parent process and members of the child job will receive notification of errors from processes in their combined assemblage.

Advice to users

Behavior of individual resource managers may differ, but it is expected that failure of any application process to start will result in termination/cleanup of *all* processes in the newly spawned job and return of an error code to the caller.

6.2.2 PMIx_Spawn_nb

Summary

Nonblocking version of the **PMIx_Spawn** routine.

Format

PMIx v1.0

C

```
pmix_status_t
PMIx_Spawn_nb(const pmix_info_t job_info[], size_t ninfo,
              const pmix_app_t apps[], size_t napps,
              pmix_spawn_cbfunc_t cbfunc, void *cbdata)
```

C

IN `job_info`
Array of info structures (array of handles)

IN `ninfo`
Number of elements in the *job_info* array (integer)

IN `apps`
Array of **pmix_app_t** structures (array of handles)

IN `cbfunc`
Callback function **pmix_spawn_cbfunc_t** (function reference)

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

Returns one of the following:

- **PMIX_SUCCESS**, indicating that the request is being processed by the host environment - result will be returned in the provided *cbfunc*. Note that the library *must not* invoke the callback function prior to returning from the API.
- a PMIx error constant indicating an error in the request - the *cbfunc* will *not* be called

Required Attributes

PMIx libraries are not required to directly support any attributes for this function. However, any provided attributes must be passed to the host SMS daemon for processing, and the PMIx library is required to add the following attributes to those provided before passing the request to the host:

PMIX_SPAWNED `"pmix.spawned"` (`bool`)
`true` if this process resulted from a call to **PMIx_Spawn**.

PMIX_PARENT_ID `"pmix.parent"` (`pmix_proc_t`)
Process identifier of the parent process of the calling process.

PMIX_REQUESTOR_IS_CLIENT `"pmix.req.client"` (`bool`)
The requesting process is a PMIx client.

PMIX_REQUESTOR_IS_TOOL **"pmix.req.tool"** (**bool**)
The requesting process is a PMIx tool.

Host environments that implement support for **PMIx_Spawn** are required to pass the **PMIX_SPAWNED** and **PMIX_PARENT_ID** attributes to all PMIx servers launching new child processes so those values can be returned to clients upon connection to the PMIx server. In addition, they are required to support the following attributes when present in either the *job_info* or the *info* array of an element of the *apps* array:

PMIX_WDIR **"pmix.wdir"** (**char***)
Working directory for spawned processes.

PMIX_SET_SESSION_CWD **"pmix.ssncwd"** (**bool**)
Set the application's current working directory to the session working directory assigned by the RM - when accessed using **PMIx_Get** , use the **PMIX_RANK_WILDCARD** value for the rank to discover the session working directory assigned to the provided namespace

PMIX_PREFIX **"pmix.prefix"** (**char***)
Prefix to use for starting spawned processes.

PMIX_HOST **"pmix.host"** (**char***)
Comma-delimited list of hosts to use for spawned processes.

PMIX_HOSTFILE **"pmix.hostfile"** (**char***)
Hostfile to use for spawned processes.

Optional Attributes

The following attributes are optional for host environments that support this operation:

PMIX_ADD_HOSTFILE **"pmix.addhostfile"** (**char***)
Hostfile listing hosts to add to existing allocation.

PMIX_ADD_HOST **"pmix.addhost"** (**char***)
Comma-delimited list of hosts to add to the allocation.

PMIX_PRELOAD_BIN **"pmix.preloadbin"** (**bool**)
Preload binaries onto nodes.

PMIX_PRELOAD_FILES **"pmix.preloadfiles"** (**char***)
Comma-delimited list of files to pre-position on nodes.

PMIX_PERSONALITY **"pmix.pers"** (**char***)
Name of personality to use.

PMIX_MAPPER **"pmix.mapper"** (**char***)
Mapping mechanism to use for placing spawned processes - when accessed using **PMIx_Get** , use the **PMIX_RANK_WILDCARD** value for the rank to discover the mapping mechanism used for the provided namespace.

PMIX_DISPLAY_MAP **"pmix.dispmap"** (**bool**)
Display process mapping upon spawn.

PMIX_PPR **"pmix.ppr"** (**char***)
Number of processes to spawn on each identified resource.

PMIX_MAPBY **"pmix.mapby"** (**char***)
Process mapping policy - when accessed using **PMIx_Get** , use the **PMIX_RANK_WILDCARD** value for the rank to discover the mapping policy used for the provided namespace

PMIX_RANKBY **"pmix.rankby"** (**char***)
Process ranking policy - when accessed using **PMIx_Get** , use the **PMIX_RANK_WILDCARD** value for the rank to discover the ranking algorithm used for the provided namespace

PMIX_BINDTO **"pmix.bindto"** (**char***)
Process binding policy - when accessed using **PMIx_Get** , use the **PMIX_RANK_WILDCARD** value for the rank to discover the binding policy used for the provided namespace

PMIX_NON_PMI **"pmix.nonpmi"** (**bool**)
Spawned processes will not call **PMIx_Init** .

PMIX_STDIN_TGT **"pmix.stdin"** (**uint32_t**)
Spawned process rank that is to receive **stdin**.

PMIX_FWD_STDIN **"pmix.fwd.stdin"** (**bool**)
Forward this process's **stdin** to the designated process.

PMIX_FWD_STDOUT **"pmix.fwd.stdout"** (**bool**)
Forward **stdout** from spawned processes to this process.

PMIX_FWD_STDERR **"pmix.fwd.stderr"** (**bool**)
Forward **stderr** from spawned processes to this process.

PMIX_DEBUGGER_DAEMONS **"pmix.debugger"** (**bool**)
Spawned application consists of debugger daemons.

PMIX_TAG_OUTPUT **"pmix.tagout"** (**bool**)
Tag application output with the identity of the source process.

PMIX_TIMESTAMP_OUTPUT **"pmix.tsout"** (**bool**)
Timestamp output from applications.

PMIX_MERGE_STDERR_STDOUT **"pmix.mergeerrout"** (**bool**)
Merge **stdout** and **stderr** streams from application processes.

PMIX_OUTPUT_TO_FILE **"pmix.outfile"** (**char***)
Output application output to the specified file.

PMIX_INDEX_ARGV **"pmix.indxargv"** (**bool**)
Mark the **argv** with the rank of the process.

PMIX_CPUS_PER_PROC **"pmix.cpuperproc"** (**uint32_t**)
Number of cpus to assign to each rank - when accessed using **PMIx_Get** , use the **PMIX_RANK_WILDCARD** value for the rank to discover the cpus/process assigned to the provided namespace

PMIX_NO_PROCS_ON_HEAD **"pmix.nolocal"** (**bool**)
Do not place processes on the head node.

PMIX_NO_OVERSUBSCRIBE **"pmix.noover"** (**bool**)
Do not oversubscribe the cpus.

PMIX_REPORT_BINDINGS **"pmix.repbind"** (**bool**)
Report bindings of the individual processes.

PMIX_CPU_LIST **"pmix.cpulist"** (**char***)
List of cpus to use for this job - when accessed using **PMIx_Get** , use the **PMIX_RANK_WILDCARD** value for the rank to discover the cpu list used for the provided namespace

PMIX_JOB_RECOVERABLE **"pmix.recover"** (**bool**)
Application supports recoverable operations.

PMIX_JOB_CONTINUOUS **"pmix.continuous"** (**bool**)
Application is continuous, all failed processes should be immediately restarted.

PMIX_MAX_RESTARTS **"pmix.maxrestarts"** (**uint32_t**)
Maximum number of times to restart a job - when accessed using **PMIx_Get** , use the **PMIX_RANK_WILDCARD** value for the rank to discover the max restarts for the provided namespace

Description

Nonblocking version of the **PMIx_Spawn** routine. The provided callback function will be executed upon successful start of *all* specified application processes.

Advice to users

Behavior of individual resource managers may differ, but it is expected that failure of any application process to start will result in termination/cleanup of *all* processes in the newly spawned job and return of an error code to the caller.

6.3 Connecting and Disconnecting Processes

This section defines functions to connect and disconnect processes in two or more separate PMIx namespaces. The PMIx definition of *connected* solely implies the following:

- job-level information for each namespace involved in the operation is to be made available to all processes in the connected assemblage
- any data posted by a process in the connected assemblage (via calls to **PMIx_Put** committed via **PMIx_Commit**) prior to execution of the **PMIx_Connect** operation is to be made accessible to all processes in the assemblage - any data posted after execution of the *connect* operation must be exchanged via a separate **PMIx_Fence** operation spanning the connected processes
- the host environment should treat the failure of any process in the assemblage as a reportable event, taking action on the assemblage as if it were a single application. For example, if the environment defaults (in the absence of any application directives) to terminating an application upon failure of any process in that application, then the environment should terminate all processes in the connected assemblage upon failure of any member.

Advice to PMIx server hosts

The host environment may choose to assign a new namespace to the connected assemblage and/or assign new ranks for its members for its own internal tracking purposes. However, it is not required to communicate such assignments to the participants (e.g., in response to an appropriate call to **PMIx_Query_info_nb**). The host environment is required to generate a **PMIX_ERR_INVALID_TERMINATION** event should any process in the assemblage terminate or call **PMIx_Finalize** without first *disconnecting* from the assemblage.

Advice to users

Attempting to *connect* processes solely within the same namespace is essentially a *no-op* operation. While not explicitly prohibited, users are advised that a PMIx implementation or host environment may return an error in such cases.

Neither the PMIx implementation nor host environment are required to provide any tracking support for the assemblage. Thus, the application is responsible for maintaining the membership list of the assemblage.

6.3.1 PMIx_Connect

Summary

Connect namespaces.

Format

PMIx v1.0

C

```c
pmix_status_t
PMIx_Connect(const pmix_proc_t procs[], size_t nprocs,
             const pmix_info_t info[], size_t ninfo)
```

C

IN **`procs`**
Array of proc structures (array of handles)

IN **`nprocs`**
Number of elements in the *procs* array (integer)

IN **`info`**
Array of info structures (array of handles)

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

Returns **`PMIX_SUCCESS`** or a negative value corresponding to a PMIx error constant.

Required Attributes

PMIx libraries are not required to directly support any attributes for this function. However, any provided attributes must be passed to the host SMS daemon for processing.

Optional Attributes

The following attributes are optional for host environments that support this operation:

`PMIX_TIMEOUT` **`"pmix.timeout"`** (**`int`**)
Time in seconds before the specified operation should time out (*0* indicating infinite) in error. The timeout parameter can help avoid “hangs” due to programming errors that prevent the target process from ever exposing its data.

`PMIX_COLLECTIVE_ALGO` **`"pmix.calgo"`** (**`char*`**)
Comma-delimited list of algorithms to use for the collective operation. PMIx does not impose any requirements on a host environment’s collective algorithms. Thus, the acceptable values for this attribute will be environment-dependent - users are encouraged to check their host environment for supported values.

`PMIX_COLLECTIVE_ALGO_REQD` **`"pmix.calreqd"`** (**`bool`**)
If **`true`**, indicates that the requested choice of algorithm is mandatory.

Advice to PMIx library implementers

We recommend that implementation of the **PMIX_TIMEOUT** attribute be left to the host environment due to race condition considerations between completion of the operation versus internal timeout in the PMIx server library. Implementers that choose to support **PMIX_TIMEOUT** directly in the PMIx server library must take care to resolve the race condition and should avoid passing **PMIX_TIMEOUT** to the host environment so that multiple competing timeouts are not created.

Description

Record the processes specified by the *procs* array as *connected* as per the PMIx definition. The function will return once all processes identified in *procs* have called either **PMIx_Connect** or its non-blocking version, *and* the host environment has completed any supporting operations required to meet the terms of the PMIx definition of *connected* processes.

Advice to users

All processes engaged in a given **PMIx_Connect** operation must provide the identical *procs* array as ordering of entries in the array and the method by which those processes are identified (e.g., use of **PMIX_RANK_WILDCARD** versus listing the individual processes) *may* impact the host environment's algorithm for uniquely identifying an operation.

Advice to PMIx library implementers

PMIx_Connect and its non-blocking form are both *collective* operations. Accordingly, the PMIx server library is required to aggregate participation by local clients, passing the request to the host environment once all local participants have executed the API.

Advice to PMIx server hosts

The host will receive a single call for each collective operation. It is the responsibility of the host to identify the nodes containing participating processes, execute the collective across all participating nodes, and notify the local PMIx server library upon completion of the global collective.

Processes that combine via **PMIx_Connect** must call **PMIx_Disconnect** prior to finalizing and/or terminating - any process in the assemblage failing to meet this requirement will cause a **PMIX_ERR_INVALID_TERMINATION** event to be generated.

A process can only engage in *one* connect operation involving the identical *procs* array at a time. However, a process *can* be simultaneously engaged in multiple connect operations, each involving a different *procs* array.

As in the case of the **PMIx_Fence** operation, the *info* array can be used to pass user-level directives regarding the algorithm to be used for any collective operation involved in the operation, timeout constraints, and other options available from the host RM.

6.3.2 PMIx_Connect_nb

Summary

Nonblocking **PMIx_Connect_nb** routine.

Format

PMIx v1.0

C

```
pmix_status_t
PMIx_Connect_nb(const pmix_proc_t procs[], size_t nprocs,
                const pmix_info_t info[], size_t ninfo,
                pmix_op_cbfunc_t cbfunc, void *cbdata)
```

C

IN **procs**
Array of proc structures (array of handles)

IN **nprocs**
Number of elements in the *procs* array (integer)

IN **info**
Array of info structures (array of handles)

IN **ninfo**
Number of element in the *info* array (integer)

IN **cbfunc**
Callback function **pmix_op_cbfunc_t** (function reference)

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

Returns one of the following:

- **PMIX_SUCCESS** , indicating that the request is being processed by the host environment - result will be returned in the provided *cbfunc*. Note that the library *must not* invoke the callback function prior to returning from the API.
- **PMIX_OPERATION_SUCCEEDED** , indicating that the request was immediately processed and returned *success* - the *cbfunc* will *not* be called
- a PMIx error constant indicating either an error in the input or that the request was immediately processed and failed - the *cbfunc* will *not* be called

Required Attributes

PMIx libraries are not required to directly support any attributes for this function. However, any provided attributes must be passed to the host SMS daemon for processing.

Optional Attributes

The following attributes are optional for host environments that support this operation:

PMIX_TIMEOUT **"pmix.timeout"** (**int**)
Time in seconds before the specified operation should time out (*0* indicating infinite) in error. The timeout parameter can help avoid "hangs" due to programming errors that prevent the target process from ever exposing its data.

PMIX_COLLECTIVE_ALGO **"pmix.calgo"** (**char***)
Comma-delimited list of algorithms to use for the collective operation. PMIx does not impose any requirements on a host environment's collective algorithms. Thus, the acceptable values for this attribute will be environment-dependent - users are encouraged to check their host environment for supported values.

PMIX_COLLECTIVE_ALGO_REQD **"pmix.calreqd"** (**bool**)
If **true**, indicates that the requested choice of algorithm is mandatory.

Advice to PMIx library implementers

We recommend that implementation of the **PMIX_TIMEOUT** attribute be left to the host environment due to race condition considerations between completion of the operation versus internal timeout in the PMIx server library. Implementers that choose to support **PMIX_TIMEOUT** directly in the PMIx server library must take care to resolve the race condition and should avoid passing **PMIX_TIMEOUT** to the host environment so that multiple competing timeouts are not created.

Description

Nonblocking version of **PMIx_Connect** . The callback function is called once all processes identified in *procs* have called either **PMIx_Connect** or its non-blocking version, *and* the host environment has completed any supporting operations required to meet the terms of the PMIx definition of *connected* processes. See the advice provided in the description for **PMIx_Connect** for more information.

6.3.3 PMIx_Disconnect

Summary

Disconnect a previously connected set of processes.

Format

PMIx v1.0

C

```
pmix_status_t
PMIx_Disconnect(const pmix_proc_t procs[], size_t nprocs,
                const pmix_info_t info[], size_t ninfo);
```

C

IN **procs**
Array of proc structures (array of handles)

IN **nprocs**
Number of elements in the *procs* array (integer)

IN **info**
Array of info structures (array of handles)

IN **ninfo**
Number of element in the *info* array (integer)

Returns **PMIX_SUCCESS** or a negative value corresponding to a PMIx error constant.

Required Attributes

PMIx libraries are not required to directly support any attributes for this function. However, any provided attributes must be passed to the host SMS daemon for processing.

Optional Attributes

The following attributes are optional for host environments that support this operation:

PMIX_TIMEOUT **"pmix.timeout" (int)**
Time in seconds before the specified operation should time out (*0* indicating infinite) in error. The timeout parameter can help avoid "hangs" due to programming errors that prevent the target process from ever exposing its data.

Advice to PMIx library implementers

We recommend that implementation of the **PMIX_TIMEOUT** attribute be left to the host environment due to race condition considerations between completion of the operation versus internal timeout in the PMIx server library. Implementers that choose to support **PMIX_TIMEOUT** directly in the PMIx server library must take care to resolve the race condition and should avoid passing **PMIX_TIMEOUT** to the host environment so that multiple competing timeouts are not created.

Description

Disconnect a previously connected set of processes. A **PMIX_ERR_INVALID_OPERATION** error will be returned if the specified set of *procs* was not previously *connected* via a call to **PMIx_Connect** or its non-blocking form. The function will return once all processes identified in *procs* have called either **PMIx_Disconnect** or its non-blocking version, *and* the host environment has completed any required supporting operations.

Advice to users

All processes engaged in a given **PMIx_Disconnect** operation must provide the identical *procs* array as ordering of entries in the array and the method by which those processes are identified (e.g., use of **PMIX_RANK_WILDCARD** versus listing the individual processes) *may* impact the host environment's algorithm for uniquely identifying an operation.

Advice to PMIx library implementers

PMIx_Disconnect and its non-blocking form are both *collective* operations. Accordingly, the PMIx server library is required to aggregate participation by local clients, passing the request to the host environment once all local participants have executed the API.

Advice to PMIx server hosts

The host will receive a single call for each collective operation. The host will receive a single call for each collective operation. It is the responsibility of the host to identify the nodes containing participating processes, execute the collective across all participating nodes, and notify the local PMIx server library upon completion of the global collective.

A process can only engage in *one* disconnect operation involving the identical *procs* array at a time. However, a process *can* be simultaneously engaged in multiple disconnect operations, each involving a different *procs* array.

As in the case of the **PMIx_Fence** operation, the *info* array can be used to pass user-level directives regarding the algorithm to be used for any collective operation involved in the operation, timeout constraints, and other options available from the host RM.

6.3.4 PMIx_Disconnect_nb

Summary

Nonblocking **PMIx_Disconnect** routine.

Format

PMIx v1.0

C

```
pmix_status_t
PMIx_Disconnect_nb(const pmix_proc_t procs[], size_t nprocs,
                   const pmix_info_t info[], size_t ninfo,
                   pmix_op_cbfunc_t cbfunc, void *cbdata);
```

C

IN **procs**
Array of proc structures (array of handles)

IN **nprocs**
Number of elements in the *procs* array (integer)

IN **info**
Array of info structures (array of handles)

IN **ninfo**
Number of element in the *info* array (integer)

IN **cbfunc**
Callback function **pmix_op_cbfunc_t** (function reference)

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

Returns one of the following:

- **PMIX_SUCCESS** , indicating that the request is being processed by the host environment - result will be returned in the provided *cbfunc*. Note that the library *must not* invoke the callback function prior to returning from the API.
- **PMIX_OPERATION_SUCCEEDED** , indicating that the request was immediately processed and returned *success* - the *cbfunc* will *not* be called
- a PMIx error constant indicating either an error in the input or that the request was immediately processed and failed - the *cbfunc* will *not* be called

Required Attributes

PMIx libraries are not required to directly support any attributes for this function. However, any provided attributes must be passed to the host SMS daemon for processing.

Optional Attributes

The following attributes are optional for host environments that support this operation:

PMIX_TIMEOUT **"pmix.timeout"** (**int**)
Time in seconds before the specified operation should time out (*0* indicating infinite) in error. The timeout parameter can help avoid "hangs" due to programming errors that prevent the target process from ever exposing its data.

Advice to PMIx library implementers

We recommend that implementation of the **PMIX_TIMEOUT** attribute be left to the host environment due to race condition considerations between completion of the operation versus internal timeout in the PMIx server library. Implementers that choose to support **PMIX_TIMEOUT** directly in the PMIx server library must take care to resolve the race condition and should avoid passing **PMIX_TIMEOUT** to the host environment so that multiple competing timeouts are not created.

Description

Nonblocking **PMIx_Disconnect** routine. The callback function is called once all processes identified in *procs* have called either **PMIx_Disconnect_nb** or its blocking version, *and* the host environment has completed any required supporting operations. See the advice provided in the description for **PMIx_Disconnect** for more information.

CHAPTER 7

Job Allocation Management and Reporting

The job management APIs provide an application with the ability to orchestrate its operation in partnership with the SMS. Members of this category include the **PMIx_Allocation_request_nb** , **PMIx_Job_control_nb** , and **PMIx_Process_monitor_nb** APIs.

7.1 Query

As the level of interaction between applications and the host SMS grows, so too does the need for the application to query the SMS regarding its capabilities and state information. PMIx provides a generalized query interface for this purpose, along with a set of standardized attribute keys to support a range of requests. This includes requests to determine the status of scheduling queues and active allocations, the scope of API and attribute support offered by the SMS, namespaces of active jobs, location and information about a job's processes, and information regarding available resources.

An example use-case for the **PMIx_Query_info_nb** API is to ensure clean job completion. Time-shared systems frequently impose maximum run times when assigning jobs to resource allocations. To shut down gracefully, e.g., to write a checkpoint before termination, it is necessary for an application to periodically query the resource manager for the time remaining in its allocation. This is especially true on systems for which allocation times may be shortened or lengthened from the original time limit. Many resource managers provide APIs to dynamically obtain this information, but each API is specific to the resource manager.

PMIx supports this use-case by defining an attribute key (**PMIX_TIME_REMAINING**) that can be used with the **PMIx_Query_info_nb** interface to obtain the number of seconds remaining in the current job allocation. Note that one could alternatively use the **PMIx_Register_event_handler** API to register for an event indicating incipient job termination, and then use the **PMIx_Job_control_nb** API to request that the host SMS generate an event a specified amount of time prior to reaching the maximum run time. PMIx provides such alternate methods as a means of maximizing the probability of a host system supporting at least one method by which the application can obtain the desired service.

The following APIs support query of various session and environment values.

7.1.1 `PMIx_Resolve_peers`

Summary

Obtain the array of processes within the specified namespace that are executing on a given node.

Format

PMIx v1.0

C

```c
pmix_status_t
PMIx_Resolve_peers(const char *nodename,
                   const pmix_nspace_t nspace,
                   pmix_proc_t **procs, size_t *nprocs)
```

C

IN `nodename`
Name of the node to query (string)

IN `nspace`
namespace (string)

OUT `procs`
Array of process structures (array of handles)

OUT `nprocs`
Number of elements in the *procs* array (integer)

Returns **`PMIX_SUCCESS`** or a negative value corresponding to a PMIx error constant.

Description

Given a *nodename*, return the array of processes within the specified *nspace* that are executing on that node. If the *nspace* is **`NULL`**, then all processes on the node will be returned. If the specified node does not currently host any processes, then the returned array will be **`NULL`**, and *nprocs* will be **`0`**. The caller is responsible for releasing the *procs* array when done with it. The **`PMIX_PROC_FREE`** macro is provided for this purpose.

7.1.2 `PMIx_Resolve_nodes`

Summary

Return a list of nodes hosting processes within the given namespace.

Format

PMIx v1.0

C

```c
pmix_status_t
PMIx_Resolve_nodes(const char *nspace, char **nodelist)
```

C

IN **nspace**
Namespace (string)

OUT **nodelist**
Comma-delimited list of nodenames (string)

Returns **PMIX_SUCCESS** or a negative value corresponding to a PMIx error constant.

Description

Given a *nspace*, return the list of nodes hosting processes within that namespace. The returned string will contain a comma-delimited list of nodenames. The caller is responsible for releasing the string when done with it.

7.1.3 PMIx_Query_info_nb

Summary

Query information about the system in general.

Format

PMIx v2.0

C

```c
pmix_status_t
PMIx_Query_info_nb(pmix_query_t queries[], size_t nqueries,
                   pmix_info_cbfunc_t cbfunc, void *cbdata)
```

C

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

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

IN **cbfunc**
Callback function **pmix_info_cbfunc_t** (function reference)

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

Function returns either:

- **PMIX_SUCCESS** indicating that the request has been accepted for processing and the provided callback function will be executed upon completion of the operation
- a non-zero PMIx error constant indicating a reason for the request to have been rejected. In this case, the provided callback function will *not* be executed

If executed, the status returned in the provided callback function will be one of the following constants:

- **PMIX_SUCCESS** All data has been returned
- **PMIX_ERR_NOT_FOUND** None of the requested data was available
- **PMIX_ERR_PARTIAL_SUCCESS** Some of the data has been returned
- **PMIX_ERR_NOT_SUPPORTED** The host RM does not support this function
- a non-zero PMIx error constant indicating a reason for the request's failure

Required Attributes

PMIx libraries that support this API are required to support the following attributes:

PMIX_QUERY_REFRESH_CACHE **"pmix.qry.rfsh"** (**bool**)
Retrieve updated information from server.

PMIX_SESSION_INFO **"pmix.ssn.info"** (**bool**)
Return information about the specified session. If information about a session other than the one containing the requesting process is desired, then the attribute array must contain a **PMIX_SESSION_ID** attribute identifying the desired target.

PMIX_JOB_INFO **"pmix.job.info"** (**bool**)
Return information about the specified job or namespace. If information about a job or namespace other than the one containing the requesting process is desired, then the attribute array must contain a **PMIX_JOBID** or **PMIX_NSPACE** attribute identifying the desired target. Similarly, if information is requested about a job or namespace in a session other than the one containing the requesting process, then an attribute identifying the target session must be provided.

PMIX_APP_INFO **"pmix.app.info"** (**bool**)
Return information about the specified application. If information about an application other than the one containing the requesting process is desired, then the attribute array must contain a **PMIX_APPNUM** attribute identifying the desired target. Similarly, if information is requested about an application in a job or session other than the one containing the requesting process, then attributes identifying the target job and/or session must be provided.

PMIX_NODE_INFO **"pmix.node.info"** (**bool**)
Return information about the specified node. If information about a node other than the one containing the requesting process is desired, then the attribute array must contain either the **PMIX_NODEID** or **PMIX_HOSTNAME** attribute identifying the desired target.

PMIx libraries are not required to directly support any other attributes for this function. However, any provided attributes must be passed to the host SMS daemon for processing, and the PMIx library is *required* to add the **PMIX_USERID** and the **PMIX_GRPID** attributes of the client process making the request.

Optional Attributes

The following attributes are optional for host environments that support this operation:

PMIX_QUERY_NAMESPACES **"pmix.qry.ns"** (**char***)
Request a comma-delimited list of active namespaces.

PMIX_QUERY_JOB_STATUS **"pmix.qry.jst"** (**pmix_status_t**)
Status of a specified, currently executing job.

PMIX_QUERY_QUEUE_LIST **"pmix.qry.qlst"** (**char***)
Request a comma-delimited list of scheduler queues.

PMIX_QUERY_QUEUE_STATUS **"pmix.qry.qst"** (**TBD**)
Status of a specified scheduler queue.

PMIX_QUERY_PROC_TABLE **"pmix.qry.ptable"** (**char***)
Input namespace of the job whose information is being requested returns (**pmix_data_array_t**) an array of **pmix_proc_info_t** .

PMIX_QUERY_LOCAL_PROC_TABLE **"pmix.qry.lptable"** (**char***)
Input namespace of the job whose information is being requested returns (**pmix_data_array_t**) an array of **pmix_proc_info_t** for processes in job on same node.

PMIX_QUERY_SPAWN_SUPPORT **"pmix.qry.spawn"** (**bool**)
Return a comma-delimited list of supported spawn attributes.

PMIX_QUERY_DEBUG_SUPPORT **"pmix.qry.debug"** (**bool**)
Return a comma-delimited list of supported debug attributes.

PMIX_QUERY_MEMORY_USAGE **"pmix.qry.mem"** (**bool**)
Return information on memory usage for the processes indicated in the qualifiers.

PMIX_QUERY_REPORT_AVG **"pmix.qry.avg"** (**bool**)
Report average values.

PMIX_QUERY_REPORT_MINMAX **"pmix.qry.minmax"** (**bool**)
Report minimum and maximum values.

PMIX_QUERY_ALLOC_STATUS **"pmix.query.alloc"** (**char***)
String identifier of the allocation whose status is being requested.

PMIX_TIME_REMAINING **"pmix.time.remaining"** (**char***)

Query number of seconds (**uint32_t**) remaining in allocation for the specified namespace.

Description

Query information about the system in general. This can include a list of active namespaces, network topology, etc. Also can be used to query node-specific info such as the list of peers executing on a given node. We assume that the host RM will exercise appropriate access control on the information.

NOTE: There is no blocking form of this API as the structures passed to query info differ from those for receiving the results.

The *status* argument to the callback function indicates if requested data was found or not. An array of **pmix_info_t** will contain each key that was provided and the corresponding value that was found. Requests for keys that are not found will return the key paired with a value of type **PMIX_UNDEF**.

Advice to users

The desire to query a list of attributes supported by the implementation and/or the host environment has been expressed and noted. The PMIx community is exploring the possibility and it will likely become available in a future release

Advice to PMIx library implementers

Information returned from **PMIx_Query_info_nb** shall be locally cached so that retrieval by subsequent calls to **PMIx_Get** or **PMIx_Query_info_nb** can succeed with minimal overhead. The local cache shall be checked prior to querying the PMIx server and/or the host environment. Queries that include the **PMIX_QUERY_REFRESH_CACHE** attribute shall bypass the local cache and retrieve a new value for the query, refreshing the values in the cache upon return.

7.1.3.1 Using **PMIx_Get** vs **PMIx_Query_info_nb**

Both **PMIx_Get** and **PMIx_Query_info_nb** can be used to retrieve information about the system. In general, the *get* operation should be used to retrieve:

- information provided by the host environment at time of job start. This includes information on the number of processes in the job, their location, and possibly their communication endpoints
- information posted by processes via the **PMIx_Put** function

This information is largely considered to be *static*, although this will not necessarily be true for environments supporting dynamic programming models or fault tolerance. Note that the **PMIx_Get** function only accesses information about execution environments - i.e., its scope is limited to values pertaining to a specific **session**, **job**, **application**, process, or node. It cannot be used to obtain information about areas such as the status of queues in the WLM.

In contrast, the *query* option should be used to access:

- system-level information (such as the available WLM queues) that would generally not be included in job-level information provided at job start
- dynamic information such as application and queue status, and resource utilization statistics. Note that the **PMIX_QUERY_REFRESH_CACHE** attribute must be provided on each query to ensure current data is returned
- information created post job start, such as process tables
- information requiring more complex search criteria than supported by the simpler **PMIx_Get** API
- queries focused on retrieving multi-attribute blocks of data with a single request, thus bypassing the single-key limitation of the **PMIx_Get** API

In theory, all information can be accessed via **PMIx_Query_info_nb** as the local cache is typically the same datastore searched by **PMIx_Get**. However, in practice, the overhead associated with the *query* operation may (depending upon implementation) be higher than the simpler *get* operation due to the need to construct and process the more complex **pmix_query_t** structure. Thus, requests for a single key value are likely to be accomplished faster with **PMIx_Get** versus the *query* operation.

7.2 Allocation Requests

This section defines functionality to request new allocations from the RM, and request modifications to existing allocations. These are primarily used in the following scenarios:

- *Evolving* applications that dynamically request and return resources as they execute

- *Malleable* environments where the scheduler redirects resources away from executing applications for higher priority jobs or load balancing
- *Resilient* applications that need to request replacement resources in the face of failures
- *Rigid* jobs where the user has requested a static allocation of resources for a fixed period of time, but realizes that they underestimated their required time while executing

PMIx attempts to address this range of use-cases with a single, flexible API.

7.2.1 `PMIx_Allocation_request_nb`

Summary

Request an allocation operation from the host resource manager.

Format

PMIx v2.0

C

```c
pmix_status_t
PMIx_Allocation_request_nb(pmix_alloc_directive_t directive,
                           pmix_info_t info[], size_t ninfo,
                           pmix_info_cbfunc_t cbfunc, void *cbdata);
```

C

IN **`directive`**
Allocation directive (handle)
IN **`info`**
Array of **`pmix_info_t`** structures (array of handles)
IN **`ninfo`**
Number of elements in the *info* array (integer)
IN **`cbfunc`**
Callback function **`pmix_info_cbfunc_t`** (function reference)
IN **`cbdata`**
Data to be passed to the callback function (memory reference)

Returns one of the following:

- **`PMIX_SUCCESS`**, indicating that the request is being processed by the host environment - result will be returned in the provided *cbfunc*. Note that the library *must not* invoke the callback function prior to returning from the API.
- **`PMIX_OPERATION_SUCCEEDED`**, indicating that the request was immediately processed and returned *success* - the *cbfunc* will *not* be called
- a PMIx error constant indicating either an error in the input or that the request was immediately processed and failed - the *cbfunc* will *not* be called

Required Attributes

PMIx libraries are not required to directly support any attributes for this function. However, any provided attributes must be passed to the host SMS daemon for processing, and the PMIx library is *required* to add the **PMIX_USERID** and the **PMIX_GRPID** attributes of the client process making the request.

Host environments that implement support for this operation are required to support the following attributes:

PMIX_ALLOC_ID **"pmix.alloc.id"** (**char***)
  Provide a string identifier for this allocation request which can later be used to query status of the request.

PMIX_ALLOC_NUM_NODES **"pmix.alloc.nnodes"** (**uint64_t**)
  The number of nodes.

PMIX_ALLOC_NUM_CPUS **"pmix.alloc.ncpus"** (**uint64_t**)
  Number of cpus.

PMIX_ALLOC_TIME **"pmix.alloc.time"** (**uint32_t**)
  Time in seconds.

Optional Attributes

The following attributes are optional for host environments that support this operation:

PMIX_ALLOC_NODE_LIST **"pmix.alloc.nlist"** (**char***)
  Regular expression of the specific nodes.

PMIX_ALLOC_NUM_CPU_LIST **"pmix.alloc.ncpulist"** (**char***)
  Regular expression of the number of cpus for each node.

PMIX_ALLOC_CPU_LIST **"pmix.alloc.cpulist"** (**char***)
  Regular expression of the specific cpus indicating the cpus involved.

PMIX_ALLOC_MEM_SIZE **"pmix.alloc.msize"** (**float**)
  Number of Megabytes.

PMIX_ALLOC_NETWORK **"pmix.alloc.net"** (**array**)
  Array of **pmix_info_t** describing requested network resources. If not given as part of an **pmix_info_t** struct that identifies the involved nodes, then the description will be applied across all nodes in the requestor's allocation.

PMIX_ALLOC_NETWORK_ID **"pmix.alloc.netid"** (**char***)
  Name of the network.

PMIX_ALLOC_BANDWIDTH **"pmix.alloc.bw"** (**float**)
  Mbits/sec.

`PMIX_ALLOC_NETWORK_QOS` `"pmix.alloc.netqos"` (`char*`)
    Quality of service level.

Description

Request an allocation operation from the host resource manager. Several broad categories are envisioned, including the ability to:

- Request allocation of additional resources, including memory, bandwidth, and compute. This should be accomplished in a non-blocking manner so that the application can continue to progress while waiting for resources to become available. Note that the new allocation will be disjoint from (i.e., not affiliated with) the allocation of the requestor - thus the termination of one allocation will not impact the other.
- Extend the reservation on currently allocated resources, subject to scheduling availability and priorities. This includes extending the time limit on current resources, and/or requesting additional resources be allocated to the requesting job. Any additional allocated resources will be considered as part of the current allocation, and thus will be released at the same time.
- Return no-longer-required resources to the scheduler. This includes the "loan" of resources back to the scheduler with a promise to return them upon subsequent request.

7.2.2 `PMIx_Job_control_nb`

The **`PMIx_Job_control_nb`** API enables the application and SMS to coordinate the response to failures and other events. This can include requesting termination of the entire job or a subset of processes within a job, but can also be used in combination with other PMIx capabilities (e.g., allocation support and event notification) for more nuanced responses. For example, an application notified of an incipient over-temperature condition on a node could use the **`PMIx_Allocation_request_nb`** interface to request replacement nodes while simultaneously using the **`PMIx_Job_control_nb`** interface to direct that a checkpoint event be delivered to all processes in the application. If replacement resources are not available, the application might use the **`PMIx_Job_control_nb`** interface to request that the job continue at a lower power setting, perhaps sufficient to avoid the over-temperature failure.

The job control API can also be used by an application to register itself as available for preemption when operating in an environment such as a cloud or where incentives, financial or otherwise, are provided to jobs willing to be preempted. Registration can include attributes indicating how many resources are being offered for preemption (e.g., all or only some portion), whether the application will require time to prepare for preemption, etc. Jobs that request a warning will receive an event notifying them of an impending preemption (possibly including information as to the resources that will be taken away, how much time the application will be given prior to being preempted, whether the preemption will be a suspension or full termination, etc.) so they have an opportunity to save their work. Once the application is ready, it calls the provided event completion callback function to indicate that the SMS is free to suspend or terminate it, and can include directives regarding any desired restart.

Summary

Request a job control action.

Format

PMIx v2.0

C

```c
pmix_status_t
PMIx_Job_control_nb(const pmix_proc_t targets[], size_t ntargets,
                    const pmix_info_t directives[], size_t ndirs,
                    pmix_info_cbfunc_t cbfunc, void *cbdata)
```

C

IN `targets`
Array of proc structures (array of handles)

IN `ntargets`
Number of element in the *targets* array (integer)

IN `directives`
Array of info structures (array of handles)

IN `ndirs`
Number of element in the *directives* array (integer)

IN `cbfunc`
Callback function `pmix_info_cbfunc_t` (function reference)

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

Returns one of the following:

- `PMIX_SUCCESS` , indicating that the request is being processed by the host environment - result will be returned in the provided *cbfunc*. Note that the library *must not* invoke the callback function prior to returning from the API.
- `PMIX_OPERATION_SUCCEEDED` , indicating that the request was immediately processed and returned *success* - the *cbfunc* will *not* be called
- a PMIx error constant indicating either an error in the input or that the request was immediately processed and failed - the *cbfunc* will *not* be called

Required Attributes

PMIx libraries are not required to directly support any attributes for this function. However, any provided attributes must be passed to the host SMS daemon for processing, and the PMIx library is *required* to add the `PMIX_USERID` and the `PMIX_GRPID` attributes of the client process making the request.

Host environments that implement support for this operation are required to support the following attributes:

`PMIX_JOB_CTRL_ID` `"pmix.jctrl.id"` (`char*`)

Provide a string identifier for this request.

PMIX_JOB_CTRL_PAUSE **"pmix.jctrl.pause"** (**bool**)
Pause the specified processes.

PMIX_JOB_CTRL_RESUME **"pmix.jctrl.resume"** (**bool**)
Resume ("un-pause") the specified processes.

PMIX_JOB_CTRL_KILL **"pmix.jctrl.kill"** (**bool**)
Forcibly terminate the specified processes and cleanup.

PMIX_JOB_CTRL_SIGNAL **"pmix.jctrl.sig"** (**int**)
Send given signal to specified processes.

PMIX_JOB_CTRL_TERMINATE **"pmix.jctrl.term"** (**bool**)
Politely terminate the specified processes.

Optional Attributes

The following attributes are optional for host environments that support this operation:

PMIX_JOB_CTRL_CANCEL **"pmix.jctrl.cancel"** (**char***)
Cancel the specified request (**NULL** implies cancel all requests from this requestor).

PMIX_JOB_CTRL_RESTART **"pmix.jctrl.restart"** (**char***)
Restart the specified processes using the given checkpoint ID.

PMIX_JOB_CTRL_CHECKPOINT **"pmix.jctrl.ckpt"** (**char***)
Checkpoint the specified processes and assign the given ID to it.

PMIX_JOB_CTRL_CHECKPOINT_EVENT **"pmix.jctrl.ckptev"** (**bool**)
Use event notification to trigger a process checkpoint.

PMIX_JOB_CTRL_CHECKPOINT_SIGNAL **"pmix.jctrl.ckptsig"** (**int**)
Use the given signal to trigger a process checkpoint.

PMIX_JOB_CTRL_CHECKPOINT_TIMEOUT **"pmix.jctrl.ckptsig"** (**int**)
Time in seconds to wait for a checkpoint to complete.

PMIX_JOB_CTRL_CHECKPOINT_METHOD
"pmix.jctrl.ckmethod" (**pmix_data_array_t**)
Array of **pmix_info_t** declaring each method and value supported by this application.

PMIX_JOB_CTRL_PROVISION **"pmix.jctrl.pvn"** (**char***)
Regular expression identifying nodes that are to be provisioned.

PMIX_JOB_CTRL_PROVISION_IMAGE **"pmix.jctrl.pvnimg"** (**char***)
Name of the image that is to be provisioned.

PMIX_JOB_CTRL_PREEMPTIBLE **"pmix.jctrl.preempt"** (**bool**)

Indicate that the job can be pre-empted.

Description

Request a job control action. The *targets* array identifies the processes to which the requested job control action is to be applied. A **NULL** value can be used to indicate all processes in the caller's namespace. The use of **PMIX_RANK_WILDARD** can also be used to indicate that all processes in the given namespace are to be included.

The directives are provided as **pmix_info_t** structures in the *directives* array. The callback function provides a *status* to indicate whether or not the request was granted, and to provide some information as to the reason for any denial in the **pmix_info_cbfunc_t** array of **pmix_info_t** structures.

7.3 Process and Job Monitoring

In addition to external faults, a common problem encountered in HPC applications is a failure to make progress due to some internal conflict in the computation. These situations can result in a significant waste of resources as the SMS is unaware of the problem, and thus cannot terminate the job. Various watchdog methods have been developed for detecting this situation, including requiring a periodic "heartbeat" from the application and monitoring a specified file for changes in size and/or modification time.

At the request of SMS vendors and members, a monitoring support interface has been included in the PMIx v2 standard. The defined API allows applications to request monitoring, directing what is to be monitored, the frequency of the associated check, whether or not the application is to be notified (via the event notification subsystem) of stall detection, and other characteristics of the operation. In addition, heartbeat and file monitoring methods have been included in the PRI but are active only when requested.

7.3.1 `PMIx_Process_monitor_nb`

Summary

Request that application processes be monitored.

Format

PMIx v2.0

C

```
pmix_status_t
PMIx_Process_monitor_nb(const pmix_info_t *monitor, pmix_status_t error,
                        const pmix_info_t directives[], size_t ndirs,
                        pmix_info_cbfunc_t cbfunc, void *cbdata)
```

C

IN `monitor`
info (handle)

IN `error`
status (integer)

IN `directives`
Array of info structures (array of handles)

IN `ndirs`
Number of elements in the *directives* array (integer)

IN `cbfunc`
Callback function `pmix_info_cbfunc_t` (function reference)

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

Returns one of the following:

- `PMIX_SUCCESS`, indicating that the request is being processed by the host environment - result will be returned in the provided *cbfunc*. Note that the library *must not* invoke the callback function prior to returning from the API.
- `PMIX_OPERATION_SUCCEEDED`, indicating that the request was immediately processed and returned *success* - the *cbfunc* will *not* be called
- a PMIx error constant indicating either an error in the input or that the request was immediately processed and failed - the *cbfunc* will *not* be called

Optional Attributes

The following attributes may be implemented by a PMIx library or by the host environment. If supported by the PMIx server library, then the library must not pass the supported attributes to the host environment. All attributes not directly supported by the server library must be passed to the host environment if it supports this operation, and the library is *required* to add the `PMIX_USERID` and the `PMIX_GRPID` attributes of the requesting process:

`PMIX_MONITOR_ID` `"pmix.monitor.id"` (`char*`)
Provide a string identifier for this request.

`PMIX_MONITOR_CANCEL` `"pmix.monitor.cancel"` (`char*`)
Identifier to be canceled (`NULL` means cancel all monitoring for this process).

PMIX_MONITOR_APP_CONTROL **"pmix.monitor.appctrl"** (**bool**)
The application desires to control the response to a monitoring event.

PMIX_MONITOR_HEARTBEAT **"pmix.monitor.mbeat"** (**void**)
Register to have the PMIx server monitor the requestor for heartbeats.

PMIX_MONITOR_HEARTBEAT_TIME **"pmix.monitor.btime"** (**uint32_t**)
Time in seconds before declaring heartbeat missed.

PMIX_MONITOR_HEARTBEAT_DROPS **"pmix.monitor.bdrop"** (**uint32_t**)
Number of heartbeats that can be missed before generating the event.

PMIX_MONITOR_FILE **"pmix.monitor.fmon"** (**char***)
Register to monitor file for signs of life.

PMIX_MONITOR_FILE_SIZE **"pmix.monitor.fsize"** (**bool**)
Monitor size of given file is growing to determine if the application is running.

PMIX_MONITOR_FILE_ACCESS **"pmix.monitor.faccess"** (**char***)
Monitor time since last access of given file to determine if the application is running.

PMIX_MONITOR_FILE_MODIFY **"pmix.monitor.fmod"** (**char***)
Monitor time since last modified of given file to determine if the application is running.

PMIX_MONITOR_FILE_CHECK_TIME **"pmix.monitor.ftime"** (**uint32_t**)
Time in seconds between checking the file.

PMIX_MONITOR_FILE_DROPS **"pmix.monitor.fdrop"** (**uint32_t**)
Number of file checks that can be missed before generating the event.

Description

Request that application processes be monitored via several possible methods. For example, that the server monitor this process for periodic heartbeats as an indication that the process has not become "wedged". When a monitor detects the specified alarm condition, it will generate an event notification using the provided error code and passing along any available relevant information. It is up to the caller to register a corresponding event handler.

The *monitor* argument is an attribute indicating the type of monitor being requested. For example, **PMIX_MONITOR_FILE** to indicate that the requestor is asking that a file be monitored.

The *error* argument is the status code to be used when generating an event notification alerting that the monitor has been triggered. The range of the notification defaults to **PMIX_RANGE_NAMESPACE** . This can be changed by providing a **PMIX_RANGE** directive.

The *directives* argument characterizes the monitoring request (e.g., monitor file size) and frequency of checking to be done

The *cbfunc* function provides a *status* to indicate whether or not the request was granted, and to provide some information as to the reason for any denial in the **pmix_info_cbfunc_t** array of **pmix_info_t** structures.

7.3.2 PMIx_Heartbeat

Summary

Send a heartbeat to the PMIx server library

Format

PMIx v2.0

C

```
void PMIx_Heartbeat(void)
```

C

Description

A simplified macro wrapping **PMIx_Process_monitor_nb** that sends a heartbeat to the PMIx server library.

7.4 Logging

The logging interface supports posting information by applications and SMS elements to persistent storage. This function is *not* intended for output of computational results, but rather for reporting status and saving state information such as inserting computation progress reports into the application's SMS job log or error reports to the local syslog.

7.4.1 PMIx_Log_nb

Summary

Log data to a data service.

Format

PMIx v2.0

C

```c
pmix_status_t
PMIx_Log_nb(const pmix_info_t data[], size_t ndata,
            const pmix_info_t directives[], size_t ndirs,
            pmix_op_cbfunc_t cbfunc, void *cbdata)
```

C

IN `data`
Array of info structures (array of handles)

IN `ndata`
Number of elements in the *data* array (`size_t`)

IN `directives`
Array of info structures (array of handles)

IN `ndirs`
Number of elements in the *directives* array (`size_t`)

IN `cbfunc`
Callback function `pmix_op_cbfunc_t` (function reference)

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

Return codes are one of the following:

`PMIX_SUCCESS` The logging request is valid and is being processed. The resulting status from the operation will be provided in the callback function. Note that the library *must not* invoke the callback function prior to returning from the API.

`PMIX_OPERATION_SUCCEEDED` , indicating that the request was immediately processed and returned *success* - the *cbfunc* will *not* be called

`PMIX_ERR_BAD_PARAM` The logging request contains at least one incorrect entry that prevents it from being processed. The callback function will *not* be called.

`PMIX_ERR_NOT_SUPPORTED` The PMIx implementation does not support this function. The callback function will *not* be called.

Required Attributes

If the PMIx library does not itself perform this operation, then it is required to pass any attributes provided by the client to the host environment for processing. In addition, it must include the following attributes in the passed *info* array:

`PMIX_USERID` `"pmix.euid"` (`uint32_t`)
Effective user id.

`PMIX_GRPID` `"pmix.egid"` (`uint32_t`)
Effective group id.

Host environments that implement support for this operation are required to support the following attributes:

PMIX_LOG_STDERR **"pmix.log.stderr"** (**char***)
Log string to **stderr**.

PMIX_LOG_STDOUT **"pmix.log.stdout"** (**char***)
Log string to **stdout**.

PMIX_LOG_SYSLOG **"pmix.log.syslog"** (**char***)
Log data to syslog. Defaults to **ERROR** priority.

Optional Attributes

The following attributes are optional for host environments that support this operation:

PMIX_LOG_MSG **"pmix.log.msg"** (**pmix_byte_object_t**)
Message blob to be sent somewhere.

PMIX_LOG_EMAIL **"pmix.log.email"** (**pmix_data_array_t**)
Log via email based on **pmix_info_t** containing directives.

PMIX_LOG_EMAIL_ADDR **"pmix.log.emaddr"** (**char***)
Comma-delimited list of email addresses that are to receive the message.

PMIX_LOG_EMAIL_SUBJECT **"pmix.log.emsub"** (**char***)
Subject line for email.

PMIX_LOG_EMAIL_MSG **"pmix.log.emmsg"** (**char***)
Message to be included in email.

Description

Log data subject to the services offered by the host environment. The data to be logged is provided in the *data* array. The (optional) *directives* can be used to direct the choice of logging channel. The callback function will be executed when the log operation has been completed. The *data* and *directives* arrays must be maintained until the callback is provided.

Advice to users

It is strongly recommended that the **PMIx_Log_nb** API not be used by applications for streaming data as it is not a “performant” transport and can perturb the application since it involves the local PMIx server and host SMS daemon.

CHAPTER 8

Event Notification

This chapter defines the PMIx event notification system. These interfaces are designed to support the reporting of events to/from clients and servers, and between library layers within a single process.

8.1 Notification and Management

PMIx event notification provides an asynchronous out-of-band mechanism for communicating events between application processes and/or elements of the SMS. Its uses span a wide range that includes fault notification, coordination between multiple programming libraries within a single process, and workflow orchestration for non-synchronous programming models. Events can be divided into two distinct classes:

- *Job-specific events* directly relate to a job executing within the session, such as a debugger attachment, process failure within a related job, or events generated by an application process. Events in this category are to be immediately delivered to the PMIx server library for relay to the related local processes.
- *Environment events* indirectly relate to a job but do not specifically target the job itself. This category includes SMS-generated events such as Error Check and Correction (ECC) errors, temperature excursions, and other non-job conditions that might directly affect a session's resources, but would never include an event generated by an application process. Note that although these do potentially impact the session's jobs, they are not directly tied to those jobs. Thus, events in this category are to be delivered to the PMIx server library only upon request.

Both SMS elements and applications can register for events of either type.

Advice to PMIx library implementers

Race conditions can cause the registration to come after events of possible interest (e.g., a memory ECC event that occurs after start of execution but prior to registration, or an application process generating an event prior to another process registering to receive it). SMS vendors are *requested* to cache environment events for some time to mitigate this situation, but are not *required* to do so. However, PMIx implementers are *required* to cache all events received by the PMIx server library and to deliver them to registering clients in the same order in which they were received

Advice to users

Applications must be aware that they may not receive environment events that occur prior to registration, depending upon the capabilities of the host SMS.

The generator of an event can specify the *target range* for delivery of that event. Thus, the generator can choose to limit notification to processes on the local node, processes within the same job as the generator, processes within the same allocation, other threads within the same process, only the SMS (i.e., not to any application processes), all application processes, or to a custom range based on specific process identifiers. Only processes within the given range that register for the provided event code will be notified. In addition, the generator can use attributes to direct that the event not be delivered to any default event handlers, or to any multi-code handler (as defined below).

Event notifications provide the process identifier of the source of the event plus the event code and any additional information provided by the generator. When an event notification is received by a process, the registered handlers are scanned for their event code(s), with matching handlers assembled into an *event chain* for servicing. Note that users can also specify a *source range* when registering an event (using the same range designators described above) to further limit when they are to be invoked. When assembled, PMIx event chains are ordered based on both the specificity of the event handler and user directives at time of handler registration. By default, handlers are grouped into three categories based on the number of event codes that can trigger the callback:

- *single-code* handlers are serviced first as they are the most specific. These are handlers that are registered against one specific event code.
- *multi-code* handlers are serviced once all single-code handlers have completed. The handler will be included in the chain upon receipt of an event matching any of the provided codes.
- *default* handlers are serviced once all multi-code handlers have completed. These handlers are always included in the chain unless the generator specifically excludes them.

Users can specify the callback order of a handler within its category at the time of registration. Ordering can be specified either by providing the relevant returned event handler registration ID or using event handler names, if the user specified an event handler name when registering the corresponding event. Thus, users can specify that a given handler be executed before or after another handler should both handlers appear in an event chain (the ordering is ignored if the other handler isn't included). Note that ordering does not imply immediate relationships. For example, multiple handlers registered to be serviced after event handler *A* will all be executed after *A*, but are not guaranteed to be executed in any particular order amongst themselves.

In addition, one event handler can be declared as the *first* handler to be executed in the chain. This handler will *always* be called prior to any other handler, regardless of category, provided the incoming event matches both the specified range and event code. Only one handler can be so designated — attempts to designate additional handlers as *first* will return an error. Deregistration of the declared *first* handler will re-open the position for subsequent assignment.

Similarly, one event handler can be declared as the *last* handler to be executed in the chain. This handler will *always* be called after all other handlers have executed, regardless of category, provided the incoming event matches both the specified range and event code. Note that this handler will not be called if the chain is terminated by an earlier handler. Only one handler can be designated as *last* — attempts to designate additional handlers as *last* will return an error. Deregistration of the declared *last* handler will re-open the position for subsequent assignment.

Advice to users

Note that the *last* handler is called *after* all registered default handlers that match the specified range of the incoming event unless a handler prior to it terminates the chain. Thus, if the application intends to define a *last* handler, it should ensure that no default handler aborts the process before it.

Upon completing its work and prior to returning, each handler *must* call the event handler completion function provided when it was invoked (including a status code plus any information to be passed to later handlers) so that the chain can continue being progressed. PMIx automatically aggregates the status and any results of each handler (as provided in the completion callback) with status from all prior handlers so that each step in the chain has full knowledge of what preceded it. An event handler can terminate all further progress along the chain by passing the **PMIX_EVENT_ACTION_COMPLETE** status to the completion callback function.

8.1.1 PMIx_Register_event_handler

Summary

Register an event handler

Format

PMIx v2.0

C

```
void
PMIx_Register_event_handler(pmix_status_t codes[], size_t ncodes,
                            pmix_info_t info[], size_t ninfo,
                            pmix_notification_fn_t evhdlr,
                            pmix_evhdlr_reg_cbfunc_t cbfunc,
                            void *cbdata);
```

C

IN **codes**
Array of status codes (array of **pmix_status_t**)

IN **ncodes**
Number of elements in the *codes* array (**size_t**)

IN **info**
Array of info structures (array of handles)

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

IN **evhdlr**
Event handler to be called **pmix_notification_fn_t** (function reference)

IN **cbfunc**
Callback function **pmix_evhdlr_reg_cbfunc_t** (function reference)

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

Required Attributes

The following attributes are required to be supported by all PMIx libraries:

PMIX_EVENT_HDLR_NAME **"pmix.evname"** (**char***)
String name identifying this handler.

PMIX_EVENT_HDLR_FIRST **"pmix.evfirst"** (**bool**)
Invoke this event handler before any other handlers.

PMIX_EVENT_HDLR_LAST **"pmix.evlast"** (**bool**)
Invoke this event handler after all other handlers have been called.

PMIX_EVENT_HDLR_FIRST_IN_CATEGORY **"pmix.evfirstcat"** (**bool**)
Invoke this event handler before any other handlers in this category.

PMIX_EVENT_HDLR_LAST_IN_CATEGORY **"pmix.evlastcat"** (**bool**)
Invoke this event handler after all other handlers in this category have been called.

PMIX_EVENT_HDLR_BEFORE **"pmix.evbefore"** (**char***)
Put this event handler immediately before the one specified in the **(char*)** value.

PMIX_EVENT_HDLR_AFTER **"pmix.evafter"** (**char***)
Put this event handler immediately after the one specified in the **(char*)** value.

PMIX_EVENT_HDLR_PREPEND **"pmix.evprepend"** (**bool**)
Prepend this handler to the precedence list within its category.

PMIX_EVENT_HDLR_APPEND **"pmix.evappend"** (**bool**)
Append this handler to the precedence list within its category.

PMIX_EVENT_CUSTOM_RANGE **"pmix.evrange"** (**pmix_data_array_t***)

Array of **pmix_proc_t** defining range of event notification.

PMIX_RANGE **"pmix.range"** (**pmix_data_range_t**)
Value for calls to publish/lookup/unpublish or for monitoring event notifications.

PMIX_EVENT_RETURN_OBJECT **"pmix.evobject"** (**void ***)
Object to be returned whenever the registered callback function **cbfunc** is invoked. The object will *only* be returned to the process that registered it.

Host environments that implement support for PMIx event notification are required to support the following attributes:

PMIX_EVENT_AFFECTED_PROC **"pmix.evproc"** (**pmix_proc_t**)
The single process that was affected.

PMIX_EVENT_AFFECTED_PROCS **"pmix.evaffected"** (**pmix_data_array_t***)
Array of **pmix_proc_t** defining affected processes.

Optional Attributes

Host environments that support PMIx event notification *may* offer notifications for environmental events impacting the job and for SMS events relating to the job. The following attributes are optional for host environments that suppport this operation:

PMIX_EVENT_TERMINATE_SESSION **"pmix.evterm.sess"** (**bool**)
The RM intends to terminate this session.

PMIX_EVENT_TERMINATE_JOB **"pmix.evterm.job"** (**bool**)
The RM intends to terminate this job.

PMIX_EVENT_TERMINATE_NODE **"pmix.evterm.node"** (**bool**)
The RM intends to terminate all processes on this node.

PMIX_EVENT_TERMINATE_PROC **"pmix.evterm.proc"** (**bool**)
The RM intends to terminate just this process.

PMIX_EVENT_ACTION_TIMEOUT **"pmix.evtimeout"** (**int**)
The time in seconds before the RM will execute error response.

PMIX_EVENT_SILENT_TERMINATION **"pmix.evsilentterm"** (**bool**)
Do not generate an event when this job normally terminates.

Description

Register an event handler to report events. Note that the codes being registered do *not* need to be PMIx error constants — any integer value can be registered. This allows for registration of non-PMIx events such as those defined by a particular SMS vendor or by an application itself.

Advice to users

In order to avoid potential conflicts, users are advised to only define codes that lie outside the range of the PMIx standard's error codes. Thus, SMS vendors and application developers should constrain their definitions to positive values or negative values beyond the **PMIX_EXTERNAL_ERR_BASE** boundary.

Upon completion, the callback will receive a status based on the following table:

PMIX_SUCCESS The event handler was successfully registered - the event handler identifier is returned in the callback.

PMIX_ERR_BAD_PARAM One or more of the directives provided in the *info* array was unrecognized.

PMIX_ERR_NOT_SUPPORTED The PMIx implementation does not support event notification, or the host SMS does not support notification of the specified event code.

The callback function *must not* be executed prior to returning from the API.

Advice to users

As previously stated, upon completing its work, and prior to returning, each handler *must* call the event handler completion function provided when it was invoked (including a status code plus any information to be passed to later handlers) so that the chain can continue being progressed. An event handler can terminate all further progress along the chain by passing the **PMIX_EVENT_ACTION_COMPLETE** status to the completion callback function. Note that the parameters passed to the event handler (e.g., the *info* and *results* arrays) will cease to be valid once the completion function has been called - thus, any information in the incoming parameters that will be referenced following the call to the completion function must be copied.

8.1.2 PMIx_Deregister_event_handler

Summary

Deregister an event handler.

Format

PMIx v2.0

C

```c
void
PMIx_Deregister_event_handler(size_t evhdlr_ref,
                              pmix_op_cbfunc_t cbfunc,
                              void *cbdata);
```

C

IN `evhdlr_ref`
Event handler ID returned by registration (`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)

Description

Deregister an event handler. If non-NULL, the provided cbfunc will be called to confirm removal of the designated handler, including a status code as per the following:

`PMIX_SUCCESS` The event handler was successfully deregistered.
`PMIX_ERR_BAD_PARAM` The provided *evhdlr_ref* was unrecognized.
`PMIX_ERR_NOT_SUPPORTED` The PMIx implementation does not support event notification.

The callback function *must not* be executed prior to returning from the API.

8.1.3 `PMIx_Notify_event`

Summary

Report an event for notification via any registered event handler.

Format

PMIx v2.0

C

```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)

`PMIX_SUCCESS` The notification request is valid and is being processed. The callback function will be called when the process-local operation is complete and will provide the resulting status of that operation. Note that this does *not* reflect the success or failure of delivering the event to any recipients. The callback function *must not* be executed prior to returning from the API.

`PMIX_OPERATION_SUCCEEDED` , indicating that the request was immediately processed and returned *success* - the *cbfunc* will *not* be called

`PMIX_ERR_BAD_PARAM` The request contains at least one incorrect entry that prevents it from being processed. The callback function will *not* be called.

`PMIX_ERR_NOT_SUPPORTED` The PMIx implementation does not support event notification, or in the case of a PMIx server calling the API, the range extended beyond the local node and the host SMS environment does not support event notification. The callback function will *not* be called.

Required Attributes

The following attributes are required to be supported by all PMIx libraries:

`PMIX_EVENT_NON_DEFAULT` `"pmix.evnondef"` (`bool`)
Event is not to be delivered to default event handlers.

`PMIX_EVENT_CUSTOM_RANGE` `"pmix.evrange"` (`pmix_data_array_t*`)
Array of `pmix_proc_t` defining range of event notification.

Host environments that implement support for PMIx event notification are required to provide the following attributes for all events generated by the environment:

`PMIX_EVENT_AFFECTED_PROC` `"pmix.evproc"` (`pmix_proc_t`)

The single process that was affected.

PMIX_EVENT_AFFECTED_PROCS **"pmix.evaffected"** (**pmix_data_array_t***)
Array of **pmix_proc_t** defining affected processes.

Description

Report an event for notification via any registered event handler. This function can be called by any PMIx process, including application processes, PMIx servers, and SMS elements. The PMIx server calls this API to report events it detected itself so that the host SMS daemon distribute and handle them, and to pass events given to it by its host down to any attached client processes for processing. Examples might include notification of the failure of another process, detection of an impending node failure due to rising temperatures, or an intent to preempt the application. Events may be locally generated or come from anywhere in the system.

Host SMS daemons call the API to pass events down to its embedded PMIx server both for transmittal to local client processes and for the server's own internal processing.

Client application processes can call this function to notify the SMS and/or other application processes of an event it encountered. Note that processes are not constrained to report status values defined in the official PMIx standard — any integer value can be used. Thus, applications are free to define their own internal events and use the notification system for their own internal purposes.

Advice to users

The callback function will be called upon completion of the **notify_event** function's actions. At that time, any messages required for executing the operation (e.g., to send the notification to the local PMIx server) will have been queued, but may not yet have been transmitted. The caller is required to maintain the input data until the callback function has been executed — the sole purpose of the callback function is to indicate when the input data is no longer required.

CHAPTER 9

Data Packing and Unpacking

PMIx intentionally does not include support for internode communications in the standard, instead relying on its host SMS environment to transfer any needed data and/or requests between nodes. These operations frequently involve PMIx-defined public data structures that include binary data. Many HPC clusters are homogeneous, and so transferring the structures can be done rather simply. However, greater effort is required in heterogeneous environments to ensure binary data is correctly transferred. PMIx buffer manipulation functions are provided for this purpose via standardized interfaces to ease adoption.

9.1 Support Macros

PMIx provides a set of convenience macros for creating, initiating, and releasing data buffers.

9.1.1 PMIX_DATA_BUFFER_CREATE

Summary

Allocate memory for a **pmix_data_buffer_t** object and initialize it

Format

PMIx v2.0

C

```
PMIX_DATA_BUFFER_CREATE(buffer);
```

C

OUT buffer
Variable to be assigned the pointer to the allocated **pmix_data_buffer_t** (handle)

Description

This macro uses *calloc* to allocate memory for the buffer and initialize all fields in it

9.1.2 PMIX_DATA_BUFFER_RELEASE

Summary

Free a **pmix_data_buffer_t** object and the data it contains

Format

PMIx v2.0

C

```
PMIX_DATA_BUFFER_RELEASE(buffer);
```

C

IN **buffer**
Pointer to the **pmix_data_buffer_t** to be released (handle)

Description

Free's the data contained in the buffer, and then free's the buffer itself

9.1.3 PMIX_DATA_BUFFER_CONSTRUCT

Summary

Initialize a statically declared **pmix_data_buffer_t** object

Format

PMIx v2.0

C

```
PMIX_DATA_BUFFER_CONSTRUCT(buffer);
```

C

IN **buffer**
Pointer to the allocated **pmix_data_buffer_t** that is to be initialized (handle)

Description

Initialize a pre-allocated buffer object

9.1.4 PMIX_DATA_BUFFER_DESTRUCT

Summary

Release the data contained in a **pmix_data_buffer_t** object

Format

PMIx v2.0 C

```
PMIX_DATA_BUFFER_DESTRUCT(buffer);
```

C

IN `buffer`
Pointer to the `pmix_data_buffer_t` whose data is to be released (handle)

Description

Free's the data contained in a `pmix_data_buffer_t` object

9.1.5 PMIX_DATA_BUFFER_LOAD

Summary

Load a blob into a `pmix_data_buffer_t` object

Format

PMIx v2.0 C

```
PMIX_DATA_BUFFER_LOAD(buffer, data, size);
```

C

IN `buffer`
Pointer to a pre-allocated `pmix_data_buffer_t` (handle)
IN `data`
Pointer to a blob (`char*`)
IN `size`
Number of bytes in the blob `size_t`

Description

Load the given data into the provided `pmix_data_buffer_t` object, usually done in preparation for unpacking the provided data. Note that the data is *not* copied into the buffer - thus, the blob must not be released until after operations on the buffer have completed.

9.1.6 PMIX_DATA_BUFFER_UNLOAD

Summary

Unload the data from a `pmix_data_buffer_t` object

Format

PMIx v2.0

C

```c
PMIX_DATA_BUFFER_UNLOAD(buffer, data, size);
```

C

IN **`buffer`**
Pointer to the **`pmix_data_buffer_t`** whose data is to be extracted (handle)

OUT **`data`**
Variable to be assigned the pointer to the extracted blob (**`void*`**)

OUT **`size`**
Variable to be assigned the number of bytes in the blob **`size_t`**

Description

Extract the data in a buffer, assigning the pointer to the data (and the number of bytes in the blob) to the provided variables, usually done to transmit the blob to a remote process for unpacking. The buffer's internal pointer will be set to NULL to protect the data upon buffer destruct or release - thus, the user is responsible for releasing the blob when done with it.

9.2 General Routines

The following routines are provided to support internode transfers in heterogeneous environments.

9.2.1 `PMIx_Data_pack`

Summary

Pack one or more values of a specified type into a buffer, usually for transmission to another process

Format

PMIx v2.0

C

```c
pmix_status_t
PMIx_Data_pack(const pmix_proc_t *target,
               pmix_data_buffer_t *buffer,
               void *src, int32_t num_vals,
               pmix_data_type_t type);
```

C

IN `target`
Pointer to a `pmix_proc_t` containing the nspace/rank of the process that will be unpacking the final buffer. A NULL value may be used to indicate that the target is based on the same PMIx version as the caller. Note that only the target's nspace is relevant. (handle)

IN `buffer`
Pointer to a `pmix_data_buffer_t` where the packed data is to be stored (handle)

IN `src`
Pointer to a location where the data resides. Strings are to be passed as (char **) — i.e., the caller must pass the address of the pointer to the string as the (void*). This allows the caller to pass multiple strings in a single call. (memory reference)

IN `num_vals`
Number of elements pointed to by the *src* pointer. A string value is counted as a single value regardless of length. The values must be contiguous in memory. Arrays of pointers (e.g., string arrays) should be contiguous, although the data pointed to need not be contiguous across array entries.(`int32_t`)

IN `type`
The type of the data to be packed (`pmix_data_type_t`)

`PMIX_SUCCESS` The data has been packed as requested
`PMIX_ERR_NOT_SUPPORTED` The PMIx implementation does not support this function.
`PMIX_ERR_BAD_PARAM` The provided buffer or src is **`NULL`**
`PMIX_ERR_UNKNOWN_DATA_TYPE` The specified data type is not known to this implementation
`PMIX_ERR_OUT_OF_RESOURCE` Not enough memory to support the operation
`PMIX_ERROR` General error

Description

The pack function packs one or more values of a specified type into the specified buffer. The buffer must have already been initialized via the `PMIX_DATA_BUFFER_CREATE` or `PMIX_DATA_BUFFER_CONSTRUCT` macros — otherwise, `PMIx_Data_pack` will return an error. Providing an unsupported type flag will likewise be reported as an error.

Note that any data to be packed that is not hard type cast (i.e., not type cast to a specific size) may lose precision when unpacked by a non-homogeneous recipient. The `PMIx_Data_pack` function will do its best to deal with heterogeneity issues between the packer and unpacker in such cases. Sending a number larger than can be handled by the recipient will return an error code (generated upon unpacking) — the error cannot be detected during packing.

The namespace of the intended recipient of the packed buffer (i.e., the process that will be unpacking it) is used solely to resolve any data type differences between PMIx versions. The recipient must, therefore, be known to the user prior to calling the pack function so that the PMIx library is aware of the version the recipient is using. Note that all processes in a given namespace are *required* to use the same PMIx version — thus, the caller must only know at least one process from the target's namespace.

9.2.2 PMIx_Data_unpack

Summary

Unpack values from a **pmix_data_buffer_t**

Format

PMIx v2.0

C

```
pmix_status_t
PMIx_Data_unpack(const pmix_proc_t *source,
                 pmix_data_buffer_t *buffer, void *dest,
                 int32_t *max_num_values,
                 pmix_data_type_t type);

```

C

IN **source**
Pointer to a **pmix_proc_t** structure containing the nspace/rank of the process that packed the provided buffer. A NULL value may be used to indicate that the source is based on the same PMIx version as the caller. Note that only the source's nspace is relevant. (handle)

IN **buffer**
A pointer to the buffer from which the value will be extracted. (handle)

INOUT **dest**
A pointer to the memory location into which the data is to be stored. Note that these values will be stored contiguously in memory. For strings, this pointer must be to (char**) to provide a means of supporting multiple string operations. The unpack function will allocate memory for each string in the array - the caller must only provide adequate memory for the array of pointers. (**void***)

INOUT **max_num_values**
The number of values to be unpacked — upon completion, the parameter will be set to the actual number of values unpacked. In most cases, this should match the maximum number provided in the parameters — but in no case will it exceed the value of this parameter. Note that unpacking fewer values than are actually available will leave the buffer in an unpackable state — the function will return an error code to warn of this condition.(**int32_t**)

IN **type**
The type of the data to be unpacked — must be one of the PMIx defined data types (**pmix_data_type_t**)

PMIX_SUCCESS The data has been unpacked as requested

PMIX_ERR_NOT_SUPPORTED The PMIx implementation does not support this function.
PMIX_ERR_BAD_PARAM The provided buffer or dest is **NULL**
PMIX_ERR_UNKNOWN_DATA_TYPE The specified data type is not known to this implementation
PMIX_ERR_OUT_OF_RESOURCE Not enough memory to support the operation
PMIX_ERROR General error

Description

The unpack function unpacks the next value (or values) of a specified type from the given buffer. The buffer must have already been initialized via an **PMIX_DATA_BUFFER_CREATE** or **PMIX_DATA_BUFFER_CONSTRUCT** call (and assumedly filled with some data) — otherwise, the unpack_value function will return an error. Providing an unsupported type flag will likewise be reported as an error, as will specifying a data type that *does not* match the type of the next item in the buffer. An attempt to read beyond the end of the stored data held in the buffer will also return an error.

NOTE: it is possible for the buffer to be corrupted and that PMIx will *think* there is a proper variable type at the beginning of an unpack region — but that the value is bogus (e.g., just a byte field in a string array that so happens to have a value that matches the specified data type flag). Therefore, the data type error check is *not* completely safe.

Unpacking values is a "nondestructive" process — i.e., the values are not removed from the buffer. It is therefore possible for the caller to re-unpack a value from the same buffer by resetting the unpack_ptr.

Warning: The caller is responsible for providing adequate memory storage for the requested data. The user must provide a parameter indicating the maximum number of values that can be unpacked into the allocated memory. If more values exist in the buffer than can fit into the memory storage, then the function will unpack what it can fit into that location and return an error code indicating that the buffer was only partially unpacked.

Note that any data that was not hard type cast (i.e., not type cast to a specific size) when packed may lose precision when unpacked by a non-homogeneous recipient. PMIx will do its best to deal with heterogeneity issues between the packer and unpacker in such cases. Sending a number larger than can be handled by the recipient will return an error code generated upon unpacking — these errors cannot be detected during packing.

The namespace of the process that packed the buffer is used solely to resolve any data type differences between PMIx versions. The packer must, therefore, be known to the user prior to calling the pack function so that the PMIx library is aware of the version the packer is using. Note that all processes in a given namespace are *required* to use the same PMIx version — thus, the caller must only know at least one process from the packer's namespace.

9.2.3 PMIx_Data_copy

Summary

Copy a data value from one location to another.

Format

PMIx v2.0

C

```c
pmix_status_t
PMIx_Data_copy(void **dest, void *src,
               pmix_data_type_t type);
```

C

IN `dest`
The address of a pointer into which the address of the resulting data is to be stored. (`void**`)

IN `src`
A pointer to the memory location from which the data is to be copied (handle)

IN `type`
The type of the data to be copied — must be one of the PMIx defined data types. (`pmix_data_type_t`)

`PMIX_SUCCESS` The data has been copied as requested
`PMIX_ERR_NOT_SUPPORTED` The PMIx implementation does not support this function.
`PMIX_ERR_BAD_PARAM` The provided src or dest is `NULL`
`PMIX_ERR_UNKNOWN_DATA_TYPE` The specified data type is not known to this implementation
`PMIX_ERR_OUT_OF_RESOURCE` Not enough memory to support the operation
`PMIX_ERROR` General error

Description

Since registered data types can be complex structures, the system needs some way to know how to copy the data from one location to another (e.g., for storage in the registry). This function, which can call other copy functions to build up complex data types, defines the method for making a copy of the specified data type.

9.2.4 PMIx_Data_print

Summary

Pretty-print a data value.

Format

PMIx v2.0

C

```c
pmix_status_t
PMIx_Data_print(char **output, char *prefix,
                void *src, pmix_data_type_t type);
```

C

IN `output`
The address of a pointer into which the address of the resulting output is to be stored. (`char**`)

IN `prefix`
String to be prepended to the resulting output (`char*`)

IN `src`
A pointer to the memory location of the data value to be printed (handle)

IN `type`
The type of the data value to be printed — must be one of the PMIx defined data types. (`pmix_data_type_t`)

`PMIX_SUCCESS` The data has been printed as requested

`PMIX_ERR_BAD_PARAM` The provided data type is not recognized.

`PMIX_ERR_NOT_SUPPORTED` The PMIx implementation does not support this function.

Description

Since registered data types can be complex structures, the system needs some way to know how to print them (i.e., convert them to a string representation). Primarily for debug purposes.

9.2.5 `PMIx_Data_copy_payload`

Summary

Copy a payload from one buffer to another

Format

PMIx v2.0

C

```c
pmix_status_t
PMIx_Data_copy_payload(pmix_data_buffer_t *dest,
                       pmix_data_buffer_t *src);
```

C

IN `dest`
Pointer to the destination `pmix_data_buffer_t` (handle)

IN `src`
Pointer to the source `pmix_data_buffer_t` (handle)

`PMIX_SUCCESS` The data has been copied as requested
`PMIX_ERR_BAD_PARAM` The src and dest `pmix_data_buffer_t` types do not match
`PMIX_ERR_NOT_SUPPORTED` The PMIx implementation does not support this function.

Description

This function will append a copy of the payload in one buffer into another buffer. Note that this is *not* a destructive procedure — the source buffer's payload will remain intact, as will any pre-existing payload in the destination's buffer. Only the unpacked portion of the source payload will be copied.

CHAPTER 10

Server-Specific Interfaces

The RM daemon that hosts the PMIx server library interacts with that library in two distinct manners. First, PMIx provides a set of APIs by which the host can request specific services from its library. This includes generating regular expressions, registering information to be passed to client processes, and requesting information on behalf of a remote process. Note that the host always has access to all PMIx client APIs - the functions listed below are in addition to those available to a PMIx client.

Second, the host can provide a set of callback functions by which the PMIx server library can pass requests upward for servicing by the host. These include notifications of client connection and finalize, as well as requests by clients for information and/or services that the PMIx server library does not itself provide.

10.1 Server Support Functions

The following APIs allow the RM daemon that hosts the PMIx server library to request specific services from the PMIx library.

10.1.1 `PMIx_generate_regex`

Summary

Generate a regular expression representation of the input string.

Format

PMIx v1.0

C

```
pmix_status_t
PMIx_generate_regex(const char *input, char **regex)
```

C

IN `input`
String to process (string)

OUT `regex`
Regular expression representation of *input* (string)

Returns **`PMIX_SUCCESS`** or a negative value corresponding to a PMIx error constant.

Description

Given a comma-separated list of *input* values, generate a regular expression that can be passed down to the PMIx client for parsing. The caller is responsible for free'ing the resulting string.

If values have leading zero's, then that is preserved, as are prefix and suffix strings. For example, an input string of
"**`odin009.org,odin010.org,odin011.org,odin012.org,odin[102-107].org`**"
will return a regular expression of "**`pmix:odin[009-012,102-107].org`**"

Advice to users

The returned regular expression will have a "**`pmix:`**" at the beginning of the string. This informs the PMIx parser that the string was produced using the PRI's regular expression generator, and thus that same plugin should be used for parsing the string

10.1.2 `PMIx_generate_ppn`

Summary

Generate a regular expression representation of the input string.

Format

PMIx v1.0 C

```
pmix_status_t PMIx_generate_ppn(const char *input, char **ppn)
```

C

IN **`input`**
String to process (string)

OUT **`regex`**
Regular expression representation of *input* (string)

Returns **`PMIX_SUCCESS`** or a negative value corresponding to a PMIx error constant.

Description

The input is expected to consist of a semicolon-separated list of ranges representing the ranks of processes on each node of the job. Thus, an input of "1-4;2-5;8,10,11,12;6,7,9" would generate a regex of "pmix:2x(3);8,10-12;6-7,9"

Advice to users

The returned regular expression will have a “pmix:” at the beginning of the string. This informs the PMIx parser that the string was produced using the PRI’s regular expression generator, and thus that same plugin should be used for parsing the string

10.1.3 PMIx_server_register_nspace

Summary

Setup the data about a particular namespace.

Format

PMIx v1.0

C

```
pmix_status_t
PMIx_server_register_nspace(const pmix_nspace_t nspace,
                            int nlocalprocs,
                            pmix_info_t info[], size_t ninfo,
                            pmix_op_cbfunc_t cbfunc, void *cbdata)
```

C

IN `nspace`
namespace (string)

IN `nlocalprocs`
number of local processes (integer)

IN `info`
Array of info structures (array of handles)

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

IN `cbfunc`
Callback function **`pmix_op_cbfunc_t`** (function reference)

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

Returns one of the following:

- **PMIX_SUCCESS** , indicating that the request is being processed by the host environment - result will be returned in the provided *cbfunc*. Note that the library *must not* invoke the callback function prior to returning from the API.
- **PMIX_OPERATION_SUCCEEDED** , indicating that the request was immediately processed and returned *success* - the *cbfunc* will *not* be called
- a PMIx error constant indicating either an error in the input or that the request was immediately processed and failed - the *cbfunc* will *not* be called

Required Attributes

The following attributes are *required* to be supported by all PMIx libraries:

PMIX_REGISTER_NODATA **"pmix.reg.nodata"** (**bool**)
Registration is for this namespace only, do not copy job data - this attribute is not accessed using the **PMIx_Get**

Host environments are *required* to provide the following attributes:

- for the session containing the given namespace:
  - **PMIX_UNIV_SIZE** **"pmix.univ.size"** (**uint32_t**)
    Number of allocated slots in a session - each slot may or may not be occupied by an executing process. Note that this attribute is the equivalent to the combination of **PMIX_SESSION_INFO_ARRAY** with the **PMIX_NUM_SLOTS** entry in the array - it is included in the Standard for historical reasons.
- for the given namespace:
  - **PMIX_JOBID** **"pmix.jobid"** (**char***)
    Job identifier assigned by the scheduler.
  - **PMIX_JOB_SIZE** **"pmix.job.size"** (**uint32_t**)
    Total number of processes in this job across all contained applications
  - **PMIX_MAX_PROCS** **"pmix.max.size"** (**uint32_t**)
    Maximum number of processes for this job.
  - **PMIX_NODE_MAP** **"pmix.nmap"** (**char***)
    Regular expression of nodes - see 10.1.3.1 for an explanation of its generation.
  - **PMIX_PROC_MAP** **"pmix.pmap"** (**char***)
    Regular expression describing processes on each node - see 10.1.3.1 for an explanation of its generation.
- for its own node:
  - **PMIX_LOCAL_SIZE** **"pmix.local.size"** (**uint32_t**)
    Number of processes in this job on this node.
  - **PMIX_LOCAL_PEERS** **"pmix.lpeers"** (**char***)

Comma-delimited list of ranks on this node within the specified namespace - referenced using **PMIX_RANK_WILDCARD** .

  - **PMIX_LOCAL_CPUSETS** **"pmix.lcpus"** (**char***)
    Colon-delimited cpusets of local peers within the specified namespace - referenced using **PMIX_RANK_WILDCARD** .

- for each process in the given namespace:
  - **PMIX_RANK** **"pmix.rank"** (**pmix_rank_t**)
    Process rank within the job.
  - **PMIX_LOCAL_RANK** **"pmix.lrank"** (**uint16_t**)
    Local rank on this node within this job.
  - **PMIX_NODE_RANK** **"pmix.nrank"** (**uint16_t**)
    Process rank on this node spanning all jobs.
  - **PMIX_NODEID** **"pmix.nodeid"** (**uint32_t**)
    Node identifier where the specified process is located, expressed as the node's index (beginning at zero) in the array resulting from expansion of the **PMIX_NODE_MAP** regular expression for the **job**

If more than one application is included in the namespace, then the host environment is also *required* to provide the following attributes:

- for each application:
  - **PMIX_APPNUM** **"pmix.appnum"** (**uint32_t**)
    Application number within the job.
  - **PMIX_APPLDR** **"pmix.aldr"** (**pmix_rank_t**)
    Lowest rank in this application within this job - referenced using **PMIX_RANK_WILDCARD** .
  - **PMIX_APP_SIZE** **"pmix.app.size"** (**uint32_t**)
    Number of processes in this application.
- for each process:
  - **PMIX_APP_RANK** **"pmix.apprank"** (**pmix_rank_t**)
    Process rank within this application.
  - **PMIX_APPNUM** **"pmix.appnum"** (**uint32_t**)
    Application number within the job.

Optional Attributes

The following attributes *may* be provided by host environments:

- for the session containing the given namespace:
  - **PMIX_SESSION_ID** **"pmix.session.id"** (**uint32_t**)
    Session identifier - referenced using **PMIX_RANK_WILDCARD** .
- for the given namespace:
  - **PMIX_SERVER_NSPACE** **"pmix.srv.nspace"** (**char***)
    Name of the namespace to use for this PMIx server.
  - **PMIX_SERVER_RANK** **"pmix.srv.rank"** (**pmix_rank_t**)
    Rank of this PMIx server
  - **PMIX_NPROC_OFFSET** **"pmix.offset"** (**pmix_rank_t**)
    Starting global rank of this job - referenced using **PMIX_RANK_WILDCARD** .
  - **PMIX_ALLOCATED_NODELIST** **"pmix.alist"** (**char***)
    Comma-delimited list of all nodes in this allocation regardless of whether or not they currently host processes - referenced using **PMIX_RANK_WILDCARD** .
  - **PMIX_JOB_NUM_APPS** **"pmix.job.napps"** (**uint32_t**)
    Number of applications in this job.
  - **PMIX_MAPBY** **"pmix.mapby"** (**char***)
    Process mapping policy - when accessed using **PMIx_Get** , use the **PMIX_RANK_WILDCARD** value for the rank to discover the mapping policy used for the provided namespace
  - **PMIX_RANKBY** **"pmix.rankby"** (**char***)
    Process ranking policy - when accessed using **PMIx_Get** , use the **PMIX_RANK_WILDCARD** value for the rank to discover the ranking algorithm used for the provided namespace
  - **PMIX_BINDTO** **"pmix.bindto"** (**char***)
    Process binding policy - when accessed using **PMIx_Get** , use the **PMIX_RANK_WILDCARD** value for the rank to discover the binding policy used for the provided namespace
- for its own node:
  - **PMIX_AVAIL_PHYS_MEMORY** **"pmix.pmem"** (**uint64_t**)
    Total available physical memory on this node.
  - **PMIX_HWLOC_XML_V1** **"pmix.hwlocxml1"** (**char***)
    XML representation of local topology using hwloc's v1.x format.
  - **PMIX_HWLOC_XML_V2** **"pmix.hwlocxml2"** (**char***)

XML representation of local topology using hwloc's v2.x format.

- **PMIX_LOCALLDR** **"pmix.lldr"** (**pmix_rank_t**)
  Lowest rank on this node within this job - referenced using **PMIX_RANK_WILDCARD** .

- **PMIX_NODE_SIZE** **"pmix.node.size"** (**uint32_t**)
  Number of processes across all jobs on this node.
- **PMIX_LOCAL_PROCS** **"pmix.lprocs"** (**pmix_proc_t array**)
  Array of **pmix_proc_t** of all processes on the specified node - referenced using **PMIX_RANK_WILDCARD** .

• for each process in the given namespace:

- **PMIX_PROCID** **"pmix.procid"** (**pmix_proc_t**)
  Process identifier
- **PMIX_GLOBAL_RANK** **"pmix.grank"** (**pmix_rank_t**)
  Process rank spanning across all jobs in this session.
- **PMIX_HOSTNAME** **"pmix.hname"** (**char***)
  Name of the host where the specified process is running.

Attributes not directly provided by the host environment *may* be derived by the PMIx server library from other required information and included in the data made available to the server library's clients.

Description

Pass job-related information to the PMIx server library for distribution to local client processes.

Advice to PMIx server hosts

Host environments are *required* to execute this operation prior to starting any local application process within the given namespace.

The PMIx server must register *all* namespaces that will participate in collective operations with local processes. This means that the server must register a namespace even if it will not host any local processes from within that namespace *if* any local process of another namespace might at some point perform an operation involving one or more processes from the new namespace. This is necessary so that the collective operation can identify the participants and know when it is locally complete.

The caller must also provide the number of local processes that will be launched within this namespace. This is required for the PMIx server library to correctly handle collectives as a collective operation call can occur before all the local processes have been started.

Advice to users

The number of local processes for any given namespace is generally fixed at the time of application launch. Calls to **PMIx_Spawn** result in processes launched in their own namespace, not that of their parent. However, it is possible for processes to *migrate* to another node via a call to **PMIx_Job_control_nb**, thus resulting in a change to the number of local processes on both the initial node and the node to which the process moved. It is therefore *critical* that applications not migrate processes without first ensuring that PMIx-based collective operations are not in progress, and that no such operations be initiated until process migration has completed.

10.1.3.1 Assembling the registration information

The following description is not intended to represent the actual layout of information in a given PMIx library. Instead, it is describes how information provided in the *info* parameter of the **PMIx_server_register_nspace** shall be organized for proper processing by a PMIx server library. The ordering of the various information elements is arbitrary - they are presented in a top-down hierarchical form solely for clarity in reading.

Advice to PMIx server hosts

Creating the *info* array of data requires knowing in advance the number of elements required for the array. This can be difficult to compute and somewhat fragile in practice. One method for resolving the problem is to create a linked list of objects, each containing a single **pmix_info_t** structure. Allocation and manipulation of the list can then be accomplished using existing standard methods. Upon completion, the final *info* array can be allocated based on the number of elements on the list, and then the values in the list object **pmix_info_t** structures transferred to the corresponding array element utilizing the **PMIX_INFO_XFER** macro.

A common building block used in several areas is the construction of a regular expression identifying the nodes involved in that area - e.g., the nodes in a **session** or **job**. PMIx provides several tools to facilitate this operation, beginning by constructing an argv-like array of node names. This array is then passed to the **PMIx_generate_regex** function to create a regular expression parseable by the PMIx server library, as shown below:

C

```c
char **nodes = NULL;
char *nodelist;
char *regex;
size_t n;
pmix_status_t rc;
pmix_info_t info;

/* loop over an array of nodes, adding each
 * name to the array */
for (n=0; n < num_nodes; n++)
    /* filter the nodes to ignore those not included
     * in the target range (session, job, etc.). In
     * this example, all nodes are accepted */
    PMIX_ARGV_APPEND(&nodes, node[n]->name);


/* join into a comma-delimited string */
nodelist = PMIX_ARGV_JOIN(nodes, ',');

/* release the array */
PMIX_ARGV_FREE(nodes);

/* generate regex */
rc = PMIx_generate_regex(nodelist, &regex);

/* release list */
free(nodelist);

/* pass the regex as the value to the PMIX_NODE_MAP key */
PMIX_INFO_LOAD(&info, PMIX_NODE_MAP, regex, PMIX_STRING);
/* release the regex */
free(regex);

```

C

Changing the filter criteria allows the construction of node maps for any level of information.

A similar method is used to construct the map of processes on each node from the namespace being registered. This may be done for each information level of interest (e.g., to identify the process map for the entire **job** or for each **application** in the job) by changing the search criteria. An example is shown below for the case of creating the process map for a **job**:

C

```c
char **ndppn;
char rank[30];
char **ppnarray = NULL;
char *ppn;
char *localranks;
char *regex;
size_t n, m;
pmix_status_t rc;
pmix_info_t info;

/* loop over an array of nodes */
for (n=0; n < num_nodes; n++)
    /* for each node, construct an array of ranks on that node */
    ndppn = NULL;
    for (m=0; m < node[n]->num_procs; m++)
        /* ignore processes that are not part of the target job */
        if (!PMIX_CHECK_NSPACE(targetjob,node[n]->proc[m].nspace))
            continue;

        snprintf(rank, 30, "%d", node[n]->proc[m].rank);
        PMIX_ARGV_APPEND(&ndppn, rank);

    /* convert the array into a comma-delimited string of ranks */
    localranks = PMIX_ARGV_JOIN(ndppn, ',');
    /* release the local array */
    PMIX_ARGV_FREE(ndppn);
    /* add this node's contribution to the overall array */
    PMIX_ARGV_APPEND(&ppnarray, localranks);
    /* release the local list */
    free(localranks);


/* join into a semicolon-delimited string */
ppn = PMIX_ARGV_JOIN(ppnarray, ';');

/* release the array */
PMIX_ARGV_FREE(ppnarray);

/* generate ppn regex */
rc = PMIx_generate_ppn(ppn, &regex);

/* release list */
```

```
free(ppn);

/* pass the regex as the value to the PMIX_PROC_MAP key */
PMIX_INFO_LOAD(&info, PMIX_PROC_MAP, regex, PMIX_STRING);
/* release the regex */
free(regex);
```

C

Note that the **PMIX_NODE_MAP** and **PMIX_PROC_MAP** attributes are linked in that the order of entries in the process map must match the ordering of nodes in the node map - i.e., there is no provision in the PMIx process map regular expression generator/parser pair supporting an out-of-order node or a node that has no corresponding process map entry (e.g., a node with no processes on it). Armed with these tools, the registration *info* array can be constructed as follows:

- Session-level information includes all session-specific values. In many cases, only two values (**PMIX_SESSION_ID** and **PMIX_UNIV_SIZE**) are included in the registration array. Since both of these values are session-specific, they can be specified independently - i.e., in their own **pmix_info_t** elements of the *info* array. Alternatively, they can be provided as a **pmix_data_array_t** array of **pmix_info_t** using the **PMIX_SESSION_INFO_ARRAY** attribute and identifed by including the **PMIX_SESSION_ID** attribute in the array - this is *required* in cases where non-specific attributes (e.g., **PMIX_NUM_NODES** or **PMIX_NODE_MAP**) are passed to describe aspects of the session. Note that the node map can include nodes not used by the job being registered as no corresponding process map is specified.

  The *info* array at this point might look like (where the labels identify the corresponding attribute - e.g., "Session ID" corresponds to the **PMIX_SESSION_ID** attribute):

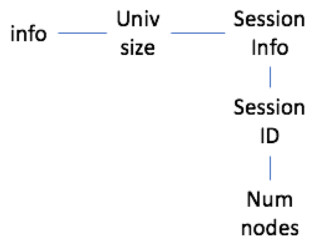

Figure 10.1.: Session-level information elements

- Job-level information includes all job-specific values such as **PMIX_JOB_SIZE** , **PMIX_JOB_NUM_APPS** , and **PMIX_JOBID** . Since each invocation of **PMIx_server_register_nspace** describes a single **job** , job-specific values can be specified independently - i.e., in their own **pmix_info_t** elements of the *info* array. Alternatively, they can be provided as a **pmix_data_array_t** array of **pmix_info_t** identified by the **PMIX_JOB_INFO_ARRAY** attribute - this is *required* in cases where non-specific attributes (e.g., **PMIX_NODE_MAP**) are passed to describe aspects of the job. Note that since the invocation only involves a single namespace, there is no need to include the **PMIX_NSPACE** attribute in the array.

Upon conclusion of this step, the *info* array might look like:

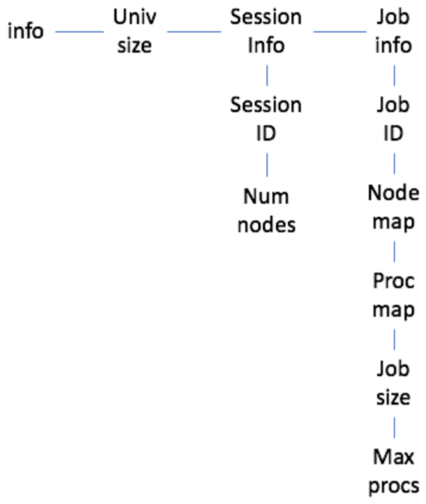

Figure 10.2.: Job-level information elements

Note that in this example, **PMIX_NUM_NODES** is not required as that information is contained in the **PMIX_NODE_MAP** attribute. Similarly, **PMIX_JOB_SIZE** is not technically required as that information is contained in the **PMIX_PROC_MAP** when combined with the corresponding node map - however, there is no issue with including the job size as a separate entry.

- Application-level information includes all application-specific values such as **PMIX_APP_SIZE** and **PMIX_APPLDR** . If the **job** contains only a single **application** , then the application-specific values can be specified independently - i.e., in their own **pmix_info_t** elements of the *info* array - or as a **pmix_data_array_t** array of **pmix_info_t** using the **PMIX_APP_INFO_ARRAY** attribute and identifed by including the **PMIX_APPNUM** attribute in the array. Use of the array format is *required* in cases where non-specific attributes (e.g., **PMIX_NODE_MAP**) are passed to describe aspects of the application.

  However, in the case of a job consisting of multiple applications, all application-specific values for each application *must* be provided using the **PMIX_APP_INFO_ARRAY** format, each identified by its **PMIX_APPNUM** value.

  Upon conclusion of this step, the *info* array might look like that shown in 10.3, assuming there are two applications in the job being registered:

- Process-level information includes an entry for each process in the job being registered, each entry marked with the **PMIX_PROC_DATA** attribute. The **rank** of the process *must* be the first

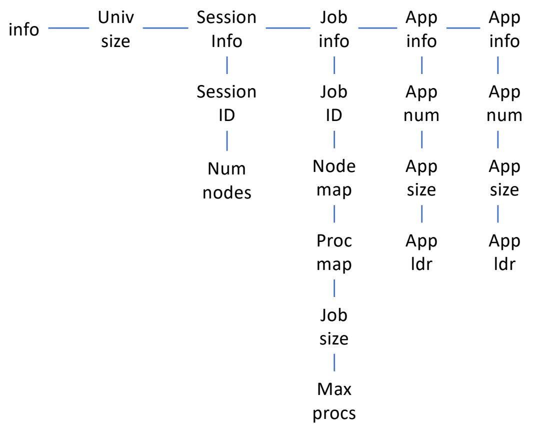

Figure 10.3.: Application-level information elements

entry in the array - this provides efficiency when storing the data. Upon conclusion of this step, the *info* array might look like the diagram in 10.4:

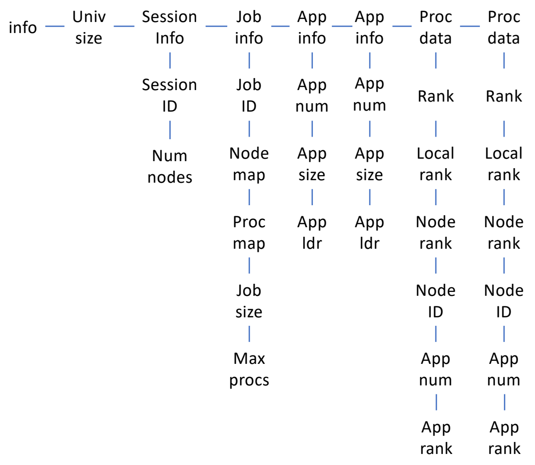

Figure 10.4.: Process-level information elements

- Node-level information only includes values describing the local node - i.e., it does not include information about other nodes in the job or session. In many cases, the values included in this level are unique to it and can be specified independently - i.e., in their own **`pmix_info_t`** elements of the *info* array. Alternatively, they can be provided as a **`pmix_data_array_t`** array of **`pmix_info_t`** using the **`PMIX_NODE_INFO_ARRAY`** attribute - this is *required* in cases where non-specific attributes are passed to describe aspects of the node.

The node-level information requires two elements that must be constructed in a manner similar to that used for the node map. The **PMIX_LOCAL_PEERS** value is computed based on the processes on the local node, filtered to select those from the job being registered, as shown below using the tools provided by PMIx:

C

```c
char **ndppn = NULL;
char rank[30];
char *localranks;
size_t m;
pmix_info_t info;

for (m=0; m < mynode->num_procs; m++)
    /* ignore processes that are not part of the target job */
    if (!PMIX_CHECK_NSPACE(targetjob,mynode->proc[m].nspace))
        continue;

    snprintf(rank, 30, "%d", mynode->proc[m].rank);
    PMIX_ARGV_APPEND(&ndppn, rank);

/* convert the array into a comma-delimited string of ranks */
localranks = PMIX_ARGV_JOIN(ndppn, ',');
/* release the local array */
PMIX_ARGV_FREE(ndppn);

/* pass the string as the value to the PMIX_LOCAL_PEERS key */
PMIX_INFO_LOAD(&info, PMIX_LOCAL_PEERS, localranks, PMIX_STRING);
/* release the list */
free(localranks);
```

C

The **PMIX_LOCAL_CPUSETS** value is constructed in a similar manner. In the provided example, it is assumed that the Hardware Locality (HWLOC) cpuset representation (a comma-delimited string of processor IDs) of the processors assigned to each process has previously been generated and stored on the process description. Thus, the value can be constructed as shown below:

C

```c
char **ndcpus = NULL;
char *localcpus;
size_t m;
pmix_info_t info;

for (m=0; m < mynode->num_procs; m++)
    /* ignore processes that are not part of the target job */
    if (!PMIX_CHECK_NSPACE(targetjob,mynode->proc[m].nspace))
        continue;

    PMIX_ARGV_APPEND(&ndcpus, mynode->proc[m].cpuset);

/* convert the array into a colon-delimited string */
localcpus = PMIX_ARGV_JOIN(ndcpus, ':');
/* release the local array */
PMIX_ARGV_FREE(ndcpus);

/* pass the string as the value to the PMIX_LOCAL_CPUSETS key */
PMIX_INFO_LOAD(&info, PMIX_LOCAL_CPUSETS, localcpus, PMIX_STRING);
/* release the list */
free(localcpus);

```

C

Note that for efficiency, these two values can be computed at the same time.

The final *info* array might therefore look like the diagram in 10.5:

10.1.4 PMIx_server_deregister_nspace

Summary

Deregister a namespace.

Format

PMIx v1.0

C

```c
void PMIx_server_deregister_nspace(const pmix_nspace_t nspace,
                                   pmix_op_cbfunc_t cbfunc, void *cbdata)
```

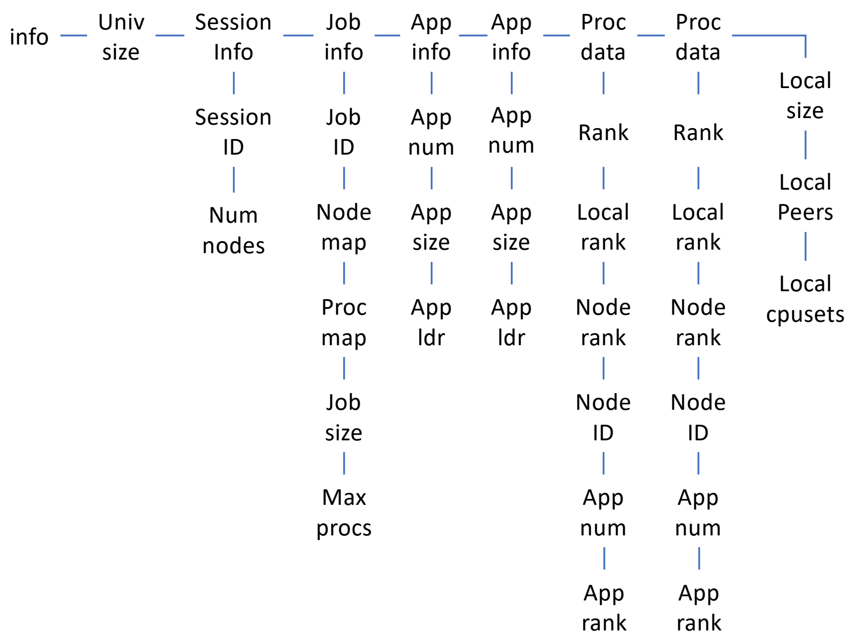

Figure 10.5.: Final information array

C

IN **`nspace`**
Namespace (string)
IN **`cbfunc`**
Callback function **`pmix_op_cbfunc_t`** (function reference)
IN **`cbdata`**
Data to be passed to the callback function (memory reference)

Description

Deregister the specified *nspace* and purge all objects relating to it, including any client information from that namespace. This is intended to support persistent PMIx servers by providing an opportunity for the host RM to tell the PMIx server library to release all memory for a completed job. Note that the library *must not* invoke the callback function prior to returning from the API.

10.1.5 `PMIx_server_register_client`

Summary

Register a client process with the PMIx server library.

Format

PMIx v1.0

C

```
pmix_status_t
PMIx_server_register_client(const pmix_proc_t *proc,
                            uid_t uid, gid_t gid,
                            void *server_object,
                            pmix_op_cbfunc_t cbfunc, void *cbdata)
```

C

IN **proc**
pmix_proc_t structure (handle)
IN **uid**
user id (integer)
IN **gid**
group id (integer)
IN **server_object**
(memory reference)
IN **cbfunc**
Callback function **pmix_op_cbfunc_t** (function reference)
IN **cbdata**
Data to be passed to the callback function (memory reference)

Returns one of the following:

- **PMIX_SUCCESS**, indicating that the request is being processed by the host environment - result will be returned in the provided *cbfunc*. Note that the library *must not* invoke the callback function prior to returning from the API.
- **PMIX_OPERATION_SUCCEEDED**, indicating that the request was immediately processed and returned *success* - the *cbfunc* will *not* be called
- a PMIx error constant indicating either an error in the input or that the request was immediately processed and failed - the *cbfunc* will *not* be called

Description

Register a client process with the PMIx server library.

The host server can also, if it desires, provide an object it wishes to be returned when a server function is called that relates to a specific process. For example, the host server may have an object that tracks the specific client. Passing the object to the library allows the library to provide that object to the host server during subsequent calls related to that client, such as a **pmix_server_client_connected_fn_t** function. This allows the host server to access the object without performing a lookup based on the client's namespace and rank.

Advice to PMIx server hosts

Host environments are *required* to execute this operation prior to starting the client process. The expected user ID and group ID of the child process allows the server library to properly authenticate clients as they connect by requiring the two values to match. Accordingly, the detected user and group ID's of the connecting process are not included in the **`pmix_server_client_connected_fn_t`** server module function.

Advice to PMIx library implementers

For security purposes, the PMIx server library should check the user and group ID's of a connecting process against those provided for the declared client process identifier via the **`PMIx_server_register_client`** prior to completing the connection.

10.1.6 `PMIx_server_deregister_client`

Summary

Deregister a client and purge all data relating to it.

Format

PMIx v1.0

C

```
void
PMIx_server_deregister_client(const pmix_proc_t *proc,
                              pmix_op_cbfunc_t cbfunc, void *cbdata)
```

C

IN **`proc`**
 `pmix_proc_t` structure (handle)

IN **`cbfunc`**
 Callback function **`pmix_op_cbfunc_t`** (function reference)

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

Description

The **`PMIx_server_deregister_nspace`** API will delete all client information for that namespace. The PMIx server library will automatically perform that operation upon disconnect of all local clients. This API is therefore intended primarily for use in exception cases, but can be called in non-exception cases if desired. Note that the library *must not* invoke the callback function prior to returning from the API.

10.1.7 `PMIx_server_setup_fork`

Summary

Setup the environment of a child process to be forked by the host.

Format

PMIx v1.0

C

```
pmix_status_t
PMIx_server_setup_fork(const pmix_proc_t *proc,
                       char ***env)
```

C

IN `proc`
`pmix_proc_t` structure (handle)

IN `env`
Environment array (array of strings)

Returns `PMIX_SUCCESS` or a negative value corresponding to a PMIx error constant.

Description

Setup the environment of a child process to be forked by the host so it can correctly interact with the PMIx server.

Advice to PMIx server hosts

Host environments are *required* to execute this operation prior to starting the client process.

The PMIx client needs some setup information so it can properly connect back to the server. This function will set appropriate environmental variables for this purpose, and will also provide any environmental variables that were specified in the launch command (e.g., via `PMIx_Spawn`) plus other values (e.g., variables required to properly initialize the client's fabric library).

10.1.8 `PMIx_server_dmodex_request`

Summary

Define a function by which the host server can request modex data from the local PMIx server.

Format

PMIx v1.0

C

```c
pmix_status_t PMIx_server_dmodex_request(const pmix_proc_t *proc,
                                 pmix_dmodex_response_fn_t cbfunc,
                                 void *cbdata)
```

C

IN `proc`
    `pmix_proc_t` structure (handle)

IN `cbfunc`
    Callback function `pmix_dmodex_response_fn_t` (function reference)

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

Returns one of the following:

- `PMIX_SUCCESS` , indicating that the request is being processed by the host environment - result will be returned in the provided *cbfunc*. Note that the library *must not* invoke the callback function prior to returning from the API.
- a PMIx error constant indicating an error in the input - the *cbfunc* will *not* be called

Description

Define a function by which the host server can request modex data from the local PMIx server. Traditional wireup procedures revolve around the per-process posting of data (e.g., location and endpoint information) via the `PMIx_Put` and `PMIx_Commit` functions followed by a `PMIx_Fence` barrier that globally exchanges the posted information. However, the barrier operation represents a signficant time impact at large scale.

PMIx supports an alternative wireup method known as *Direct Modex* that replaces the barrier-based exchange of all process-posted information with on-demand fetch of a peer's data. In place of the barrier operation, data posted by each process is cached on the local PMIx server. When a process requests the information posted by a particular peer, it first checks the local cache to see if the data is already available. If not, then the request is passed to the local PMIx server, which subsequently requests that its RM host request the data from the RM daemon on the node where the specified peer process is located. Upon receiving the request, the RM daemon passes the request into its PMIx server library using the `PMIx_server_dmodex_request` function, receiving the response in the provided *cbfunc* once the indicated process has posted its information. The RM daemon then returns the data to the requesting daemon, who subsequently passes the data to its PMIx server library for transfer to the requesting client.

Advice to users

While direct modex allows for faster launch times by eliminating the barrier operation, per-peer retrieval of posted information is less efficient. Optimizations can be implemented - e.g., by returning posted information from all processes on a node upon first request - but in general direct modex remains best suited for sparsely connected applications.

10.1.9 `PMIx_server_setup_application`

Summary

Provide a function by which the resource manager can request application-specific setup data prior to launch of an application.

Format

PMIx v2.0

C
```
pmix_status_t
PMIx_server_setup_application(const pmix_nspace_t nspace,
                              pmix_info_t info[], size_t ninfo,
                              pmix_setup_application_cbfunc_t cbfunc,
                              void *cbdata)
```
C

IN `nspace`
namespace (string)

IN `info`
Array of info structures (array of handles)

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

IN `cbfunc`
Callback function `pmix_setup_application_cbfunc_t` (function reference)

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

Returns one of the following:

- `PMIX_SUCCESS`, indicating that the request is being processed by the host environment - result will be returned in the provided *cbfunc*. Note that the library *must not* invoke the callback function prior to returning from the API.
- a PMIx error constant indicating either an error in the input - the *cbfunc* will *not* be called

Description

Provide a function by which the RM can request application-specific setup data (e.g., environmental variables, fabric configuration and security credentials) from supporting PMIx server library subsystems prior to initiating launch of an application.

Advice to PMIx server hosts

Host environments are *required* to execute this operation prior to launching an application.

This is defined as a non-blocking operation in case contributing subsystems need to perform some potentially time consuming action (e.g., query a remote service) before responding. The returned data must be distributed by the RM and subsequently delivered to the local PMIx server on each node where application processes will execute prior to initiating execution of those processes.

In the callback function, the returned *info* array is owned by the PMIx server library and will be free'd when the provided *cbfunc* is called.

Advice to PMIx library implementers

Support for harvesting of environmental variables and providing of local configuration information by the PMIx implementation is optional.

10.1.10 `PMIx_server_setup_local_support`

Summary

Provide a function by which the local PMIx server can perform any application-specific operations prior to spawning local clients of a given application.

Format

PMIx v2.0

C

```
pmix_status_t
PMIx_server_setup_local_support(const pmix_nspace_t nspace,
                                pmix_info_t info[], size_t ninfo,
                                pmix_op_cbfunc_t cbfunc,
                                void *cbdata);
```

C

IN `nspace`
Namespace (string)

IN `info`
Array of info structures (array of handles)

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

IN `cbfunc`
Callback function `pmix_op_cbfunc_t` (function reference)

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

Returns one of the following:

- `PMIX_SUCCESS` , indicating that the request is being processed by the host environment - result will be returned in the provided *cbfunc*. Note that the library *must not* invoke the callback function prior to returning from the API.
- `PMIX_OPERATION_SUCCEEDED` , indicating that the request was immediately processed and returned *success* - the *cbfunc* will *not* be called
- a PMIx error constant indicating either an error in the input or that the request was immediately processed and failed - the *cbfunc* will *not* be called

Description

Provide a function by which the local PMIx server can perform any application-specific operations prior to spawning local clients of a given application. For example, a network library might need to setup the local driver for "instant on" addressing. The data provided in the *info* array is the data provided to there host RM from the a call to `PMIx_server_setup_application` .

Advice to PMIx server hosts

Host environments are *required* to execute this operation prior to starting any local application processes from the specified namespace.

10.2 Server Function Pointers

PMIx utilizes a "function-shipping" approach to support for implementing the server-side of the protocol. This method allows RMs to implement the server without being burdened with PMIx internal details. When a request is received from the client, the corresponding server function will be called with the information.

Any functions not supported by the RM can be indicated by a **NULL** for the function pointer. Client calls to such functions will return a **PMIX_ERR_NOT_SUPPORTED** status.

The host RM will provide the function pointers in a **pmix_server_module_t** structure passed to **PMIx_server_init** . That module structure and associated function references are defined in this section.

Advice to PMIx server hosts

For performance purposes, the host server is required to return as quickly as possible from all functions. Execution of the function is thus to be done asynchronously so as to allow the PMIx server support library to handle multiple client requests as quickly and scalably as possible.

All data passed to the host server functions is “owned” by the PMIX server support library and *MUST NOT* be free’d. Data returned by the host server via callback function is owned by the host server, which is free to release it upon return from the callback

10.2.1 pmix_server_module_t Module

Summary

List of function pointers that a PMIx server passes to **PMIx_server_init** during startup.

Format

C

```c
typedef struct pmix_server_module_2_0_0_t
    /* v1x interfaces */
    pmix_server_client_connected_fn_t   client_connected;
    pmix_server_client_finalized_fn_t   client_finalized;
    pmix_server_abort_fn_t              abort;
    pmix_server_fencenb_fn_t            fence_nb;
    pmix_server_dmodex_req_fn_t         direct_modex;
    pmix_server_publish_fn_t            publish;
    pmix_server_lookup_fn_t             lookup;
    pmix_server_unpublish_fn_t          unpublish;
    pmix_server_spawn_fn_t              spawn;
    pmix_server_connect_fn_t            connect;
    pmix_server_disconnect_fn_t         disconnect;
    pmix_server_register_events_fn_t    register_events;
    pmix_server_deregister_events_fn_t  deregister_events;
    pmix_server_listener_fn_t           listener;
    /* v2x interfaces */
    pmix_server_notify_event_fn_t       notify_event;
    pmix_server_query_fn_t              query;
    pmix_server_tool_connection_fn_t    tool_connected;
    pmix_server_log_fn_t                log;
    pmix_server_alloc_fn_t              allocate;
    pmix_server_job_control_fn_t        job_control;
    pmix_server_monitor_fn_t            monitor;
 pmix_server_module_t;
```

C

10.2.2 pmix_server_client_connected_fn_t

Summary

Notify the host server that a client connected to this server.

Format

PMIx v1.0

C

```c
typedef pmix_status_t (*pmix_server_client_connected_fn_t)(
                            const pmix_proc_t *proc,
                            void* server_object,
                            pmix_op_cbfunc_t cbfunc,
                            void *cbdata)
```

C

IN `proc`
`pmix_proc_t` structure (handle)

IN `server_object`
object reference (memory reference)

IN `cbfunc`
Callback function `pmix_op_cbfunc_t` (function reference)

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

Returns one of the following:

- `PMIX_SUCCESS`, indicating that the request is being processed by the host environment - result will be returned in the provided *cbfunc*. Note that the host *must not* invoke the callback function prior to returning from the API.
- `PMIX_OPERATION_SUCCEEDED`, indicating that the request was immediately processed and returned *success* - the *cbfunc* will *not* be called
- a PMIx error constant indicating either an error in the input or that the request was immediately processed and failed - the *cbfunc* will *not* be called

Description

Notify the host environment that a client has called `PMIx_Init`. Note that the client will be in a blocked state until the host server executes the callback function, thus allowing the PMIx server support library to release the client. The server_object parameter will be the value of the server_object parameter passed to `PMIx_server_register_client` by the host server when registering the connecting client. If provided, an implementation of `pmix_server_client_connected_fn_t` is only required to call the callback function designated. A host server can choose to not be notified when clients connect by setting `pmix_server_client_connected_fn_t` to **NULL**.

It is possible that only a subset of the clients in a namespace call `PMIx_Init`. The server's `pmix_server_client_connected_fn_t` implementation should not depend on being called once per rank in a namespace or delay calling the callback function until all ranks have connected. However, if a rank makes any PMIx calls, it must first call `PMIx_Init` and therefore the server's `pmix_server_client_connected_fn_t` will be called before any other server functions specific to the rank.

Advice to PMIx server hosts

This operation is an opportunity for a host environment to update the status of the ranks it manages. It is also a convenient and well defined time to perform initialization necessary to support further calls into the server related to that rank.

10.2.3 `pmix_server_client_finalized_fn_t`

Summary

Notify the host environment that a client called **`PMIx_Finalize`** .

Format

PMIx v1.0

C

```c
typedef pmix_status_t (*pmix_server_client_finalized_fn_t)(
                              const pmix_proc_t *proc,
                              void* server_object,
                              pmix_op_cbfunc_t cbfunc,
                              void *cbdata)
```

C

IN `proc`
`pmix_proc_t` structure (handle)

IN `server_object`
object reference (memory reference)

IN `cbfunc`
Callback function `pmix_op_cbfunc_t` (function reference)

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

Returns one of the following:

- **`PMIX_SUCCESS`** , indicating that the request is being processed by the host environment - result will be returned in the provided *cbfunc*. Note that the host *must not* invoke the callback function prior to returning from the API.
- **`PMIX_OPERATION_SUCCEEDED`** , indicating that the request was immediately processed and returned *success* - the *cbfunc* will *not* be called
- a PMIx error constant indicating either an error in the input or that the request was immediately processed and failed - the *cbfunc* will *not* be called

Description

Notify the host environment that a client called **`PMIx_Finalize`** . Note that the client will be in a blocked state until the host server executes the callback function, thus allowing the PMIx server support library to release the client. The server_object parameter will be the value of the server_object parameter passed to **`PMIx_server_register_client`** by the host server when registering the connecting client. If provided, an implementation of `pmix_server_client_finalized_fn_t` is only required to call the callback function designated. A host server can choose to not be notified when clients finalize by setting **pmix_server_client_finalized_fn_t** to **NULL**.

Note that the host server is only being informed that the client has called **PMIx_Finalize** . The client might not have exited. If a client exits without calling **PMIx_Finalize** , the server support library will not call the **pmix_server_client_finalized_fn_t** implementation.

Advice to PMIx server hosts

This operation is an opportunity for a host server to update the status of the tasks it manages. It is also a convenient and well defined time to release resources used to support that client.

10.2.4 pmix_server_abort_fn_t

Summary

Notify the host environment that a local client called **PMIx_Abort** .

Format

PMIx v1.0

C

```
typedef pmix_status_t (*pmix_server_abort_fn_t)(
                                    const pmix_proc_t *proc,
                                    void *server_object,
                                    int status,
                                    const char msg[],
                                    pmix_proc_t procs[],
                                    size_t nprocs,
                                    pmix_op_cbfunc_t cbfunc,
                                    void *cbdata)
```

C

IN **proc**
pmix_proc_t structure identifying the process requesting the abort (handle)

IN **server_object**
object reference (memory reference)

IN **status**
exit status (integer)

IN **msg**
exit status message (string)

IN **procs**
Array of **pmix_proc_t** structures identifying the processes to be terminated (array of handles)

IN **nprocs**
Number of elements in the *procs* array (integer)

IN **cbfunc**
Callback function **pmix_op_cbfunc_t** (function reference)

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

Returns one of the following:

- **PMIX_SUCCESS** , indicating that the request is being processed by the host environment - result will be returned in the provided *cbfunc*. Note that the host *must not* invoke the callback function prior to returning from the API.
- **PMIX_OPERATION_SUCCEEDED** , indicating that the request was immediately processed and returned *success* - the *cbfunc* will *not* be called
- a PMIx error constant indicating either an error in the input or that the request was immediately processed and failed - the *cbfunc* will *not* be called

Description

A local client called **PMIx_Abort** . Note that the client will be in a blocked state until the host server executes the callback function, thus allowing the PMIx server library to release the client. The array of *procs* indicates which processes are to be terminated. A **NULL** indicates that all processes in the client's namespace are to be terminated.

10.2.5 pmix_server_fencenb_fn_t

Summary

At least one client called either **PMIx_Fence** or **PMIx_Fence_nb** .

Format

PMIx v1.0

C

```
typedef pmix_status_t (*pmix_server_fencenb_fn_t)(
                             const pmix_proc_t procs[],
                             size_t nprocs,
                             const pmix_info_t info[],
                             size_t ninfo,
                             char *data, size_t ndata,
                             pmix_modex_cbfunc_t cbfunc,
                             void *cbdata)
```

C

IN **`procs`**
Array of **`pmix_proc_t`** structures identifying operation participants(array of handles)
IN **`nprocs`**
Number of elements in the *procs* array (integer)
IN **`info`**
Array of info structures (array of handles)
IN **`ninfo`**
Number of elements in the *info* array (integer)
IN **`data`**
(string)
IN **`ndata`**
(integer)
IN **`cbfunc`**
Callback function **`pmix_modex_cbfunc_t`** (function reference)
IN **`cbdata`**
Data to be passed to the callback function (memory reference)

Returns one of the following:

- **`PMIX_SUCCESS`** , indicating that the request is being processed by the host environment - result will be returned in the provided *cbfunc*. Note that the host *must not* invoke the callback function prior to returning from the API.
- **`PMIX_OPERATION_SUCCEEDED`** , indicating that the request was immediately processed and returned *success* - the *cbfunc* will *not* be called
- a PMIx error constant indicating either an error in the input or that the request was immediately processed and failed - the *cbfunc* will *not* be called

Required Attributes

PMIx libraries are required to pass any provided attributes to the host environment for processing.

The following attributes are required to be supported by all host environments:

`PMIX_COLLECT_DATA` **`"pmix.collect"`** (**`bool`**)
Collect data and return it at the end of the operation.

Optional Attributes

The following attributes are optional for host environments:

PMIX_TIMEOUT **"pmix.timeout"** (**int**)
Time in seconds before the specified operation should time out (*0* indicating infinite) in error. The timeout parameter can help avoid "hangs" due to programming errors that prevent the target process from ever exposing its data.

PMIX_COLLECTIVE_ALGO **"pmix.calgo"** (**char***)
Comma-delimited list of algorithms to use for the collective operation. PMIx does not impose any requirements on a host environment's collective algorithms. Thus, the acceptable values for this attribute will be environment-dependent - users are encouraged to check their host environment for supported values.

PMIX_COLLECTIVE_ALGO_REQD **"pmix.calreqd"** (**bool**)
If **true**, indicates that the requested choice of algorithm is mandatory.

Advice to PMIx server hosts

Host environment are *required* to return **PMIX_ERR_NOT_SUPPORTED** if passed an attributed marked as **PMIX_INFO_REQD** that they do not support, even if support for that attribute is optional.

Description

All local clients in the provided array of *procs* called either **PMIx_Fence** or **PMIx_Fence_nb**. In either case, the host server will be called via a non-blocking function to execute the specified operation once all participating local processes have contributed. All processes in the specified *procs* array are required to participate in the **PMIx_Fence** / **PMIx_Fence_nb** operation. The callback is to be executed once every daemon hosting at least one participant has called the host server's **pmix_server_fencenb_fn_t** function.

Advice to PMIx library implementers

The PMIx server library is required to aggregate participation by local clients, passing the request to the host environment once all local participants have executed the API.

Advice to PMIx server hosts

The host will receive a single call for each collective operation. It is the responsibility of the host to identify the nodes containing participating processes, execute the collective across all participating nodes, and notify the local PMIx server library upon completion of the global collective.

The provided data is to be collectively shared with all PMIx servers involved in the fence operation, and returned in the modex *cbfunc*. A **NULL** data value indicates that the local processes had no data to contribute.

The array of *info* structs is used to pass user-requested options to the server. This can include directives as to the algorithm to be used to execute the fence operation. The directives are optional *unless* the **PMIX_INFO_REQD** flag has been set - in such cases, the host RM is required to return an error if the directive cannot be met.

10.2.6 pmix_server_dmodex_req_fn_t

Summary

Used by the PMIx server to request its local host contact the PMIx server on the remote node that hosts the specified proc to obtain and return a direct modex blob for that proc.

Format

PMIx v1.0

C

```c
typedef pmix_status_t (*pmix_server_dmodex_req_fn_t)(
                            const pmix_proc_t *proc,
                            const pmix_info_t info[],
                            size_t ninfo,
                            pmix_modex_cbfunc_t cbfunc,
                            void *cbdata)
```

C

IN **proc**
  pmix_proc_t structure identifying the process whose data is being requested (handle)
IN **info**
  Array of info structures (array of handles)
IN **ninfo**
  Number of elements in the *info* array (integer)
IN **cbfunc**
  Callback function **pmix_modex_cbfunc_t** (function reference)
IN **cbdata**
  Data to be passed to the callback function (memory reference)

Returns one of the following:

- **PMIX_SUCCESS**, indicating that the request is being processed by the host environment - result will be returned in the provided *cbfunc*. Note that the host *must not* invoke the callback function prior to returning from the API.

- `PMIX_OPERATION_SUCCEEDED` , indicating that the request was immediately processed and returned *success* - the *cbfunc* will *not* be called
- a PMIx error constant indicating either an error in the input or that the request was immediately processed and failed - the *cbfunc* will *not* be called

Required Attributes

PMIx libraries are required to pass any provided attributes to the host environment for processing.

Optional Attributes

The following attributes are optional for host environments that support this operation:

`PMIX_TIMEOUT` **"pmix.timeout"** (**int**)
  Time in seconds before the specified operation should time out (*0* indicating infinite) in error. The timeout parameter can help avoid "hangs" due to programming errors that prevent the target process from ever exposing its data.

Description

Used by the PMIx server to request its local host contact the PMIx server on the remote node that hosts the specified proc to obtain and return any information that process posted via calls to `PMIx_Put` and `PMIx_Commit` .

The array of *info* structs is used to pass user-requested options to the server. This can include a timeout to preclude an indefinite wait for data that may never become available. The directives are optional *unless* the *mandatory* flag has been set - in such cases, the host RM is required to return an error if the directive cannot be met.

10.2.7 `pmix_server_publish_fn_t`

Summary

Publish data per the PMIx API specification.

Format

PMIx v1.0

C

```
typedef pmix_status_t (*pmix_server_publish_fn_t)(
                                const pmix_proc_t *proc,
                                const pmix_info_t info[],
                                size_t ninfo,
                                pmix_op_cbfunc_t cbfunc,
                                void *cbdata)
```

C

IN **`proc`**
    `pmix_proc_t` structure of the process publishing the data (handle)

IN **`info`**
    Array of info structures (array of handles)

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

IN **`cbfunc`**
    Callback function **`pmix_op_cbfunc_t`** (function reference)

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

Returns one of the following:

- **`PMIX_SUCCESS`**, indicating that the request is being processed by the host environment - result will be returned in the provided *cbfunc*. Note that the host *must not* invoke the callback function prior to returning from the API.
- **`PMIX_OPERATION_SUCCEEDED`**, indicating that the request was immediately processed and returned *success* - the *cbfunc* will *not* be called
- a PMIx error constant indicating either an error in the input or that the request was immediately processed and failed - the *cbfunc* will *not* be called

Required Attributes

PMIx libraries are required to pass any provided attributes to the host environment for processing. In addition, the following attributes are required to be included in the passed *info* array:

`PMIX_USERID` **`"pmix.euid"`** (**`uint32_t`**)
    Effective user id.

`PMIX_GRPID` **`"pmix.egid"`** (**`uint32_t`**)
    Effective group id.

Host environments that implement this entry point are required to support the following attributes:

`PMIX_RANGE` **`"pmix.range"`** (**`pmix_data_range_t`**)
    Value for calls to publish/lookup/unpublish or for monitoring event notifications.

`PMIX_PERSISTENCE` **"pmix.persist"** (**pmix_persistence_t**)
Value for calls to **PMIx_Publish** .

Optional Attributes

The following attributes are optional for host environments that support this operation:

`PMIX_TIMEOUT` **"pmix.timeout"** (**int**)
Time in seconds before the specified operation should time out (*0* indicating infinite) in error. The timeout parameter can help avoid "hangs" due to programming errors that prevent the target process from ever exposing its data.

Description

Publish data per the **PMIx_Publish** specification. The callback is to be executed upon completion of the operation. The default data range is left to the host environment, but expected to be **PMIX_SESSION** , and the default persistence **PMIX_PERSIST_SESSION** or their equivalent. These values can be specified by including the respective attributed in the *info* array.

The persistence indicates how long the server should retain the data.

Advice to PMIx server hosts

The host environment is not required to guarantee support for any specific range - i.e., the environment does not need to return an error if the data store doesn't support a specified range so long as it is covered by some internally defined range. However, the server must return an error (a) if the key is duplicative within the storage range, and (b) if the server does not allow overwriting of published info by the original publisher - it is left to the discretion of the host environment to allow info-key-based flags to modify this behavior.

The **PMIX_USERID** and **PMIX_GRPID** of the publishing process will be provided to support authorization-based access to published information and must be returned on any subsequent lookup request.

10.2.8 `pmix_server_lookup_fn_t`

Summary

Lookup published data.

Format

PMIx v1.0

C

```c
typedef pmix_status_t (*pmix_server_lookup_fn_t)(
                            const pmix_proc_t *proc,
                            char **keys,
                            const pmix_info_t info[],
                            size_t ninfo,
                            pmix_lookup_cbfunc_t cbfunc,
                            void *cbdata)
```

C

IN `proc`
 `pmix_proc_t` structure of the process seeking the data (handle)
IN `keys`
 (array of strings)
IN `info`
 Array of info structures (array of handles)
IN `ninfo`
 Number of elements in the *info* array (integer)
IN `cbfunc`
 Callback function `pmix_lookup_cbfunc_t` (function reference)
IN `cbdata`
 Data to be passed to the callback function (memory reference)

Returns one of the following:

- `PMIX_SUCCESS` , indicating that the request is being processed by the host environment - result will be returned in the provided *cbfunc*. Note that the host *must not* invoke the callback function prior to returning from the API.
- `PMIX_OPERATION_SUCCEEDED` , indicating that the request was immediately processed and returned *success* - the *cbfunc* will *not* be called
- a PMIx error constant indicating either an error in the input or that the request was immediately processed and failed - the *cbfunc* will *not* be called

Required Attributes

PMIx libraries are required to pass any provided attributes to the host environment for processing. In addition, the following attributes are required to be included in the passed *info* array:

`PMIX_USERID` `"pmix.euid"` (`uint32_t`)
 Effective user id.

`PMIX_GRPID` `"pmix.egid"` (`uint32_t`)
 Effective group id.

Host environments that implement this entry point are required to support the following attributes:

PMIX_RANGE **"pmix.range"** (**pmix_data_range_t**)
Value for calls to publish/lookup/unpublish or for monitoring event notifications.

PMIX_WAIT **"pmix.wait"** (**int**)
Caller requests that the PMIx server wait until at least the specified number of values are found (*0* indicates all and is the default).

Optional Attributes

The following attributes are optional for host environments that support this operation:

PMIX_TIMEOUT **"pmix.timeout"** (**int**)
Time in seconds before the specified operation should time out (*0* indicating infinite) in error. The timeout parameter can help avoid "hangs" due to programming errors that prevent the target process from ever exposing its data.

Description

Lookup published data. The host server will be passed a **NULL**-terminated array of string keys identifying the data being requested.

The array of *info* structs is used to pass user-requested options to the server. The default data range is left to the host environment, but expected to be **PMIX_SESSION** . This can include a wait flag to indicate that the server should wait for all data to become available before executing the callback function, or should immediately callback with whatever data is available. In addition, a timeout can be specified on the wait to preclude an indefinite wait for data that may never be published.

Advice to PMIx server hosts

The **PMIX_USERID** and **PMIX_GRPID** of the requesting process will be provided to support authorization-based access to published information. The host environment is not required to guarantee support for any specific range - i.e., the environment does not need to return an error if the data store doesn't support a specified range so long as it is covered by some internally defined range.

10.2.9 pmix_server_unpublish_fn_t

Summary

Delete data from the data store.

Format

PMIx v1.0

C

```c
typedef pmix_status_t (*pmix_server_unpublish_fn_t)(
                              const pmix_proc_t *proc,
                              char **keys,
                              const pmix_info_t info[],
                              size_t ninfo,
                              pmix_op_cbfunc_t cbfunc,
                              void *cbdata)
```

C

IN `proc`
`pmix_proc_t` structure identifying the process making the request (handle)

IN `keys`
(array of strings)

IN `info`
Array of info structures (array of handles)

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

IN `cbfunc`
Callback function `pmix_op_cbfunc_t` (function reference)

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

Returns one of the following:

- `PMIX_SUCCESS`, indicating that the request is being processed by the host environment - result will be returned in the provided *cbfunc*. Note that the host *must not* invoke the callback function prior to returning from the API.
- `PMIX_OPERATION_SUCCEEDED`, indicating that the request was immediately processed and returned *success* - the *cbfunc* will *not* be called
- a PMIx error constant indicating either an error in the input or that the request was immediately processed and failed - the *cbfunc* will *not* be called

Required Attributes

PMIx libraries are required to pass any provided attributes to the host environment for processing. In addition, the following attributes are required to be included in the passed *info* array:

`PMIX_USERID` `"pmix.euid"` (`uint32_t`)
Effective user id.

`PMIX_GRPID` `"pmix.egid"` (`uint32_t`)
Effective group id.

Host environments that implement this entry point are required to support the following attributes:

PMIX_RANGE **"pmix.range"** (**pmix_data_range_t**)
Value for calls to publish/lookup/unpublish or for monitoring event notifications.

Optional Attributes

The following attributes are optional for host environments that support this operation:

PMIX_TIMEOUT **"pmix.timeout"** (**int**)
Time in seconds before the specified operation should time out (*0* indicating infinite) in error. The timeout parameter can help avoid "hangs" due to programming errors that prevent the target process from ever exposing its data.

Description

Delete data from the data store. The host server will be passed a **NULL**-terminated array of string keys, plus potential directives such as the data range within which the keys should be deleted. The default data range is left to the host environment, but expected to be **PMIX_SESSION**. The callback is to be executed upon completion of the delete procedure.

Advice to PMIx server hosts

The **PMIX_USERID** and **PMIX_GRPID** of the requesting process will be provided to support authorization-based access to published information. The host environment is not required to guarantee support for any specific range - i.e., the environment does not need to return an error if the data store doesn't support a specified range so long as it is covered by some internally defined range.

10.2.10 pmix_server_spawn_fn_t

Summary

Spawn a set of applications/processes as per the **PMIx_Spawn** API.

Format

PMIx v1.0

C

```c
typedef pmix_status_t (*pmix_server_spawn_fn_t)(
                            const pmix_proc_t *proc,
                            const pmix_info_t job_info[],
                            size_t ninfo,
                            const pmix_app_t apps[],
                            size_t napps,
                            pmix_spawn_cbfunc_t cbfunc,
                            void *cbdata)
```

C

IN `proc`
  `pmix_proc_t` structure of the process making the request (handle)

IN `job_info`
  Array of info structures (array of handles)

IN `ninfo`
  Number of elements in the *jobinfo* array (integer)

IN `apps`
  Array of `pmix_app_t` structures (array of handles)

IN `napps`
  Number of elements in the *apps* array (integer)

IN `cbfunc`
  Callback function `pmix_spawn_cbfunc_t` (function reference)

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

Returns one of the following:

- `PMIX_SUCCESS` , indicating that the request is being processed by the host environment - result will be returned in the provided *cbfunc*. Note that the host *must not* invoke the callback function prior to returning from the API.
- `PMIX_OPERATION_SUCCEEDED` , indicating that the request was immediately processed and returned *success* - the *cbfunc* will *not* be called
- a PMIx error constant indicating either an error in the input or that the request was immediately processed and failed - the *cbfunc* will *not* be called

Required Attributes

PMIx libraries are required to pass any provided attributes to the host environment for processing. In addition, the following attributes are required to be included in the passed *info* array:

`PMIX_USERID` `"pmix.euid"` (`uint32_t`)
  Effective user id.

PMIX_GRPID "**pmix.egid**" (**uint32_t**)
Effective group id.

Host environments that provide this module entry point are required to pass the **PMIX_SPAWNED** and **PMIX_PARENT_ID** attributes to all PMIx servers launching new child processes so those values can be returned to clients upon connection to the PMIx server. In addition, they are required to support the following attributes when present in either the *job_info* or the *info* array of an element of the *apps* array:

PMIX_WDIR "**pmix.wdir**" (**char***)
Working directory for spawned processes.

PMIX_SET_SESSION_CWD "**pmix.ssncwd**" (**bool**)
Set the application's current working directory to the session working directory assigned by the RM - when accessed using **PMIx_Get** , use the **PMIX_RANK_WILDCARD** value for the rank to discover the session working directory assigned to the provided namespace

PMIX_PREFIX "**pmix.prefix**" (**char***)
Prefix to use for starting spawned processes.

PMIX_HOST "**pmix.host**" (**char***)
Comma-delimited list of hosts to use for spawned processes.

PMIX_HOSTFILE "**pmix.hostfile**" (**char***)
Hostfile to use for spawned processes.

Optional Attributes

The following attributes are optional for host environments that support this operation:

PMIX_ADD_HOSTFILE "**pmix.addhostfile**" (**char***)
Hostfile listing hosts to add to existing allocation.

PMIX_ADD_HOST "**pmix.addhost**" (**char***)
Comma-delimited list of hosts to add to the allocation.

PMIX_PRELOAD_BIN "**pmix.preloadbin**" (**bool**)
Preload binaries onto nodes.

PMIX_PRELOAD_FILES "**pmix.preloadfiles**" (**char***)
Comma-delimited list of files to pre-position on nodes.

PMIX_PERSONALITY "**pmix.pers**" (**char***)
Name of personality to use.

PMIX_MAPPER "**pmix.mapper**" (**char***)
Mapping mechanism to use for placing spawned processes - when accessed using **PMIx_Get** , use the **PMIX_RANK_WILDCARD** value for the rank to discover the mapping mechanism used for the provided namespace.

PMIX_DISPLAY_MAP **"pmix.dispmap"** (**bool**)
Display process mapping upon spawn.

PMIX_PPR **"pmix.ppr"** (**char***)
Number of processes to spawn on each identified resource.

PMIX_MAPBY **"pmix.mapby"** (**char***)
Process mapping policy - when accessed using **PMIx_Get** , use the **PMIX_RANK_WILDCARD** value for the rank to discover the mapping policy used for the provided namespace

PMIX_RANKBY **"pmix.rankby"** (**char***)
Process ranking policy - when accessed using **PMIx_Get** , use the **PMIX_RANK_WILDCARD** value for the rank to discover the ranking algorithm used for the provided namespace

PMIX_BINDTO **"pmix.bindto"** (**char***)
Process binding policy - when accessed using **PMIx_Get** , use the **PMIX_RANK_WILDCARD** value for the rank to discover the binding policy used for the provided namespace

PMIX_NON_PMI **"pmix.nonpmi"** (**bool**)
Spawned processes will not call **PMIx_Init** .

PMIX_STDIN_TGT **"pmix.stdin"** (**uint32_t**)
Spawned process rank that is to receive **stdin**.

PMIX_FWD_STDIN **"pmix.fwd.stdin"** (**bool**)
Forward this process's **stdin** to the designated process.

PMIX_FWD_STDOUT **"pmix.fwd.stdout"** (**bool**)
Forward **stdout** from spawned processes to this process.

PMIX_FWD_STDERR **"pmix.fwd.stderr"** (**bool**)
Forward **stderr** from spawned processes to this process.

PMIX_DEBUGGER_DAEMONS **"pmix.debugger"** (**bool**)
Spawned application consists of debugger daemons.

PMIX_TAG_OUTPUT **"pmix.tagout"** (**bool**)
Tag application output with the identity of the source process.

PMIX_TIMESTAMP_OUTPUT **"pmix.tsout"** (**bool**)
Timestamp output from applications.

PMIX_MERGE_STDERR_STDOUT **"pmix.mergeerrout"** (**bool**)
Merge **stdout** and **stderr** streams from application processes.

PMIX_OUTPUT_TO_FILE **"pmix.outfile"** (**char***)
Output application output to the specified file.

PMIX_INDEX_ARGV **"pmix.indxargv"** (**bool**)
Mark the **argv** with the rank of the process.

PMIX_CPUS_PER_PROC **"pmix.cpuperproc"** (**uint32_t**)
Number of cpus to assign to each rank - when accessed using **PMIx_Get** , use the **PMIX_RANK_WILDCARD** value for the rank to discover the cpus/process assigned to the provided namespace

PMIX_NO_PROCS_ON_HEAD **"pmix.nolocal"** (**bool**)
Do not place processes on the head node.

PMIX_NO_OVERSUBSCRIBE **"pmix.noover"** (**bool**)
Do not oversubscribe the cpus.

PMIX_REPORT_BINDINGS **"pmix.repbind"** (**bool**)
Report bindings of the individual processes.

PMIX_CPU_LIST **"pmix.cpulist"** (**char***)
List of cpus to use for this job - when accessed using **PMIx_Get** , use the **PMIX_RANK_WILDCARD** value for the rank to discover the cpu list used for the provided namespace

PMIX_JOB_RECOVERABLE **"pmix.recover"** (**bool**)
Application supports recoverable operations.

PMIX_JOB_CONTINUOUS **"pmix.continuous"** (**bool**)
Application is continuous, all failed processes should be immediately restarted.

PMIX_MAX_RESTARTS **"pmix.maxrestarts"** (**uint32_t**)
Maximum number of times to restart a job - when accessed using **PMIx_Get** , use the **PMIX_RANK_WILDCARD** value for the rank to discover the max restarts for the provided namespace

PMIX_TIMEOUT **"pmix.timeout"** (**int**)
Time in seconds before the specified operation should time out (*0* indicating infinite) in error. The timeout parameter can help avoid “hangs” due to programming errors that prevent the target process from ever exposing its data.

Description

Spawn a set of applications/processes as per the **PMIx_Spawn** API. Note that applications are not required to be MPI or any other programming model. Thus, the host server cannot make any assumptions as to their required support. The callback function is to be executed once all processes have been started. An error in starting any application or process in this request shall cause all applications and processes in the request to be terminated, and an error returned to the originating caller.

Note that a timeout can be specified in the job_info array to indicate that failure to start the requested job within the given time should result in termination to avoid hangs.

10.2.11 pmix_server_connect_fn_t

Summary

Record the specified processes as *connected*.

Format

PMIx v1.0

C

```c
typedef pmix_status_t (*pmix_server_connect_fn_t)(
                             const pmix_proc_t procs[],
                             size_t nprocs,
                             const pmix_info_t info[],
                             size_t ninfo,
                             pmix_op_cbfunc_t cbfunc,
                             void *cbdata)
```

C

IN **procs**
Array of **pmix_proc_t** structures identifying participants (array of handles)

IN **nprocs**
Number of elements in the *procs* array (integer)

IN **info**
Array of info structures (array of handles)

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

IN **cbfunc**
Callback function **pmix_op_cbfunc_t** (function reference)

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

Returns one of the following:

- **PMIX_SUCCESS** , indicating that the request is being processed by the host environment - result will be returned in the provided *cbfunc*. Note that the host *must not* invoke the callback function prior to returning from the API.
- **PMIX_OPERATION_SUCCEEDED** , indicating that the request was immediately processed and returned *success* - the *cbfunc* will *not* be called
- a PMIx error constant indicating either an error in the input or that the request was immediately processed and failed - the *cbfunc* will *not* be called

Required Attributes

PMIx libraries are required to pass any provided attributes to the host environment for processing.

Optional Attributes

The following attributes are optional for host environments that support this operation:

PMIX_TIMEOUT **"pmix.timeout"** (**int**)
Time in seconds before the specified operation should time out (*0* indicating infinite) in error. The timeout parameter can help avoid "hangs" due to programming errors that prevent the target process from ever exposing its data.

PMIX_COLLECTIVE_ALGO **"pmix.calgo"** (**char***)
Comma-delimited list of algorithms to use for the collective operation. PMIx does not impose any requirements on a host environment's collective algorithms. Thus, the acceptable values for this attribute will be environment-dependent - users are encouraged to check their host environment for supported values.

PMIX_COLLECTIVE_ALGO_REQD **"pmix.calreqd"** (**bool**)
If **true**, indicates that the requested choice of algorithm is mandatory.

Description

Record the processes specified by the *procs* array as *connected* as per the PMIx definition. The callback is to be executed once every daemon hosting at least one participant has called the host server's **`pmix_server_connect_fn_t`** function, *and* the host environment has completed any supporting operations required to meet the terms of the PMIx definition of *connected* processes.

Advice to PMIx library implementers

The PMIx server library is required to aggregate participation by local clients, passing the request to the host environment once all local participants have executed the API.

Advice to PMIx server hosts

The host will receive a single call for each collective operation. It is the responsibility of the host to identify the nodes containing participating processes, execute the collective across all participating nodes, and notify the local PMIx server library upon completion of the global collective.

10.2.12 `pmix_server_disconnect_fn_t`

Summary

Disconnect a previously connected set of processes.

Format

PMIx v1.0

C

```c
typedef pmix_status_t (*pmix_server_disconnect_fn_t)(
                              const pmix_proc_t procs[],
                              size_t nprocs,
                              const pmix_info_t info[],
                              size_t ninfo,
                              pmix_op_cbfunc_t cbfunc,
                              void *cbdata)
```

C

IN `procs`
Array of `pmix_proc_t` structures identifying participants (array of handles)

IN `nprocs`
Number of elements in the *procs* array (integer)

IN `info`
Array of info structures (array of handles)

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

IN `cbfunc`
Callback function `pmix_op_cbfunc_t` (function reference)

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

Returns one of the following:

- `PMIX_SUCCESS` , indicating that the request is being processed by the host environment - result will be returned in the provided *cbfunc*. Note that the host *must not* invoke the callback function prior to returning from the API.
- `PMIX_OPERATION_SUCCEEDED` , indicating that the request was immediately processed and returned *success* - the *cbfunc* will *not* be called
- a PMIx error constant indicating either an error in the input or that the request was immediately processed and failed - the *cbfunc* will *not* be called

Required Attributes

PMIx libraries are required to pass any provided attributes to the host environment for processing.

Optional Attributes

The following attributes are optional for host environments that support this operation:

PMIX_TIMEOUT **"pmix.timeout"** (**int**)
Time in seconds before the specified operation should time out (*0* indicating infinite) in error. The timeout parameter can help avoid "hangs" due to programming errors that prevent the target process from ever exposing its data.

Description

Disconnect a previously connected set of processes. The callback is to be executed once every daemon hosting at least one participant has called the host server's has called the **pmix_server_disconnect_fn_t** function, *and* the host environment has completed any required supporting operations.

Advice to PMIx library implementers

The PMIx server library is required to aggregate participation by local clients, passing the request to the host environment once all local participants have executed the API.

Advice to PMIx server hosts

The host will receive a single call for each collective operation. It is the responsibility of the host to identify the nodes containing participating processes, execute the collective across all participating nodes, and notify the local PMIx server library upon completion of the global collective.

A **PMIX_ERR_INVALID_OPERATION** error must be returned if the specified set of *procs* was not previously *connected* via a call to the **pmix_server_connect_fn_t** function.

10.2.13 pmix_server_register_events_fn_t

Summary

Register to receive notifications for the specified events.

Format

PMIx v1.0

C

```
typedef pmix_status_t (*pmix_server_register_events_fn_t)(
                             pmix_status_t *codes,
                             size_t ncodes,
                             const pmix_info_t info[],
                             size_t ninfo,
                             pmix_op_cbfunc_t cbfunc,
                             void *cbdata)
```

C

IN `codes`
Array of `pmix_status_t` values (array of handles)

IN `ncodes`
Number of elements in the *codes* array (integer)

IN `info`
Array of info structures (array of handles)

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

IN `cbfunc`
Callback function `pmix_op_cbfunc_t` (function reference)

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

Returns one of the following:

- `PMIX_SUCCESS` , indicating that the request is being processed by the host environment - result will be returned in the provided *cbfunc*. Note that the host *must not* invoke the callback function prior to returning from the API.
- `PMIX_OPERATION_SUCCEEDED` , indicating that the request was immediately processed and returned *success* - the *cbfunc* will *not* be called
- a PMIx error constant indicating either an error in the input or that the request was immediately processed and failed - the *cbfunc* will *not* be called

Required Attributes

PMIx libraries are required to pass any provided attributes to the host environment for processing. In addition, the following attributes are required to be included in the passed *info* array:

`PMIX_USERID` `"pmix.euid"` (`uint32_t`)
Effective user id.

`PMIX_GRPID` `"pmix.egid"` (`uint32_t`)
Effective group id.

Description

Register to receive notifications for the specified status codes. The *info* array included in this API is reserved for possible future directives to further steer notification.

Advice to PMIx library implementers

The PMIx server library must track all client registrations for subsequent notification. This module function shall only be called when:

- the client has requested notification of an environmental code (i.e., a PMIx code in the range between **PMIX_ERR_SYS_BASE** and **PMIX_ERR_SYS_OTHER** , inclusive) or a code that lies outside the defined PMIx range of constants; and
- the PMIx server library has not previously requested notification of that code - i.e., the host environment is to be contacted only once a given unique code value

Advice to PMIx server hosts

The host environment is *required* to pass to its PMIx server library all non-environmental events that directly relate to a registered namespace without the PMIx server library explicitly requesting them. Environmental events are to be translated to their nearest PMIx equivalent code as defined in the range between **PMIX_ERR_SYS_BASE** and **PMIX_ERR_SYS_OTHER** (inclusive).

10.2.14 `pmix_server_deregister_events_fn_t`

Summary

Deregister to receive notifications for the specified events.

Format

PMIx v1.0

C

```
typedef pmix_status_t (*pmix_server_deregister_events_fn_t)(
                                    pmix_status_t *codes,
                                    size_t ncodes,
                                    pmix_op_cbfunc_t cbfunc,
                                    void *cbdata)
```

C

IN `codes`
Array of `pmix_status_t` values (array of handles)

IN `ncodes`
Number of elements in the *codes* array (integer)

IN `cbfunc`
Callback function `pmix_op_cbfunc_t` (function reference)

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

Returns one of the following:

- `PMIX_SUCCESS`, indicating that the request is being processed by the host environment - result will be returned in the provided *cbfunc*. Note that the host *must not* invoke the callback function prior to returning from the API.
- `PMIX_OPERATION_SUCCEEDED`, indicating that the request was immediately processed and returned *success* - the *cbfunc* will *not* be called
- a PMIx error constant indicating either an error in the input or that the request was immediately processed and failed - the *cbfunc* will *not* be called

Description

Deregister to receive notifications for the specified events to which the PMIx server has previously registered.

Advice to PMIx library implementers

The PMIx server library must track all client registrations. This module function shall only be called when:

- the library is deregistering environmental codes (i.e., a PMIx codes in the range between `PMIX_ERR_SYS_BASE` and `PMIX_ERR_SYS_OTHER`, inclusive) or codes that lies outside the defined PMIx range of constants; and
- no client (including the server library itself) remains registered for notifications on any included code - i.e., a code should be included in this call only when no registered notifications against it remain.

10.2.15 pmix_server_notify_event_fn_t

Summary

Notify the specified processes of an event.

Format

PMIx v2.0

C

```c
typedef pmix_status_t (*pmix_server_notify_event_fn_t)(pmix_status_t code,
                                                       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 `code`
The **`pmix_status_t`** event code being referenced structure (handle)

IN `source`
`pmix_proc_t` of process that generated the event (handle)

IN `range`
`pmix_data_range_t` range over which the event is to be distributed (handle)

IN `info`
Optional array of **`pmix_info_t`** structures containing additional information on the event (array of handles)

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

IN `cbfunc`
Callback function **`pmix_op_cbfunc_t`** (function reference)

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

Returns one of the following:

- **`PMIX_SUCCESS`**, indicating that the request is being processed by the host environment - result will be returned in the provided *cbfunc*. Note that the host *must not* invoke the callback function prior to returning from the API.
- **`PMIX_OPERATION_SUCCEEDED`**, indicating that the request was immediately processed and returned *success* - the *cbfunc* will *not* be called

- a PMIx error constant indicating either an error in the input or that the request was immediately processed and failed - the *cbfunc* will *not* be called

Required Attributes

PMIx libraries are required to pass any provided attributes to the host environment for processing.

Host environments that provide this module entry point are required to support the following attributes:

`PMIX_RANGE` **`"pmix.range"`** (**`pmix_data_range_t`**)
    Value for calls to publish/lookup/unpublish or for monitoring event notifications.

Description

Notify the specified processes (described through a combination of *range* and attributes provided in the *info* array) of an event generated either by the PMIx server itself or by one of its local clients. The process generating the event is provided in the *source* parameter, and any further descriptive information is included in the *info* array.

Advice to PMIx server hosts

The callback function is to be executed once the host environment no longer requires that the PMIx server library maintain the provided data structures. It does *not* necessarily indicate that the event has been delivered to any process, nor that the event has been distributed for delivery

10.2.16 `pmix_server_listener_fn_t`

Summary

Register a socket the host server can monitor for connection requests.

Format

PMIx v1.0

C

```c
typedef pmix_status_t (*pmix_server_listener_fn_t)(
                              int listening_sd,
                              pmix_connection_cbfunc_t cbfunc,
                              void *cbdata)
```

C

IN `incoming_sd`
(integer)

IN `cbfunc`
Callback function `pmix_connection_cbfunc_t` (function reference)

IN `cbdata`
(memory reference)

Returns `PMIX_SUCCESS` indicating that the request is accepted, or a negative value corresponding to a PMIx error constant indicating that the request has been rejected.

Description

Register a socket the host environment can monitor for connection requests, harvest them, and then call the PMIx server library's internal callback function for further processing. A listener thread is essential to efficiently harvesting connection requests from large numbers of local clients such as occur when running on large SMPs. The host server listener is required to call accept on the incoming connection request, and then pass the resulting socket to the provided cbfunc. A `NULL` for this function will cause the internal PMIx server to spawn its own listener thread.

10.2.17 `pmix_server_query_fn_t`

Summary

Query information from the resource manager.

Format

PMIx v2.0

C

```c
typedef pmix_status_t (*pmix_server_query_fn_t)(
                              pmix_proc_t *proct,
                              pmix_query_t *queries, size_t nqueries,
                              pmix_info_cbfunc_t cbfunc,
                              void *cbdata)
```

C

IN `proct`
`pmix_proc_t` structure of the requesting process (handle)

IN `queries`
Array of `pmix_query_t` structures (array of handles)

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

IN `cbfunc`
Callback function `pmix_info_cbfunc_t` (function reference)

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

Returns one of the following:

- `PMIX_SUCCESS` , indicating that the request is being processed by the host environment - result will be returned in the provided *cbfunc*. Note that the host *must not* invoke the callback function prior to returning from the API.
- `PMIX_OPERATION_SUCCEEDED` , indicating that the request was immediately processed and returned *success* - the *cbfunc* will *not* be called
- a PMIx error constant indicating either an error in the input or that the request was immediately processed and failed - the *cbfunc* will *not* be called

Required Attributes

PMIx libraries are required to pass any provided attributes to the host environment for processing. In addition, the following attributes are required to be included in the passed *info* array:

`PMIX_USERID` `"pmix.euid"` (`uint32_t`)
Effective user id.

`PMIX_GRPID` `"pmix.egid"` (`uint32_t`)
Effective group id.

Optional Attributes

The following attributes are optional for host environments that support this operation:

`PMIX_QUERY_NAMESPACES` `"pmix.qry.ns"` (`char*`)
Request a comma-delimited list of active namespaces.

`PMIX_QUERY_JOB_STATUS` `"pmix.qry.jst"` (`pmix_status_t`)
Status of a specified, currently executing job.

`PMIX_QUERY_QUEUE_LIST` `"pmix.qry.qlst"` (`char*`)
Request a comma-delimited list of scheduler queues.

`PMIX_QUERY_QUEUE_STATUS` `"pmix.qry.qst"` (`TBD`)

Status of a specified scheduler queue.

PMIX_QUERY_PROC_TABLE **"pmix.qry.ptable"** (**char***)
Input namespace of the job whose information is being requested returns (**pmix_data_array_t**) an array of **pmix_proc_info_t** .

PMIX_QUERY_LOCAL_PROC_TABLE **"pmix.qry.lptable"** (**char***)
Input namespace of the job whose information is being requested returns (**pmix_data_array_t**) an array of **pmix_proc_info_t** for processes in job on same node.

PMIX_QUERY_SPAWN_SUPPORT **"pmix.qry.spawn"** (**bool**)
Return a comma-delimited list of supported spawn attributes.

PMIX_QUERY_DEBUG_SUPPORT **"pmix.qry.debug"** (**bool**)
Return a comma-delimited list of supported debug attributes.

PMIX_QUERY_MEMORY_USAGE **"pmix.qry.mem"** (**bool**)
Return information on memory usage for the processes indicated in the qualifiers.

PMIX_QUERY_LOCAL_ONLY **"pmix.qry.local"** (**bool**)
Constrain the query to local information only.

PMIX_QUERY_REPORT_AVG **"pmix.qry.avg"** (**bool**)
Report average values.

PMIX_QUERY_REPORT_MINMAX **"pmix.qry.minmax"** (**bool**)
Report minimum and maximum values.

PMIX_QUERY_ALLOC_STATUS **"pmix.query.alloc"** (**char***)
String identifier of the allocation whose status is being requested.

PMIX_TIME_REMAINING **"pmix.time.remaining"** (**char***)
Query number of seconds (**uint32_t**) remaining in allocation for the specified namespace.

Description

Query information from the host environment. The query will include the namespace/rank of the process that is requesting the info, an array of **pmix_query_t** describing the request, and a callback function/data for the return.

Advice to PMIx library implementers

The PMIx server library should not block in this function as the host environment may, depending upon the information being requested, require significant time to respond.

10.2.18 `pmix_server_tool_connection_fn_t`

Summary

Register that a tool has connected to the server.

Format

PMIx v2.0

C

```c
typedef void (*pmix_server_tool_connection_fn_t)(
                                    pmix_info_t info[], size_t ninfo,
                                    pmix_tool_connection_cbfunc_t cbfunc,
                                    void *cbdata)
```

C

IN `info`
Array of `pmix_info_t` structures (array of handles)

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

IN `cbfunc`
Callback function `pmix_tool_connection_cbfunc_t` (function reference)

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

Required Attributes

PMIx libraries are required to pass the following attributes in the *info* array:

`PMIX_USERID` `"pmix.euid"` (`uint32_t`)
Effective user id.

`PMIX_GRPID` `"pmix.egid"` (`uint32_t`)
Effective group id.

Optional Attributes

The following attributes are optional for host environments that support this operation:

`PMIX_FWD_STDOUT` `"pmix.fwd.stdout"` (`bool`)
Forward `stdout` from spawned processes to this process.

`PMIX_FWD_STDERR` `"pmix.fwd.stderr"` (`bool`)
Forward `stderr` from spawned processes to this process.

`PMIX_FWD_STDIN` `"pmix.fwd.stdin"` (`bool`)
Forward this process's `stdin` to the designated process.

Description

Register that a tool has connected to the server, and request that the tool be assigned a namespace/rank identifier for further interactions. The **pmix_info_t** array is used to pass qualifiers for the connection request, including the effective uid and gid of the calling tool for authentication purposes.

Advice to PMIx server hosts

The host environment is solely responsible for authenticating and authorizing the connection, and for authorizing all subsequent tool requests. The host *must not* execute the callback function prior to returning from the API.

10.2.19 pmix_server_log_fn_t

Summary

Log data on behalf of a client.

Format

PMIx v2.0

C

```
typedef void (*pmix_server_log_fn_t)(
                            const pmix_proc_t *client,
                            const pmix_info_t data[], size_t ndata,
                            const pmix_info_t directives[], size_t ndirs,
                            pmix_op_cbfunc_t cbfunc, void *cbdata)
```

C

IN **client**
  pmix_proc_t structure (handle)
IN **data**
  Array of info structures (array of handles)
IN **ndata**
  Number of elements in the *data* array (integer)
IN **directives**
  Array of info structures (array of handles)
IN **ndirs**
  Number of elements in the *directives* array (integer)
IN **cbfunc**
  Callback function **pmix_op_cbfunc_t** (function reference)
IN **cbdata**
  Data to be passed to the callback function (memory reference)

Required Attributes

PMIx libraries are required to pass any provided attributes to the host environment for processing. In addition, the following attributes are required to be included in the passed *info* array:

PMIX_USERID **"pmix.euid"** (**uint32_t**)
Effective user id.

PMIX_GRPID **"pmix.egid"** (**uint32_t**)
Effective group id.

Host environments that provide this module entry point are required to support the following attributes:

PMIX_LOG_STDERR **"pmix.log.stderr"** (**char***)
Log string to **stderr**.

PMIX_LOG_STDOUT **"pmix.log.stdout"** (**char***)
Log string to **stdout**.

PMIX_LOG_SYSLOG **"pmix.log.syslog"** (**char***)
Log data to syslog. Defaults to **ERROR** priority.

Optional Attributes

The following attributes are optional for host environments that support this operation:

PMIX_LOG_MSG **"pmix.log.msg"** (**pmix_byte_object_t**)
Message blob to be sent somewhere.

PMIX_LOG_EMAIL **"pmix.log.email"** (**pmix_data_array_t**)
Log via email based on **pmix_info_t** containing directives.

PMIX_LOG_EMAIL_ADDR **"pmix.log.emaddr"** (**char***)
Comma-delimited list of email addresses that are to receive the message.

PMIX_LOG_EMAIL_SUBJECT **"pmix.log.emsub"** (**char***)
Subject line for email.

PMIX_LOG_EMAIL_MSG **"pmix.log.emmsg"** (**char***)
Message to be included in email.

Description

Log data on behalf of a client. This function is *not* intended for output of computational results, but rather for reporting status and error messages. The host *must not* execute the callback function prior to returning from the API.

10.2.20 `pmix_server_alloc_fn_t`

Summary

Request allocation operations on behalf of a client.

Format

PMIx v2.0

C

```
typedef pmix_status_t (*pmix_server_alloc_fn_t)(
                                  const pmix_proc_t *client,
                                  pmix_alloc_directive_t directive,
                                  const pmix_info_t data[], size_t ndata,
                                  pmix_info_cbfunc_t cbfunc, void *cbdata)
```

C

IN `client`
  `pmix_proc_t` structure of process making request (handle)

IN `directive`
  Specific action being requested (`pmix_alloc_directive_t`)

IN `data`
  Array of info structures (array of handles)

IN `ndata`
  Number of elements in the *data* array (integer)

IN `cbfunc`
  Callback function `pmix_info_cbfunc_t` (function reference)

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

Returns one of the following:

- `PMIX_SUCCESS` , indicating that the request is being processed by the host environment - result will be returned in the provided *cbfunc*. Note that the host *must not* invoke the callback function prior to returning from the API.
- `PMIX_OPERATION_SUCCEEDED` , indicating that the request was immediately processed and returned *success* - the *cbfunc* will *not* be called
- a PMIx error constant indicating either an error in the input or that the request was immediately processed and failed - the *cbfunc* will *not* be called

Required Attributes

PMIx libraries are required to pass any provided attributes to the host environment for processing. In addition, the following attributes are required to be included in the passed *info* array:

PMIX_USERID **"pmix.euid"** (**uint32_t**)
    Effective user id.

PMIX_GRPID **"pmix.egid"** (**uint32_t**)
    Effective group id.

Host environments that provide this module entry point are required to support the following attributes:

PMIX_ALLOC_ID **"pmix.alloc.id"** (**char***)
    Provide a string identifier for this allocation request which can later be used to query status of the request.

PMIX_ALLOC_NUM_NODES **"pmix.alloc.nnodes"** (**uint64_t**)
    The number of nodes.

PMIX_ALLOC_NUM_CPUS **"pmix.alloc.ncpus"** (**uint64_t**)
    Number of cpus.

PMIX_ALLOC_TIME **"pmix.alloc.time"** (**uint32_t**)
    Time in seconds.

Optional Attributes

The following attributes are optional for host environments that support this operation:

PMIX_ALLOC_NODE_LIST **"pmix.alloc.nlist"** (**char***)
    Regular expression of the specific nodes.

PMIX_ALLOC_NUM_CPU_LIST **"pmix.alloc.ncpulist"** (**char***)
    Regular expression of the number of cpus for each node.

PMIX_ALLOC_CPU_LIST **"pmix.alloc.cpulist"** (**char***)
    Regular expression of the specific cpus indicating the cpus involved.

PMIX_ALLOC_MEM_SIZE **"pmix.alloc.msize"** (**float**)
    Number of Megabytes.

PMIX_ALLOC_NETWORK **"pmix.alloc.net"** (**array**)
    Array of **pmix_info_t** describing requested network resources. If not given as part of an **pmix_info_t** struct that identifies the involved nodes, then the description will be applied across all nodes in the requestor's allocation.

PMIX_ALLOC_NETWORK_ID **"pmix.alloc.netid"** (**char***)
    Name of the network.

PMIX_ALLOC_BANDWIDTH **"pmix.alloc.bw" (float)**
Mbits/sec.

PMIX_ALLOC_NETWORK_QOS **"pmix.alloc.netqos" (char*)**
Quality of service level.

Description

Request new allocation or modifications to an existing allocation on behalf of a client. Several broad categories are envisioned, including the ability to:

- Request allocation of additional resources, including memory, bandwidth, and compute for an existing allocation. Any additional allocated resources will be considered as part of the current allocation, and thus will be released at the same time.
- Request a new allocation of resources. Note that the new allocation will be disjoint from (i.e., not affiliated with) the allocation of the requestor - thus the termination of one allocation will not impact the other.
- Extend the reservation on currently allocated resources, subject to scheduling availability and priorities.
- Return no-longer-required resources to the scheduler. This includes the *loan* of resources back to the scheduler with a promise to return them upon subsequent request.

The callback function provides a *status* to indicate whether or not the request was granted, and to provide some information as to the reason for any denial in the **pmix_info_cbfunc_t** array of **pmix_info_t** structures.

10.2.21 pmix_server_job_control_fn_t

Summary

Execute a job control action on behalf of a client.

Format

PMIx v2.0

C

```
typedef pmix_status_t (*pmix_server_job_control_fn_t)(
                            const pmix_proc_t *requestor,
                            const pmix_proc_t targets[], size_t ntargets,
                            const pmix_info_t directives[], size_t ndirs,
                            pmix_info_cbfunc_t cbfunc, void *cbdata)
```

C

IN `requestor`
  `pmix_proc_t` structure of requesting process (handle)

IN `targets`
  Array of proc structures (array of handles)

IN `ntargets`
  Number of elements in the *targets* array (integer)

IN `directives`
  Array of info structures (array of handles)

IN `ndirs`
  Number of elements in the *info* array (integer)

IN `cbfunc`
  Callback function `pmix_op_cbfunc_t` (function reference)

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

Returns one of the following:

- `PMIX_SUCCESS` , indicating that the request is being processed by the host environment - result will be returned in the provided *cbfunc*. Note that the host *must not* invoke the callback function prior to returning from the API.
- `PMIX_OPERATION_SUCCEEDED` , indicating that the request was immediately processed and returned *success* - the *cbfunc* will *not* be called
- a PMIx error constant indicating either an error in the input or that the request was immediately processed and failed - the *cbfunc* will *not* be called

Required Attributes

PMIx libraries are required to pass any provided attributes to the host environment for processing. In addition, the following attributes are required to be included in the passed *info* array:

`PMIX_USERID` **"pmix.euid"** (**uint32_t**)
  Effective user id.

`PMIX_GRPID` **"pmix.egid"** (**uint32_t**)
  Effective group id.

Host environments that provide this module entry point are required to support the following attributes:

`PMIX_JOB_CTRL_ID` **"pmix.jctrl.id"** (**char***)
  Provide a string identifier for this request.

`PMIX_JOB_CTRL_PAUSE` **"pmix.jctrl.pause"** (**bool**)
  Pause the specified processes.

`PMIX_JOB_CTRL_RESUME` **"pmix.jctrl.resume"** (**bool**)

Resume ("un-pause") the specified processes.

PMIX_JOB_CTRL_KILL **"pmix.jctrl.kill"** (**bool**)
Forcibly terminate the specified processes and cleanup.

PMIX_JOB_CTRL_SIGNAL **"pmix.jctrl.sig"** (**int**)
Send given signal to specified processes.

PMIX_JOB_CTRL_TERMINATE **"pmix.jctrl.term"** (**bool**)
Politely terminate the specified processes.

Optional Attributes

The following attributes are optional for host environments that support this operation:

PMIX_JOB_CTRL_CANCEL **"pmix.jctrl.cancel"** (**char***)
Cancel the specified request (**NULL** implies cancel all requests from this requestor).

PMIX_JOB_CTRL_RESTART **"pmix.jctrl.restart"** (**char***)
Restart the specified processes using the given checkpoint ID.

PMIX_JOB_CTRL_CHECKPOINT **"pmix.jctrl.ckpt"** (**char***)
Checkpoint the specified processes and assign the given ID to it.

PMIX_JOB_CTRL_CHECKPOINT_EVENT **"pmix.jctrl.ckptev"** (**bool**)
Use event notification to trigger a process checkpoint.

PMIX_JOB_CTRL_CHECKPOINT_SIGNAL **"pmix.jctrl.ckptsig"** (**int**)
Use the given signal to trigger a process checkpoint.

PMIX_JOB_CTRL_CHECKPOINT_TIMEOUT **"pmix.jctrl.ckptsig"** (**int**)
Time in seconds to wait for a checkpoint to complete.

PMIX_JOB_CTRL_CHECKPOINT_METHOD
"pmix.jctrl.ckmethod" (**pmix_data_array_t**)
Array of **pmix_info_t** declaring each method and value supported by this application.

PMIX_JOB_CTRL_PROVISION **"pmix.jctrl.pvn"** (**char***)
Regular expression identifying nodes that are to be provisioned.

PMIX_JOB_CTRL_PROVISION_IMAGE **"pmix.jctrl.pvnimg"** (**char***)
Name of the image that is to be provisioned.

PMIX_JOB_CTRL_PREEMPTIBLE **"pmix.jctrl.preempt"** (**bool**)
Indicate that the job can be pre-empted.

Description

Execute a job control action on behalf of a client. The *targets* array identifies the processes to which the requested job control action is to be applied. A **NULL** value can be used to indicate all processes in the caller's namespace. The use of **PMIX_RANK_WILDARD** can also be used to indicate that all processes in the given namespace are to be included.

The directives are provided as **pmix_info_t** structures in the *directives* array. The callback function provides a *status* to indicate whether or not the request was granted, and to provide some information as to the reason for any denial in the **pmix_info_cbfunc_t** array of **pmix_info_t** structures.

10.2.22 pmix_server_monitor_fn_t

Summary

Request that a client be monitored for activity.

Format

PMIx v2.0

C

```
/* Request that a client be monitored for activity */
typedef pmix_status_t (*pmix_server_monitor_fn_t)(
                              const pmix_proc_t *requestor,
                              const pmix_info_t *monitor, pmix_status_t error
                              const pmix_info_t directives[], size_t ndirs,
                              pmix_info_cbfunc_t cbfunc, void *cbdata);
```

C

IN **requestor**
pmix_proc_t structure of requesting process (handle)
IN **monitor**
pmix_info_t identifying the type of monitor being requested (handle)
IN **error**
Status code to use in generating event if alarm triggers (integer)
IN **directives**
Array of info structures (array of handles)
IN **ndirs**
Number of elements in the *info* array (integer)
IN **cbfunc**
Callback function **pmix_op_cbfunc_t** (function reference)
IN **cbdata**
Data to be passed to the callback function (memory reference)

Returns one of the following:

- **PMIX_SUCCESS** , indicating that the request is being processed by the host environment - result will be returned in the provided *cbfunc*. Note that the host *must not* invoke the callback function prior to returning from the API.
- **PMIX_OPERATION_SUCCEEDED** , indicating that the request was immediately processed and returned *success* - the *cbfunc* will *not* be called
- a PMIx error constant indicating either an error in the input or that the request was immediately processed and failed - the *cbfunc* will *not* be called

This entry point is only called for monitoring requests that are not directly supported by the PMIx server library itself.

Required Attributes

If supported by the PMIx server library, then the library must not pass any supported attributes to the host environment. All attributes not directly supported by the server library must be passed to the host environment if it provides this module entry. In addition, the following attributes are required to be included in the passed *info* array:

PMIX_USERID **"pmix.euid"** (**uint32_t**)
Effective user id.

PMIX_GRPID **"pmix.egid"** (**uint32_t**)
Effective group id.

Host environments are not required to support any specific monitoring attributes.

Optional Attributes

The following attributes may be implemented by a host environment.

PMIX_MONITOR_ID **"pmix.monitor.id"** (**char***)
Provide a string identifier for this request.

PMIX_MONITOR_CANCEL **"pmix.monitor.cancel"** (**char***)
Identifier to be canceled (**NULL** means cancel all monitoring for this process).

PMIX_MONITOR_APP_CONTROL **"pmix.monitor.appctrl"** (**bool**)
The application desires to control the response to a monitoring event.

PMIX_MONITOR_HEARTBEAT **"pmix.monitor.mbeat"** (**void**)
Register to have the PMIx server monitor the requestor for heartbeats.

PMIX_MONITOR_HEARTBEAT_TIME **"pmix.monitor.btime"** (**uint32_t**)
Time in seconds before declaring heartbeat missed.

PMIX_MONITOR_HEARTBEAT_DROPS **"pmix.monitor.bdrop"** (**uint32_t**)

Number of heartbeats that can be missed before generating the event.

PMIX_MONITOR_FILE **"pmix.monitor.fmon"** (**char***)
Register to monitor file for signs of life.

PMIX_MONITOR_FILE_SIZE **"pmix.monitor.fsize"** (**bool**)
Monitor size of given file is growing to determine if the application is running.

PMIX_MONITOR_FILE_ACCESS **"pmix.monitor.faccess"** (**char***)
Monitor time since last access of given file to determine if the application is running.

PMIX_MONITOR_FILE_MODIFY **"pmix.monitor.fmod"** (**char***)
Monitor time since last modified of given file to determine if the application is running.

PMIX_MONITOR_FILE_CHECK_TIME **"pmix.monitor.ftime"** (**uint32_t**)
Time in seconds between checking the file.

PMIX_MONITOR_FILE_DROPS **"pmix.monitor.fdrop"** (**uint32_t**)
Number of file checks that can be missed before generating the event.

Description

Request that a client be monitored for activity.

Advice to PMIx server hosts

If this module entry is provided and called by the PMIx server library, then the host environment must either provide the requested services or return **PMIX_ERR_NOT_SUPPORTED** to the provided *cbfunc*.

APPENDIX A

Acknowledgements

This document represents the work of many people who have contributed to the PMIx community. Without the hard work and dedication of these people this document would not have been possible. The sections below list some of the active participants and organizations in the various PMIx standard iterations.

A.1 Version 2.0

The following list includes some of the active participants in the PMIx v2 standardization process.

- Ralph H. Castain, Annapurna Dasari, Christopher A. Holguin, Andrew Friedley, Michael Klemm and Terry Wilmarth
- Joshua Hursey, David Solt, Alexander Eichenberger, Geoff Paulsen, and Sameh Sharkawi
- Aurelien Bouteiller and George Bosilca
- Artem Polyakov, Igor Ivanov and Boris Karasev
- Gilles Gouaillardet
- Michael A Raymond and Jim Stoffel
- Dirk Schubert
- Moe Jette
- Takahiro Kawashima and Shinji Sumimoto
- Howard Pritchard
- David Beer
- Brice Goglin
- Geoffroy Vallee, Swen Boehm, Thomas Naughton and David Bernholdt
- Adam Moody and Martin Schulz
- Ryan Grant and Stephen Olivier
- Michael Karo

The following institutions supported this effort through time and travel support for the people listed above.

- Intel Corporation
- IBM, Inc.
- University of Tennessee, Knoxville
- The Exascale Computing Project, an initiative of the US Department of Energy
- National Science Foundation
- Mellanox, Inc.
- Research Organization for Information Science and Technology
- HPE Co.
- Allinea (ARM)
- SchedMD, Inc.
- Fujitsu Limited
- Los Alamos National Laboratory
- Adaptive Solutions, Inc.
- INRIA
- Oak Ridge National Laboratory
- Lawrence Livermore National Laboratory
- Sandia National Laboratory
- Altair

A.2 Version 1.0

The following list includes some of the active participants in the PMIx v1 standardization process.

- Ralph H. Castain, Annapurna Dasari and Christopher A. Holguin
- Joshua Hursey and David Solt
- Aurelien Bouteiller and George Bosilca
- Artem Polyakov, Elena Shipunova, Igor Ivanov, and Joshua Ladd
- Gilles Gouaillardet
- Gary Brown

- Moe Jette

The following institutions supported this effort through time and travel support for the people listed above.

- Intel Corporation
- IBM, Inc.
- University of Tennessee, Knoxville
- Mellanox, Inc.
- Research Organization for Information Science and Technology
- Adaptive Solutions, Inc.
- SchedMD, Inc.

Bibliography

[1] Ralph H. Castain, David Solt, Joshua Hursey, and Aurelien Bouteiller. PMIx: Process management for exascale environments. In *Proceedings of the 24th European MPI Users' Group Meeting*, EuroMPI '17, pages 14:1–14:10, New York, NY, USA, 2017. ACM.

Index