PMIx 10^{18}

Process Management Interface for Exascale (PMIx) Standard

Version 4.0 (Draft)

Created on June 22, 2020

This document describes the Process Management Interface for Exascale (PMIx) Standard, version 4.0 (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.

This page intentionally left blank

Contents

CHAPTER 1
Introduction

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

7 • the physical scale of the machines, and the corresponding number of total processes they support,
8   is expected to reach levels approaching 1 million processes executing across 100 thousand nodes.
9   Prior methods for initiating applications relied on exchanging communication endpoint
10   information between the processes, either directly or in some form of hierarchical collective
11   operation. Regardless of the specific mechanism employed, the exchange across such large
12   applications would consume considerable time, with estimates running in excess of 5-10
13   minutes; and

14 • whether it be hybrid applications that combine OpenMP threading operations with MPI, or
15   application-steered workflow computations, the HPC community is experiencing an
16   unprecedented wave of new approaches for computing at exascale levels. One common thread
17   across the proposed methods is an increasing need for orchestration between the application and
18   the system management software stack (SMS) comprising the scheduler (a.k.a. the workload
19   manager (WLM)), the resource manager (RM), global file system, fabric, and other subsystems.
20   The lack of available support for application-to-SMS integration has forced researchers to
21   develop "virtual" environments that hide the SMS behind a customized abstraction layer, but this
22   results in considerable duplication of effort and a lack of portability.

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

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

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

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

1.1   Charter

10 The charter of the PMIx community is to:

11 • Define a set of agnostic APIs (not affiliated with any specific programming model or code base)
12   to support interactions between application processes and the SMS.

13 • Develop an open source (non-copy-left licensed) standalone "reference" library implementation
14   to facilitate adoption of the PMIx standard.

15 • Retain transparent backward compatibility with the existing PMI-1 and PMI-2 definitions, any
16   future PMI releases, and across all PMIx versions.

17 • Support the "Instant On" initiative for rapid startup of applications at exascale and beyond.

18 • Work with the HPC community to define and implement new APIs that support evolving
19   programming model requirements for application interactions with the SMS.

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

1.2   PMIx Standard Overview

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

1.2.1   Who should use the standard?

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

30 PMIx clients (e.g., tools, Message Passing Environment (MPE) libraries) can use this standard to
31 understand the set of attributes provided by various APIs of the PRI and their intended behavior.

1   Additional information about the rationale for the selection of specific interfaces and attributes is
2   also provided.

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

1.2.2   What is defined in the standard?

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

1.2.3   What is *not* defined in the standard?

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

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

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

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

32   It is therefore critical that users and application implementers:

33   a) consider whether or not a given attribute is required, marking it accordingly; and
34   b) check the return status on all PMIx function calls to ensure support was present and that the
35   request was accepted. Note that for non-blocking APIs, a return of **PMIX_SUCCESS** only
36   indicates that the request had no obvious errors and is being processed – the eventual callback
37   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.
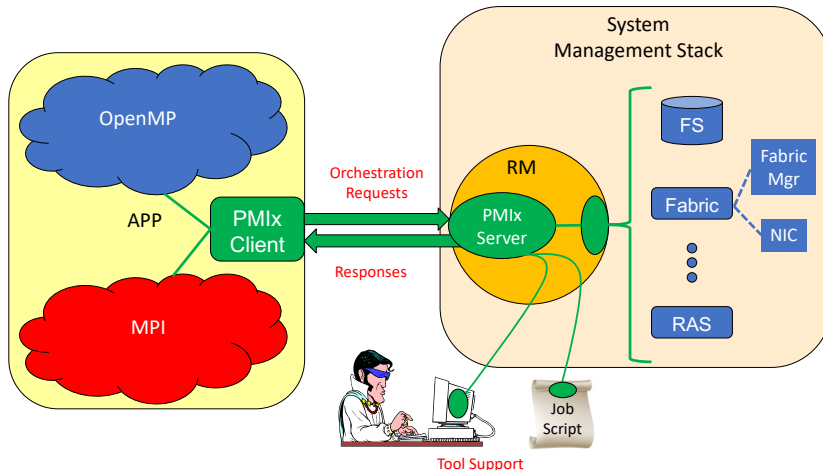
Figure 1.1.: PMIx-SMS Interactions

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/

1.3.3 PMIx Roles

The role of a PMIx process in the PMIx universe is grouped into three categories based on how it operates in the PMIx environment namely as a *client*, *server*, or *tool*. As a result, there are three corresponding sets of initialization and finalization functions. If a process initializes as either a *server* or a *tool* that process may also access all of the *client* APIs.

A process operating as a *client* is started (directly or indirectly, for example, by an intermediate script) by the RM and is connected to the PMIx server instance within that RM when the client calls the client PMIx initialization routine. A process operating as a *server* is responsible for starting client processes and coordinating with other server and tool processes in the same PMIx universe. Often processes operating as a *server* are part of the resource manager (RM) infrastructure. A process operating as a *tool* will connect to a PMIx *server* to interact with the processes in the PMIx universe. An example of a *tool* process is a parallel debugger that will connect to the server to assist with attaching to a set of client processes.

PMIx serves as a conduit between processes acting in these three different roles. As such, an API is often described in how it interacts with processes operating in other roles in the PMIx universe.

───────── Advice to PMIx library implementers ─────────

A PMIx implementation may support all or a subset of the API role groupings defined in the standard. A common nomenclature is defined here to aid in identifying levels of conformance of an implementation.

1 A PMIx implementation that supports all three sets of the API role groupings is said to be *fully*
2 *PMIx standard compliant*. These *fully PMIx standard compliant* implementations have the
3 advantage of being able to support a broad set of PMIx consumers in the different roles.

4 Alternatively, a PMIx implementation may choose to support fewer than all three sets of the API
5 role groupings. PMIx implementations that support only the *client* APIs are said to be *client-only*
6 *PMIx standard compliant*. Similarly, an implementation that only supports the *client* and *tool* APIs
7 are said to be *client-and-tool-only PMIx standard compliant*. Finally, an implementation that only
8 supports the *client* and *server* APIs are said to be *client-and-server-only PMIx standard compliant*.
9 Note that it would not make sense for an implementation to exclude the *client* interfaces from their
10 implementation since they are also used by the *server* and *tool* roles.

1.4 Organization of this document

12 The remainder of this document is structured as follows:

13 • Introduction and Overview in Chapter 1 on page 1

14 • Terms and Conventions in Chapter 2 on page 15

15 • Data Structures and Types in Chapter 14 on page 287

16 • PMIx Initialization and Finalization in Chapter 4 on page 23

17 • Key/Value Management in Chapter 5 on page 33

18 • Process Management in Chapter 6 on page 62

19 • Job Management in Chapter 7 on page 88

20 • Event Notification in Chapter 8 on page 124

21 • Data Packing and Unpacking in Chapter 9 on page 134

22 • Security in Chapter 10 on page 144

23 • PMIx Server Specific Interfaces in Chapter 11 on page 153

24 • Scheduler-Specific Interface in Chapter **??** on page **??**

25 • Process Sets and Groups in Chapter 13 on page 259

26 • Network Coordinates in Chapter **??** on page **??**

27 • Python Bindings in Appendix A on page 382

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 from the v2.0 document, 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

1   • Clarify definitions of **PMIX_UNIV_SIZE** versus **PMIX_JOB_SIZE**

2   • Clarify server module function return values

3   • Provide examples of the use of **PMIx_Get** for retrieval of information

4   • Clarify the use of **PMIx_Get** versus **PMIx_Query_info_nb**

5   • Clarify return values for non-blocking APIs and emphasize that callback functions must not be
6   invoked prior to return from the API

7   • Provide detailed example for construction of the **PMIx_server_register_nspace** input
8   information array

9   • Define information levels (e.g., **session** vs **job**) and associated attributes for both storing
10   and retrieving values

11   • Clarify roles of PMIx server library and host environment for collective operations

12   • Clarify definition of **PMIX_UNIV_SIZE**

1.8 Version 2.2: Jan 2019

14   The v2.2 update includes the following clarifications and corrections from the v2.1 document:

15   • Direct modex upcall function (**pmix_server_dmodex_req_fn_t**) cannot complete
16   atomically as the API cannot return the requested information except via the provided callback
17   function

18   • Add missing **pmix_data_array_t** definition and support macros

19   • Add a rule divider between implementer and host environment required attributes for clarity

20   • Add **PMIX_QUERY_QUALIFIERS_CREATE** macro to simplify creation of **pmix_query_t**
21   qualifiers

22   • Add **PMIX_APP_INFO_CREATE** macro to simplify creation of **pmix_app_t** directives

23   • Add flag and **PMIX_INFO_IS_END** macro for marking and detecting the end of a
24   **pmix_info_t** array

25   • Clarify the allowed hierarchical nesting of the **PMIX_SESSION_INFO_ARRAY** ,
26   **PMIX_JOB_INFO_ARRAY** , and associated attributes

1.9 Version 3.0: Dec. 2018

28   The following APIs were introduced in v3.0 of the PMIx Standard:

29   • Client APIs

30   – **PMIx_Log** , **PMIx_Job_control**

1   – **PMIx_Allocation_request** , **PMIx_Process_monitor**

2   – **PMIx_Get_credential** , **PMIx_Validate_credential**

3   • Server APIs

4   – **PMIx_server_IOF_deliver**

5   – **PMIx_server_collect_inventory** , **PMIx_server_deliver_inventory**

6   • Tool APIs

7   – **PMIx_IOF_pull** , **PMIx_IOF_push** , **PMIx_IOF_deregister**

8   – **PMIx_tool_connect_to_server**

9   • Common APIs

10   – **PMIx_IOF_channel_string**

11 The document added a chapter on security credentials, a new section for Input/Output (IO)
12 forwarding to the Process Management chapter, and a few blocking forms of previously-existing
13 non-blocking APIs. Attributes supporting the new APIs were introduced, as well as additional
14 attributes for a few existing functions.

15   1.10   Version 3.1: Jan. 2019

16 The v3.1 update includes clarifications and corrections from the v3.0 document:

17   • Direct modex upcall function (**pmix_server_dmodex_req_fn_t**) cannot complete
18   atomically as the API cannot return the requested information except via the provided callback
19   function

20   • Fix typo in name of **PMIX_FWD_STDDIAG** attribute

21   • Correctly identify the information retrieval and storage attributes as "new" to v3 of the standard

22   • Add missing **pmix_data_array_t** definition and support macros

23   • Add a rule divider between implementer and host environment required attributes for clarity

24   • Add **PMIX_QUERY_QUALIFIERS_CREATE** macro to simplify creation of **pmix_query_t**
25   qualifiers

26   • Add **PMIX_APP_INFO_CREATE** macro to simplify creation of **pmix_app_t** directives

27   • Add new attributes to specify the level of information being requested where ambiguity may exist
28   (see 14.4.11)

29   • Add new attributes to assemble information by its level for storage where ambiguity may exist
30   (see 14.4.12)

- Add flag and **PMIX_INFO_IS_END** macro for marking and detecting the end of a **pmix_info_t** array

- Clarify that **PMIX_NUM_SLOTS** is duplicative of (a) **PMIX_UNIV_SIZE** when used at the **session** level and (b) **PMIX_MAX_PROCS** when used at the **job** and **application** levels, but leave it in for backward compatibility.

- Clarify difference between **PMIX_JOB_SIZE** and **PMIX_MAX_PROCS**

- Clarify that **PMIx_server_setup_application** must be called per-**job** instead of per-**application** as the name implies. Unfortunately, this is a historical artifact. Note that both **PMIX_NODE_MAP** and **PMIX_PROC_MAP** must be included as input in the *info* array provided to that function. Further descriptive explanation of the "instant on" procedure will be provided in the next version of the PMIx Standard.

- Clarify how the PMIx server expects data passed to the host by **pmix_server_fencenb_fn_t** should be aggregated across nodes, and provide a code snippet example

1.11 Version 3.2: Oct. 2019

The v3.2 update includes clarifications and corrections from the v3.1 document:

- Correct an error in the **PMIx_Allocation_request** function signature, and clarify the allocation ID attributes

- Rename the **PMIX_ALLOC_ID** attribute to **PMIX_ALLOC_REQ_ID** to clarify that this is a string the user provides as a means to identify their request to query status

- Add a new **PMIX_ALLOC_ID** attribute that contains the identifier (provided by the host environment) for the resulting allocation which can later be used to reference the allocated resources in, for example, a call to **PMIx_Spawn**

1.12 Version 4.0: June 2019

The following changes were introduced in v4.0 of the PMIx Standard:

- Clarified that the **PMIx_Fence_nb** operation can immediately return **PMIX_OPERATION_SUCCEEDED** in lieu of passing the request to a PMIx server if only the calling process is involved in the operation

- Added the **PMIx_Register_attributes** API by which a host environment can register the attributes it supports for each server-to-host operation

- Added the ability to query supported attributes from the PMIx tool, client and server libraries, as well as the host environment via the new **pmix_regattr_t** structure. Both human-readable and machine-parsable output is supported. New attributes to support this operation include:

- – **PMIX_CLIENT_ATTRIBUTES** , **PMIX_SERVER_ATTRIBUTES** , **PMIX_TOOL_ATTRIBUTES** , and **PMIX_HOST_ATTRIBUTES** to identify which library supports the attribute; and

- – **PMIX_MAX_VALUE** , **PMIX_MIN_VALUE** , and **PMIX_ENUM_VALUE** to provide machine-parsable description of accepted values

- Add **PMIX_APP_WILDCARD** to reference all applications within a given job

- Fix signature of blocking APIs **PMIx_Allocation_request** , **PMIx_Job_control** , **PMIx_Process_monitor** , **PMIx_Get_credential** , and **PMIx_Validate_credential** to allow return of results

- Update description to provide an option for blocking behavior of the **PMIx_Register_event_handler** , **PMIx_Deregister_event_handler** , **PMIx_Notify_event** , **PMIx_IOF_pull** , **PMIx_IOF_deregister** , and **PMIx_IOF_push** APIs. The need for blocking forms of these functions was not initially anticipated but has emerged over time. For these functions, the return value is sufficient to provide the caller with information otherwise returned via callback. Thus, use of a **NULL** value as the callback function parameter was deemed a minimal disruption method for providing the desired capability

CHAPTER 2

PMIx Terms and Conventions

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

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

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

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

◄———————————— Advice to users ————————————►

25 It is critical that users and application developers consider whether or not a given attribute is
26 required (marking it accordingly) and always check the return status on all PMIx function calls to
27 ensure support was present and that the request was accepted. Note that for non-blocking APIs, a
28 return of **PMIX_SUCCESS** only indicates that the request had no obvious errors and is being
29 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.
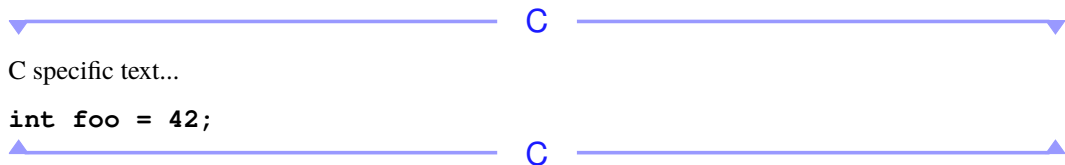
- *scheduler* refers to the component of the SMS responsible for scheduling of resource allocations. This is also generally referred to as the *system workflow manager* - for the purposes of this document, the *WLM* acronym will be used interchangeably to refer to the scheduler.

- *resource manager* is used in a generic sense to represent the subsystem 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.

- *fabric* is used in a generic sense to refer to the networks within the system regardless of speed or protocol. Any use of the term *network* in the document should be considered interchangeable with *fabric*.

- *fabric plane* refers to a collection of devices (Network Interface Cards (NICs)) and switches in a common logical or physical configuration. Fabric planes are often implemented in HPC clusters as separate overlay or physical networks controlled by a dedicated fabric manager.

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.

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

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

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

2.2 Semantics

12 The following terms will be taken to mean:

13 • *shall*, *must* and *will* indicate that the specified behavior is *required* of all conforming
14   implementations

15 • *should* and *may* indicate behaviors that a complete implementation would include, but are not
16   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.

Many PMIx interfaces, particularly nonblocking interfaces, use a **void*** cbdata object passed to the function that is then passed to the associated callback. In a client-side API, the cbdata is a client-provided context (opaque object) that the client can pass to the nonblocking call (e.g., **PMIx_Get_nb**). When the nonblocking call (e.g., **pmix_value_cbfunc_t**) completes, the cbdata is passed back to the client without modification by the PMIx library, thus allowing the client to associate a context with that callback. This is useful if there are many outstanding nonblocking calls.

A similar model is used for the server module functions (see 11.3.1). In this case, the PMIx library is making an upcall into its host via the PMIx server module function and passing a specific cbfunc

1   and cbdata. The PMIx library expects the host to call the cbfunc with the necessary arguments and
2   pass back the original cbdata upon completing the operation. This gives the server-side PMIx
3   library the ability to associate a context with the call back (since multiple operations may be
4   outstanding). The host has no visibility into the contents of the cbdata object, nor is permitted to
5   alter it in any way.

2.5   Standard vs Reference Implementation

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

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

General Information Interfaces

1 The APIs defined in this chapter can be used by any PMIx process, regardless of their role in the
2 PMIx universe.

3.1 Initialization Status

4 The APIs defined in this section return information about the status of the PMIx library.

3.1.1 **PMIx_Initialized**

6 **Format**

PMIx v1.0 ▼─────────────────────── C ───────────────────────▼

7 **int PMIx_Initialized(void)**

▲─────────────────────── C ───────────────────────▲

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

▼ - Rationale - ▼

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

▲ - ▲

10 **Description**
11 Check to see if the PMIx library has been initialized using any of the initialization functions:
12 **PMIx_Init** , **PMIx_server_init** , or **PMIx_tool_init** . It is valid to call this API
13 outside of a region of initialization.

3.2 Library Information

15 The APIs defined in this section return information about the PMIx library.

3.2.1 **PMIx_Get_version**

17 **Summary**
18 Get the PMIx version information.

1   **Format**

PMIx v1.0

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

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

Client-Specific Interfaces

1   The APIs defined in this chapter are dedicated to PMIx consumers in the *client* role.

2 4.1 Client Initialization and Finalization

3   The PMIx APIs may only be used between the completion of the initialization function and the start
4   of the finalization function, unless otherwise noted. The initialization and finalization functions are
5   paired, and the initialized regions defined by them must not overlap.

6 4.1.1 **PMIx_Init**

7   **Summary**
8   Initialize the PMIx client library

9   **Format**

PMIx v1.2 ▼ ─────────────────── C ─────────────────── ▼

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

▲ ─────────────────── C ─────────────────── ▲

13   **INOUT proc**
14   **pmix_proc_t** structure (handle)
15   **IN   info**
16   Array of **pmix_info_t** structures (array of handles)
17   **IN   ninfo**
18   Number of element in the *info* array (**size_t**)

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

1   The following attributes are optional for implementers of PMIx libraries:

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

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

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

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

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

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

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

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

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

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

36 **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 global data storage (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 associated 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.

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 detected.

◆──────────────── 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 of **PMIx_Init** is not included in this document to avoid confusion.

▲──▲

4.1.2 **PMIx_Finalize**

Summary
Finalize the PMIx client library.

─────────────────────────────────

[1]http://libevent.org/

1 **Format**

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

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

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

▼----------------- Optional Attributes -----------------▼

9 The following attributes are optional for implementers of PMIx libraries:

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

14 **Description**
15 Decrement the PMIx client library reference count. When the reference count reaches zero, the
16 library will finalize the PMIx client, closing the connection with the local PMIx server and
17 releasing internally allocated resources.

18 4.2 Tool Initialization and Finalization

19 The APIs defined in this chapter are dedicated to PMIx consumers in the *client* role.

20 **NOTE: THIS SECTION WILL MOVE TO THE NEW TOOLS CHAPTER WHEN**
21 **MERGED**

22 The PMIx APIs may only be used between the completion of the initialization function and the start
23 of the finalization function, unless otherwise noted. The initialization and finalization functions are
24 paired, and the initialized regions defined by them must not overlap.

———————————————— Advice to users ————————————————

25 Tool initialization automatically searches for a server to which it can connect. If the tool is declared
26 as a *launcher* (via **PMIX_LAUNCHER**), the PMIx library sets up the required "hooks" for other
27 tools (e.g., debuggers) to attach to it.

4.2.1 `PMIx_tool_init`

Summary

Initialize the PMIx library for operating as a tool.

Format

PMIx v2.0

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

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.
▲--▲

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**)
> process identifier (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 initialization 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, by calling the **PMIx_tool_connect_to_server** function. In all other cases, the PMIx tool library will attempt to connect to a PMIx server 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 is not 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/

1   • if **PMIX_CONNECT_TO_SYSTEM** is given, then the tool will search for a system-level
2   rendezvous file created by a PMIx server in the directory specified by the
3   **PMIX_SYSTEM_TMPDIR** environmental variable. If found, then the tool will attempt to
4   connect to it. An error is returned if the rendezvous file cannot be found or the connection is
5   refused.

6   • if **PMIX_CONNECT_SYSTEM_FIRST** is given, then the tool will search for a system-level
7   rendezvous file created by a PMIx server in the directory specified by the
8   **PMIX_SYSTEM_TMPDIR** environmental variable. If found, then the tool will attempt to
9   connect to it. In this case, no error will be returned if the rendezvous file is not found or
10   connection is refused — the PMIx library will silently continue to the next option

11   • lastly and by default, the tool will search the directory tree under the directory provided by the
12   **PMIX_SERVER_TMPDIR** environmental variable for rendezvous files of PMIx servers,
13   attempting to connect to each it finds until one accepts the connection. If no rendezvous files are
14   found, or all contacted servers refuse connection, then the PMIx library will return an error.

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

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

4.2.2 **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.

1 **4.2.3 `PMIx_tool_connect_to_server`**

2 **Summary**

3 Switch connection from the current PMIx server to another one, or initialize a connection to a
4 specified server.

5 **Format**

C

```
6    pmix_status_t
7    PMIx_tool_connect_to_server(pmix_proc_t *proc,
8                                pmix_info_t info[], size_t ninfo)
```
                                          C

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

▼------------------   Required Attributes   ------------------▼

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

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

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

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

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

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

▲---▲

Description

A tool may call **PMIx_tool_init** with the **PMIX_TOOL_DO_NOT_CONNECT** attribute in which case they can use this function to connect to a specific server. Additionally, a tool may use this function to switch connection from the current PMIx server to another one Closes the connection, if existing, to a server and establishes a connection to the specified server. This function can be called at any time by a PMIx tool to shift connections between servers. The process identifier assigned to this tool is returned in the provided **pmix_proc_t** struct. Passing a value of **NULL** for this parameter is allowed if the user wishes solely to connect to the PMIx server and does not require return of the identifier at that time.

——————————————— Advice to PMIx library implementers ———————————————

PMIx tools and clients are prohibited from being connected to more than one server at a time to avoid confusion in subsystems such as event notification.

When a tool connects to a server that is under a different namespace manager (e.g., host RM) as the prior server, the identifier of the tool must remain unique in the namespaces. This may require the identifier of the tool to be changed on-the-fly, that is, the *proc* parameter would be filled (if non-NULL) with a different nspace/rank from the current tool identifier.

———————————————————————— Advice to users ————————————————————————

Passing a **NULL** value for the *info* pointer is not allowed and will result in returning an error.

Some PMIx implementations may not support connecting to a server that is not under the same namespace manager (e.g., host RM) as the tool.

Key/Value Management

1 Management of key-value pairs in PMIx is a distributed responsibility. While the stated objective of
2 the PMIx community is to eliminate collective operations, it is recognized that the traditional
3 method of posting/exchanging data must be supported until that objective can be met. This method
4 relies on processes to discover and post their local information which is collected by the local PMIx
5 server library. Global exchange of the posted information is then executed via a collective operation
6 performed by the host SMS servers. The **PMIx_Put** and **PMIx_Commit** APIs, plus an attribute
7 directing **PMIx_Fence** to globally collect the data posted by processes, are provided for this
8 purpose.

5.1 Setting and Accessing Key/Value Pairs

5.1.1 `PMIx_Put`

11 **Summary**
12 Push a key/value pair into the client's namespace.

13 **Format**

PMIx v1.0 ▼ ─────────── C ─────────── ▼

```
14   pmix_status_t
15   PMIx_Put(pmix_scope_t scope,
16            const pmix_key_t key,
17            pmix_value_t *val)
```

▲ ─────────── C ─────────── ▲

18 **IN**   **scope**
19   Distribution scope of the provided value (handle)
20 **IN**   **key**
21   key (**pmix_key_t**)
22 **IN**   **value**
23   Reference to a **pmix_value_t** structure (handle)

24 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 14.2.9 on page 301. 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

─────────────────────────── 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.

PMIX_GET_STATIC_VALUES **"pmix.get.static"** (**bool**)
Request that any pointers in the returned value point directly to values in the key-value store and indicate that the address provided for the return value points to a statically defined memory location. Returned non-pointer values should therefore be copied directly into the provided memory. Pointers in the returned value should point directly to values in the key-value store. User is responsible for *not* releasing memory on any returned pointer value. Note that a return status of **PMIX_ERR_GET_MALLOC_REQD** indicates that direct pointers could not be supported - thus, the returned data contains allocated memory that the user must release.

▲--▲

▼----------------- 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.

▲--▲

1 We recommend that implementation of the **PMIX_TIMEOUT** attribute be left to the host
2 environment due to race condition considerations between delivery of the data by the host
3 environment versus internal timeout in the PMIx server library. Implementers that choose to
4 support **PMIX_TIMEOUT** directly in the PMIx server library must take care to resolve the race
5 condition and should avoid passing **PMIX_TIMEOUT** to the host environment so that multiple
6 competing timeouts are not created.

Description

8 Retrieve information for the specified *key* as published by the process identified in the given
9 **pmix_proc_t** , returning a pointer to the value in the given address.

10 This is a blocking operation - the caller will block until either the specified data becomes available
11 from the specified rank in the *proc* structure or the operation times out should the **PMIX_TIMEOUT**
12 attribute have been given. The caller is responsible for freeing all memory associated with the
13 returned *value* when no longer required.

14 The *info* array is used to pass user requests regarding the get operation.

15 Information provided by the PMIx server at time of process start is accessed by providing the
16 namespace of the job with the rank set to **PMIX_RANK_WILDCARD** . The list of data referenced in
17 this way is maintained on the PMIx web site at https://pmix.org/support/faq/wildcard-rank-access/
18 but includes items such as the number of processes in the namespace (**PMIX_JOB_SIZE**), total
19 available slots in the allocation (**PMIX_UNIV_SIZE**), and the number of nodes in the allocation (
20 **PMIX_NUM_NODES**).

21 Data posted by a process via **PMIx_Put** needs to be retrieved by specifying the rank of the
22 posting process. All other information is retrievable using a rank of **PMIX_RANK_WILDCARD**
23 when the information being retrieved refers to something non-rank specific (e.g., number of
24 processes on a node, number of processes in a job), and using the rank of the relevant process when
25 requesting information that is rank-specific (e.g., the URI of the process, or the node upon which it
26 is executing). Each subsection of Section 14.4 indicates the appropriate rank value for referencing
27 the defined attribute.

28 ## 5.1.3 **PMIx_Get_nb**

29 **Summary**
30 Nonblocking **PMIx_Get** operation.

Format

```
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)
```

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)

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

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.

PMIX_GET_STATIC_VALUES **"pmix.get.static"** (**bool**)
    Request that any pointers in the returned value point directly to values in the key-value store
    and indicate that user takes responsibility for properly releasing memory on the returned
    value (i.e., free'ing the value structure but not the pointer fields). Note that a return status of
    PMIX_ERR_GET_MALLOC_REQD indicates that direct pointers could not be supported -
    thus, the returned data contains allocated memory that the user must release.

▲---▲

▼----------------- 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.

1 We recommend that implementation of the **PMIX_TIMEOUT** attribute be left to the host
2 environment due to race condition considerations between delivery of the data by the host
3 environment versus internal timeout in the PMIx server library. Implementers that choose to
4 support **PMIX_TIMEOUT** directly in the PMIx server library must take care to resolve the race
5 condition and should avoid passing **PMIX_TIMEOUT** to the host environment so that multiple
6 competing timeouts are not created.

Description

8 The callback function will be executed once the specified data becomes available from the
9 identified process and retrieved by the local server. The *info* array is used as described by the
10 **PMIx_Get** routine.

11 Information provided by the PMIx server at time of process start is accessed by providing the
12 namespace of the job with the rank set to **PMIX_RANK_WILDCARD** . Attributes referenced in this
13 way are identified in 14.4 but includes items such as the number of processes in the namespace (
14 **PMIX_JOB_SIZE**), total available slots in the allocation (**PMIX_UNIV_SIZE**), and the number
15 of nodes in the allocation (**PMIX_NUM_NODES**).

16 In general, data posted by a process via **PMIx_Put** and data that refers directly to a
17 process-related value needs to be retrieved by specifying the rank of the posting process. All other
18 information is retrievable using a rank of **PMIX_RANK_WILDCARD** , as illustrated in 5.1.5. See
19 14.4.11 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.

— C —

```
2   pmix_status_t
3   PMIx_Store_internal(const pmix_proc_t *proc,
4                       const pmix_key_t key,
5                       pmix_value_t *val);
```

— C —

6   **IN**   **proc**
7   process reference (handle)
8   **IN**   **key**
9   key to retrieve (string)
10   **IN**   **val**
11   Value to store (handle)

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

13   **Description**
14   Store some data locally for retrieval by other areas of the proc. This is data that has only internal
15   scope - it will never be "pushed" externally.

16   5.1.5   Accessing information: examples

17   This section provides examples illustrating methods for accessing information at various levels.
18   The intent of the examples is not to provide comprehensive coding guidance, but rather to illustrate
19   how **PMIx_Get** can be used to obtain information on a **session**, **job**, **application**,
20   process, and node.

21   5.1.5.1   Session-level information

22   The **PMIx_Get** API does not include an argument for specifying the **session** associated with
23   the information being requested. Information regarding the session containing the requestor can be
24   obtained by the following methods:

25   • for session-level attributes (e.g., **PMIX_UNIV_SIZE**), specifying the requestor's namespace
26   and a rank of **PMIX_RANK_WILDCARD**; or

27   • for non-specific attributes (e.g., **PMIX_NUM_NODES**), including the **PMIX_SESSION_INFO**
28   attribute to indicate that the session-level information for that attribute is being requested

29   Example requests are shown below:

———————————————————— C ————————————————————

```
1    pmix_info_t info;
2    pmix_value_t *value;
3    pmix_status_t rc;
4    pmix_proc_t myproc, wildcard;
5
6    /* initialize the client library */
7    PMIx_Init(&myproc, NULL, 0);
8
9    /* get the #slots in our session */
10   PMIX_PROC_LOAD(&wildcard, myproc.nspace, PMIX_RANK_WILDCARD);
11   rc = PMIx_Get(&wildcard, PMIX_UNIV_SIZE, NULL, 0, &value);
12
13   /* get the #nodes in our session */
14   PMIX_INFO_LOAD(&info, PMIX_SESSION_INFO, NULL, PMIX_BOOL);
15   rc = PMIx_Get(&wildcard, PMIX_NUM_NODES, &info, 1, &value);
```

———————————————————— C ————————————————————

16 Information regarding a different session can be requested by either specifying the namespace and a
17 rank of **PMIX_RANK_WILDCARD** for a process in the target session, or adding the
18 **PMIX_SESSION_ID** attribute identifying the target session. In the latter case, the *proc* argument
19 to **PMIx_Get** will be ignored:

———————————————————— C ————————————————————

```
20   pmix_info_t info[2];
21   pmix_value_t *value;
22   pmix_status_t rc;
23   pmix_proc_t myproc;
24   uint32_t sid;
25
26   /* initialize the client library */
27   PMIx_Init(&myproc, NULL, 0);
28
29   /* get the #nodes in a different session */
30   sid = 12345;
31   PMIX_INFO_LOAD(&info[0], PMIX_SESSION_INFO, NULL, PMIX_BOOL);
32   PMIX_INFO_LOAD(&info[1], PMIX_SESSION_ID, &sid, PMIX_UINT32);
33   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 ----------------------------------

```
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 ———————————————————————

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_MAX_PROCS**), 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 ————————————————————————

```
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_MAX_PROCS, info, 2, &value);
```

———————————————————————— C ————————————————————————

———————————————————————— Advice to users ————————————————————————

An explanation of the use of **PMIx_Get** versus **PMIx_Query_info_nb** is provided in 7.1.4.1.

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

———————————————————— 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

――――――――――――――――――― 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.

▲――▲

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.

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.

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

$$C$$

```
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.

- **PMIX_OPERATION_SUCCEEDED** , indicating that the request was immediately processed and returned *success* - the *cbfunc* will *not* be called. This can occur if the collective involved only processes on the local node.

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

1   The following attributes are optional for host environments that support this operation:

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

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

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

------------------- Advice to PMIx library implementers -------------------

13   We recommend that implementation of the **PMIX_TIMEOUT** attribute be left to the host
14   environment due to race condition considerations between completion of the operation versus
15   internal timeout in the PMIx server library. Implementers that choose to support **PMIX_TIMEOUT**
16   directly in the PMIx server library must take care to resolve the race condition and should avoid
17   passing **PMIX_TIMEOUT** to the host environment so that multiple competing timeouts are not
18   created.

19   Note that PMIx libraries may choose to implement an optimization for the case where only the
20   calling process is involved in the fence operation by immediately returning
21   **PMIX_OPERATION_SUCCEEDED** from the client's call in lieu of passing the fence operation to a
22   PMIx server. Fence operations involving more than just the calling process must be communicated
23   to the PMIx server for proper execution of the included barrier behavior.

24   Similarly, fence operations that involve only processes that are clients of the same PMIx server may
25   be resolved by that server without referral to its host environment as no inter-node coordination is
26   required.

27   **Description**
28   Nonblocking **PMIx_Fence** routine. Note that the function will return an error if a **NULL** callback
29   function is given.

30   Note that for scalability reasons, the default behavior for **PMIx_Fence_nb** is to not collect the
31   data.

32   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.

1   The following attributes are optional for host environments that support this operation:

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

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

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

10   We recommend that implementation of the **PMIX_TIMEOUT** attribute be left to the host
11   environment due to race condition considerations between completion of the operation versus
12   internal timeout in the PMIx server library. Implementers that choose to support **PMIX_TIMEOUT**
13   directly in the PMIx server library must take care to resolve the race condition and should avoid
14   passing **PMIX_TIMEOUT** to the host environment so that multiple competing timeouts are not
15   created.

16   **Description**
17   Publish the data in the *info* array for subsequent lookup. By default, the data will be published into
18   the **PMIX_RANGE_SESSION** range and with **PMIX_PERSIST_APP** persistence. Changes to
19   those values, and any additional directives, can be included in the **pmix_info_t** array. Attempts
20   to access the data by processes outside of the provided data range will be rejected. The persistence
21   parameter instructs the server as to how long the data is to be retained.

22   The blocking form will block until the server confirms that the data has been sent to the PMIx
23   server and that it has obtained confirmation from its host SMS daemon that the data is ready to be
24   looked up. Data is copied into the backing key-value data store, and therefore the *info* array can be
25   released upon return from the blocking function call.

26   Publishing duplicate keys is permitted provided they are published to different ranges.

27   Implementations should, to the best of their ability, detect duplicate keys being posted on the same
28   data range and protect the user from unexpected behavior by returning the
29   **PMIX_ERR_DUPLICATE_KEY** error.

5.3.2 `PMIx_Publish_nb`

Summary

Nonblocking **PMIx_Publish** routine.

Format

PMIx v1.0

```
pmix_status_t
PMIx_Publish_nb(const pmix_info_t info[], size_t ninfo,
                pmix_op_cbfunc_t cbfunc, void *cbdata)
```

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.

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_status_t
PMIx_Lookup(pmix_pdata_t data[], size_t ndata,
            const pmix_info_t info[], size_t ninfo)
```

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).

▲--▲

1 We recommend that implementation of the **PMIX_TIMEOUT** attribute be left to the host
2 environment due to race condition considerations between completion of the operation versus
3 internal timeout in the PMIx server library. Implementers that choose to support **PMIX_TIMEOUT**
4 directly in the PMIx server library must take care to resolve the race condition and should avoid
5 passing **PMIX_TIMEOUT** to the host environment so that multiple competing timeouts are not
6 created.

Description

8 Lookup information published by this or another process. By default, the search will be conducted
9 across the **PMIX_RANGE_SESSION** range. Changes to the range, and any additional directives,
10 can be provided in the **pmix_info_t** array. Data is returned provided the following conditions
11 are met:

12 • the requesting process resides within the range specified by the publisher. For example, data
13   published to **PMIX_RANGE_LOCAL** can only be discovered by a process executing on the same
14   node

15 • the provided key matches the published key within that data range

16 • the data was published by a process with corresponding user and/or group IDs as the one looking
17   up the data. There currently is no option to override this behavior - such an option may become
18   available later via an appropriate **pmix_info_t** directive.

19 The *data* parameter consists of an array of **pmix_pdata_t** struct with the keys specifying the
20 requested information. Data will be returned for each key in the associated *value* struct. Any key
21 that cannot be found will return with a data type of **PMIX_UNDEF**. The function will return
22 **PMIX_SUCCESS** if any values can be found, so the caller must check each data element to ensure
23 it was returned.

24 The proc field in each **pmix_pdata_t** struct will contain the namespace/rank of the process that
25 published the data.

26 Although this is a blocking function, it will not wait by default for the requested data to be
27 published. Instead, it will block for the time required by the server to lookup its current data and
28 return any found items. Thus, the caller is responsible for ensuring that data is published prior to
29 executing a lookup, using **PMIX_WAIT** to instruct the server to wait for the data to be published, or
30 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 ───────────────── ▼

```
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.

▲- -▲

1   The following attributes are optional for host environments that support this operation:

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

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

8   **PMIX_WAIT** **"pmix.wait"** (**int**)
9   Caller requests that the PMIx server wait until at least the specified number of values are
10   found (*0* indicates all and is the default).

--------------------- Advice to PMIx library implementers ---------------------

11   We recommend that implementation of the **PMIX_TIMEOUT** attribute be left to the host
12   environment due to race condition considerations between completion of the operation versus
13   internal timeout in the PMIx server library. Implementers that choose to support **PMIX_TIMEOUT**
14   directly in the PMIx server library must take care to resolve the race condition and should avoid
15   passing **PMIX_TIMEOUT** to the host environment so that multiple competing timeouts are not
16   created.

17   **Description**
18   Non-blocking form of the **PMIx_Lookup** function. Data for the provided NULL-terminated *keys*
19   array will be returned in the provided callback function. As with **PMIx_Lookup**, the default
20   behavior is to not wait for data to be published. The *info* array can be used to modify the behavior
21   as previously described by **PMIx_Lookup**. Both the *info* and *keys* arrays must be maintained until
22   the callback is provided.

23   ## 5.3.5 **PMIx_Unpublish**

24   **Summary**
25   Unpublish data posted by this process using the given keys.

Format

```
pmix_status_t
PMIx_Unpublish(char **keys,
                const pmix_info_t info[], size_t ninfo)
```

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.

2 Unpublish data posted by this process using the given *keys*. The function will block until the data
3 has been removed by the server (i.e., it is safe to publish that key again). A value of **NULL** for the
4 *keys* parameter instructs the server to remove all data published by this process.

5 By default, the range is assumed to be **PMIX_RANGE_SESSION** . Changes to the range, and any
6 additional directives, can be provided in the *info* array.

7 5.3.6 `PMIx_Unpublish_nb`

8 **Summary**
9 Nonblocking version of **PMIx_Unpublish** .

10 **Format**

PMIx v1.0 ▼ ─────────── C ───────────── ▼

```
11      pmix_status_t
12      PMIx_Unpublish_nb(char **keys,
13                        const pmix_info_t info[], size_t ninfo,
14                        pmix_op_cbfunc_t cbfunc, void *cbdata)
```

▲ ─────────── C ───────────── ▲

15 **IN   keys**
16   (array of strings)
17 **IN   info**
18   Array of info structures (array of handles)
19 **IN   ninfo**
20   Number of element in the *info* array (integer)
21 **IN   cbfunc**
22   Callback function **pmix_op_cbfunc_t** (function reference)
23 **IN   cbdata**
24   Data to be passed to the callback function (memory reference)

25 Returns one of the following:

26 • **PMIX_SUCCESS** , indicating that the request is being processed by the host environment - result
27   will be returned in the provided *cbfunc*. Note that the library must not invoke the callback
28   function prior to returning from the API.

29 • **PMIX_OPERATION_SUCCEEDED** , indicating that the request was immediately processed and
30   returned *success* - the *cbfunc* will *not* be called

31 • a PMIx error constant indicating either an error in the input or that the request was immediately
32   processed and failed - the *cbfunc* will *not* be called

1   PMIx libraries are not required to directly support any attributes for this function. However, any
2   provided attributes must be passed to the host SMS daemon for processing, and the PMIx library is
3   *required* to add the **PMIX_USERID** and the **PMIX_GRPID** attributes of the client process that is
4   requesting the operation.

5   The following attributes are optional for host environments that support this operation:

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

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

12   We recommend that implementation of the **PMIX_TIMEOUT** attribute be left to the host
13   environment due to race condition considerations between completion of the operation versus
14   internal timeout in the PMIx server library. Implementers that choose to support **PMIX_TIMEOUT**
15   directly in the PMIx server library must take care to resolve the race condition and should avoid
16   passing **PMIX_TIMEOUT** to the host environment so that multiple competing timeouts are not
17   created.

18   **Description**
19   Non-blocking form of the **PMIx_Unpublish** function. The callback function will be executed
20   once the server confirms removal of the specified data. The *info* array must be maintained until the
21   callback is provided.

Process Management

1   This chapter defines functionality used by clients to create and destroy/abort processes in the PMIx
2   universe.

6.1   Abort

4   PMIx provides a dedicated API by which an application can request that specified processes be
5   aborted by the system.

6.1.1   **PMIx_Abort**

7   **Summary**
8   Abort the specified processes

9   **Format**

PMIx v1.0 ──────────────────── C ────────────────────

```
pmix_status_t
PMIx_Abort(int status, const char msg[],
           pmix_proc_t procs[], size_t nprocs)
```

──────────────────── C ────────────────────

13 **IN   status**
14   Error code to return to invoking environment (integer)
15 **IN   msg**
16   String message to be returned to user (string)
17 **IN   procs**
18   Array of **pmix_proc_t** structures (array of handles)
19 **IN   nprocs**
20   Number of elements in the *procs* array (integer)

21 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

```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[])
```

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

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

▲--▲

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.

1 Behavior of individual resource managers may differ, but it is expected that failure of any
2 application process to start will result in termination/cleanup of all processes in the newly spawned
3 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

1 PMIx libraries are not required to directly support any attributes for this function. However, any
2 provided attributes must be passed to the host SMS daemon for processing, and the PMIx library is
3 required to add the following attributes to those provided before passing the request to the host:

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

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

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

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

12 ───

13 Host environments that implement support for **PMIx_Spawn** are required to pass the
14 **PMIX_SPAWNED** and **PMIX_PARENT_ID** attributes to all PMIx servers launching new child
15 processes so those values can be returned to clients upon connection to the PMIx server. In
16 addition, they are required to support the following attributes when present in either the *job_info* or
17 the *info* array of an element of the *apps* array:

18 **PMIX_WDIR** **"pmix.wdir"** (**char***)
19   Working directory for spawned processes.

20 **PMIX_SET_SESSION_CWD** **"pmix.ssncwd"** (**bool**)
21   Set the application's current working directory to the session working directory assigned by
22   the RM - when accessed using **PMIx_Get** , use the **PMIX_RANK_WILDCARD** value for
23   the rank to discover the session working directory assigned to the provided namespace
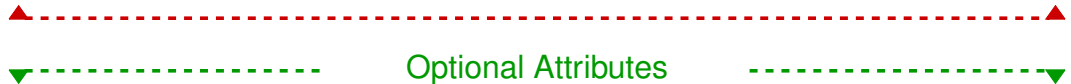
24 **PMIX_PREFIX** **"pmix.prefix"** (**char***)
25   Prefix to use for starting spawned processes.

26 **PMIX_HOST** **"pmix.host"** (**char***)
27   Comma-delimited list of hosts to use for spawned processes.

28 **PMIX_HOSTFILE** **"pmix.hostfile"** (**char***)
29   Hostfile to use for spawned processes.
   ▲---▲

1   The following attributes are optional for host environments that support this operation:

2   **PMIX_ADD_HOSTFILE** **"pmix.addhostfile"** (**char***)
3   Hostfile listing hosts to add to existing allocation.

4   **PMIX_ADD_HOST** **"pmix.addhost"** (**char***)
5   Comma-delimited list of hosts to add to the allocation.

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

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

10   **PMIX_PERSONALITY** **"pmix.pers"** (**char***)
11   Name of personality to use.

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

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

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

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

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

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

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

34   **PMIX_STDIN_TGT** **"pmix.stdin"** (**uint32_t**)
35   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 that 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.

The *connect* operation does not require the exchange of job-level information nor the inclusion of
information posted by participating processes via **PMIx_Put** . Indeed, the callback function
utilized in **pmix_server_connect_fn_t** cannot pass information back into the PMIx server
library. However, host environments are advised that collecting such information at the
participating daemons represents an optimization opportunity as participating processes are likely
to request such information after the connect operation completes.

1 Attempting to *connect* processes solely within the same namespace is essentially a *no-op* operation.
2 While not explicitly prohibited, users are advised that a PMIx implementation or host environment
3 may return an error in such cases.

4 Neither the PMIx implementation nor host environment are required to provide any tracking
5 support for the assemblage. Thus, the application is responsible for maintaining the membership
6 list of the assemblage.

7 6.3.1 `PMIx_Connect`

8 **Summary**
9 Connect namespaces.

10 **Format**

PMIx v1.0 ▼———————————————— C ————————————————▼

```
11  pmix_status_t
12  PMIx_Connect(const pmix_proc_t procs[], size_t nprocs,
13                const pmix_info_t info[], size_t ninfo)
```

△———————————————— C ————————————————△

14 **IN   procs**
15   Array of proc structures (array of handles)
16 **IN   nprocs**
17   Number of elements in the *procs* array (integer)
18 **IN   info**
19   Array of info structures (array of handles)
20 **IN   ninfo**
21   Number of elements in the *info* array (integer)

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

▼- - - - - - - - - - - - - - - - - - Required Attributes - - - - - - - - - - - - - - - - - -▼

23 PMIx libraries are not required to directly support any attributes for this function. However, any
24 provided attributes must be passed to the host SMS daemon for processing.

1　The following attributes are optional for host environments that support this operation:

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

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

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

——————— Advice to PMIx library implementers ———————

13　We recommend that implementation of the **PMIX_TIMEOUT** attribute be left to the host
14　environment due to race condition considerations between completion of the operation versus
15　internal timeout in the PMIx server library. Implementers that choose to support **PMIX_TIMEOUT**
16　directly in the PMIx server library must take care to resolve the race condition and should avoid
17　passing **PMIX_TIMEOUT** to the host environment so that multiple competing timeouts are not
18　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_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)
```

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.

1   The following attributes are optional for host environments that support this operation:

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

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

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

▲- -▲

◆- - - - - - - - - - - - - Advice to PMIx library implementers - - - - - - - - - - - - -▼

13   We recommend that implementation of the **PMIX_TIMEOUT** attribute be left to the host
14   environment due to race condition considerations between completion of the operation versus
15   internal timeout in the PMIx server library. Implementers that choose to support **PMIX_TIMEOUT**
16   directly in the PMIx server library must take care to resolve the race condition and should avoid
17   passing **PMIX_TIMEOUT** to the host environment so that multiple competing timeouts are not
18   created.

▲- -▲

19   **Description**
20   Nonblocking version of **PMIx_Connect** . The callback function is called once all processes
21   identified in *procs* have called either **PMIx_Connect** or its non-blocking version, *and* the host
22   environment has completed any supporting operations required to meet the terms of the PMIx
23   definition of *connected* processes. See the advice provided in the description for **PMIx_Connect**
24   for more information.

25 **6.3.3** **PMIx_Disconnect**

26   **Summary**
27   Disconnect a previously connected set of processes.

Format

C

```
2   pmix_status_t
3   PMIx_Disconnect(const pmix_proc_t procs[], size_t nprocs,
4                   const pmix_info_t info[], size_t ninfo);
```

C

5   **IN   procs**
6   Array of proc structures (array of handles)
7   **IN   nprocs**
8   Number of elements in the *procs* array (integer)
9   **IN   info**
10   Array of info structures (array of handles)
11 **IN   ninfo**
12   Number of element in the *info* array (integer)

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

▼----------------- Required Attributes -------------------▼

14 PMIx libraries are not required to directly support any attributes for this function. However, any
15 provided attributes must be passed to the host SMS daemon for processing.

▼----------------- Optional Attributes -------------------▼

16 The following attributes are optional for host environments that support this operation:

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

——————————— Advice to PMIx library implementers ———————————

21 We recommend that implementation of the **PMIX_TIMEOUT** attribute be left to the host
22 environment due to race condition considerations between completion of the operation versus
23 internal timeout in the PMIx server library. Implementers that choose to support **PMIX_TIMEOUT**
24 directly in the PMIx server library must take care to resolve the race condition and should avoid
25 passing **PMIX_TIMEOUT** to the host environment so that multiple competing timeouts are not
26 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

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 ——————————

1   We recommend that implementation of the **PMIX_TIMEOUT** attribute be left to the host
2   environment due to race condition considerations between completion of the operation versus
3   internal timeout in the PMIx server library. Implementers that choose to support **PMIX_TIMEOUT**
4   directly in the PMIx server library must take care to resolve the race condition and should avoid
5   passing **PMIX_TIMEOUT** to the host environment so that multiple competing timeouts are not
6   created.

Description

7   Nonblocking **PMIx_Disconnect** routine. The callback function is called once all processes
8   identified in *procs* have called either **PMIx_Disconnect_nb** or its blocking version, *and* the
9   host environment has completed any required supporting operations. See the advice provided in the
10 description for **PMIx_Disconnect** for more information.

6.4   IO Forwarding

13 This section defines functions by which tools (e.g., debuggers) can request forwarding of
14 input/output to/from other processes. The term "tool" widely refers to non-computational programs
15 executed by the user or system administrator to monitor or control a principal computational
16 program. Tools almost always interact with either the host environment, user applications, or both
17 to perform administrative and support functions. For example, a debugger tool might be used to
18 remotely control the processes of a parallel application, monitoring their behavior on a step-by-step
19 basis.

20 Underlying the operation of many tools is a common need to forward stdin from the tool to targeted
21 processes, and to return stdout/stderr from those processes for display on the user's console.
22 Historically, each tool developer was responsible for creating their own IO forwarding subsystem.
23 However, with the introduction of PMIx as a standard mechanism for interacting between
24 applications and the host environment, it has become possible to relieve tool developers of this
25 burden.

—————————— Advice to PMIx server hosts ——————————

26 The responsibility of the host environment in forwarding of IO falls into the following areas:

27 • Capturing output from specified child processes

28 • Forwarding that output to the host of the PMIx server library that requested it

29 • Delivering that payload to the PMIx server library via the **PMIx_server_IOF_deliver**
30   API for final dispatch

31 It is the responsibility of the PMIx library to buffer, format, and deliver the payload to the
32 requesting client.

1 The forwarding of IO via PMIx requires that both the host environment and the tool support PMIx,
2 but does not impose any similar requirements on the application itself.

3 ### 6.4.1 `PMIx_IOF_pull`

4 **Summary**
5 Register to receive output forwarded from a set of remote processes.

6 **Format**

PMIx v3.0 ▼ ──────────────────── C ──────────────────── ▼

```
pmix_status_t
PMIx_IOF_pull(const pmix_proc_t procs[], size_t nprocs,
              const pmix_info_t directives[], size_t ndirs,
              pmix_iof_channel_t channel, pmix_iof_cbfunc_t cbfunc,
              pmix_hdlr_reg_cbfunc_t regcbfunc, void *regcbdata)
```

▲ ──────────────────── C ──────────────────── ▲

12 **IN   procs**
13   Array of proc structures identifying desired source processes (array of handles)
14 **IN   nprocs**
15   Number of elements in the *procs* array (integer)
16 **IN   directives**
17   Array of **pmix_info_t** structures (array of handles)
18 **IN   ndirs**
19   Number of elements in the *directives* array (integer)
20 **IN   channel**
21   Bitmask of IO channels included in the request (**pmix_iof_channel_t**)
22 **IN   cbfunc**
23   Callback function for delivering relevant output (**pmix_iof_cbfunc_t** function
24   reference)
25 **IN   regcbfunc**
26   Function to be called when registration is completed (**pmix_hdlr_reg_cbfunc_t**
27   function reference)
28 **IN   regcbdata**
29   Data to be passed to the *regcbfunc* callback function (memory reference)

If *regcbfunc* is **NULL**, the function call will be treated as a *blocking* call. In this case, the returned
status will be either (a) the IOF handler reference identifier if the value is greater than or equal to
zero, or (b) a negative error code indicative of the reason for the failure.

If the *regcbfunc* is non-**NULL**, the function call will be treated as a *non-blocking* call and will return
the following:

PMIX_SUCCESS indicating that the request has been accepted for processing and the provided
    callback function will be executed upon completion of the operation. Note that the library
    must not invoke the callback function prior to returning from the API. The IOF handler
    identifier will be returned in the callback

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.

▼------------------   Required Attributes   ------------------▼

The following attributes are required for PMIx libraries that support IO forwarding:

PMIX_IOF_CACHE_SIZE   **"pmix.iof.csize"** (**uint32_t**)
    The requested size of the server cache in bytes for each specified channel. By default, the
    server is allowed (but not required) to drop all bytes received beyond the max size.

PMIX_IOF_DROP_OLDEST   **"pmix.iof.old"** (**bool**)
    In an overflow situation, drop the oldest bytes to make room in the cache.

PMIX_IOF_DROP_NEWEST   **"pmix.iof.new"** (**bool**)
    In an overflow situation, drop any new bytes received until room becomes available in the
    cache (default).

▲---▲

▼------------------   Optional Attributes   ------------------▼

The following attributes are optional for PMIx libraries that support IO forwarding:

PMIX_IOF_BUFFERING_SIZE   **"pmix.iof.bsize"** (**uint32_t**)
    Controls grouping of IO on the specified channel(s) to avoid being called every time a bit of
    IO arrives. The library will execute the callback whenever the specified number of bytes
    becomes available. Any remaining buffered data will be "flushed" upon call to deregister the
    respective channel.

PMIX_IOF_BUFFERING_TIME   **"pmix.iof.btime"** (**uint32_t**)
    Max time in seconds to buffer IO before delivering it. Used in conjunction with buffering
    size, this prevents IO from being held indefinitely while waiting for another payload to
    arrive.

PMIX_IOF_TAG_OUTPUT   **"pmix.iof.tag"** (**bool**)
    Tag output with the channel it comes from.

PMIX_IOF_TIMESTAMP_OUTPUT   **"pmix.iof.ts"** (**bool**)

1	Timestamp output
2	**PMIX_IOF_XML_OUTPUT** **"pmix.iof.xml"** (**bool**)
3	Format output in eXtensible Markup Language (XML)

▲ - ▲

4	**Description**
5	Register to receive output forwarded from a set of remote processes.

───────────────── Advice to users ─────────────────

6	Providing a **NULL** function pointer for the *cbfunc* parameter will cause output for the indicated
7	channels to be written to their corresponding stdout/stderr file descriptors. Use of
8	**PMIX_RANK_WILDCARD** to specify all processes in a given namespace is supported but should
9	be used carefully due to bandwidth considerations.

10 6.4.2 **PMIx_IOF_deregister**

11	**Summary**
12	Deregister from output forwarded from a set of remote processes.

13	**Format**

PMIx v3.0 ▼ ───────────────── C ───────────────── ▼

14	**pmix_status_t**
15	**PMIx_IOF_deregister(size_t iofhdlr,**
16	**const pmix_info_t directives[], size_t ndirs,**
17	**pmix_op_cbfunc_t cbfunc, void *cbdata)**

───────────────── C ─────────────────

18	**IN iofhdlr**
19	Registration number returned from the **pmix_hdlr_reg_cbfunc_t** callback from the
20	call to **PMIx_IOF_pull** (**size_t**)
21	**IN directives**
22	Array of **pmix_info_t** structures (array of handles)
23	**IN ndirs**
24	Number of elements in the *directives* array (integer)
25	**IN cbfunc**
26	Callback function to be called when deregistration has been completed. (function reference)
27	**IN cbdata**
28	Data to be passed to the *cbfunc* callback function (memory reference)

1 If *cbfunc* is **NULL**, the function will be treated as a *blocking* call and the result of the operation
2 returned in the status code.

3 If *cbfunc* is non-**NULL**, the function will be treated as a *non-blocking* call and return one of the
4 following:

5 • **PMIX_SUCCESS** , indicating that the request is being processed - result will be returned in the
6   provided *cbfunc*. Note that the library must not invoke the callback function prior to returning
7   from the API.

8 • **PMIX_OPERATION_SUCCEEDED** , indicating that the request was immediately processed and
9   returned *success* - the *cbfunc* will *not* be called

10 • a PMIx error constant indicating either an error in the input or that the request was immediately
11   processed and failed - the *cbfunc* will *not* be called

12 The returned status code will be one of the following:

13 **PMIX_SUCCESS** The IOF handler was successfully deregistered.
14 **PMIX_ERR_BAD_PARAM** The provided *iofhdlr* was unrecognized.
15 **PMIX_ERR_NOT_SUPPORTED** The PMIx implementation does not support this function.

16 **Description**
17 Deregister from output forwarded from a set of remote processes.

———————————— Advice to PMIx library implementers ————————————

18 Any currently buffered IO should be flushed upon receipt of a deregistration request. All received
19 IO after receipt of the request shall be discarded.

20 6.4.3 `PMIx_IOF_push`

21 **Summary**
22 Push data collected locally (typically from stdin or a file) to stdin of the target recipients.

Format

```
2    pmix_status_t
3    PMIx_IOF_push(const pmix_proc_t targets[], size_t ntargets,
4                    pmix_byte_object_t *bo,
5                    const pmix_info_t directives[], size_t ndirs,
6                    pmix_op_cbfunc_t cbfunc, void *cbdata)
```

7   **IN   targets**
8   Array of proc structures identifying desired target processes (array of handles)
9   **IN   ntargets**
10   Number of elements in the *targets* array (integer)
11 **IN   bo**
12   Pointer to **pmix_byte_object_t** containing the payload to be delivered (handle)
13 **IN   directives**
14   Array of **pmix_info_t** structures (array of handles)
15 **IN   ndirs**
16   Number of elements in the *directives* array (integer)
17 **IN   directives**
18   Array of **pmix_info_t** structures (array of handles)
19 **IN   cbfunc**
20   Callback function to be called when operation has been completed. (**pmix_op_cbfunc_t**
21   function reference)
22 **IN   cbdata**
23   Data to be passed to the *cbfunc* callback function (memory reference)

24   If *cbfunc* is **NULL**, the function will be treated as a *blocking* call and the result of the operation
25   returned in the status code.

26   If *cbfunc* is non-**NULL**, the function will be treated as a *non-blocking* call and return one of the
27   following:

28   • **PMIX_SUCCESS** , indicating that the request is being processed - result will be returned in the
29   provided *cbfunc*. Note that the library must not invoke the callback function prior to returning
30   from the API.

31   • **PMIX_OPERATION_SUCCEEDED** , indicating that the request was immediately processed and
32   returned *success* - the *cbfunc* will *not* be called

33   • a PMIx error constant indicating either an error in the input or that the request was immediately
34   processed and failed - the *cbfunc* will *not* be called

35   The returned status code will be one of the following:

36   **PMIX_SUCCESS** The provided data has been accepted for transmission - it is not indicative of
37   the payload being delivered to any member of the provided *targets*

1   **PMIX_ERR_NOT_SUPPORTED** The PMIx implementation does not support this function.
2   a PMIx error constant indicating the nature of the error

▼----------------- Required Attributes -----------------▼

3   The following attributes are required for PMIx libraries that support IO forwarding:

4   **PMIX_IOF_CACHE_SIZE "pmix.iof.csize"** (**uint32_t**)
5   The requested size of the server cache in bytes for each specified channel. By default, the
6   server is allowed (but not required) to drop all bytes received beyond the max size.

7   **PMIX_IOF_DROP_OLDEST "pmix.iof.old"** (**bool**)
8   In an overflow situation, drop the oldest bytes to make room in the cache.

9   **PMIX_IOF_DROP_NEWEST "pmix.iof.new"** (**bool**)
10   In an overflow situation, drop any new bytes received until room becomes available in the
11   cache (default).

▲---▲

▼----------------- Optional Attributes -----------------▼

12   The following attributes are optional for PMIx libraries that support IO forwarding:

13   **PMIX_IOF_BUFFERING_SIZE "pmix.iof.bsize"** (**uint32_t**)
14   Controls grouping of IO on the specified channel(s) to avoid being called every time a bit of
15   IO arrives. The library will execute the callback whenever the specified number of bytes
16   becomes available. Any remaining buffered data will be "flushed" upon call to deregister the
17   respective channel.

18   **PMIX_IOF_BUFFERING_TIME "pmix.iof.btime"** (**uint32_t**)
19   Max time in seconds to buffer IO before delivering it. Used in conjunction with buffering
20   size, this prevents IO from being held indefinitely while waiting for another payload to
21   arrive.

▲---▲

22   **Description**
23   Push data collected locally (typically from stdin or a file) to stdin of the target recipients.

◆─────────────────────── Advice to users ───────────────────────

24   Execution of the *cbfunc* callback function serves as notice that the PMIx library no longer requires
25   the caller to maintain the *bo* data object - it does *not* indicate delivery of the payload to the targets.
26   Use of **PMIX_RANK_WILDCARD** to specify all processes in a given namespace is supported but
27   should be used carefully due to bandwidth considerations.

◆───

Job Management and Reporting

1   The job management APIs provide an application with the ability to orchestrate its operation in
2   partnership with the SMS. Members of this category include the
3   **PMIx_Allocation_request_nb** , **PMIx_Job_control_nb** , and
4   **PMIx_Process_monitor_nb** APIs.

7.1 Query

6   As the level of interaction between applications and the host SMS grows, so too does the need for
7   the application to query the SMS regarding its capabilities and state information. PMIx provides a
8   generalized query interface for this purpose, along with a set of standardized attribute keys to
9   support a range of requests. This includes requests to determine the status of scheduling queues and
10   active allocations, the scope of API and attribute support offered by the SMS, namespaces of active
11   jobs, location and information about a job's processes, and information regarding available
12   resources.

13   An example use-case for the **PMIx_Query_info_nb** API is to ensure clean job completion.
14   Time-shared systems frequently impose maximum run times when assigning jobs to resource
15   allocations. To shut down gracefully, e.g., to write a checkpoint before termination, it is necessary
16   for an application to periodically query the resource manager for the time remaining in its
17   allocation. This is especially true on systems for which allocation times may be shortened or
18   lengthened from the original time limit. Many resource managers provide APIs to dynamically
19   obtain this information, but each API is specific to the resource manager.

20   PMIx supports this use-case by defining an attribute key (**PMIX_TIME_REMAINING**) that can be
21   used with the **PMIx_Query_info_nb** interface to obtain the number of seconds remaining in
22   the current job allocation. Note that one could alternatively use the
23   **PMIx_Register_event_handler** API to register for an event indicating incipient job
24   termination, and then use the **PMIx_Job_control_nb** API to request that the host SMS
25   generate an event a specified amount of time prior to reaching the maximum run time. PMIx
26   provides such alternate methods as a means of maximizing the probability of a host system
27   supporting at least one method by which the application can obtain the desired service.

28   The following APIs support query of various session and environment values.

7.1.1 **PMIx_Resolve_peers**

Summary
31   Obtain the array of processes within the specified namespace that are executing on a given node.

Format

PMIx v1.0 ──────────────────── 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 ────────────────────

```
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`

Summary

Query information about the system in general.

Format

PMIx v4.0 ▼ ─────────────── C ───────────────── ▼

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

▲ ─────────────── C ───────────────── ▲

IN   `queries`
    Array of query structures (array of handles)
IN   `nqueries`
    Number of elements in the *queries* array (integer)
INOUT `info`
    Address where a pointer to an array of **`pmix_info_t`** containing the results of the query
    can be returned (memory reference)
INOUT `ninfo`
    Address where the number of elements in *info* can be returned (handle)

Returns one of the following:

- **`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_PROCID **"pmix.procid"** (**pmix_proc_t**)
   Process identifier Specifies the process ID whose information is being requested - e.g., a query asking for the **PMIX_LOCAL_RANK** of a specified process. Only required when the request is for information on a specific process.

PMIX_NSPACE **"pmix.nspace"** (**char***)
   Namespace of the job. Specifies the namespace of the process whose information is being requested - e.g., a query asking for the **PMIX_LOCAL_RANK** of a specified process. Must be accompanied by the **PMIX_RANK** attribute. Only required when the request is for information on a specific process.

PMIX_RANK **"pmix.rank"** (**pmix_rank_t**)
   Process rank within the job. Specifies the rank of the process whose information is being requested - e.g., a query asking for the **PMIX_LOCAL_RANK** of a specified process. Must be accompanied by the **PMIX_NSPACE** attribute. Only required when the request is for information on a specific process.

PMIX_QUERY_ATTRIBUTE_SUPPORT **"pmix.qry.attrs"** (**bool**)
   Query list of supported attributes for specified APIs

PMIX_CLIENT_ATTRIBUTES **"pmix.client.attrs"** (**bool**)
   Request attributes supported by the PMIx client library

PMIX_SERVER_ATTRIBUTES **"pmix.srvr.attrs"** (**bool**)

Request attributes supported by the PMIx server library

PMIX_HOST_ATTRIBUTES **"pmix.host.attrs"** (**bool**)
    Request attributes supported by the host environment

PMIX_TOOL_ATTRIBUTES **"pmix.setup.env"** (**bool**)
    Request attributes supported by the PMIx tool library functions

Note that inclusion of the **PMIX_PROCID** directive and either the **PMIX_NSPACE** or the **PMIX_RANK** attribute will return a **PMIX_ERR_BAD_PARAM** result, and that the inclusion of a process identifier must apply to all keys in that **pmix_query_t** . Queries for information on multiple specific processes therefore requires submitting multiple **pmix_query_t** structures, each referencing one process.

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.

Host environments that support this operation are required to support the following attributes as qualifiers to the request:

PMIX_PROCID **"pmix.procid"** (**pmix_proc_t**)
    Process identifier   Specifies the process ID whose information is being requested - e.g., a query asking for the **PMIX_LOCAL_RANK** of a specified process. Only required when the request is for information on a specific process.

PMIX_NSPACE **"pmix.nspace"** (**char***)
    Namespace of the job.   Specifies the namespace of the process whose information is being requested - e.g., a query asking for the **PMIX_LOCAL_RANK** of a specified process. Must be accompanied by the **PMIX_RANK** attribute. Only required when the request is for information on a specific process.

PMIX_RANK **"pmix.rank"** (**pmix_rank_t**)
    Process rank within the job.   Specifies the rank of the process whose information is being requested - e.g., a query asking for the **PMIX_LOCAL_RANK** of a specified process. Must be accompanied by the **PMIX_NSPACE** attribute. Only required when the request is for information on a specific process.

Note that inclusion of the **PMIX_PROCID** directive and either the **PMIX_NSPACE** or the **PMIX_RANK** attribute will return a **PMIX_ERR_BAD_PARAM** result, and that the inclusion of a process identifier must apply to all keys in that **pmix_query_t** . Queries for information on multiple specific processes therefore requires submitting multiple **pmix_query_t** structures, each referencing one process.

1   The following attributes are optional for host environments that support this operation:

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

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

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

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

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

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

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

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

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

23   **PMIX_QUERY_REPORT_AVG** **"pmix.qry.avg"** (**bool**)
24   Report only average values for sampled information.

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

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

29   **PMIX_TIME_REMAINING** **"pmix.time.remaining"** (**char***)
30   Query number of seconds (**uint32_t**) remaining in allocation for the specified namespace.
31

32   **PMIX_SERVER_URI** **"pmix.srvr.uri"** (**char***)
33   URI of the PMIx server to be contacted.   Requests the URI of the specified PMIx server's
34   PMIx connection. Defaults to requesting the information for the local PMIx server.

PMIX_PROC_URI `"pmix.puri"` (**char***)
> URI containing contact information for a given process. Requests the URI of the specified
> PMIx server's out-of-band connection. Defaults to requesting the information for the local
> PMIx server.

▲- -▲

Description

Query information about the system in general. This can include a list of active namespaces, fabric
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.

The returned *status* indicates if requested data was found or not. The returned 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. The caller is responsible for releasing the returned array.

──────────── Advice to PMIx library implementers ────────────

Information returned from **PMIx_Query_info** shall be locally cached so that retrieval by
subsequent calls to **PMIx_Get**, **PMIx_Query_info**, 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.

7.1.4 `PMIx_Query_info_nb`

Summary

Query information about the system in general.

Format

```
pmix_status_t
PMIx_Query_info_nb(pmix_query_t queries[], size_t nqueries,
                   pmix_info_cbfunc_t cbfunc, void *cbdata)
```

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)

Returns one of the following:

- **PMIX_SUCCESS** indicating that the request has been accepted for processing and the provided callback function will be executed upon completion of the operation. Note that the library must not invoke the callback function prior to returning from the API.

- 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_PROCID **"pmix.procid"** (**pmix_proc_t**)
   Process identifier Specifies the process ID whose information is being requested - e.g., a
   query asking for the **PMIX_LOCAL_RANK** of a specified process. Only required when the
   request is for information on a specific process.

PMIX_NSPACE **"pmix.nspace"** (**char***)
   Namespace of the job. Specifies the namespace of the process whose information is being
   requested - e.g., a query asking for the **PMIX_LOCAL_RANK** of a specified process. Must
   be accompanied by the **PMIX_RANK** attribute. Only required when the request is for
   information on a specific process.

PMIX_RANK **"pmix.rank"** (**pmix_rank_t**)
   Process rank within the job. Specifies the rank of the process whose information is being
   requested - e.g., a query asking for the **PMIX_LOCAL_RANK** of a specified process. Must
   be accompanied by the **PMIX_NSPACE** attribute. Only required when the request is for
   information on a specific process.

PMIX_QUERY_ATTRIBUTE_SUPPORT **"pmix.qry.attrs"** (**bool**)
   Query list of supported attributes for specified APIs

PMIX_CLIENT_ATTRIBUTES **"pmix.client.attrs"** (**bool**)
   Request attributes supported by the PMIx client library

PMIX_SERVER_ATTRIBUTES **"pmix.srvr.attrs"** (**bool**)
   Request attributes supported by the PMIx server library

PMIX_HOST_ATTRIBUTES **"pmix.host.attrs"** (**bool**)
   Request attributes supported by the host environment

PMIX_TOOL_ATTRIBUTES **"pmix.setup.env"** (**bool**)
    Request attributes supported by the PMIx tool library functions

Note that inclusion of the **PMIX_PROCID** directive and either the **PMIX_NSPACE** or the **PMIX_RANK** attribute will return a **PMIX_ERR_BAD_PARAM** result, and that the inclusion of a process identifier must apply to all keys in that **pmix_query_t** . Queries for information on multiple specific processes therefore requires submitting multiple **pmix_query_t** structures, each referencing one process.

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.

Host environments that support this operation are required to support the following attributes as qualifiers to the request:

PMIX_PROCID **"pmix.procid"** (**pmix_proc_t**)
    Process identifier   Specifies the process ID whose information is being requested - e.g., a query asking for the **PMIX_LOCAL_RANK** of a specified process. Only required when the request is for information on a specific process.

PMIX_NSPACE **"pmix.nspace"** (**char***)
    Namespace of the job.   Specifies the namespace of the process whose information is being requested - e.g., a query asking for the **PMIX_LOCAL_RANK** of a specified process. Must be accompanied by the **PMIX_RANK** attribute. Only required when the request is for information on a specific process.

PMIX_RANK **"pmix.rank"** (**pmix_rank_t**)
    Process rank within the job.   Specifies the rank of the process whose information is being requested - e.g., a query asking for the **PMIX_LOCAL_RANK** of a specified process. Must be accompanied by the **PMIX_NSPACE** attribute. Only required when the request is for information on a specific process.

Note that inclusion of the **PMIX_PROCID** directive and either the **PMIX_NSPACE** or the **PMIX_RANK** attribute will return a **PMIX_ERR_BAD_PARAM** result, and that the inclusion of a process identifier must apply to all keys in that **pmix_query_t** . Queries for information on multiple specific processes therefore requires submitting multiple **pmix_query_t** structures, each referencing one process.

▼----------------- 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 only average values for sampled information.

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.

PMIX_SERVER_URI **"pmix.srvr.uri"** (**char***)
    URI of the PMIx server to be contacted.   Requests the URI of the specified PMIx server's
    PMIx connection. Defaults to requesting the information for the local PMIx server.

PMIX_PROC_URI **"pmix.puri"** (**char***)
    URI containing contact information for a given process.   Requests the URI of the specified
    PMIx server's out-of-band connection. Defaults to requesting the information for the local
    PMIx server.
    ▲--▲

Description

Non-blocking form of the **PMIx_Query_info** API

7.1.4.1 Using **PMIx_Get** vs **PMIx_Query_info**

Both **PMIx_Get** and **PMIx_Query_info** 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** 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.

7.1.4.2 Accessing attribute support information

Information as to attributes supported by either the PMIx implementation or its host environment can be obtained via the **PMIx_Query_info_nb** API. The **PMIX_QUERY_ATTRIBUTE_SUPPORT** attribute must be listed as the first entry in the *keys* field of the **pmix_query_t** structure, followed by the name of the function whose attribute support is being requested - support for multiple functions can be requested simultaneously by simply adding

1 the function names to the array of *keys*. Function names *must* be given as user-level API names -
2 e.g., "PMIx_Get", "PMIx_server_setup_application", or "PMIx_tool_connect_to_server".

3 The desired levels (see 14.4.33) of attribute support are provided as qualifiers. Multiple levels can
4 be requested simultaneously by simply adding elements to the *qualifiers* array. Each qualifier
5 should contain the desired level attribute with the boolean value set to indicate whether or not that
6 level is to be included in the returned information. Failure to provide any levels is equivalent to a
7 request for all levels.

8 Unlike other queries, queries for attribute support can result in the number of returned
9 `pmix_info_t` structures being different from the number of queries. Each element in the
10 returned array will correspond to a pair of specified attribute level and function in the query, where
11 the *key* is the function and the *value* contains a `pmix_data_array_t` of `pmix_info_t` .
12 Each element of the array is marked by a *key* indicating the requested attribute *level* with a *value*
13 composed of a `pmix_data_array_t` of `pmix_regattr_t` , each describing a supported
14 attribute for that function, as illustrated in Fig. 7.1 below where the requestor asked for supported
15 attributes of `PMIx_Get` at the *client* and *server* levels, plus attributes of
16 `PMIx_Allocation_request` at all levels:
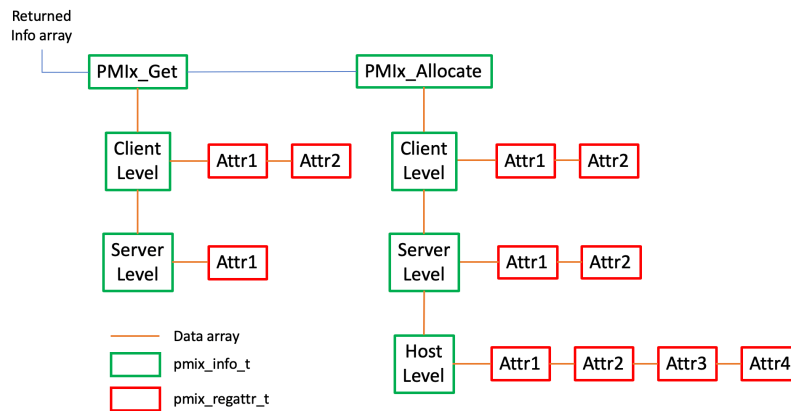
Figure 7.1.: Returned information hierarchy for attribute support request

17 The array of returned structures, and their child arrays, are subject to the return rules for the
18 `PMIx_Query_info_nb` API. For example, a request for supported attributes of the `PMIx_Get`
19 function that includes the *host* level will return values for the *client* and *server* levels, plus an array
20 element with a *key* of `PMIX_HOST_ATTRIBUTES` and a value type of `PMIX_UNDEF` indicating
21 that no attributes are supported at that level.

22 7.2 Allocation Requests

23 This section defines functionality to request new allocations from the RM, and request
24 modifications to existing allocations. These are primarily used in the following scenarios:

1. • *Evolving* applications that dynamically request and return resources as they execute

2. • *Malleable* environments where the scheduler redirects resources away from executing
3. applications for higher priority jobs or load balancing

4. • *Resilient* applications that need to request replacement resources in the face of failures

5. • *Rigid* jobs where the user has requested a static allocation of resources for a fixed period of time,
6. but realizes that they underestimated their required time while executing

7. PMIx attempts to address this range of use-cases with a flexible API.

7.2.1 `PMIx_Allocation_request`

Summary

Request an allocation operation from the host resource manager.

Format

PMIx v3.0
—————————————————— C ——————————————————

```
pmix_status_t
PMIx_Allocation_request(pmix_alloc_directive_t directive,
                        pmix_info_t info[], size_t ninfo,
                        pmix_info_t *results[], size_t *nresults);
```
—————————————————— 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)
INOUT results
    Address where a pointer to an array of **pmix_info_t** containing the results of the request
    can be returned (memory reference)
INOUT nresults
    Address where the number of elements in *results* can be returned (handle)

Returns one of the following:

• **PMIX_SUCCESS** , indicating that the request was processed and returned *success*

• a PMIx error constant indicating either an error in the input or that the request was refused

1 PMIx libraries are not required to directly support any attributes for this function. However, any
2 provided attributes must be passed to the host SMS daemon for processing, and the PMIx library is
3 *required* to add the **PMIX_USERID** and the **PMIX_GRPID** attributes of the client process making
4 the request.

5

6 Host environments that implement support for this operation are required to support the following
7 attributes:

8 **PMIX_ALLOC_REQ_ID** **"pmix.alloc.reqid"** (**char***)
9   User-provided string identifier for this allocation request which can later be used to query
10   status of the request.

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

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

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

------------------ Optional Attributes ------------------

17 The following attributes are optional for host environments that support this operation:

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

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

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

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

26 **PMIX_ALLOC_FABRIC** **"pmix.alloc.net"** (**array**)
27   Array of **pmix_info_t** describing requested fabric resources. This must include at least:
28   **PMIX_ALLOC_FABRIC_ID**, **PMIX_ALLOC_FABRIC_TYPE**, and
29   **PMIX_ALLOC_FABRIC_ENDPTS**, plus whatever other descriptors are desired.

30 **PMIX_ALLOC_FABRIC_ID** **"pmix.alloc.netid"** (**char***)

The key to be used when accessing this requested fabric allocation. The allocation will be returned/stored as a **`pmix_data_array_t`** of **`pmix_info_t`** indexed by this key and containing at least one entry with the same key and the allocated resource description. The type of the included value depends upon the fabric support. For example, a TCP allocation might consist of a comma-delimited string of socket ranges such as *"32000-32100,33005,38123-38146"*. Additional entries will consist of any provided resource request directives, along with their assigned values. Examples include: **`PMIX_ALLOC_FABRIC_TYPE`** - the type of resources provided; **`PMIX_ALLOC_FABRIC_PLANE`** - if applicable, what plane the resources were assigned from; **`PMIX_ALLOC_FABRIC_QOS`** - the assigned QoS; **`PMIX_ALLOC_BANDWIDTH`** - the allocated bandwidth; **`PMIX_ALLOC_FABRIC_SEC_KEY`** - a security key for the requested fabric allocation. NOTE: the assigned values may differ from those requested, especially if **`PMIX_INFO_REQD`** was not set in the request.

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

`PMIX_ALLOC_FABRIC_QOS` **`"pmix.alloc.netqos"`** (**`char*`**)
    Quality of service level.

`PMIX_ALLOC_FABRIC_TYPE` **`"pmix.alloc.nettype"`** (**`char*`**)
    Type of desired transport (e.g., *"tcp"*, *"udp"*)

`PMIX_ALLOC_FABRIC_PLANE` **`"pmix.alloc.netplane"`** (**`char*`**)
    ID string for the NIC (aka *plane*) to be used for this allocation (e.g., CIDR for Ethernet)

`PMIX_ALLOC_FABRIC_ENDPTS` **`"pmix.alloc.endpts"`** (**`size_t`**)
    Number of endpoints to allocate per process

`PMIX_ALLOC_FABRIC_ENDPTS_NODE` **`"pmix.alloc.endpts.nd"`** (**`size_t`**)
    Number of endpoints to allocate per node

`PMIX_ALLOC_FABRIC_SEC_KEY` **`"pmix.alloc.nsec"`** (**`pmix_byte_object_t`**)
    Fabric security key

▲ -- ▲

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.

1   • Extend the reservation on currently allocated resources, subject to scheduling availability and
2   priorities. This includes extending the time limit on current resources, and/or requesting
3   additional resources be allocated to the requesting job. Any additional allocated resources will be
4   considered as part of the current allocation, and thus will be released at the same time.

5   • Return no-longer-required resources to the scheduler. This includes the "loan" of resources back
6   to the scheduler with a promise to return them upon subsequent request.

7   If successful, the returned results for a request for additional resources must include the host
8   resource manager's identifier (**PMIX_ALLOC_ID**) that the requester can use to specify the
9   resources in, for example, a call to **PMIx_Spawn** .

10 7.2.2 **PMIx_Allocation_request_nb**

11   **Summary**
12   Request an allocation operation from the host resource manager.

13   **Format**

PMIx v2.0 ▼ ──────────────── C ──────────────────────── ▼

```
14    pmix_status_t
15    PMIx_Allocation_request_nb(pmix_alloc_directive_t directive,
16                               pmix_info_t info[], size_t ninfo,
17                               pmix_info_cbfunc_t cbfunc, void *cbdata);
```

▲ ──────────────── C ──────────────────────── ▲

18   **IN directive**
19   Allocation directive (handle)
20   **IN info**
21   Array of **pmix_info_t** structures (array of handles)
22   **IN ninfo**
23   Number of elements in the *info* array (integer)
24   **IN cbfunc**
25   Callback function **pmix_info_cbfunc_t** (function reference)
26   **IN cbdata**
27   Data to be passed to the callback function (memory reference)

28   Returns one of the following:

29   • **PMIX_SUCCESS** , indicating that the request is being processed by the host environment - result
30   will be returned in the provided *cbfunc*. Note that the library must not invoke the callback
31   function prior to returning from the API.

32   • **PMIX_OPERATION_SUCCEEDED** , indicating that the request was immediately processed and
33   returned *success* - the *cbfunc* will *not* be called

34   • a PMIx error constant indicating either an error in the input or that the request was immediately
35   processed and failed - the *cbfunc* will *not* be called

1   PMIx libraries are not required to directly support any attributes for this function. However, any
2   provided attributes must be passed to the host SMS daemon for processing, and the PMIx library is
3   *required* to add the **PMIX_USERID** and the **PMIX_GRPID** attributes of the client process making
4   the request.

5

6   Host environments that implement support for this operation are required to support the following
7   attributes:

8   **PMIX_ALLOC_REQ_ID** **"pmix.alloc.reqid"** (**char***)
9   User-provided string identifier for this allocation request which can later be used to query
10   status of the request.

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

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

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

17   The following attributes are optional for host environments that support this operation:

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

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

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

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

26   **PMIX_ALLOC_FABRIC** **"pmix.alloc.net"** (**array**)
27   Array of **pmix_info_t** describing requested fabric resources. This must include at least:
28   **PMIX_ALLOC_FABRIC_ID**, **PMIX_ALLOC_FABRIC_TYPE**, and
29   **PMIX_ALLOC_FABRIC_ENDPTS**, plus whatever other descriptors are desired.

30   **PMIX_ALLOC_FABRIC_ID** **"pmix.alloc.netid"** (**char***)

1	The key to be used when accessing this requested fabric allocation. The allocation will be
2	returned/stored as a **pmix_data_array_t** of **pmix_info_t** indexed by this key and
3	containing at least one entry with the same key and the allocated resource description. The
4	type of the included value depends upon the fabric support. For example, a TCP allocation
5	might consist of a comma-delimited string of socket ranges such as
6	*"32000-32100,33005,38123-38146"*. Additional entries will consist of any provided
7	resource request directives, along with their assigned values. Examples include:
8	**PMIX_ALLOC_FABRIC_TYPE** - the type of resources provided;
9	**PMIX_ALLOC_FABRIC_PLANE** - if applicable, what plane the resources were assigned
10	from; **PMIX_ALLOC_FABRIC_QOS** - the assigned QoS; **PMIX_ALLOC_BANDWIDTH** -
11	the allocated bandwidth; **PMIX_ALLOC_FABRIC_SEC_KEY** - a security key for the
12	requested fabric allocation. NOTE: the assigned values may differ from those requested,
13	especially if **PMIX_INFO_REQD** was not set in the request.

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

16 **PMIX_ALLOC_FABRIC_QOS** **"pmix.alloc.netqos"** (**char***)
17   Quality of service level.

18 **PMIX_ALLOC_FABRIC_TYPE** **"pmix.alloc.nettype"** (**char***)
19   Type of desired transport (e.g., *"tcp"*, *"udp"*)

20 **PMIX_ALLOC_FABRIC_PLANE** **"pmix.alloc.netplane"** (**char***)
21   ID string for the NIC (aka *plane*) to be used for this allocation (e.g., CIDR for Ethernet)

22 **PMIX_ALLOC_FABRIC_ENDPTS** **"pmix.alloc.endpts"** (**size_t**)
23   Number of endpoints to allocate per process

24 **PMIX_ALLOC_FABRIC_ENDPTS_NODE** **"pmix.alloc.endpts.nd"** (**size_t**)
25   Number of endpoints to allocate per node

26 **PMIX_ALLOC_FABRIC_SEC_KEY** **"pmix.alloc.nsec"** (**pmix_byte_object_t**)
27   Fabric security key

▲--▲

28 **Description**
29 Non-blocking form of the **PMIx_Allocation_request** API.

1 7.3 Job Control

2 This section defines APIs that enable the application and host environment to coordinate the
3 response to failures and other events. This can include requesting termination of the entire job or a
4 subset of processes within a job, but can also be used in combination with other PMIx capabilities
5 (e.g., allocation support and event notification) for more nuanced responses. For example, an
6 application notified of an incipient over-temperature condition on a node could use the
7 **PMIx_Allocation_request_nb** interface to request replacement nodes while
8 simultaneously using the **PMIx_Job_control_nb** interface to direct that a checkpoint event be
9 delivered to all processes in the application. If replacement resources are not available, the
10 application might use the **PMIx_Job_control_nb** interface to request that the job continue at
11 a lower power setting, perhaps sufficient to avoid the over-temperature failure.

12 The job control APIs can also be used by an application to register itself as available for preemption
13 when operating in an environment such as a cloud or where incentives, financial or otherwise, are
14 provided to jobs willing to be preempted. Registration can include attributes indicating how many
15 resources are being offered for preemption (e.g., all or only some portion), whether the application
16 will require time to prepare for preemption, etc. Jobs that request a warning will receive an event
17 notifying them of an impending preemption (possibly including information as to the resources that
18 will be taken away, how much time the application will be given prior to being preempted, whether
19 the preemption will be a suspension or full termination, etc.) so they have an opportunity to save
20 their work. Once the application is ready, it calls the provided event completion callback function to
21 indicate that the SMS is free to suspend or terminate it, and can include directives regarding any
22 desired restart.

23 7.3.1 `PMIx_Job_control`

24 **Summary**
25 Request a job control action.

26 **Format**
PMIx v3.0 ▼───────────────── C ─────────────────▼

```
27  pmix_status_t
28  PMIx_Job_control(const pmix_proc_t targets[], size_t ntargets,
29                   const pmix_info_t directives[], size_t ndirs,
30                   pmix_info_t *results[], size_t *nresults)
```
▲───────────────── C ─────────────────▲

31 **IN   targets**
32 Array of proc structures (array of handles)
33 **IN   ntargets**
34 Number of element in the *targets* array (integer)
35 **IN   directives**
36 Array of info structures (array of handles)

1   **IN**   **ndirs**
2   Number of element in the *directives* array (integer)
3   **INOUT** **results**
4   Address where a pointer to an array of **pmix_info_t** containing the results of the request
5   can be returned (memory reference)
6   **INOUT** **nresults**
7   Address where the number of elements in *results* can be returned (handle)

8 Returns one of the following:

9 • **PMIX_SUCCESS** , indicating that the request was processed by the host environment and
10   returned *success*. Details of the result will be returned in the *results* array

11 • a PMIx error constant indicating either an error in the input or that the request was refused

▼-------------------   Required Attributes   -------------------▼

12 PMIx libraries are not required to directly support any attributes for this function. However, any
13 provided attributes must be passed to the host SMS daemon for processing, and the PMIx library is
14 *required* to add the **PMIX_USERID** and the **PMIX_GRPID** attributes of the client process making
15 the request.

16 ───

17 Host environments that implement support for this operation are required to support the following
18 attributes:

19 **PMIX_JOB_CTRL_ID**   **"pmix.jctrl.id"** (**char***)
20   Provide a string identifier for this request. The user can provide an identifier for the
21   requested operation, thus allowing them to later request status of the operation or to
22   terminate it. The host, therefore, shall track it with the request for future reference.

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

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

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

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

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

33 **PMIX_REGISTER_CLEANUP**   **"pmix.reg.cleanup"** (**char***)
34   Comma-delimited list of files to be removed upon process termination

35 **PMIX_REGISTER_CLEANUP_DIR**   **"pmix.reg.cleanupdir"** (**char***)

1   Comma-delimited list of directories to be removed upon process termination

PMIX_CLEANUP_RECURSIVE **"pmix.clnup.recurse"** (**bool**)
        Recursively cleanup all subdirectories under the specified one(s)

PMIX_CLEANUP_EMPTY **"pmix.clnup.empty"** (**bool**)
        Only remove empty subdirectories

PMIX_CLEANUP_IGNORE **"pmix.clnup.ignore"** (**char***)
        Comma-delimited list of filenames that are not to be removed

PMIX_CLEANUP_LEAVE_TOPDIR **"pmix.clnup.lvtop"** (**bool**)
        When recursively cleaning subdirectories, do not remove the top-level directory (the one
        given in the cleanup request)

▲--▲

▼----------------   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 - the provided request ID must match the
        PMIX_JOB_CTRL_ID provided to a previous call to **PMIx_Job_control** . An ID of
        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**)

1   Indicate that the job can be pre-empted.

▲ .... ▲

2   **Description**
3   Request a job control action. The *targets* array identifies the processes to which the requested job
4   control action is to be applied. A **NULL** value can be used to indicate all processes in the caller's
5   namespace. The use of **PMIX_RANK_WILDCARD** can also be used to indicate that all processes in
6   the given namespace are to be included.

7   The directives are provided as **pmix_info_t** structures in the *directives* array. The callback
8   function provides a *status* to indicate whether or not the request was granted, and to provide some
9   information as to the reason for any denial in the **pmix_info_cbfunc_t** array of
10   **pmix_info_t** structures.

11   7.3.2 **PMIx_Job_control_nb**

12   **Summary**
13   Request a job control action.

14   **Format**

PMIx v2.0

─────────────────────────── C ───────────────────────────

```
15   pmix_status_t
16   PMIx_Job_control_nb(const pmix_proc_t targets[], size_t ntargets,
17                       const pmix_info_t directives[], size_t ndirs,
18                       pmix_info_cbfunc_t cbfunc, void *cbdata)
```

─────────────────────────── C ───────────────────────────

19   **IN   targets**
20   Array of proc structures (array of handles)
21   **IN   ntargets**
22   Number of element in the *targets* array (integer)
23   **IN   directives**
24   Array of info structures (array of handles)
25   **IN   ndirs**
26   Number of element in the *directives* array (integer)
27   **IN   cbfunc**
28   Callback function **pmix_info_cbfunc_t** (function reference)
29   **IN   cbdata**
30   Data to be passed to the callback function (memory reference)

31   Returns one of the following:

32   • **PMIX_SUCCESS**, indicating that the request is being processed by the host environment - result
33   will be returned in the provided *cbfunc*. Note that the library must not invoke the callback
34   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. The user can provide an identifier for the requested operation, thus allowing them to later request status of the operation or to terminate it. The host, therefore, shall track it with the request for future reference.

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.

PMIX_REGISTER_CLEANUP **"pmix.reg.cleanup"** (**char***)
    Comma-delimited list of files to be removed upon process termination

PMIX_REGISTER_CLEANUP_DIR **"pmix.reg.cleanupdir"** (**char***)
    Comma-delimited list of directories to be removed upon process termination

PMIX_CLEANUP_RECURSIVE **"pmix.clnup.recurse"** (**bool**)
    Recursively cleanup all subdirectories under the specified one(s)

PMIX_CLEANUP_EMPTY **"pmix.clnup.empty"** (**bool**)
    Only remove empty subdirectories

PMIX_CLEANUP_IGNORE **"pmix.clnup.ignore"** (**char***)

1   Comma-delimited list of filenames that are not to be removed

2   **PMIX_CLEANUP_LEAVE_TOPDIR** **"pmix.clnup.lvtop"** (**bool**)
3   When recursively cleaning subdirectories, do not remove the top-level directory (the one
4   given in the cleanup request)

▲---▲

▼-----------------   Optional Attributes   -----------------▼

5 The following attributes are optional for host environments that support this operation:

6   **PMIX_JOB_CTRL_CANCEL** **"pmix.jctrl.cancel"** (**char***)
7   Cancel the specified request - the provided request ID must match the
8   **PMIX_JOB_CTRL_ID** provided to a previous call to **PMIx_Job_control** . An ID of
9   **NULL** implies cancel all requests from this requestor.

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

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

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

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

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

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

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

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

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

▲---▲

Description

Non-blocking form of the **PMIx_Job_control** API. 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_WILDCARD** 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.

7.4 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.

7.4.1 PMIx_Process_monitor

Summary

Request that application processes be monitored.

Format

PMIx v3.0

```
pmix_status_t
PMIx_Process_monitor(const pmix_info_t *monitor, pmix_status_t error,
                     const pmix_info_t directives[], size_t ndirs,
                     pmix_info_t *results[], size_t *nresults)
```

IN **monitor**
    info (handle)
IN **error**
    status (integer)

1	**IN** `directives`
2	Array of info structures (array of handles)
3	**IN** `ndirs`
4	Number of elements in the *directives* array (integer)
5	**INOUT** `results`
6	Address where a pointer to an array of **`pmix_info_t`** containing the results of the request
7	can be returned (memory reference)
8	**INOUT** `nresults`
9	Address where the number of elements in *results* can be returned (handle)

10 Returns one of the following:

11 • **PMIX_SUCCESS** , indicating that the request was processed and returned *success*. Details of the
12 result will be returned in the *results* array

13 • a PMIx error constant indicating either an error in the input or that the request was refused

▼------------------ Optional Attributes ------------------▼

14 The following attributes may be implemented by a PMIx library or by the host environment. If
15 supported by the PMIx server library, then the library must not pass the supported attributes to the
16 host environment. All attributes not directly supported by the server library must be passed to the
17 host environment if it supports this operation, and the library is *required* to add the
18 **PMIX_USERID** and the **PMIX_GRPID** attributes of the requesting process:

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

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

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

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

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

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

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

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

35 **PMIX_MONITOR_FILE_ACCESS** `"pmix.monitor.faccess"` (**char***)

1   Monitor time since last access of given file to determine if the application is running.

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

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

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

8   **Description**
9   Request that application processes be monitored via several possible methods. For example, that
10 the server monitor this process for periodic heartbeats as an indication that the process has not
11 become "wedged". When a monitor detects the specified alarm condition, it will generate an event
12 notification using the provided error code and passing along any available relevant information. It
13 is up to the caller to register a corresponding event handler.

14 The *monitor* argument is an attribute indicating the type of monitor being requested. For example,
15 **PMIX_MONITOR_FILE** to indicate that the requestor is asking that a file be monitored.

16 The *error* argument is the status code to be used when generating an event notification alerting that
17 the monitor has been triggered. The range of the notification defaults to
18 **PMIX_RANGE_NAMESPACE** . This can be changed by providing a **PMIX_RANGE** directive.

19 The *directives* argument characterizes the monitoring request (e.g., monitor file size) and frequency
20 of checking to be done

21 ## 7.4.2 `PMIx_Process_monitor_nb`

22 **Summary**
23 Request that application processes be monitored.

24 **Format**

PMIx v2.0   C

```
25  pmix_status_t
26  PMIx_Process_monitor_nb(const pmix_info_t *monitor, pmix_status_t error,
27                          const pmix_info_t directives[], size_t ndirs,
28                          pmix_info_cbfunc_t cbfunc, void *cbdata)
```

1   **IN**   **monitor**
2   info (handle)
3   **IN**   **error**
4   status (integer)
5   **IN**   **directives**
6   Array of info structures (array of handles)
7   **IN**   **ndirs**
8   Number of elements in the *directives* array (integer)
9   **IN**   **cbfunc**
10   Callback function **pmix_info_cbfunc_t** (function reference)
11   **IN**   **cbdata**
12   Data to be passed to the callback function (memory reference)

13 Returns one of the following:

14 • **PMIX_SUCCESS** , indicating that the request is being processed by the host environment - result
15   will be returned in the provided *cbfunc*. Note that the library must not invoke the callback
16   function prior to returning from the API.

17 • **PMIX_OPERATION_SUCCEEDED** , indicating that the request was immediately processed and
18   returned *success* - the *cbfunc* will *not* be called

19 • a PMIx error constant indicating either an error in the input or that the request was immediately
20   processed and failed - the *cbfunc* will *not* be called

▼ -----------------   Optional Attributes   ----------------- ▼

21 The following attributes may be implemented by a PMIx library or by the host environment. If
22 supported by the PMIx server library, then the library must not pass the supported attributes to the
23 host environment. All attributes not directly supported by the server library must be passed to the
24 host environment if it supports this operation, and the library is *required* to add the
25 **PMIX_USERID** and the **PMIX_GRPID** attributes of the requesting process:

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

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

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

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

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

1	**PMIX_MONITOR_HEARTBEAT_DROPS** **"pmix.monitor.bdrop"** (**uint32_t**)
2	Number of heartbeats that can be missed before generating the event.
3	**PMIX_MONITOR_FILE** **"pmix.monitor.fmon"** (**char***)
4	Register to monitor file for signs of life.
5	**PMIX_MONITOR_FILE_SIZE** **"pmix.monitor.fsize"** (**bool**)
6	Monitor size of given file is growing to determine if the application is running.
7	**PMIX_MONITOR_FILE_ACCESS** **"pmix.monitor.faccess"** (**char***)
8	Monitor time since last access of given file to determine if the application is running.
9	**PMIX_MONITOR_FILE_MODIFY** **"pmix.monitor.fmod"** (**char***)
10	Monitor time since last modified of given file to determine if the application is running.
11	**PMIX_MONITOR_FILE_CHECK_TIME** **"pmix.monitor.ftime"** (**uint32_t**)
12	Time in seconds between checking the file.
13	**PMIX_MONITOR_FILE_DROPS** **"pmix.monitor.fdrop"** (**uint32_t**)
14	Number of file checks that can be missed before generating the event.

▲--▲

Description

16 Non-blocking form of the **PMIx_Process_monitor** API. The *cbfunc* function provides a
17 *status* to indicate whether or not the request was granted, and to provide some information as to the
18 reason for any denial in the **pmix_info_cbfunc_t** array of **pmix_info_t** structures.

19 7.4.3 PMIx_Heartbeat

20 Summary
21 Send a heartbeat to the PMIx server library

22 Format

PMIx v2.0

───────────── C ─────────────

23 **PMIx_Heartbeat(void)**

───────────── C ─────────────

24 Description
25 A simplified macro wrapping **PMIx_Process_monitor_nb** that sends a heartbeat to the
26 PMIx server library.

1 7.5   Logging

2   The logging interface supports posting information by applications and SMS elements to persistent
3   storage. This function is *not* intended for output of computational results, but rather for reporting
4   status and saving state information such as inserting computation progress reports into the
5   application's SMS job log or error reports to the local syslog.

6 7.5.1   `PMIx_Log`

7   **Summary**
8   Log data to a data service.

9   **Format**

PMIx v3.0

```
pmix_status_t
PMIx_Log(const pmix_info_t data[], size_t ndata,
         const pmix_info_t directives[], size_t ndirs)
```

13   **IN   data**
14   Array of info structures (array of handles)
15   **IN   ndata**
16   Number of elements in the *data* array (**size_t**)
17   **IN   directives**
18   Array of info structures (array of handles)
19   **IN   ndirs**
20   Number of elements in the *directives* array (**size_t**)

21   Return codes are one of the following:

22   **PMIX_SUCCESS** The logging request was successful.
23   **PMIX_ERR_BAD_PARAM** The logging request contains at least one incorrect entry.
24   **PMIX_ERR_NOT_SUPPORTED** The PMIx implementation or host environment does not
25   support this function.

▼----------------- Required Attributes -----------------▼

26   If the PMIx library does not itself perform this operation, then it is required to pass any attributes
27   provided by the client to the host environment for processing. In addition, it must include the
28   following attributes in the passed *info* array:

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

31   **PMIX_GRPID   "pmix.egid" (uint32_t)**
32   Effective group id.

Host environments or PMIx libraries 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. Will log to global syslog if available, otherwise to local syslog

PMIX_LOG_LOCAL_SYSLOG **"pmix.log.lsys"** (**char***)
    Log data to local syslog. Defaults to **ERROR** priority.

PMIX_LOG_GLOBAL_SYSLOG **"pmix.log.gsys"** (**char***)
    Forward data to system "gateway" and log msg to that syslog Defaults to **ERROR** priority.

PMIX_LOG_SYSLOG_PRI **"pmix.log.syspri"** (**int**)
    Syslog priority level

PMIX_LOG_ONCE **"pmix.log.once"** (**bool**)
    Only log this once with whichever channel can first support it, taking the channels in priority order

▲ --- ▲

▼------------------ Optional Attributes ------------------▼

The following attributes are optional for host environments or PMIx libraries that support this operation:

PMIX_LOG_SOURCE **"pmix.log.source"** (**pmix_proc_t***)
    ID of source of the log request

PMIX_LOG_TIMESTAMP **"pmix.log.tstmp"** (**time_t**)
    Timestamp for log report

PMIX_LOG_GENERATE_TIMESTAMP **"pmix.log.gtstmp"** (**bool**)
    Generate timestamp for log

PMIX_LOG_TAG_OUTPUT **"pmix.log.tag"** (**bool**)
    Label the output stream with the channel name (e.g., "stdout")

PMIX_LOG_TIMESTAMP_OUTPUT **"pmix.log.tsout"** (**bool**)
    Print timestamp in output string

PMIX_LOG_XML_OUTPUT **"pmix.log.xml"** (**bool**)
    Print the output stream in XML format

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.

PMIX_LOG_JOB_RECORD "pmix.log.jrec" (bool)
      Log the provided information to the host environment's job record

PMIX_LOG_GLOBAL_DATASTORE "pmix.log.gstore" (bool)
      Store the log data in a global data store (e.g., database)
▲ - ▲

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.

◆————————————— Advice to users ——————————————◆

It is strongly recommended that the **PMIx_Log** 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. Note that a return of **PMIX_SUCCESS** only denotes that the data
was successfully handed to the appropriate system call (for local channels) or the host environment
and does not indicate receipt at the final destination.

▲   ▲

7.5.2 PMIx_Log_nb

Summary
Log data to a data service.

Format

```
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)
```

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 or PMIx libraries 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. Will log to global syslog if available, otherwise to local syslog

PMIX_LOG_LOCAL_SYSLOG **"pmix.log.lsys"** (**char***)
    Log data to local syslog. Defaults to **ERROR** priority.

PMIX_LOG_GLOBAL_SYSLOG **"pmix.log.gsys"** (**char***)
    Forward data to system "gateway" and log msg to that syslog Defaults to **ERROR** priority.

PMIX_LOG_SYSLOG_PRI **"pmix.log.syspri"** (**int**)
    Syslog priority level

PMIX_LOG_ONCE **"pmix.log.once"** (**bool**)
    Only log this once with whichever channel can first support it, taking the channels in priority order

▲--▲

▼------------------ Optional Attributes ------------------▼

The following attributes are optional for host environments or PMIx libraries that support this operation:

PMIX_LOG_SOURCE **"pmix.log.source"** (**pmix_proc_t***)
    ID of source of the log request

PMIX_LOG_TIMESTAMP **"pmix.log.tstmp"** (**time_t**)
    Timestamp for log report

PMIX_LOG_GENERATE_TIMESTAMP **"pmix.log.gtstmp"** (**bool**)
    Generate timestamp for log

PMIX_LOG_TAG_OUTPUT **"pmix.log.tag"** (**bool**)
    Label the output stream with the channel name (e.g., "stdout")

PMIX_LOG_TIMESTAMP_OUTPUT **"pmix.log.tsout"** (**bool**)
    Print timestamp in output string

PMIX_LOG_XML_OUTPUT **"pmix.log.xml"** (**bool**)
    Print the output stream in XML format

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.

PMIX_LOG_JOB_RECORD   "pmix.log.jrec" (bool)
          Log the provided information to the host environment's job record

PMIX_LOG_GLOBAL_DATASTORE   "pmix.log.gstore" (bool)
          Store the log data in a global data store (e.g., database)

──

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. Note that a return of PMIX_SUCCESS only denotes that the
data was successfully handed to the appropriate system call (for local channels) or the host
environment and does not indicate receipt at the final destination.

Event Notification

1 This chapter defines the PMIx event notification system. These interfaces are designed to support
2 the reporting of events to/from clients and servers, and between library layers within a single
3 process.

8.1 Notification and Management

5 PMIx event notification provides an asynchronous out-of-band mechanism for communicating
6 events between application processes and/or elements of the SMS. Its uses span a wide range that
7 includes fault notification, coordination between multiple programming libraries within a single
8 process, and workflow orchestration for non-synchronous programming models. Events can be
9 divided into two distinct classes:

10 • *Job-specific events* directly relate to a job executing within the session, such as a debugger
11   attachment, process failure within a related job, or events generated by an application process.
12   Events in this category are to be immediately delivered to the PMIx server library for relay to the
13   related local processes.

14 • *Environment events* indirectly relate to a job but do not specifically target the job itself. This
15   category includes SMS-generated events such as Error Check and Correction (ECC) errors,
16   temperature excursions, and other non-job conditions that might directly affect a session's
17   resources, but would never include an event generated by an application process. Note that
18   although these do potentially impact the session's jobs, they are not directly tied to those jobs.
19   Thus, events in this category are to be delivered to the PMIx server library only upon request.

20 Both SMS elements and applications can register for events of either type.

—————◆ Advice to PMIx library implementers ◆—————

21 Race conditions can cause the registration to come after events of possible interest (e.g., a memory
22 ECC event that occurs after start of execution but prior to registration, or an application process
23 generating an event prior to another process registering to receive it). SMS vendors are *requested* to
24 cache environment events for some time to mitigate this situation, but are not *required* to do so.
25 However, PMIx implementers are *required* to cache all events received by the PMIx server library
26 and to deliver them to registering clients in the same order in which they were received

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.

1	Similarly, one event handler can be declared as the *last* handler to be executed in the chain. This
2	handler will *always* be called after all other handlers have executed, regardless of category,
3	provided the incoming event matches both the specified range and event code. Note that this
4	handler will not be called if the chain is terminated by an earlier handler. Only one handler can be
5	designated as *last* — attempts to designate additional handlers as *last* will return an error.
6	Deregistration of the declared *last* handler will re-open the position for subsequent assignment.

———————————————— Advice to users ————————————————

7	Note that the *last* handler is called *after* all registered default handlers that match the specified
8	range of the incoming event unless a handler prior to it terminates the chain. Thus, if the application
9	intends to define a *last* handler, it should ensure that no default handler aborts the process before it.

10	Upon completing its work and prior to returning, each handler *must* call the event handler
11	completion function provided when it was invoked (including a status code plus any information to
12	be passed to later handlers) so that the chain can continue being progressed. PMIx automatically
13	aggregates the status and any results of each handler (as provided in the completion callback) with
14	status from all prior handlers so that each step in the chain has full knowledge of what preceded it.
15	An event handler can terminate all further progress along the chain by passing the
16	**PMIX_EVENT_ACTION_COMPLETE** status to the completion callback function.

17 8.1.1 `PMIx_Register_event_handler`

18	**Summary**
19	Register an event handler

20	**Format**

PMIx v2.0

———————————————————— C ————————————————————

```
pmix_status_t
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 ————————————————————

27	**IN**	**codes**
28		Array of status codes (array of **pmix_status_t**)
29	**IN**	**ncodes**
30		Number of elements in the *codes* array (**size_t**)
31	**IN**	**info**
32		Array of info structures (array of handles)

	IN	ninfo
1		
2		Number of elements in the *info* array (**size_t**)
3	**IN**	**evhdlr**
4		Event handler to be called **pmix_notification_fn_t** (function reference)
5	**IN**	**cbfunc**
6		Callback function **pmix_evhdlr_reg_cbfunc_t** (function reference)
7	**IN**	**cbdata**
8		Data to be passed to the cbfunc callback function (memory reference)

If *cbfunc* is **NULL**, the function call will be treated as a *blocking* call. In this case, the returned status will be either (a) the event handler reference identifier if the value is greater than or equal to zero, or (b) a negative error code indicative of the reason for the failure.

If the *cbfunc* is non-**NULL**, the function call will be treated as a *non-blocking* call and will return the following:

PMIX_SUCCESS indicating that the request has been accepted for processing and the provided callback function will be executed upon completion of the operation. Note that the library must not invoke the callback function prior to returning from the API. The event handler identifier will be returned in the callback

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.

The callback function must not be executed prior to returning from the API, and no events corresponding to this registration may be delivered prior to the completion of the registration callback function (*cbfunc*).

▼------------------- 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***)

1   Put this event handler immediately after the one specified in the **(char*)** value.

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

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

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

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

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

13   _____

14   Host environments that implement support for PMIx event notification are required to support the
15   following attributes:

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

18   **PMIX_EVENT_AFFECTED_PROCS**   **"pmix.evaffected"** (**pmix_data_array_t***)
19   Array of **pmix_proc_t** defining affected processes.
     ▲- ▲

     ▼- - - - - - - - - - - - - - - -   Optional Attributes   - - - - - - - - - - - - - - - -▼

20   Host environments that support PMIx event notification *may* offer notifications for environmental
21   events impacting the job and for SMS events relating to the job. The following attributes are
22   optional for host environments that suppport this operation:

23   **PMIX_EVENT_TERMINATE_SESSION**   **"pmix.evterm.sess"** (**bool**)
24   The RM intends to terminate this session.

25   **PMIX_EVENT_TERMINATE_JOB**   **"pmix.evterm.job"** (**bool**)
26   The RM intends to terminate this job.

27   **PMIX_EVENT_TERMINATE_NODE**   **"pmix.evterm.node"** (**bool**)
28   The RM intends to terminate all processes on this node.

29   **PMIX_EVENT_TERMINATE_PROC**   **"pmix.evterm.proc"** (**bool**)
30   The RM intends to terminate just this process.

31   **PMIX_EVENT_ACTION_TIMEOUT**   **"pmix.evtimeout"** (**int**)
32   The time in seconds before the RM will execute error response.

33   **PMIX_EVENT_SILENT_TERMINATION**   **"pmix.evsilentterm"** (**bool**)

1   Do not generate an event when this job normally terminates.

▲ --- ▲

2   **Description**
3   Register an event handler to report events. Note that the codes being registered do *not* need to be
4   PMIx error constants — any integer value can be registered. This allows for registration of
5   non-PMIx events such as those defined by a particular SMS vendor or by an application itself.

━━━━━━━━━━━━━━━━━━━━━━━━━━ Advice to users ━━━━━━━━━━━━━━━━━━━━━━━━

6   In order to avoid potential conflicts, users are advised to only define codes that lie outside the range
7   of the PMIx standard's error codes. Thus, SMS vendors and application developers should
8   constrain their definitions to positive values or negative values beyond the
9   **PMIX_EXTERNAL_ERR_BASE** boundary.

▲   ▲

━━━━━━━━━━━━━━━━━━━━━━━━━━ Advice to users ━━━━━━━━━━━━━━━━━━━━━━━━

10   As previously stated, upon completing its work, and prior to returning, each handler *must* call the
11   event handler completion function provided when it was invoked (including a status code plus any
12   information to be passed to later handlers) so that the chain can continue being progressed. An
13   event handler can terminate all further progress along the chain by passing the
14   **PMIX_EVENT_ACTION_COMPLETE** status to the completion callback function. Note that the
15   parameters passed to the event handler (e.g., the *info* and *results* arrays) will cease to be valid once
16   the completion function has been called - thus, any information in the incoming parameters that
17   will be referenced following the call to the completion function must be copied.

▲   ▲

18   8.1.2   PMIx_Deregister_event_handler

19   **Summary**
20   Deregister an event handler.

Format

```
pmix_status_t
PMIx_Deregister_event_handler(size_t evhdlr_ref,
                              pmix_op_cbfunc_t cbfunc,
                              void *cbdata);
```

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)

If *cbfunc* is **NULL**, the function will be treated as a *blocking* call and the result of the operation
returned in the status code.

If *cbfunc* is non-**NULL**, the function will be treated as a *non-blocking* call and return one of the
following:

- **PMIX_SUCCESS** , indicating that the request is being processed - 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

The returned status code will be one of 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.

Description
Deregister an event handler. Note that no events corresponding to the referenced registration may
be delivered following completion of the deregistration operation (either return from the API with
PMIX_OPERATION_SUCCEEDED or execution of the *cbfunc*).

8.1.3   **PMIx_Notify_event**

Summary
Report an event for notification via any registered event handler.

1   **Format**
————————————————————————————— C —————————————————————————————

2   `pmix_status_t`
3   `PMIx_Notify_event(pmix_status_t status,`
4   `const pmix_proc_t *source,`
5   `pmix_data_range_t range,`
6   `pmix_info_t info[], size_t ninfo,`
7   `pmix_op_cbfunc_t cbfunc, void *cbdata);`
————————————————————————————— C —————————————————————————————

8   **IN   status**
9   Status code of the event (**`pmix_status_t`**)
10   **IN   source**
11   Pointer to a **`pmix_proc_t`** identifying the original reporter of the event (handle)
12   **IN   range**
13   Range across which this notification shall be delivered (**`pmix_data_range_t`**)
14   **IN   info**
15   Array of **`pmix_info_t`** structures containing any further info provided by the originator of
16   the event (array of handles)
17   **IN   ninfo**
18   Number of elements in the *info* array (**`size_t`**)
19   **IN   cbfunc**
20   Callback function to be executed upon completion of operation **`pmix_op_cbfunc_t`**
21   (function reference)
22   **IN   cbdata**
23   Data to be passed to the cbfunc callback function (memory reference)

24   If *cbfunc* is **`NULL`**, the function will be treated as a *blocking* call and the result of the operation
25   returned in the status code.

26   If *cbfunc* is non-**`NULL`**, the function will be treated as a *non-blocking* call and return one of the
27   following:

28   **`PMIX_SUCCESS`** The notification request is valid and is being processed. The callback function
29   will be called when the process-local operation is complete and will provide the resulting
30   status of that operation. Note that this does *not* reflect the success or failure of delivering the
31   event to any recipients. The callback function must not be executed prior to returning from the
32   API.
33   **`PMIX_OPERATION_SUCCEEDED`** , indicating that the request was immediately processed and
34   returned *success* - the *cbfunc* will *not* be called
35   **`PMIX_ERR_BAD_PARAM`** The request contains at least one incorrect entry that prevents it from
36   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.

1   The callback function will be called upon completion of the **notify_event** function's actions.
2   At that time, any messages required for executing the operation (e.g., to send the notification to the
3   local PMIx server) will have been queued, but may not yet have been transmitted. The caller is
4   required to maintain the input data until the callback function has been executed — the sole purpose
5   of the callback function is to indicate when the input data is no longer required.

Data Packing and Unpacking

1 PMIx intentionally does not include support for internode communications in the standard, instead
2 relying on its host SMS environment to transfer any needed data and/or requests between nodes.
3 These operations frequently involve PMIx-defined public data structures that include binary data.
4 Many HPC clusters are homogeneous, and so transferring the structures can be done rather simply.
5 However, greater effort is required in heterogeneous environments to ensure binary data is correctly
6 transferred. PMIx buffer manipulation functions are provided for this purpose via standardized
7 interfaces to ease adoption.

9.1 Data Buffer Type

9 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;
```

▲ ──────────────── C ────────────────── ▲

9.2 Support Macros

PMIx provides a set of convenience macros for creating, initiating, and releasing data buffers.

9.2.1 `PMIX_DATA_BUFFER_CREATE`

Summary

Allocate memory for a **`pmix_data_buffer_t`** object and initialize it

Format

PMIx v2.0
```
PMIX_DATA_BUFFER_CREATE(buffer);
```

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

9.2.2 `PMIX_DATA_BUFFER_RELEASE`

Summary

Free a **`pmix_data_buffer_t`** object and the data it contains

Format

PMIx v2.0
```
PMIX_DATA_BUFFER_RELEASE(buffer);
```

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

9.2.3 `PMIX_DATA_BUFFER_CONSTRUCT`

Summary

Initialize a statically declared **`pmix_data_buffer_t`** object

Format

PMIx v2.0

`PMIX_DATA_BUFFER_CONSTRUCT(buffer);`

IN   buffer
Pointer to the allocated **`pmix_data_buffer_t`** that is to be initialized (handle)

Description
Initialize a pre-allocated buffer object

9.2.4 `PMIX_DATA_BUFFER_DESTRUCT`

Summary
Release the data contained in a **`pmix_data_buffer_t`** object

Format

PMIx v2.0

`PMIX_DATA_BUFFER_DESTRUCT(buffer);`

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

9.2.5 `PMIX_DATA_BUFFER_LOAD`

Summary
Load a blob into a **`pmix_data_buffer_t`** object

Format

PMIx v2.0

`PMIX_DATA_BUFFER_LOAD(buffer, data, size);`

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`**
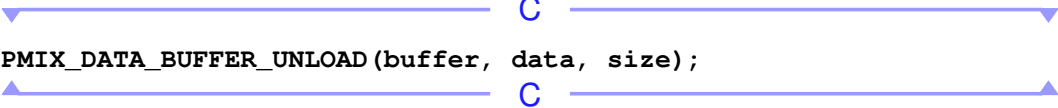
Description

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.

9.2.6 `PMIX_DATA_BUFFER_UNLOAD`

Summary

Unload the data from a **pmix_data_buffer_t** object

Format

PMIx v2.0

```
PMIX_DATA_BUFFER_UNLOAD(buffer, data, size);
```

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.

9.3 General Routines

The following routines are provided to support internode transfers in heterogeneous environments.

9.3.1 `PMIx_Data_pack`

Summary

Pack one or more values of a specified type into a buffer, usually for transmission to another process

Format

—————————————————————— 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**)

Returns one of the following:

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

1 will do its best to deal with heterogeneity issues between the packer and unpacker in such cases.
2 Sending a number larger than can be handled by the recipient will return an error code (generated
3 upon unpacking) — the error cannot be detected during packing.

4 The namespace of the intended recipient of the packed buffer (i.e., the process that will be
5 unpacking it) is used solely to resolve any data type differences between PMIx versions. The
6 recipient must, therefore, be known to the user prior to calling the pack function so that the PMIx
7 library is aware of the version the recipient is using. Note that all processes in a given namespace
8 are *required* to use the same PMIx version — thus, the caller must only know at least one process
9 from the target's namespace.

9.3.2 `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`**)

1 **IN   type**
2   The type of the data to be unpacked — must be one of the PMIx defined data types (
3   **pmix_data_type_t**)

4 Returns one of the following:

5 **PMIX_SUCCESS** The data has been unpacked as requested
6 **PMIX_ERR_NOT_SUPPORTED** The PMIx implementation does not support this function.
7 **PMIX_ERR_BAD_PARAM** The provided buffer or dest is **NULL**
8 **PMIX_ERR_UNKNOWN_DATA_TYPE** The specified data type is not known to this
9   implementation
10 **PMIX_ERR_OUT_OF_RESOURCE** Not enough memory to support the operation
11 **PMIX_ERROR** General error

12 **Description**
13 The unpack function unpacks the next value (or values) of a specified type from the given buffer.
14 The buffer must have already been initialized via an **PMIX_DATA_BUFFER_CREATE** or
15 **PMIX_DATA_BUFFER_CONSTRUCT** call (and assumedly filled with some data) — otherwise,
16 the unpack_value function will return an error. Providing an unsupported type flag will likewise be
17 reported as an error, as will specifying a data type that *does not* match the type of the next item in
18 the buffer. An attempt to read beyond the end of the stored data held in the buffer will also return an
19 error.

20 NOTE: it is possible for the buffer to be corrupted and that PMIx will *think* there is a proper
21 variable type at the beginning of an unpack region — but that the value is bogus (e.g., just a byte
22 field in a string array that so happens to have a value that matches the specified data type flag).
23 Therefore, the data type error check is *not* completely safe.

24 Unpacking values is a "nondestructive" process — i.e., the values are not removed from the buffer.
25 It is therefore possible for the caller to re-unpack a value from the same buffer by resetting the
26 unpack_ptr.

27 Warning: The caller is responsible for providing adequate memory storage for the requested data.
28 The user must provide a parameter indicating the maximum number of values that can be unpacked
29 into the allocated memory. If more values exist in the buffer than can fit into the memory storage,
30 then the function will unpack what it can fit into that location and return an error code indicating
31 that the buffer was only partially unpacked.

32 Note that any data that was not hard type cast (i.e., not type cast to a specific size) when packed may
33 lose precision when unpacked by a non-homogeneous recipient. PMIx will do its best to deal with
34 heterogeneity issues between the packer and unpacker in such cases. Sending a number larger than
35 can be handled by the recipient will return an error code generated upon unpacking — these errors
36 cannot be detected during packing.

37 The namespace of the process that packed the buffer is used solely to resolve any data type
38 differences between PMIx versions. The packer must, therefore, be known to the user prior to
39 calling the pack function so that the PMIx library is aware of the version the packer is using. Note

1   that all processes in a given namespace are *required* to use the same PMIx version — thus, the
2   caller must only know at least one process from the packer's namespace.

3 ## 9.3.3   `PMIx_Data_copy`

4 **Summary**
5 Copy a data value from one location to another.

6 **Format**

PMIx v2.0

─────────────────────── C ───────────────────────

```
7 pmix_status_t
8 PMIx_Data_copy(void **dest, void *src,
9                 pmix_data_type_t type);
```

─────────────────────── C ───────────────────────

10 **IN   dest**
11   The address of a pointer into which the address of the resulting data is to be stored. (**void****)
12 **IN   src**
13   A pointer to the memory location from which the data is to be copied (handle)
14 **IN   type**
15   The type of the data to be copied — must be one of the PMIx defined data types. (
16   **pmix_data_type_t**)

17 Returns one of the following:

18   **PMIX_SUCCESS** The data has been copied as requested
19   **PMIX_ERR_NOT_SUPPORTED** The PMIx implementation does not support this function.
20   **PMIX_ERR_BAD_PARAM** The provided src or dest is **NULL**
21   **PMIX_ERR_UNKNOWN_DATA_TYPE** The specified data type is not known to this
22   implementation
23   **PMIX_ERR_OUT_OF_RESOURCE** Not enough memory to support the operation
24   **PMIX_ERROR** General error

25 **Description**
26 Since registered data types can be complex structures, the system needs some way to know how to
27 copy the data from one location to another (e.g., for storage in the registry). This function, which
28 can call other copy functions to build up complex data types, defines the method for making a copy
29 of the specified data type.

30 ## 9.3.4   `PMIx_Data_print`

31 **Summary**
32 Pretty-print a data value.

Format

▼ ———————————————— 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)

Returns one of the following:

  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.

9.3.5 **PMIx_Data_copy_payload**

Summary
Copy a payload from one buffer to another

Format

```
1    pmix_status_t
2    PMIx_Data_copy_payload(pmix_data_buffer_t *dest,
3                               pmix_data_buffer_t *src);
```

4   **IN   dest**
5   Pointer to the destination **pmix_data_buffer_t** (handle)
6   **IN   src**
7   Pointer to the source **pmix_data_buffer_t** (handle)

8   Returns one of the following:

9   **PMIX_SUCCESS** The data has been copied as requested
10   **PMIX_ERR_BAD_PARAM** The src and dest **pmix_data_buffer_t** types do not match
11   **PMIX_ERR_NOT_SUPPORTED** The PMIx implementation does not support this function.

Description

13   This function will append a copy of the payload in one buffer into another buffer. Note that this is
14   *not* a destructive procedure — the source buffer's payload will remain intact, as will any pre-existing
15   payload in the destination's buffer. Only the unpacked portion of the source payload will be copied.

CHAPTER 10

Security

---

1 PMIx utilizes a multi-layered approach toward security that differs for client versus tool processes.
2 *Client* processes (i.e., processes started by the host environment) must be preregistered with the
3 PMIx server library via the **PMIx_server_register_client** API before they are spawned.
4 This API requires that you pass the expected uid/gid of the client process.

5 When the client attempts to connect to the PMIx server, the server uses available standard
6 Operating System (OS) methods to determine the effective uid/gid of the process requesting the
7 connection. PMIx implementations shall not rely on any values reported by the client process itself
8 as that would be unsafe. The effective uid/gid reported by the OS is compared to the values
9 provided by the host during registration - if they don't match, the PMIx server is required to drop
10 the connection request. This ensures that the PMIx server does not allow connection from a client
11 that doesn't at least meet some minimal security requirement.

12 Once the requesting client passes the initial test, the PMIx server can, at the choice of the
13 implementor, perform additional security checks. This may involve a variety of methods such as
14 exchange of a system-provided key or credential. At the conclusion of that process, the PMIx server
15 reports the client connection request to the host via the
16 **pmix_server_client_connected_fn_t** interface. The host may then perform any
17 additional checks and operations before responding with either **PMIX_SUCCESS** to indicate that
18 the connection is approved, or a PMIx error constant indicating that the connection request is
19 refused. In this latter case, the PMIx server is required to drop the connection.

20 Tools started by the host environment are classed as a subgroup of client processes and follow the
21 client process procedure. However, tools that are not started by the host environment must be
22 handled differently as registration information is not available prior to the connection request. In
23 these cases, the PMIx server library is required to use available standard OS methods to get the
24 effective uid/gid and report them upwards as part of invoking the
25 **pmix_server_tool_connection_fn_t** interface, deferring initial security screening to
26 the host. It is recognized that this may represent a security risk - for this reason, PMIx server
27 libraries must not enable tool connections by default. Instead, the host has to explicitly enable them
28 via the **PMIX_SERVER_TOOL_SUPPORT** attribute, thus recognizing the associated risk. Once
29 the host has completed its authentication procedure, it again informs the PMIx server of the result.

30 Applications and tools often interact with the host environment in ways that require security beyond
31 just verifying the user's identity - e.g., access to that user's relevant authorizations. This is
32 particularly important when tools connect directly to a system-level PMIx server that may be
33 operating at a privileged level. A variety of system management software packages provide
34 authorization services, but the lack of standardized interfaces makes portability problematic.

1 This section defines two PMIx client-side APIs for this purpose. These are most likely to be used
2 by user-space applications/tools, but are not restricted to that realm.

10.1 Obtaining Credentials

4 The API for obtaining a credential is a non-blocking operation since the host environment may have
5 to contact a remote credential service. The definition takes into account the potential that the
6 returned credential could be sent via some mechanism to another application that resides in an
7 environment using a different security mechanism. Thus, provision is made for the system to return
8 additional information (e.g., the identity of the issuing agent) outside of the credential itself and
9 visible to the application.

10.1.1 PMIx_Get_credential

Summary

Request a credential from the PMIx server library or the host environment

Format

PMIx v3.0

```
pmix_status_t
PMIx_Get_credential(const pmix_info_t info[], size_t ninfo,
                    pmix_byte_object_t *credential)
```

IN   info
   Array of **pmix_info_t** structures (array of handles)
IN   ninfo
   Number of elements in the *info* array (**size_t**)
IN   credential
   Address of a **pmix_byte_object_t** within which to return credential (handle)

Returns one of the following:

- **PMIX_SUCCESS** , indicating that the credential has been returned in the provided
  pmix_byte_object_t

- a PMIx error constant indicating either an error in the input or that the request is unsupported

1 PMIx libraries that choose not to support this operation *must* return
2 **PMIX_ERR_NOT_SUPPORTED** when the function is called.

3 There are no required attributes for this API. Note that implementations may choose to internally
4 execute integration for some security environments (e.g., directly contacting a *munge* server).

5 Implementations that support the operation but cannot directly process the client's request must
6 pass any attributes that are provided by the client to the host environment for processing. In
7 addition, the following attributes are required to be included in the *info* array passed from the PMIx
8 library to the host environment:

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

11 **PMIX_GRPID** **"pmix.egid"** (**uint32_t**)
12   Effective group id.

------------------- Optional Attributes -------------------

13 The following attributes are optional for host environments that support this operation:

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

--------- Advice to PMIx library implementers ---------

18 We recommend that implementation of the **PMIX_TIMEOUT** attribute be left to the host
19 environment due to race condition considerations between completion of the operation versus
20 internal timeout in the PMIx server library. Implementers that choose to support **PMIX_TIMEOUT**
21 directly in the PMIx server library must take care to resolve the race condition and should avoid
22 passing **PMIX_TIMEOUT** to the host environment so that multiple competing timeouts are not
23 created.

24 **Description**
25 Request a credential from the PMIx server library or the host environment

26 10.1.2 **PMIx_Get_credential_nb**

27 **Summary**
28 Request a credential from the PMIx server library or the host environment

Format

```
                                         C
pmix_status_t
PMIx_Get_credential_nb(const pmix_info_t info[], size_t ninfo,
                       pmix_credential_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 (**size_t**)
IN   cbfunc
    Callback function to return credential (**pmix_credential_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 has been communicated to the local PMIx server - result will be returned in the provided *cbfunc*

- a PMIx error constant indicating either an error in the input or that the request is unsupported - the *cbfunc* will *not* be called

▼----------------- Required Attributes -----------------▼

PMIx libraries that choose not to support this operation *must* return
PMIX_ERR_NOT_SUPPORTED when the function is called.

There are no required attributes for this API. Note that implementations may choose to internally execute integration for some security environments (e.g., directly contacting a *munge* server).

Implementations that support the operation but cannot directly process the client's request must pass any attributes that are provided by the client to the host environment for processing. In addition, the following attributes are required to be included in the *info* array passed from the PMIx library to the host environment:

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

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

▲--▲

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
Request a credential from the PMIx server library or the host environment

10.2 Validating Credentials

The API for validating a credential is a non-blocking operation since the host environment may
have to contact a remote credential service. Provision is made for the system to return additional
information regarding possible authorization limitations beyond simple authentication.

10.2.1 `PMIx_Validate_credential`

Summary
Request validation of a credential by the PMIx server library or the host environment

Format

─────────────────────────── C ───────────────────────────

```
pmix_status_t
PMIx_Validate_credential(const pmix_byte_object_t *cred,
                         const pmix_info_t info[], size_t ninfo,
                         pmix_info_t **results, size_t *nresults)
```

─────────────────────────── C ───────────────────────────

IN   cred
   Pointer to **pmix_byte_object_t** containing the credential (handle)
IN   info
   Array of **pmix_info_t** structures (array of handles)
IN   ninfo
   Number of elements in the *info* array (**size_t**)
INOUT results
   Address where a pointer to an array of **pmix_info_t** containing the results of the request
   can be returned (memory reference)
INOUT nresults
   Address where the number of elements in *results* can be returned (handle)

Returns one of the following:

- **PMIX_SUCCESS** , indicating that the request was processed and returned *success*. Details of the
  result will be returned in the *results* array

- a PMIx error constant indicating either an error in the input or that the request was refused

▼------------------ Required Attributes ------------------▼

PMIx libraries that choose not to support this operation *must* return
PMIX_ERR_NOT_SUPPORTED when the function is called.

There are no required attributes for this API. Note that implementations may choose to internally
execute integration for some security environments (e.g., directly contacting a *munge* server).

Implementations that support the operation but cannot directly process the client's request must
pass any attributes that are provided by the client to the host environment for processing. In
addition, the following attributes are required to be included in the *info* array passed from the PMIx
library to the host environment:

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

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

▲--▲

1   The following attributes are optional for host environments that support this operation:

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

▲--▲

6   We recommend that implementation of the **PMIX_TIMEOUT** attribute be left to the host
7   environment due to race condition considerations between completion of the operation versus
8   internal timeout in the PMIx server library. Implementers that choose to support **PMIX_TIMEOUT**
9   directly in the PMIx server library must take care to resolve the race condition and should avoid
10   passing **PMIX_TIMEOUT** to the host environment so that multiple competing timeouts are not
11   created.

▲

12   **Description**
13   Request validation of a credential by the PMIx server library or the host environment.

10.2.2 `PMIx_Validate_credential_nb`

15   **Summary**
16   Request validation of a credential by the PMIx server library or the host environment

Format

```
pmix_status_t
PMIx_Validate_credential_nb(const pmix_byte_object_t *cred,
                             const pmix_info_t info[], size_t ninfo,
                             pmix_validation_cbfunc_t cbfunc,
                             void *cbdata)
```

IN   cred
     Pointer to **pmix_byte_object_t** containing the credential (handle)
IN   info
     Array of **pmix_info_t** structures (array of handles)
IN   ninfo
     Number of elements in the *info* array (**size_t**)
IN   cbfunc
     Callback function to return result (**pmix_validation_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 has been communicated to the local PMIx server - result will be returned in the provided *cbfunc*

- a PMIx error constant indicating either an error in the input or that the request is unsupported - the *cbfunc* will *not* be called

▼- - - - - - - - - - - - - - - - - - Required Attributes - - - - - - - - - - - - - - - - - -▼

PMIx libraries that choose not to support this operation *must* return
PMIX_ERR_NOT_SUPPORTED when the function is called.

There are no required attributes for this API. Note that implementations may choose to internally execute integration for some security environments (e.g., directly contacting a *munge* server).

Implementations that support the operation but cannot directly process the client's request must pass any attributes that are provided by the client to the host environment for processing. In addition, the following attributes are required to be included in the *info* array passed from the PMIx library to the host environment:

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

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

▲- -▲

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
Request validation of a credential by the PMIx server library or the host environment.

Server-Specific Interfaces

1   The RM daemon that hosts the PMIx server library interacts with that library in two distinct
2   manners. First, PMIx provides a set of APIs by which the host can request specific services from its
3   library. This includes generating regular expressions, registering information to be passed to client
4   processes, and requesting information on behalf of a remote process. Note that the host always has
5   access to all PMIx client APIs - the functions listed below are in addition to those available to a
6   PMIx client.

7   Second, the host can provide a set of callback functions by which the PMIx server library can pass
8   requests upward for servicing by the host. These include notifications of client connection and
9   finalize, as well as requests by clients for information and/or services that the PMIx server library
10 does not itself provide.

11.1 Server Initialization and Finalization

12 The PMIx APIs may only be used between the completion of the initialization function and the start
13 of the finalization function, unless otherwise noted. The initialization and finalization functions are
14 paired, and the initialized regions defined by them must not overlap.

---------------------------- Advice to users ----------------------------

15 Server initialization includes setting up the infrastructure to support local clients, Therefore, server
16 initialization will likely result in additional overhead and an increased memory footprint than client
17 initialization alone.

11.1.1 `PMIx_server_init`

19 **Summary**
20 Initialize the PMIx server.

Format

```
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.

▲--▲

1   The following attributes are optional for implementers of PMIx libraries:

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

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

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

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

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

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

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

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

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

32 **PMIX_SERVER_REMOTE_CONNECTIONS** **"pmix.srvr.remote"** (**bool**)
33   Allow connections from remote tools. Forces the PMIx server to not exclusively use
34   loopback device.   If the library supports connections from remote tools, this attribute may
35   be supported for enabling or disabling it.

36 **PMIX_EVENT_BASE** **"pmix.evbase"** (**struct event_base ***)

1   Pointer to libevent[1] **event_base** to use in place of the internal progress thread.

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

▲--▲

6   **Description**
7   Initialize the PMIx server support library, and provide a pointer to a **pmix_server_module_t**
8   structure containing the caller's callback functions. The array of **pmix_info_t** structs is used to
9   pass additional info that may be required by the server when initializing. For example, it may
10   include the **PMIX_SERVER_TOOL_SUPPORT** attribute, thereby indicating that the daemon is
11   willing to accept connection requests from PMIx tools.

─────────────────── Advice to PMIx server hosts ───────────────────

12   Providing a value of **NULL** for the *module* argument is permitted, as is passing an empty *module*
13   structure. Doing so indicates that the host environment will not provide support for multi-node
14   operations such as **PMIx_Fence** , but does intend to support local clients access to information.

11.1.2 `PMIx_server_finalize`

16   **Summary**
17   Finalize the PMIx server library.

18   **Format**

PMIx v1.0   ▼────────────────── C ──────────────────▼

19   **pmix_status_t**
20   **PMIx_server_finalize(void)**

▲────────────────── C ──────────────────▲

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

22   **Description**
23   Finalize the PMIx server support library, terminating all connections to the attached tools and any
24   local clients. All allocated resources are released.

11.2 Server Support Functions

26   The following APIs allow the RM daemon that hosts the PMIx server library to request specific
27   services from the PMIx library.

───────────────────────────────

[1]http://libevent.org/

11.2.1 `PMIx_generate_regex`

Summary

Generate a compressed representation of the input string.

Format

PMIx v1.0

─────────────────────────────── C ───────────────────────────────

```
pmix_status_t
PMIx_generate_regex(const char *input, char **output)
```
─────────────────────────────── C ───────────────────────────────

IN   input
    String to process (string)
OUT   output
    Compressed representation of *input* (array of bytes)

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

Description

Given a comma-separated list of *input* values, generate a reduced size representation of the input that can be passed down to the PMIx server library's **PMIx_server_register_nspace** API for parsing. The order of the individual values in the *input* string is preserved across the operation. The caller is responsible for releasing the returned data.

The precise compressed representations will be implementation specific. However, all PMIx implementations are required to include a **NULL**-terminated string in the output representation that can be printed for diagnostic purposes.

──────────────────── Advice to PMIx server hosts ────────────────────

The returned representation may be an arbitrary array of bytes as opposed to a valid NULL-terminated string. However, the method used to generate the representation shall be identified with a colon-delimited string at the beginning of the output. For example, an output starting with **"pmix:\0"** might indicate that the representation is a PMIx-defined regular expression represented as a **NULL**-terminated string following the **"pmix:\0"** prefix. In contrast, an output starting with **"blob:\0"** might indicate a compressed binary array follows the prefix.

Communicating the resulting output should be done by first packing the returned expression using the **PMIx_Data_pack** , declaring the input to be of type **PMIX_REGEX** , and then obtaining the resulting blob to be communicated using the **PMIX_DATA_BUFFER_UNLOAD** macro. The reciprocal method can be used on the remote end prior to passing the regex into **PMIx_server_register_nspace** . The pack/unpack routines will ensure proper handling of the data based on the regex prefix.

11.2.2 `PMIx_generate_ppn`

Summary

Generate a compressed representation of the input identifying the processes on each node.

Format

PMIx v1.0

———————————————————— C ————————————————————

```
pmix_status_t PMIx_generate_ppn(const char *input, char **ppn)
```

———————————————————— C ————————————————————

IN `input`
  String to process (string)
OUT `ppn`
  Compressed representation of *input* (array of bytes)

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

Description

The input shall consist of a semicolon-separated list of ranges representing the ranks of processes on each node of the job - e.g., "1-4;2-5;8,10,11,12;6,7,9". Each field of the input must correspond to the node name provided at that position in the input to **PMIx_generate_regex**. Thus, in the example, ranks 1-4 would be located on the first node of the comma-separated list of names provided to **PMIx_generate_regex**, and ranks 2-5 would be on the second name in the list.

———————————————————— Advice to PMIx server hosts ————————————————————

The returned representation may be an arbitrary array of bytes as opposed to a valid NULL-terminated string. However, the method used to generate the representation shall be identified with a colon-delimited string at the beginning of the output. For example, an output starting with **`"pmix:"`** indicates that the representation is a PMIx-defined regular expression represented as a NULL-terminated string. In contrast, an output starting with **`"blob:\0size=1234:"`** is a compressed binary array.

Communicating the resulting output should be done by first packing the returned expression using the **PMIx_Data_pack**, declaring the input to be of type **PMIX_REGEX**, and then obtaining the blob to be communicated using the **PMIX_DATA_BUFFER_UNLOAD** macro. The pack/unpack routines will ensure proper handling of the data based on the regex prefix.

11.2.3 `PMIx_server_register_nspace`

Summary

Setup the data about a particular namespace.

Format

```
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)
```

IN   nspace
Character array of maximum size **PMIX_MAX_NSLEN** containing the namespace identifier (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:

1   – **PMIX_UNIV_SIZE** **"pmix.univ.size"** (**uint32_t**)
2   Number of allocated slots in a session - each slot may or may not be occupied by an
3   executing process. Note that this attribute is the equivalent to the combination of
4   **PMIX_SESSION_INFO_ARRAY** with the **PMIX_MAX_PROCS** entry in the array - it
5   is included in the Standard for historical reasons.

6   • for the given namespace:

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

9   – **PMIX_JOB_SIZE** **"pmix.job.size"** (**uint32_t**)
10   Total number of processes in this job across all contained applications. Note that this
11   value can be different from **PMIX_MAX_PROCS** . For example, users may choose to
12   subdivide an allocation (running several jobs in parallel within it), and dynamic
13   programming models may support adding and removing processes from a running **job**
14   on-they-fly. In the latter case, PMIx events must be used to notify processes within the
15   job that the job size has changed.

16   – **PMIX_MAX_PROCS** **"pmix.max.size"** (**uint32_t**)
17   Maximum number of processes that can be executed in this context (session,
18   namespace, application, or node). Typically, this is a constraint imposed by a scheduler
19   or by user settings in a hostfile or other resource description.

20   – **PMIX_NODE_MAP** **"pmix.nmap"** (**char***)
21   Regular expression of nodes - see 11.2.3.1 for an explanation of its generation.

22   – **PMIX_PROC_MAP** **"pmix.pmap"** (**char***)
23   Regular expression describing processes on each node - see 11.2.3.1 for an explanation
24   of its generation.

25   • for its own node:

26   – **PMIX_LOCAL_SIZE** **"pmix.local.size"** (**uint32_t**)
27   Number of processes in this job or application on this node.

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

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

34   • for each process in the given namespace:

35   – **PMIX_RANK** **"pmix.rank"** (**pmix_rank_t**)
36   Process rank within the job.

37   – **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 expressed as the node's index (beginning at zero) in an array of nodes within the active session. The value must be unique and directly correlate to the **PMIX_HOSTNAME** of the node - i.e., users can interchangeably reference the same location using either the **PMIX_HOSTNAME** or corresponding **PMIX_NODEID** .

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

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

- 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 (e.g., where a specified process is running, or a given device is located).

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.

The following optional attributes may be provided by the host environment to identify the programming model (as specified by the user) being executed within the namespace. The PMIx server library may utilize this information to customize the environment to fit that model (e.g., adding environmental variables specified by the corresponding standard for that model):

- **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")

▲ - ▲

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.

1 The number of local processes for any given namespace is generally fixed at the time of application
2 launch. Calls to **PMIx_Spawn** result in processes launched in their own namespace, not that of
3 their parent. However, it is possible for processes to *migrate* to another node via a call to
4 **PMIx_Job_control_nb**, thus resulting in a change to the number of local processes on both
5 the initial node and the node to which the process moved. It is therefore critical that applications
6 not migrate processes without first ensuring that PMIx-based collective operations are not in
7 progress, and that no such operations be initiated until process migration has completed.

11.2.3.1 Assembling the registration information

9 The following description is not intended to represent the actual layout of information in a given
10 PMIx library. Instead, it is describes how information provided in the *info* parameter of the
11 **PMIx_server_register_nspace** shall be organized for proper processing by a PMIx server
12 library. The ordering of the various information elements is arbitrary - they are presented in a
13 top-down hierarchical form solely for clarity in reading.

14 Creating the *info* array of data requires knowing in advance the number of elements required for the
15 array. This can be difficult to compute and somewhat fragile in practice. One method for resolving
16 the problem is to create a linked list of objects, each containing a single **pmix_info_t** structure.
17 Allocation and manipulation of the list can then be accomplished using existing standard methods.
18 Upon completion, the final *info* array can be allocated based on the number of elements on the list,
19 and then the values in the list object **pmix_info_t** structures transferred to the corresponding
20 array element utilizing the **PMIX_INFO_XFER** macro.

21 A common building block used in several areas is the construction of a regular expression
22 identifying the nodes involved in that area - e.g., the nodes in a **session** or **job**. PMIx provides
23 several tools to facilitate this operation, beginning by constructing an argv-like array of node
24 names. This array is then passed to the **PMIx_generate_regex** function to create a regular
25 expression parseable by the PMIx server library, as shown below:

```
1    char **nodes = NULL;
2    char *nodelist;
3    char *regex;
4    size_t n;
5    pmix_status_t rc;
6    pmix_info_t info;
7
8    /* loop over an array of nodes, adding each
9     * name to the array */
10   for (n=0; n < num_nodes; n++)
11       /* filter the nodes to ignore those not included
12        * in the target range (session, job, etc.). In
13        * this example, all nodes are accepted */
14       PMIX_ARGV_APPEND(&nodes, node[n]->name);
15
16
17   /* join into a comma-delimited string */
18   nodelist = PMIX_ARGV_JOIN(nodes, ',');
19
20   /* release the array */
21   PMIX_ARGV_FREE(nodes);
22
23   /* generate regex */
24   rc = PMIx_generate_regex(nodelist, &regex);
25
26   /* release list */
27   free(nodelist);
28
29   /* pass the regex as the value to the PMIX_NODE_MAP key */
30   PMIX_INFO_LOAD(&info, PMIX_NODE_MAP, regex, PMIX_STRING);
31   /* release the regex */
32   free(regex);
33
```

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** :

```
1          char **ndppn;
2          char rank[30];
3          char **ppnarray = NULL;
4          char *ppn;
5          char *localranks;
6          char *regex;
7          size_t n, m;
8          pmix_status_t rc;
9          pmix_info_t info;
10
11         /* loop over an array of nodes */
12         for (n=0; n < num_nodes; n++)
13             /* for each node, construct an array of ranks on that node */
14             ndppn = NULL;
15             for (m=0; m < node[n]->num_procs; m++)
16                 /* ignore processes that are not part of the target job */
17                 if (!PMIX_CHECK_NSPACE(targetjob,node[n]->proc[m].nspace))
18                     continue;
19
20                 snprintf(rank, 30, "%d", node[n]->proc[m].rank);
21                 PMIX_ARGV_APPEND(&ndppn, rank);
22
23             /* convert the array into a comma-delimited string of ranks */
24             localranks = PMIX_ARGV_JOIN(ndppn, ',');
25             /* release the local array */
26             PMIX_ARGV_FREE(ndppn);
27             /* add this node's contribution to the overall array */
28             PMIX_ARGV_APPEND(&ppnarray, localranks);
29             /* release the local list */
30             free(localranks);
31
32
33         /* join into a semicolon-delimited string */
34         ppn = PMIX_ARGV_JOIN(ppnarray, ';');
35
36         /* release the array */
37         PMIX_ARGV_FREE(ppnarray);
38
39         /* generate ppn regex */
40         rc = PMIx_generate_ppn(ppn, &regex);
41
42         /* release list */
```

```
1    free(ppn);
2
3    /* pass the regex as the value to the PMIX_PROC_MAP key */
4    PMIX_INFO_LOAD(&info, PMIX_PROC_MAP, regex, PMIX_STRING);
5    /* release the regex */
6    free(regex);
```

7 ──────────────────────── 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):
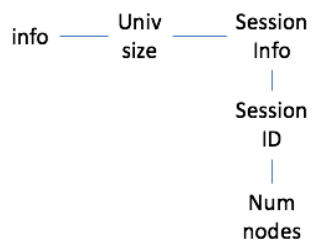
Figure 11.1.: Session-level information elements

- 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

1   non-specific attributes (e.g., **PMIX_NODE_MAP**) are passed to describe aspects of the job. Note
2   that since the invocation only involves a single namespace, there is no need to include the
3   **PMIX_NSPACE** attribute in the array.

4   Upon conclusion of this step, the *info* array might look like:
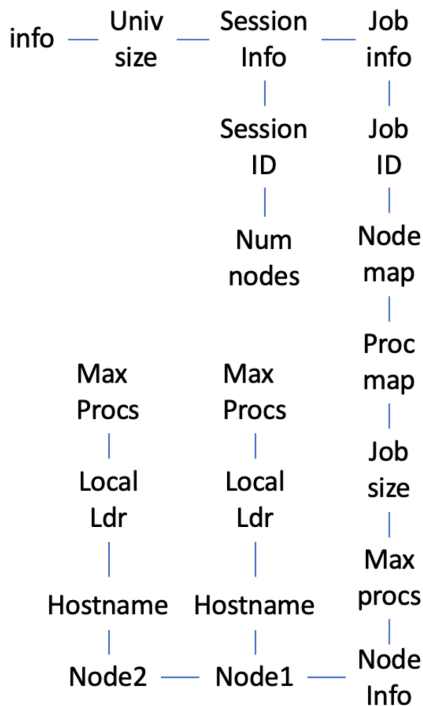
Figure 11.2.: Job-level information elements

5   Note that in this example, **PMIX_NUM_NODES** is not required as that information is contained
6   in the **PMIX_NODE_MAP** attribute. Similarly, **PMIX_JOB_SIZE** is not technically required as
7   that information is contained in the **PMIX_PROC_MAP** when combined with the corresponding
8   node map - however, there is no issue with including the job size as a separate entry.

9   The example also illustrates the hierarchical use of the **PMIX_NODE_INFO_ARRAY** attribute.
10   In this case, we have chosen to pass several job-related values for each node - since those values
11   are non-unique across the job, they must be passed in a node-info container. Note that the choice
12   of what information to pass into the PMIx server library versus what information to derive from
13   other values at time of request is left to the host environment. PMIx implementors in turn may, if
14   they choose, pre-parse registration data to create expanded views (thus enabling faster response
15   to requests at the expense of memory footprint) or to compress views into tighter representations
16   (thus trading minimized footprint for longer response times).

- 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 must 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 11.3, assuming there are two applications in the job being registered:
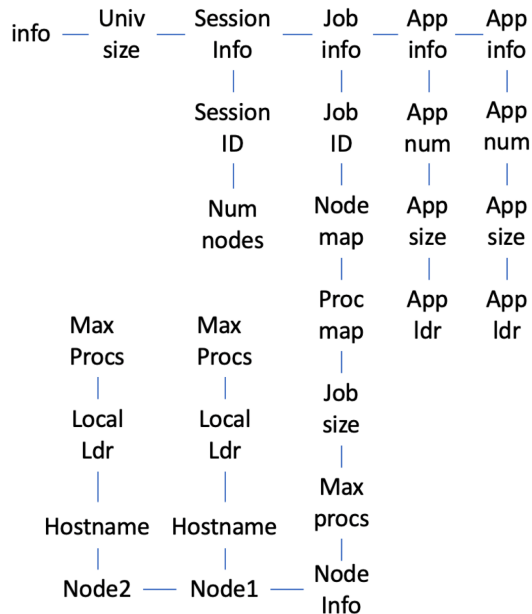
Figure 11.3.: Application-level information elements

- 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 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 11.4:

- For purposes of this example, 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

info	Univ size	Session Info	Job info	App info	App info	Proc data	Proc data
		Session ID	Job ID	App num	App num	Rank	Rank
		Num nodes	Node map	App size	App size	Local rank	Local rank
			Proc map	App ldr	App ldr	Node rank	Node rank
	Max Procs	Max Procs	Job size			Node ID	Node ID
	Local Ldr	Local Ldr	Max procs			App num	App num
Hostname	Hostname		Node Info			App rank	App rank
Node2 — Node1 —							
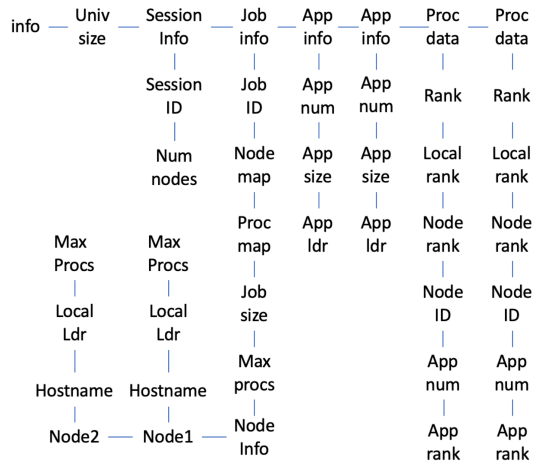
Figure 11.4.: Process-level information elements

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, or where values for multiple nodes are being provided.

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 ————————————————————————

```
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 11.5:
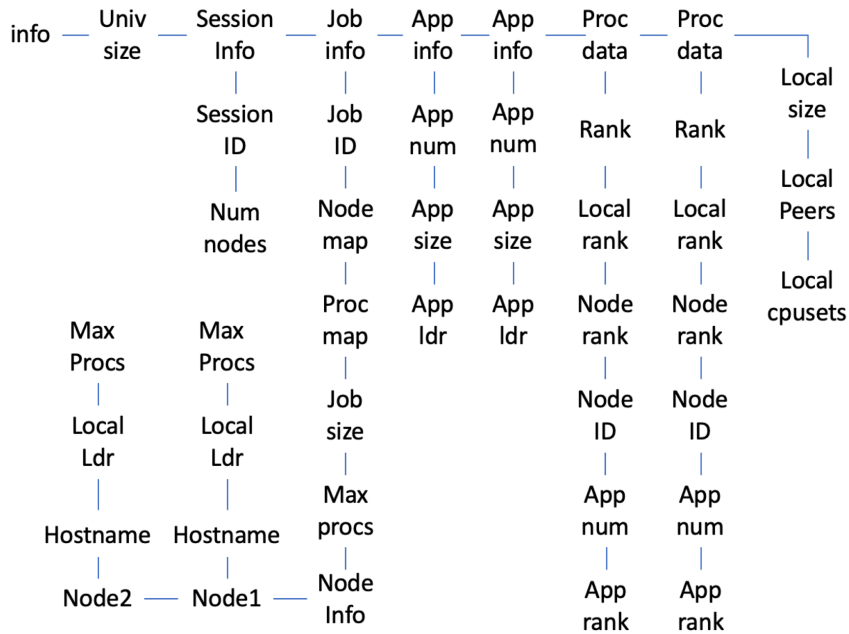
Figure 11.5.: Final information array

11.2.4 `PMIx_server_deregister_nspace`

Summary

Deregister a namespace.

Format

PMIx v1.0

────────────────────────── C ──────────────────────────

```
void PMIx_server_deregister_nspace(const pmix_nspace_t nspace,
                                   pmix_op_cbfunc_t cbfunc, void *cbdata)
```

────────────────────────── 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.

11.2.5 `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.

11.2.6 `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.

11.2.7 `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).

11.2.8 `PMIx_server_dmodex_request`

Summary

Define a function by which the host server can request modex data from the local PMIx server.

Format

```
pmix_status_t PMIx_server_dmodex_request(const pmix_proc_t *proc,
                         pmix_dmodex_response_fn_t cbfunc,
                         void *cbdata)
```

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.

11.2.9 `PMIx_server_setup_application`

Summary

Provide a function by which the resource manager can request application-specific setup data prior to launch of a **job** .

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

▼------------------   Required Attributes   ------------------▼

PMIx libraries that support this operation are required to support the following:

PMIX_SETUP_APP_ENVARS   "pmix.setup.env" (**bool**)
        Harvest and include relevant environmental variables

PMIX_SETUP_APP_NONENVARS   ""pmix.setup.nenv" (**bool**)
        Include all relevant data other than environmental variables

PMIX_SETUP_APP_ALL   "pmix.setup.all" (**bool**)

1   Include all relevant data

2 **PMIX_ALLOC_FABRIC**   **"pmix.alloc.net"** (**array**)
3   Array of **pmix_info_t** describing requested fabric resources. This must include at least:
4   **PMIX_ALLOC_FABRIC_ID** , **PMIX_ALLOC_FABRIC_TYPE** , and
5   **PMIX_ALLOC_FABRIC_ENDPTS** , plus whatever other descriptors are desired.

6 **PMIX_ALLOC_FABRIC_ID**   **"pmix.alloc.netid"** (**char***)
7   The key to be used when accessing this requested fabric allocation. The allocation will be
8   returned/stored as a **pmix_data_array_t** of **pmix_info_t** indexed by this key and
9   containing at least one entry with the same key and the allocated resource description. The
10   type of the included value depends upon the fabric support. For example, a TCP allocation
11   might consist of a comma-delimited string of socket ranges such as
12   *"32000-32100,33005,38123-38146"*. Additional entries will consist of any provided
13   resource request directives, along with their assigned values. Examples include:
14   **PMIX_ALLOC_FABRIC_TYPE** - the type of resources provided;
15   **PMIX_ALLOC_FABRIC_PLANE** - if applicable, what plane the resources were assigned
16   from; **PMIX_ALLOC_FABRIC_QOS** - the assigned QoS; **PMIX_ALLOC_BANDWIDTH** -
17   the allocated bandwidth; **PMIX_ALLOC_FABRIC_SEC_KEY** - a security key for the
18   requested fabric allocation. NOTE: the assigned values may differ from those requested,
19   especially if **PMIX_INFO_REQD** was not set in the request.

20 **PMIX_ALLOC_FABRIC_SEC_KEY**   **"pmix.alloc.nsec"** (**pmix_byte_object_t**)
21   Fabric security key

22 **PMIX_ALLOC_FABRIC_TYPE**   **"pmix.alloc.nettype"** (**char***)
23   Type of desired transport (e.g., *"tcp"*, *"udp"*)

24 **PMIX_ALLOC_FABRIC_PLANE**   **"pmix.alloc.netplane"** (**char***)
25   ID string for the NIC (aka *plane*) to be used for this allocation (e.g., CIDR for Ethernet)

26 **PMIX_ALLOC_FABRIC_ENDPTS**   **"pmix.alloc.endpts"** (**size_t**)
27   Number of endpoints to allocate per process

28 **PMIX_ALLOC_FABRIC_ENDPTS_NODE**   **"pmix.alloc.endpts.nd"** (**size_t**)
29   Number of endpoints to allocate per node

▲--▲
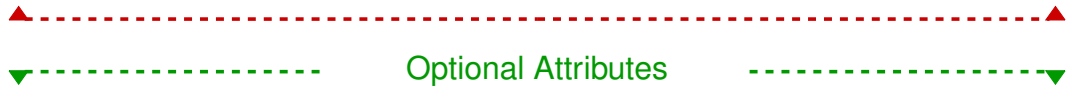
▼----------------- Optional Attributes -----------------▼

30 PMIx libraries that support this operation may support the following:

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

33 **PMIX_ALLOC_FABRIC_QOS**   **"pmix.alloc.netqos"** (**char***)
34   Quality of service level.

35 **PMIX_ALLOC_TIME**   **"pmix.alloc.time"** (**uint32_t**)

Time in seconds.

The following optional attributes may be provided by the host environment to identify the programming model (as specified by the user) being executed within the application. The PMIx server library may utilize this information to harvest/forward model-specific environmental variables, record the programming model associated with the application, etc.

- **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")

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 a job.

▼——————————— Advice to PMIx server hosts ———————————▼

Host environments are required to execute this operation prior to launching a job. In addition to supported directives, the *info* array must include a description of the **job** using the **PMIX_NODE_MAP** and **PMIX_PROC_MAP** attributes.

▲

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.

▼——————————— Advice to PMIx library implementers ———————————▼

Support for harvesting of environmental variables and providing of local configuration information by the PMIx implementation is optional.

▲

11.2.10 **PMIx_Register_attributes**

Summary

Register host environment attribute support for a function.

1 **Format**

```
pmix_status_t
PMIx_Register_attributes(char *function,
                         pmix_regattr_t attrs[],
                         size_t nattrs)
```

6 **IN   function**
7   String name of function (string)
8 **IN   attrs**
9   Array of **pmix_regattr_t** describing the supported attributes (handle)
10 **IN   nattrs**
11   Number of elements in *attrs* (**size_t**)

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

13 **Description**
14 The **PMIx_Register_attributes** function is used by the host environment to register with
15 its PMIx server library the attributes it supports for each **pmix_server_module_t** function.
16 The *function* is the string name of the server module function (e.g., "register_events",
17 "validate_credential", or "allocate") whose attributes are being registered. See the
18 **pmix_regattr_t** entry for a description of the *attrs* array elements.

19 Note that the host environment can also query the library (using the **PMIx_Query_info_nb**
20 API) for its attribute support both at the server, client, and tool levels once the host has executed
21 **PMIx_server_init** since the server will internally register those values.

——————— Advice to PMIx server hosts ———————

22 Host environments are strongly encouraged to register all supported attributes immediately after
23 initializing the library to ensure that user requests are correctly serviced.

1   PMIx implementations are *required* to register all internally supported attributes for each API
2   during initialization of the library (i.e., when the process calls their respective PMIx init function).
3   Specifically, the implementation *must not* register supported attributes upon first call to a given API
4   as this would prevent users from discovering supported attributes prior to first use of an API.

5   It is the implementation's responsibility to associate registered attributes for a given
6   **pmix_server_module_t** function with their corresponding user-facing API. Supported
7   attributes *must* be reported to users in terms of their support for user-facing APIs, broken down by
8   the level (see 14.4.33) at which the attribute is supported.

9   Note that attributes can/will be registered on an API for each level. It is *required* that the
10   implementation support user queries for supported attributes on a per-level basis. Duplicate
11   registrations at the *same* level for a function *shall* return an error - however, duplicate registrations
12   at *different* levels *shall* be independently tracked.

11.2.11 `PMIx_server_setup_local_support`

13

Summary

14

15   Provide a function by which the local PMIx server can perform any application-specific operations
16   prior to spawning local clients of a given application.

Format

17

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

23   **IN**   **nspace**
24   Namespace (string)
25   **IN**   **info**
26   Array of info structures (array of handles)
27   **IN**   **ninfo**
28   Number of elements in the *info* array (**size_t**)
29   **IN**   **cbfunc**
30   Callback function **pmix_op_cbfunc_t** (function reference)
31   **IN**   **cbdata**
32   Data to be passed to the callback function (memory reference)

1   Returns one of the following:

2   • **PMIX_SUCCESS** , indicating that the request is being processed by the host environment - result
3   will be returned in the provided *cbfunc*. Note that the library must not invoke the callback
4   function prior to returning from the API.

5   • **PMIX_OPERATION_SUCCEEDED** , indicating that the request was immediately processed and
6   returned *success* - the *cbfunc* will not be called

7   • a PMIx error constant indicating either an error in the input or that the request was immediately
8   processed and failed - the *cbfunc* will not be called

9   **Description**
10   Provide a function by which the local PMIx server can perform any application-specific operations
11   prior to spawning local clients of a given application. For example, a fabric library might need to
12   setup the local driver for "instant on" addressing. The data provided in the *info* array is the data
13   returned to the host RM by the callback function executed as a result of a call to
14   **PMIx_server_setup_application** .

———————————— Advice to PMIx server hosts ————————————

15   Host environments are required to execute this operation prior to starting any local application
16   processes from the specified namespace.

11.2.12 **PMIx_server_IOF_deliver**

18   **Summary**
19   Provide a function by which the host environment can pass forwarded IO to the PMIx server library
20   for distribution to its clients.

21   **Format**
PMIx v3.0 ▼ ———————————————— C ———————————————— ▼

```
22   pmix_status_t
23   PMIx_server_IOF_deliver(const pmix_proc_t *source,
24                               pmix_iof_channel_t channel,
25                               const pmix_byte_object_t *bo,
26                               const pmix_info_t info[], size_t ninfo,
27                               pmix_op_cbfunc_t cbfunc, void *cbdata);
```

1 **IN   source**
2   Pointer to **pmix_proc_t** identifying source of the IO (handle)
3 **IN   channel**
4   IO channel of the data (**pmix_iof_channel_t**)
5 **IN   bo**
6   Pointer to **pmix_byte_object_t** containing the payload to be delivered (handle)
7 **IN   info**
8   Array of **pmix_info_t** metadata describing the data (array of handles)
9 **IN   ninfo**
10   Number of elements in the *info* array (**size_t**)
11 **IN   cbfunc**
12   Callback function **pmix_op_cbfunc_t** (function reference)
13 **IN   cbdata**
14   Data to be passed to the callback function (memory reference)

15 Returns one of the following:

16 • **PMIX_SUCCESS** , indicating that the request is being processed by the host environment - result
17   will be returned in the provided *cbfunc*. Note that the library must not invoke the callback
18   function prior to returning from the API.

19 • **PMIX_OPERATION_SUCCEEDED** , indicating that the request was immediately processed and
20   returned *success* - the *cbfunc* will not be called

21 • a PMIx error constant indicating either an error in the input or that the request was immediately
22   processed and failed - the *cbfunc* will not be called

23 **Description**
24 Provide a function by which the host environment can pass forwarded IO to the PMIx server library
25 for distribution to its clients. The PMIx server library is responsible for determining which of its
26 clients have actually registered for the provided data and delivering it. The *cbfunc* callback function
27 will be called once the PMIx server library no longer requires access to the provided data.

28 ## 11.2.13 **PMIx_server_collect_inventory**

29 **Summary**
30 Collect inventory of resources on a node

Format

```
pmix_status_t
PMIx_server_collect_inventory(const pmix_info_t directives[],
                              size_t ndirs,
                              pmix_info_cbfunc_t cbfunc,
                              void *cbdata);
```

IN   directives
    Array of **pmix_info_t** directing the request (array of handles)
IN   ndirs
    Number of elements in the *directives* array (**size_t**)
IN   cbfunc
    Callback function to return collected data (**pmix_info_cbfunc_t** function reference)
IN   cbdata
    Data to be passed to the callback function (memory reference)

Returns **PMIX_SUCCESS** or a negative value corresponding to a PMIx error constant. In the event the function returns an error, the *cbfunc* will not be called.

Description
Provide a function by which the host environment can request its PMIx server library collect an inventory of local resources. Supported resources depends upon the PMIx implementation, but may include the local node topology and fabric interfaces.

———————————— Advice to PMIx server hosts ————————————

This is a non-blocking API as it may involve somewhat lengthy operations to obtain the requested information. Inventory collection is expected to be a rare event – at system startup and upon command from a system administrator. Inventory updates are expected to initiate a smaller operation involving only the changed information. For example, replacement of a node would generate an event to notify the scheduler with an inventory update without invoking a global inventory operation.

11.2.14 `PMIx_server_deliver_inventory`

Summary
Pass collected inventory to the PMIx server library for storage

Format

─────────────────────────── C ───────────────────────────

```
pmix_status_t
PMIx_server_deliver_inventory(const pmix_info_t info[],
                              size_t ninfo,
                              const pmix_info_t directives[],
                              size_t ndirs,
                              pmix_op_cbfunc_t cbfunc,
                              void *cbdata);
```

─────────────────────────── C ───────────────────────────

IN   **info**
      Array of **pmix_info_t** containing the inventory (array of handles)
IN   **ninfo**
      Number of elements in the *info* array (**size_t**)
IN   **directives**
      Array of **pmix_info_t** directing the request (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)

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 host environment can pass inventory information obtained from a
node to the PMIx server library for storage. Inventory data is subsequently used by the PMIx server
library for allocations in response to **PMIx_server_setup_application** , and may be
available to the library's host via the **PMIx_Get** API (depending upon PMIx implementation).
The *cbfunc* callback function will be called once the PMIx server library no longer requires access
to the provided data.

11.3 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. PMIx implementations are required to return a **PMIX_ERR_NOT_SUPPORTED** status to all calls to functions that require host environment support and are not backed by a corresponding server module entry.

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

11.3.1 `pmix_server_module_t` Module

Summary

List of function pointers that a PMIx server passes to **PMIx_server_init** during startup.

Format

```
1    typedef struct pmix_server_module_3_0_0_t
2        /* v1x interfaces */
3        pmix_server_client_connected_fn_t     client_connected;
4        pmix_server_client_finalized_fn_t     client_finalized;
5        pmix_server_abort_fn_t                abort;
6        pmix_server_fencenb_fn_t              fence_nb;
7        pmix_server_dmodex_req_fn_t           direct_modex;
8        pmix_server_publish_fn_t              publish;
9        pmix_server_lookup_fn_t               lookup;
10       pmix_server_unpublish_fn_t            unpublish;
11       pmix_server_spawn_fn_t                spawn;
12       pmix_server_connect_fn_t              connect;
13       pmix_server_disconnect_fn_t           disconnect;
14       pmix_server_register_events_fn_t      register_events;
15       pmix_server_deregister_events_fn_t    deregister_events;
16       pmix_server_listener_fn_t             listener;
17       /* v2x interfaces */
18       pmix_server_notify_event_fn_t         notify_event;
19       pmix_server_query_fn_t                query;
20       pmix_server_tool_connection_fn_t      tool_connected;
21       pmix_server_log_fn_t                  log;
22       pmix_server_alloc_fn_t                allocate;
23       pmix_server_job_control_fn_t          job_control;
24       pmix_server_monitor_fn_t              monitor;
25       /* v3x interfaces */
26       pmix_server_get_cred_fn_t             get_credential;
27       pmix_server_validate_cred_fn_t        validate_credential;
28       pmix_server_iof_fn_t                  iof_pull;
29       pmix_server_stdin_fn_t                push_stdin;
30       /* v4x interfaces */
31       pmix_server_grp_fn_t                  group;
32       pmix_server_fabric_fn_t               fabric;
33   pmix_server_module_t;
```

———————— C ————————

34   11.3.2   `pmix_server_client_connected_fn_t`

35 **Summary**

36 Notify the host server that a client connected to this server.

———————————————————— 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.

1   This operation is an opportunity for a host environment to update the status of the ranks it manages.
2   It is also a convenient and well defined time to perform initialization necessary to support further
3   calls into the server related to that rank.

4 ## 11.3.3   `pmix_server_client_finalized_fn_t`

5 **Summary**
6 Notify the host environment that a client called **PMIx_Finalize**.

7 **Format**

PMIx v1.0

― 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 ―

13 **IN   proc**
14   **pmix_proc_t** structure (handle)
15 **IN   server_object**
16   object reference (memory reference)
17 **IN   cbfunc**
18   Callback function **pmix_op_cbfunc_t** (function reference)
19 **IN   cbdata**
20   Data to be passed to the callback function (memory reference)

21 Returns one of the following:

22 • **PMIX_SUCCESS**, indicating that the request is being processed by the host environment - result
23   will be returned in the provided *cbfunc*. Note that the host must not invoke the callback function
24   prior to returning from the API.

25 • **PMIX_OPERATION_SUCCEEDED**, indicating that the request was immediately processed and
26   returned *success* - the *cbfunc* will not be called

27 • a PMIx error constant indicating either an error in the input or that the request was immediately
28   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.

11.3.4 `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)
```

1   **IN** `proc`
2   **`pmix_proc_t`** structure identifying the process requesting the abort (handle)
3   **IN** `server_object`
4   object reference (memory reference)
5   **IN** `status`
6   exit status (integer)
7   **IN** `msg`
8   exit status message (string)
9   **IN** `procs`
10   Array of **`pmix_proc_t`** structures identifying the processes to be terminated (array of
11   handles)
12   **IN** `nprocs`
13   Number of elements in the *procs* array (integer)
14   **IN** `cbfunc`
15   Callback function **`pmix_op_cbfunc_t`** (function reference)
16   **IN** `cbdata`
17   Data to be passed to the callback function (memory reference)

18   Returns one of the following:

19   - **PMIX_SUCCESS** , indicating that the request is being processed by the host environment - result
20   will be returned in the provided *cbfunc*. Note that the host must not invoke the callback function
21   prior to returning from the API.

22   - **PMIX_OPERATION_SUCCEEDED** , indicating that the request was immediately processed and
23   returned *success* - the *cbfunc* will not be called

24   - **PMIX_ERR_NOT_SUPPORTED** , indicating that the host environment does not support the
25   request, even though the function entry was provided in the server module - the *cbfunc* will not
26   be called

27   - a PMIx error constant indicating either an error in the input or that the request was immediately
28   processed and failed - the *cbfunc* will not be called

Description
30   A local client called **PMIx_Abort** . Note that the client will be in a blocked state until the host
31   server executes the callback function, thus allowing the PMIx server library to release the client.
32   The array of *procs* indicates which processes are to be terminated. A **NULL** indicates that all
33   processes in the client's namespace are to be terminated.

11.3.5 `pmix_server_fencenb_fn_t`

Summary
36   At least one client called either **PMIx_Fence** or **PMIx_Fence_nb** .

```
2    typedef pmix_status_t (*pmix_server_fencenb_fn_t)(
3                                      const pmix_proc_t procs[],
4                                      size_t nprocs,
5                                      const pmix_info_t info[],
6                                      size_t ninfo,
7                                      char *data, size_t ndata,
8                                      pmix_modex_cbfunc_t cbfunc,
9                                      void *cbdata)
```

10   **IN   procs**
11   Array of **pmix_proc_t** structures identifying operation participants(array of handles)
12   **IN   nprocs**
13   Number of elements in the *procs* array (integer)
14   **IN   info**
15   Array of info structures (array of handles)
16   **IN   ninfo**
17   Number of elements in the *info* array (integer)
18   **IN   data**
19   (string)
20   **IN   ndata**
21   (integer)
22   **IN   cbfunc**
23   Callback function **pmix_modex_cbfunc_t** (function reference)
24   **IN   cbdata**
25   Data to be passed to the callback function (memory reference)

26   Returns one of the following:

27   • **PMIX_SUCCESS** , indicating that the request is being processed by the host environment - result
28   will be returned in the provided *cbfunc*. Note that the host must not invoke the callback function
29   prior to returning from the API.

30   • **PMIX_OPERATION_SUCCEEDED** , indicating that the request was immediately processed and
31   returned *success* - the *cbfunc* will not be called

32   • **PMIX_ERR_NOT_SUPPORTED** , indicating that the host environment does not support the
33   request, even though the function entry was provided in the server module - the *cbfunc* will not
34   be called

35   • a PMIx error constant indicating either an error in the input or that the request was immediately
36   processed and failed - the *cbfunc* will not be called

1 PMIx libraries are required to pass any provided attributes to the host environment for processing.

2

3 The following attributes are required to be supported by all host environments:

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

6 The following attributes are optional for host environments:

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

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

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

18 Host environment are required to return **PMIX_ERR_NOT_SUPPORTED** if passed an attributed
19 marked as **PMIX_INFO_REQD** that they do not support, even if support for that attribute is
20 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. Data received from each node must be simply concatenated to form an aggregated unit, as shown in the following example:

———————————————— C ————————————————

```
uint8_t *blob1, *blob2, *total;
size_t sz_blob1, sz_blob2, sz_total;

sz_total = sz_blob1 + sz_blob2;
total = (uint8_t*)malloc(sz_total);
memcpy(total, blob1, sz_blob1);
memcpy(&total[sz_blob1], blob2, sz_blob2);
```

———————————————— C ————————————————

Note that the ordering of the data blobs does not matter.

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.

11.3.6 `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

```
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)
```

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_ERR_NOT_SUPPORTED** , indicating that the host environment does not support the request, even though the function entry was provided in the server module - 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.

CHAPTER 11. SERVER-SPECIFIC INTERFACES   **195**

1   The following attributes are optional for host environments that support this operation:

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

6   **Description**
7   Used by the PMIx server to request its local host contact the PMIx server on the remote node that
8   hosts the specified proc to obtain and return any information that process posted via calls to
9   **PMIx_Put** and **PMIx_Commit** .

10   The array of *info* structs is used to pass user-requested options to the server. This can include a
11   timeout to preclude an indefinite wait for data that may never become available. The directives are
12   optional unless the *mandatory* flag has been set - in such cases, the host RM is required to return an
13   error if the directive cannot be met.

14   ## 11.3.7 **pmix_server_publish_fn_t**

15   **Summary**
16   Publish data per the PMIx API specification.

17   **Format**
     PMIx v1.0 ──────────────────────── C ────────────────────────
18   ```
     typedef pmix_status_t (*pmix_server_publish_fn_t)(
19   const pmix_proc_t *proc,
20   const pmix_info_t info[],
21   size_t ninfo,
22   pmix_op_cbfunc_t cbfunc,
23   void *cbdata)
     ```
     ──────────────────────── C ────────────────────────

24   **IN   proc**
25       **pmix_proc_t** structure of the process publishing the data (handle)
26   **IN   info**
27       Array of info structures (array of handles)
28   **IN   ninfo**
29       Number of elements in the *info* array (integer)
30   **IN   cbfunc**
31       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

- **PMIX_ERR_NOT_SUPPORTED** , indicating that the host environment does not support the
  request, even though the function entry was provided in the server module - 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_RANGE_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.

## 11.3.8  `pmix_server_lookup_fn_t`

**Summary**

Lookup published data.

**Format**

*PMIx v1.0*  ――――――――――――――――――――― 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)
```

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

- **PMIX_ERR_NOT_SUPPORTED** , indicating that the host environment does not support the request, even though the function entry was provided in the server module - 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**)

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

▲--▲

▼----------------   Optional Attributes   ----------------▼

3   The following attributes are optional for host environments that support this operation:

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

▲--▲

8   **Description**
9   Lookup published data. The host server will be passed a **NULL**-terminated array of string keys
10   identifying the data being requested.

11   The array of *info* structs is used to pass user-requested options to the server. The default data range
12   is left to the host environment, but expected to be **PMIX_RANGE_SESSION** . This can include a
13   wait flag to indicate that the server should wait for all data to become available before executing the
14   callback function, or should immediately callback with whatever data is available. In addition, a
15   timeout can be specified on the wait to preclude an indefinite wait for data that may never be
16   published.

───────────────   Advice to PMIx server hosts   ───────────────

17   The **PMIX_USERID** and **PMIX_GRPID** of the requesting process will be provided to support
18   authorization-based access to published information. The host environment is not required to
19   guarantee support for any specific range - i.e., the environment does not need to return an error if
20   the data store doesn't support a specified range so long as it is covered by some internally defined
21   range.

───

22   11.3.9   **pmix_server_unpublish_fn_t**

23   **Summary**
24   Delete data from the data store.

Format

———————————————————— C ————————————————————

```
2    typedef pmix_status_t (*pmix_server_unpublish_fn_t)(
3                                        const pmix_proc_t *proc,
4                                        char **keys,
5                                        const pmix_info_t info[],
6                                        size_t ninfo,
7                                        pmix_op_cbfunc_t cbfunc,
8                                        void *cbdata)
```

———————————————————— C ————————————————————

9   IN   proc
10   **pmix_proc_t** structure identifying the process making the request (handle)
11 IN   keys
12   (array of strings)
13 IN   info
14   Array of info structures (array of handles)
15 IN   ninfo
16   Number of elements in the *info* array (integer)
17 IN   cbfunc
18   Callback function **pmix_op_cbfunc_t** (function reference)
19 IN   cbdata
20   Data to be passed to the callback function (memory reference)

21 Returns one of the following:

22 • **PMIX_SUCCESS** , indicating that the request is being processed by the host environment - result
23   will be returned in the provided *cbfunc*. Note that the host must not invoke the callback function
24   prior to returning from the API.

25 • **PMIX_OPERATION_SUCCEEDED** , indicating that the request was immediately processed and
26   returned *success* - the *cbfunc* will not be called

27 • **PMIX_ERR_NOT_SUPPORTED** , indicating that the host environment does not support the
28   request, even though the function entry was provided in the server module - the *cbfunc* will not
29   be called

30 • a PMIx error constant indicating either an error in the input or that the request was immediately
31   processed and failed - the *cbfunc* will not be called

▼----------------- Required Attributes -----------------▼

32 PMIx libraries are required to pass any provided attributes to the host environment for processing.
33 In addition, the following attributes are required to be included in the passed *info* array:

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

1 **`PMIX_GRPID`** **`"pmix.egid"`** (**`uint32_t`**)
2   Effective group id.

3 ──

4 Host environments that implement this entry point are required to support the following attributes:

5 **`PMIX_RANGE`** **`"pmix.range"`** (**`pmix_data_range_t`**)
6   Value for calls to publish/lookup/unpublish or for monitoring event notifications.
▲--▲

▼----------------- Optional Attributes -----------------▼

7 The following attributes are optional for host environments that support this operation:

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

12 **Description**
13 Delete data from the data store. The host server will be passed a **NULL**-terminated array of string
14 keys, plus potential directives such as the data range within which the keys should be deleted. The
15 default data range is left to the host environment, but expected to be **`PMIX_RANGE_SESSION`** .
16 The callback is to be executed upon completion of the delete procedure.

──────────────── Advice to PMIx server hosts ────────────────

17 The **`PMIX_USERID`** and **`PMIX_GRPID`** of the requesting process will be provided to support
18 authorization-based access to published information. The host environment is not required to
19 guarantee support for any specific range - i.e., the environment does not need to return an error if
20 the data store doesn't support a specified range so long as it is covered by some internally defined
21 range.

11.3.10   `pmix_server_spawn_fn_t`

23 **Summary**
24 Spawn a set of applications/processes as per the **`PMIx_Spawn`** API.

PMIx v1.0

—————————————————— C ——————————————————

```
2    typedef pmix_status_t (*pmix_server_spawn_fn_t)(
3                                    const pmix_proc_t *proc,
4                                    const pmix_info_t job_info[],
5                                    size_t ninfo,
6                                    const pmix_app_t apps[],
7                                    size_t napps,
8                                    pmix_spawn_cbfunc_t cbfunc,
9                                    void *cbdata)
```

—————————————————— C ——————————————————

10   **IN** **proc**
11   **pmix_proc_t** structure of the process making the request (handle)
12   **IN** **job_info**
13   Array of info structures (array of handles)
14   **IN** **ninfo**
15   Number of elements in the *jobinfo* array (integer)
16   **IN** **apps**
17   Array of **pmix_app_t** structures (array of handles)
18   **IN** **napps**
19   Number of elements in the *apps* array (integer)
20   **IN** **cbfunc**
21   Callback function **pmix_spawn_cbfunc_t** (function reference)
22   **IN** **cbdata**
23   Data to be passed to the callback function (memory reference)

24   Returns one of the following:

25   • **PMIX_SUCCESS** , indicating that the request is being processed by the host environment - result
26   will be returned in the provided *cbfunc*. Note that the host must not invoke the callback function
27   prior to returning from the API.

28   • **PMIX_OPERATION_SUCCEEDED** , indicating that the request was immediately processed and
29   returned *success* - the *cbfunc* will not be called

30   • **PMIX_ERR_NOT_SUPPORTED** , indicating that the host environment does not support the
31   request, even though the function entry was provided in the server module - the *cbfunc* will not
32   be called

33   • a PMIx error constant indicating either an error in the input or that the request was immediately
34   processed and failed - the *cbfunc* will not be called

1 PMIx libraries are required to pass any provided attributes to the host environment for processing.
2 In addition, the following attributes are required to be included in the passed *info* array:

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

5 **PMIX_GRPID** **"pmix.egid"** (**uint32_t**)
6   Effective group id.

7 _____

8 Host environments that provide this module entry point are required to pass the **PMIX_SPAWNED**
9 and **PMIX_PARENT_ID** attributes to all PMIx servers launching new child processes so those
10 values can be returned to clients upon connection to the PMIx server. In addition, they are required
11 to support the following attributes when present in either the *job_info* or the *info* array of an
12 element of the *apps* array:

13 **PMIX_WDIR** **"pmix.wdir"** (**char***)
14   Working directory for spawned processes.

15 **PMIX_SET_SESSION_CWD** **"pmix.ssncwd"** (**bool**)
16   Set the application's current working directory to the session working directory assigned by
17   the RM - when accessed using **PMIx_Get** , use the **PMIX_RANK_WILDCARD** value for
18   the rank to discover the session working directory assigned to the provided namespace

19 **PMIX_PREFIX** **"pmix.prefix"** (**char***)
20   Prefix to use for starting spawned processes.

21 **PMIX_HOST** **"pmix.host"** (**char***)
22   Comma-delimited list of hosts to use for spawned processes.

23 **PMIX_HOSTFILE** **"pmix.hostfile"** (**char***)
24   Hostfile to use for spawned processes.

25 The following attributes are optional for host environments that support this operation:

26 **PMIX_ADD_HOSTFILE** **"pmix.addhostfile"** (**char***)
27   Hostfile listing hosts to add to existing allocation.

28 **PMIX_ADD_HOST** **"pmix.addhost"** (**char***)
29   Comma-delimited list of hosts to add to the allocation.

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

32 **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.

11.3.11 `pmix_server_connect_fn_t`

Summary

Record the specified processes as *connected*.

Format

PMIx v1.0

```
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)
```

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

- **`PMIX_ERR_NOT_SUPPORTED`** , indicating that the host environment does not support the request, even though the function entry was provided in the server module - 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.

11.3.12 `pmix_server_disconnect_fn_t`

Summary

Disconnect a previously connected set of processes.

1 **Format**

C

```
2      typedef pmix_status_t (*pmix_server_disconnect_fn_t)(
3                                          const pmix_proc_t procs[],
4                                          size_t nprocs,
5                                          const pmix_info_t info[],
6                                          size_t ninfo,
7                                          pmix_op_cbfunc_t cbfunc,
8                                          void *cbdata)
```

C

9   **IN   procs**
10   Array of **pmix_proc_t** structures identifying participants (array of handles)
11 **IN   nprocs**
12   Number of elements in the *procs* array (integer)
13 **IN   info**
14   Array of info structures (array of handles)
15 **IN   ninfo**
16   Number of elements in the *info* array (integer)
17 **IN   cbfunc**
18   Callback function **pmix_op_cbfunc_t** (function reference)
19 **IN   cbdata**
20   Data to be passed to the callback function (memory reference)

21 Returns one of the following:

22 • **PMIX_SUCCESS** , indicating that the request is being processed by the host environment - result
23   will be returned in the provided *cbfunc*. Note that the host must not invoke the callback function
24   prior to returning from the API.

25 • **PMIX_OPERATION_SUCCEEDED** , indicating that the request was immediately processed and
26   returned *success* - the *cbfunc* will not be called

27 • **PMIX_ERR_NOT_SUPPORTED** , indicating that the host environment does not support the
28   request, even though the function entry was provided in the server module - the *cbfunc* will not
29   be called

30 • a PMIx error constant indicating either an error in the input or that the request was immediately
31   processed and failed - the *cbfunc* will not be called

▼----------------- Required Attributes -----------------▼

32 PMIx libraries are required to pass any provided attributes to the host environment for processing.

1   The following attributes are optional for host environments that support this operation:

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

6   **Description**
7   Disconnect a previously connected set of processes. The callback is to be executed once every
8   daemon hosting at least one participant has called the host server's has called the
9   **pmix_server_disconnect_fn_t** function, and the host environment has completed any
10   required supporting operations.

────────────────── Advice to PMIx library implementers ──────────────────

11   The PMIx server library is required to aggregate participation by local clients, passing the request
12   to the host environment once all local participants have executed the API.

────────────────── Advice to PMIx server hosts ──────────────────

13   The host will receive a single call for each collective operation. It is the responsibility of the host to
14   identify the nodes containing participating processes, execute the collective across all participating
15   nodes, and notify the local PMIx server library upon completion of the global collective.

16   A **PMIX_ERR_INVALID_OPERATION** error must be returned if the specified set of *procs* was
17   not previously *connected* via a call to the **pmix_server_connect_fn_t** function.

18   # 11.3.13 `pmix_server_register_events_fn_t`

19   **Summary**
20   Register to receive notifications for the specified events.

Format

```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)
```

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

- **PMIX_ERR_NOT_SUPPORTED** , indicating that the host environment does not support the request, even though the function entry was provided in the server module - 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 beyond **PMIX_ERR_SYS_OTHER**) 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).

▲ ▲

11.3.14 `pmix_server_deregister_events_fn_t`

Summary

Deregister to receive notifications for the specified events.

Format

```
typedef pmix_status_t (*pmix_server_deregister_events_fn_t)(
                                   pmix_status_t *codes,
                                   size_t ncodes,
                                   pmix_op_cbfunc_t cbfunc,
                                   void *cbdata)
```

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

- **PMIX_ERR_NOT_SUPPORTED** , indicating that the host environment does not support the request, even though the function entry was provided in the server module - 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

1 • no client (including the server library itself) remains registered for notifications on any included
2   code - i.e., a code should be included in this call only when no registered notifications against it
3   remain.

11.3.15 `pmix_server_notify_event_fn_t`

Summary

Notify the specified processes of an event.

Format

PMIx v2.0

```
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);
```

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

- **`PMIX_ERR_NOT_SUPPORTED`** , indicating that the host environment does not support the
  request, even though the function entry was provided in the server module - 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

11.3.16 `pmix_server_listener_fn_t`

Summary
Register a socket the host server can monitor for connection requests.

Format

PMIx v1.0

```
typedef pmix_status_t (*pmix_server_listener_fn_t)(
                                    int listening_sd,
                                    pmix_connection_cbfunc_t cbfunc,
                                    void *cbdata)
```

6   **IN incoming_sd**
7   (integer)
8   **IN cbfunc**
9   Callback function **pmix_connection_cbfunc_t** (function reference)
10   **IN cbdata**
11   (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.

11.3.17 `pmix_server_query_fn_t`

Summary
Query information from the resource manager.

Format

PMIx v2.0

```
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)
```

30   **IN proct**
31   **pmix_proc_t** structure of the requesting process (handle)
32   **IN queries**
33   Array of **pmix_query_t** structures (array of handles)

1   **IN   nqueries**
2   Number of elements in the *queries* array (integer)
3   **IN   cbfunc**
4   Callback function **pmix_info_cbfunc_t** (function reference)
5   **IN   cbdata**
6   Data to be passed to the callback function (memory reference)

7   Returns one of the following:

8   • **PMIX_SUCCESS** , indicating that the request is being processed by the host environment - result
9   will be returned in the provided *cbfunc*. Note that the host must not invoke the callback function
10   prior to returning from the API.

11   • **PMIX_OPERATION_SUCCEEDED** , indicating that the request was immediately processed and
12   returned *success* - the *cbfunc* will not be called

13   • **PMIX_ERR_NOT_SUPPORTED** , indicating that the host environment does not support the
14   request, even though the function entry was provided in the server module - the *cbfunc* will not
15   be called

16   • a PMIx error constant indicating either an error in the input or that the request was immediately
17   processed and failed - the *cbfunc* will not be called

▼------------------   Required Attributes   ------------------▼

18   PMIx libraries are required to pass any provided attributes to the host environment for processing.
19   In addition, the following attributes are required to be included in the passed *info* array:

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

22   **PMIX_GRPID   "pmix.egid"** (**uint32_t**)
23   Effective group id.

▲--▲

▼------------------   Optional Attributes   ------------------▼

24   The following attributes are optional for host environments that support this operation:

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

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

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

31   **PMIX_QUERY_QUEUE_STATUS   "pmix.qry.qst"** (**TBD**)
32   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 only average values for sampled information.

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.

▲

11.3.18 `pmix_server_tool_connection_fn_t`

Summary

Register that a tool has connected to the server.

Format

PMIx v2.0

```
typedef void (*pmix_server_tool_connection_fn_t)(
                              pmix_info_t info[], size_t ninfo,
                              pmix_tool_connection_cbfunc_t cbfunc,
                              void *cbdata)
```

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.

11.3.19 `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)

1 PMIx libraries are required to pass any provided attributes to the host environment for processing.
2 In addition, the following attributes are required to be included in the passed *info* array:

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

5 **PMIX_GRPID** **"pmix.egid"** (**uint32_t**)
6   Effective group id.

7 _____

8 Host environments that provide this module entry point are required to support the following
9 attributes:

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

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

14 **PMIX_LOG_SYSLOG** **"pmix.log.syslog"** (**char***)
15   Log data to syslog. Defaults to **ERROR** priority. Will log to global syslog if available,
16   otherwise to local syslog

17 The following attributes are optional for host environments that support this operation:

18 **PMIX_LOG_MSG** **"pmix.log.msg"** (**pmix_byte_object_t**)
19   Message blob to be sent somewhere.

20 **PMIX_LOG_EMAIL** **"pmix.log.email"** (**pmix_data_array_t**)
21   Log via email based on **pmix_info_t** containing directives.

22 **PMIX_LOG_EMAIL_ADDR** **"pmix.log.emaddr"** (**char***)
23   Comma-delimited list of email addresses that are to receive the message.

24 **PMIX_LOG_EMAIL_SUBJECT** **"pmix.log.emsub"** (**char***)
25   Subject line for email.

26 **PMIX_LOG_EMAIL_MSG** **"pmix.log.emmsg"** (**char***)
27   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.

11.3.20 `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

- **PMIX_ERR_NOT_SUPPORTED** , indicating that the host environment does not support the request, even though the function entry was provided in the server module - 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

1 PMIx libraries are required to pass any provided attributes to the host environment for processing.
2 In addition, the following attributes are required to be included in the passed *info* array:

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

5 **PMIX_GRPID** **"pmix.egid"** (**uint32_t**)
6   Effective group id.

7 _____

8 Host environments that provide this module entry point are required to support the following
9 attributes:

10 **PMIX_ALLOC_ID** **"pmix.alloc.id"** (**char***)
11   A string identifier (provided by the host environment) for the resulting allocation which can
12   later be used to reference the allocated resources in, for example, a call to **PMIx_Spawn** .

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

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

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

19 The following attributes are optional for host environments that support this operation:

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

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

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

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

28 **PMIX_ALLOC_FABRIC** **"pmix.alloc.net"** (**array**)
29   Array of **pmix_info_t** describing requested fabric resources. This must include at least:
30   **PMIX_ALLOC_FABRIC_ID** , **PMIX_ALLOC_FABRIC_TYPE** , and
31   **PMIX_ALLOC_FABRIC_ENDPTS** , plus whatever other descriptors are desired.

32 **PMIX_ALLOC_FABRIC_ID** **"pmix.alloc.netid"** (**char***)

1	The key to be used when accessing this requested fabric allocation. The allocation will be
2	returned/stored as a **`pmix_data_array_t`** of **`pmix_info_t`** indexed by this key and
3	containing at least one entry with the same key and the allocated resource description. The
4	type of the included value depends upon the fabric support. For example, a TCP allocation
5	might consist of a comma-delimited string of socket ranges such as
6	*"32000-32100,33005,38123-38146"*. Additional entries will consist of any provided
7	resource request directives, along with their assigned values. Examples include:
8	**`PMIX_ALLOC_FABRIC_TYPE`** - the type of resources provided;
9	**`PMIX_ALLOC_FABRIC_PLANE`** - if applicable, what plane the resources were assigned
10	from; **`PMIX_ALLOC_FABRIC_QOS`** - the assigned QoS; **`PMIX_ALLOC_BANDWIDTH`** -
11	the allocated bandwidth; **`PMIX_ALLOC_FABRIC_SEC_KEY`** - a security key for the
12	requested fabric allocation. NOTE: the assigned values may differ from those requested,
13	especially if **`PMIX_INFO_REQD`** was not set in the request.

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

16 **`PMIX_ALLOC_FABRIC_QOS`** **`"pmix.alloc.netqos"`** (**`char*`**)
17   Quality of service level.

▲ - ▲

Description

19 Request new allocation or modifications to an existing allocation on behalf of a client. Several
20 broad categories are envisioned, including the ability to:

21 • Request allocation of additional resources, including memory, bandwidth, and compute for an
22   existing allocation. Any additional allocated resources will be considered as part of the current
23   allocation, and thus will be released at the same time.
24 • Request a new allocation of resources. Note that the new allocation will be disjoint from (i.e., not
25   affiliated with) the allocation of the requestor - thus the termination of one allocation will not
26   impact the other.
27 • Extend the reservation on currently allocated resources, subject to scheduling availability and
28   priorities.
29 • Return no-longer-required resources to the scheduler. This includes the *loan* of resources back to
30   the scheduler with a promise to return them upon subsequent request.

31 The callback function provides a *status* to indicate whether or not the request was granted, and to
32 provide some information as to the reason for any denial in the **`pmix_info_cbfunc_t`** array of
33 **`pmix_info_t`** structures.

11.3.21   `pmix_server_job_control_fn_t`

Summary
36 Execute a job control action on behalf of a client.

2   **typedef pmix_status_t (*pmix_server_job_control_fn_t)(**
3   **const pmix_proc_t *requestor,**
4   **const pmix_proc_t targets[], size_t ntargets,**
5   **const pmix_info_t directives[], size_t ndirs,**
6   **pmix_info_cbfunc_t cbfunc, void *cbdata)**

7   **IN   requestor**
8   **pmix_proc_t** structure of requesting process (handle)
9   **IN   targets**
10   Array of proc structures (array of handles)
11   **IN   ntargets**
12   Number of elements in the *targets* array (integer)
13   **IN   directives**
14   Array of info structures (array of handles)
15   **IN   ndirs**
16   Number of elements in the *info* array (integer)
17   **IN   cbfunc**
18   Callback function **pmix_op_cbfunc_t** (function reference)
19   **IN   cbdata**
20   Data to be passed to the callback function (memory reference)

21   Returns one of the following:

22   • **PMIX_SUCCESS** , indicating that the request is being processed by the host environment - result
23   will be returned in the provided *cbfunc*. Note that the host must not invoke the callback function
24   prior to returning from the API.

25   • **PMIX_OPERATION_SUCCEEDED** , indicating that the request was immediately processed and
26   returned *success* - the *cbfunc* will not be called

27   • **PMIX_ERR_NOT_SUPPORTED** , indicating that the host environment does not support the
28   request, even though the function entry was provided in the server module - the *cbfunc* will not
29   be called

30   • a PMIx error constant indicating either an error in the input or that the request was immediately
31   processed and failed - the *cbfunc* will not be called

▼------------------   Required Attributes   ------------------▼

32   PMIx libraries are required to pass any attributes provided by the client to the host environment for
33   processing. In addition, the following attributes are required to be included in the passed *info* array:

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

```
1   PMIX_GRPID  "pmix.egid" (uint32_t)
2         Effective group id.
```

3 ──

4   Host environments that provide this module entry point are required to support the following
5   attributes:

```
6   PMIX_JOB_CTRL_ID  "pmix.jctrl.id" (char*)
7         Provide a string identifier for this request. The user can provide an identifier for the
8         requested operation, thus allowing them to later request status of the operation or to
9         terminate it. The host, therefore, shall track it with the request for future reference.

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

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

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

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

18  PMIX_JOB_CTRL_TERMINATE  "pmix.jctrl.term" (bool)
19        Politely terminate the specified processes.
```
▲---▲

▼----------------- Optional Attributes -----------------▼

20 The following attributes are optional for host environments that support this operation:

```
21  PMIX_JOB_CTRL_CANCEL  "pmix.jctrl.cancel" (char*)
22        Cancel the specified request - the provided request ID must match the
23        PMIX_JOB_CTRL_ID provided to a previous call to PMIx_Job_control . An ID of
24        NULL implies cancel all requests from this requestor.

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

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

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

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

33  PMIX_JOB_CTRL_CHECKPOINT_TIMEOUT  "pmix.jctrl.ckptsig" (int)
```

1	Time in seconds to wait for a checkpoint to complete.
2	**PMIX_JOB_CTRL_CHECKPOINT_METHOD**
3	**"pmix.jctrl.ckmethod"** (**pmix_data_array_t**)
4	Array of **pmix_info_t** declaring each method and value supported by this application.
5	**PMIX_JOB_CTRL_PROVISION** **"pmix.jctrl.pvn"** (**char***)
6	Regular expression identifying nodes that are to be provisioned.
7	**PMIX_JOB_CTRL_PROVISION_IMAGE** **"pmix.jctrl.pvnimg"** (**char***)
8	Name of the image that is to be provisioned.
9	**PMIX_JOB_CTRL_PREEMPTIBLE** **"pmix.jctrl.preempt"** (**bool**)
10	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_WILDCARD** 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.

11.3.22 **pmix_server_monitor_fn_t**

Summary

Request that a client be monitored for activity.

Format

PMIx v2.0 ▼―――――――――――――――――― C ――――――――――――――――――▼

```
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);
```

1	**IN** `requestor`
2	**pmix_proc_t** structure of requesting process (handle)
3	**IN** `monitor`
4	**pmix_info_t** identifying the type of monitor being requested (handle)
5	**IN** `error`
6	Status code to use in generating event if alarm triggers (integer)
7	**IN** `directives`
8	Array of info structures (array of handles)
9	**IN** `ndirs`
10	Number of elements in the *info* array (integer)
11	**IN** `cbfunc`
12	Callback function **pmix_op_cbfunc_t** (function reference)
13	**IN** `cbdata`
14	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

- **PMIX_ERR_NOT_SUPPORTED** , indicating that the host environment does not support the request, even though the function entry was provided in the server module - 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. Any attributes provided by the client that are 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.

1 ───

2   Host environments are not required to support any specific monitoring attributes.
   ▲--▲

   ▼------------------   Optional Attributes   ------------------▼

3   The following attributes may be implemented by a host environment.

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

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

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

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

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

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

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

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

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

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

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

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

28   **Description**
29   Request that a client be monitored for activity.

11.3.23 `pmix_server_get_cred_fn_t`

Summary

Request a credential from the host environment

Format

PMIx v3.0

```
typedef pmix_status_t (*pmix_server_get_cred_fn_t)(
                                   const pmix_proc_t *proc,
                                   const pmix_info_t directives[],
                                   size_t ndirs,
                                   pmix_credential_cbfunc_t cbfunc,
                                   void *cbdata);
```

IN   proc
    pmix_proc_t structure of requesting process (handle)
IN   directives
    Array of info structures (array of handles)
IN   ndirs
    Number of elements in the *info* array (integer)
IN   cbfunc
    Callback function to return the credential (**pmix_credential_cbfunc_t** function
    reference)
IN   cbdata
    Data to be passed to the callback function (memory reference)

Returns **PMIX_SUCCESS** , **PMIX_ERR_NOT_SUPPORTED** indicating that the host environment
does not support the request (even though the function entry was provided in the server module), or
a negative value corresponding to a PMIx error constant. In the event the function returns an error,
the *cbfunc* will not be called.

▼------------------ Required Attributes ------------------▼

If the PMIx library does not itself provide the requested credential, 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.
▲--▲

The following attributes are optional for host environments that support this operation:

PMIX_CRED_TYPE   **"pmix.sec.ctype"** (**char***)
    When passed in **PMIx_Get_credential** , a prioritized, comma-delimited list of desired
    credential types for use in environments where multiple authentication mechanisms may be
    available. When returned in a callback function, a string identifier of the credential type.

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
Request a credential from the host environment

11.3.24 `pmix_server_validate_cred_fn_t`

Summary
Request validation of a credential

Format

——————————————————— C ———————————————————

```
2    typedef pmix_status_t (*pmix_server_validate_cred_fn_t)(
3                                     const pmix_proc_t *proc,
4                                     const pmix_byte_object_t *cred,
5                                     const pmix_info_t directives[],
6                                     size_t ndirs,
7                                     pmix_validation_cbfunc_t cbfunc,
8                                     void *cbdata);
```

——————————————————— C ———————————————————

9   **IN   proc**
10   **pmix_proc_t** structure of requesting process (handle)
11   **IN   cred**
12   Pointer to **pmix_byte_object_t** containing the credential (handle)
13   **IN   directives**
14   Array of info structures (array of handles)
15   **IN   ndirs**
16   Number of elements in the *info* array (integer)
17   **IN   cbfunc**
18   Callback function to return the result (**pmix_validation_cbfunc_t** function
19   reference)
20   **IN   cbdata**
21   Data to be passed to the callback function (memory reference)

22   Returns one of the following:

23   • **PMIX_SUCCESS** , indicating that the request is being processed by the host environment - result
24   will be returned in the provided *cbfunc*

25   • **PMIX_OPERATION_SUCCEEDED** , indicating that the request was immediately processed and
26   returned *success* - the *cbfunc* will not be called

27   • **PMIX_ERR_NOT_SUPPORTED** , indicating that the host environment does not support the
28   request, even though the function entry was provided in the server module - the *cbfunc* will not
29   be called

30   • a PMIx error constant indicating either an error in the input or that the request was immediately
31   processed and failed - the *cbfunc* will not be called

▼----------------- Required Attributes -----------------▼

32   If the PMIx library does not itself validate the credential, then it is required to pass any attributes
33   provided by the client to the host environment for processing. In addition, it must include the
34   following attributes in the passed *info* array:

35   **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 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
Request validation of a credential obtained from the host environment via a prior call to the
pmix_server_get_cred_fn_t module entry.

11.3.25 **pmix_server_iof_fn_t**

Summary
Request the specified IO channels be forwarded from the given array of processes.

Format

———————————————— C ————————————————

```
2   typedef pmix_status_t (*pmix_server_iof_fn_t)(
3                                   const pmix_proc_t procs[], size_t nprocs,
4                                   const pmix_info_t directives[], size_t ndirs,
5                                   pmix_iof_channel_t channels,
6                                   pmix_op_cbfunc_t cbfunc, void *cbdata);
```

———————————————— C ————————————————

7   **IN   procs**
8   Array **pmix_proc_t** identifiers whose IO is being requested (handle)
9   **IN   nprocs**
10   Number of elements in *procs* (**size_t**)
11 **IN   directives**
12   Array of **pmix_info_t** structures further defining the request (array of handles)
13 **IN   ndirs**
14   Number of elements in the *info* array (integer)
15 **IN   channels**
16   Bitmask identifying the channels to be forwarded (**pmix_iof_channel_t**)
17 **IN   cbfunc**
18   Callback function **pmix_op_cbfunc_t** (function reference)
19 **IN   cbdata**
20   Data to be passed to the callback function (memory reference)

21 Returns one of the following:

22 • **PMIX_SUCCESS** , indicating that the request is being processed by the host environment - result
23   will be returned in the provided *cbfunc*. Note that the library must not invoke the callback
24   function prior to returning from the API.

25 • **PMIX_OPERATION_SUCCEEDED** , indicating that the request was immediately processed and
26   returned *success* - the *cbfunc* will not be called

27 • **PMIX_ERR_NOT_SUPPORTED** , indicating that the host environment does not support the
28   request, even though the function entry was provided in the server module - the *cbfunc* will not
29   be called

30 • a PMIx error constant indicating either an error in the input or that the request was immediately
31   processed and failed - the *cbfunc* will not be called

▼------------------   Required Attributes   ------------------▼

32 The following attributes are required to be included in the passed *info* array:

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

35 **PMIX_GRPID   "pmix.egid"** (**uint32_t**)

1   Effective group id.

2   _____

3   Host environments that provide this module entry point are required to support the following
4   attributes:

5   **PMIX_IOF_CACHE_SIZE** **"pmix.iof.csize"** (**uint32_t**)
6   The requested size of the server cache in bytes for each specified channel. By default, the
7   server is allowed (but not required) to drop all bytes received beyond the max size.

8   **PMIX_IOF_DROP_OLDEST** **"pmix.iof.old"** (**bool**)
9   In an overflow situation, drop the oldest bytes to make room in the cache.

10   **PMIX_IOF_DROP_NEWEST** **"pmix.iof.new"** (**bool**)
11   In an overflow situation, drop any new bytes received until room becomes available in the
12   cache (default).

▲---▲

▼----------------- Optional Attributes -----------------▼

13   The following attributes may be supported by a host environment.

14   **PMIX_IOF_BUFFERING_SIZE** **"pmix.iof.bsize"** (**uint32_t**)
15   Controls grouping of IO on the specified channel(s) to avoid being called every time a bit of
16   IO arrives. The library will execute the callback whenever the specified number of bytes
17   becomes available. Any remaining buffered data will be "flushed" upon call to deregister the
18   respective channel.

19   **PMIX_IOF_BUFFERING_TIME** **"pmix.iof.btime"** (**uint32_t**)
20   Max time in seconds to buffer IO before delivering it. Used in conjunction with buffering
21   size, this prevents IO from being held indefinitely while waiting for another payload to
22   arrive.

▲---▲

23   **Description**
24   Request the specified IO channels be forwarded from the given array of processes. An error shall be
25   returned in the callback function if the requested service from any of the requested processes cannot
26   be provided.

─────────── Advice to PMIx library implementers ───────────

27   The forwarding of stdin is a *push* process - processes cannot request that it be *pulled* from some
28   other source. Requests including the **PMIX_FWD_STDIN_CHANNEL** channel will return a
29   **PMIX_ERR_NOT_SUPPORTED** error.

11.3.26 `pmix_server_stdin_fn_t`

Summary

Pass standard input data to the host environment for transmission to specified recipients.

Format

PMIx v3.0

———————————————————— C ————————————————————

```
typedef pmix_status_t (*pmix_server_stdin_fn_t)(
                                 const pmix_proc_t *source,
                                 const pmix_proc_t targets[],
                                 size_t ntargets,
                                 const pmix_info_t directives[],
                                 size_t ndirs,
                                 const pmix_byte_object_t *bo,
                                 pmix_op_cbfunc_t cbfunc, void *cbdata);
```

———————————————————— C ————————————————————

IN   source
    pmix_proc_t structure of source process (handle)
IN   targets
    Array of **pmix_proc_t** target identifiers (handle)
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   bo
    Pointer to **pmix_byte_object_t** containing the payload (handle)
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

- **PMIX_ERR_NOT_SUPPORTED** , indicating that the host environment does not support the request, even though the function entry was provided in the server module - the *cbfunc* will not be called

1　　• a PMIx error constant indicating either an error in the input or that the request was immediately
2　　　processed and failed - the *cbfunc* will not be called

▼----------------　　Required Attributes　　----------------▼

3　　The following attributes are required to be included in the passed *info* array:

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

6　　**PMIX_GRPID** **"pmix.egid"** (**uint32_t**)
7　　　　Effective group id.

▲--▲

Description
9　　Passes stdin to the host environment for transmission to specified recipients. The host environment
10　　is responsible for forwarding the data to all locations that host the specified *targets* and delivering
11　　the payload to the PMIx server library connected to those clients.

11.3.27 `pmix_server_grp_fn_t`

Summary
14　　Request group operations (construct, destruct, etc.) on behalf of a set of processes.

Format
PMIx v4.0

▼――――――――――――― C ―――――――――――――▼
```
typedef pmix_status_t (*pmix_server_grp_fn_t)(
                                 pmix_group_operation_t op, char grp[],
                                 const pmix_proc_t procs[], size_t nprocs,
                                 const pmix_info_t directives[],
                                 size_t ndirs,
                                 pmix_info_cbfunc_t cbfunc, void *cbdata);
```
▲――――――――――――― C ―――――――――――――▲

22　　**IN**　op
23　　　　**pmix_group_operation_t** value indicating operation the host is requested to perform
24　　　　(integer)
25　　**IN**　grp
26　　　　Character string identifying the group (string)
27　　**IN**　procs
28　　　　Array of **pmix_proc_t** identifiers of participants (handle)
29　　**IN**　nprocs
30　　　　Number of elements in the *procs* array (integer)
31　　**IN**　directives
32　　　　Array of info structures (array of handles)

1   **IN**   **ndirs**
2   Number of elements in the *info* array (integer)
3   **IN**   **cbfunc**
4   Callback function **pmix_info_cbfunc_t** (function reference)
5   **IN**   **cbdata**
6   Data to be passed to the callback function (memory reference)

7 Returns one of the following:

8 • **PMIX_SUCCESS** , indicating that the request is being processed by the host environment - result
9   will be returned in the provided *cbfunc*. Note that the library must not invoke the callback
10   function prior to returning from the API.

11 • **PMIX_OPERATION_SUCCEEDED** , indicating that the request was immediately processed and
12   returned *success* - the *cbfunc* will not be called

13 • **PMIX_ERR_NOT_SUPPORTED** , indicating that the host environment does not support the
14   request, even though the function entry was provided in the server module - the *cbfunc* will not
15   be called

16 • a PMIx error constant indicating either an error in the input or that the request was immediately
17   processed and failed - the *cbfunc* will not be called

▼-----------------   Optional Attributes   -----------------▼

18 The following attributes may be supported by a host environment.

19 **PMIX_GROUP_ASSIGN_CONTEXT_ID** **"pmix.grp.actxid"** (**bool**)
20   Requests that the RM assign a new context identifier to the newly created group. The
21   identifier is an unsigned, **size_t** value that the RM guarantees to be unique across the range
22   specified in the request. Thus, the value serves as a means of identifying the group within
23   that range. If no range is specified, then the request defaults to **PMIX_RANGE_SESSION** .

24 **PMIX_GROUP_LOCAL_ONLY** **"pmix.grp.lcl"** (**bool**)
25   Group operation only involves local processes. PMIx implementations are *required* to
26   automatically scan an array of group members for local vs remote processes - if only local
27   processes are detected, the implementation need not execute a global collective for the
28   operation unless a context ID has been requested from the host environment. This can result
29   in significant time savings. This attribute can be used to optimize the operation by indicating
30   whether or not only local processes are represented, thus allowing the implementation to
31   bypass the scan. The default is false

32 **PMIX_GROUP_ENDPT_DATA** **"pmix.grp.endpt"** (**pmix_byte_object_t**)
33   Data collected to be shared during group construction

34 **PMIX_GROUP_OPTIONAL** **"pmix.grp.opt"** (**bool**)
35   Participation is optional - do not return an error if any of the specified processes terminate
36   without having joined. The default is false

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

3 The following attributes may be included in the host's response:

4   **PMIX_GROUP_ID** **"pmix.grp.id"** (**char***)
5   User-provided group identifier

6   **PMIX_GROUP_MEMBERSHIP** **"pmix.grp.mbrs"** (**pmix_data_array_t***)
7   Array of group member ID's

8   **PMIX_GROUP_CONTEXT_ID** **"pmix.grp.ctxid"** (**size_t**)
9   Context identifier assigned to the group by the host RM.

10   **PMIX_GROUP_ENDPT_DATA** **"pmix.grp.endpt"** (**pmix_byte_object_t**)
11   Data collected to be shared during group construction

▲--▲

Description

12

13 Perform the specified operation across the identified processes, plus any special actions included in
14 the directives. Return the result of any special action requests in the callback function when the
15 operation is completed. Actions may include a request (**PMIX_GROUP_ASSIGN_CONTEXT_ID**
16) that the host assign a unique numerical (size_t) ID to this group - if given, the **PMIX_RANGE**
17 attribute will specify the range across which the ID must be unique (default to
18 **PMIX_RANGE_SESSION**).

11.3.28 `pmix_server_fabric_fn_t`

Summary

21 Request fabric-related operations (e.g., information on a fabric) on behalf of a tool or other process.

Format

PMIx v4.0 ▼———————————————— C ————————————————▼

```
typedef pmix_status_t (*pmix_server_fabric_fn_t)(
                              const pmix_proc_t *requestor,
                              pmix_fabric_operation_t op,
                              const pmix_info_t directives[],
                              size_t ndirs,
                              pmix_info_cbfunc_t cbfunc, void *cbdata);
```

IN `requestor`
  `pmix_proc_t` identifying the requestor (handle)
IN `op`
  `pmix_fabric_operation_t` value indicating operation the host is requested to perform
  (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_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

- `PMIX_ERR_NOT_SUPPORTED`, indicating that the host environment does not support the
  request, even though the function entry was provided in the server module - 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 directives are required to be supported by all hosts to aid users in identifying the
fabric to whom the operation is to be applied:

`PMIX_FABRIC_VENDOR` `"pmix.fab.vndr"` (`string`)
  Name of fabric vendor (e.g., Amazon, Mellanox, Cray, Intel)

`PMIX_FABRIC_IDENTIFIER` `"pmix.fab.id"` (`string`)
  An identifier for the fabric (e.g., MgmtEthernet, Slingshot-11, OmniPath-1)

`PMIX_FABRIC_PLANE` `"pmix.fab.plane"` (`char*`)
  ID string of a fabric plane (e.g., CIDR for Ethernet). When used as a modifier in a request
  for information, specifies the plane whose information is to be returned. When used directly
  in a request, returns a `pmix_data_array_t` of string identifiers for all fabric planes in
  the system.

▲--▲

Description

Perform the specified operation. Return the result of any requests in the callback function when the operation is completed. Operations may, for example, include a request for fabric information. See `pmix_fabric_t` for a list of expected information to be included in the response.

Fabric Support Definitions

1   As the drive for performance continues, interest has grown in both scheduling algorithms that take
2   into account network locality of the allocated resources, and in optimizing collective
3   communication patterns by structuring them to follow fabric topology. Several interfaces have been
4   defined that are specifically intended to support WLMs (also known as *schedulers*) by providing
5   access to information of potential use to scheduling algorithms - e.g., information on
6   communication costs between different points on the fabric.

7   In contrast, hierarchical collective operations require each process have global information about
8   both its peers and the fabric. For example, one might aggregate the contribution from all processes
9   on a node, then again across all nodes on a common switch, and finally across all switches. Creating
10 such optimized patterns relies on detailed knowledge of the fabric location of each participant.

11 PMIx supports these efforts by defining datatypes and attributes by which fabric coordinates for
12 processes and devices can be obtained from the host SMS. When used in conjunction with the
13 PMIx *instant on* methods, this results in the ability of a process to obtain the fabric coordinate of all
14 other processes without incurring additional overhead associated with the publish/exchange of that
15 information.

12.1 Fabric Support Constants

17 The following constants are defined for use in fabric-related events.

18 **PMIX_FABRIC_UPDATE_PENDING**   The PMIx server library has been alerted to a change in
19   the fabric that requires updating of one or more registered **pmix_fabric_t** objects.
20 **PMIX_FABRIC_UPDATED**   The PMIx server library has completed updating the entries of all
21   affected **pmix_fabric_t** objects registered with the library. Access to the entries of those
22   objects may now resume.
23 **PMIX_FABRIC_COORDS_UPDATED**   Fabric coordinates have been updated - the affected
24   fabrics/planes are identified in the notification. Coordinates of processes and devices on those
25   affected components should be refreshed prior to next use.

12.2 Fabric Support Datatypes

27 Several datatype definitions have been created to support fabric-related operations and information.

12.2.1 Fabric Coordinate Structure

The **pmix_coord_t** structure describes the fabric coordinates of a specified process in a given view

──────────────────────────────── C ────────────────────────────────

```
typedef struct pmix_coord {
    char *fabric;
    char *plane;
    pmix_coord_view_t view;
    uint32_t *coord;
    size_t dims;
} pmix_coord_t;
```

──────────────────────────────── C ────────────────────────────────

All coordinate values shall be expressed as unsigned integers due to their units being defined in fabric devices and not physical distances. The coordinate is therefore an indicator of connectivity and not relative communication distance.

The fabric and plane fields are assigned by the fabric provider to help the user identify the fabric to which the coordinates refer. Note that providers are not required to assign any particular value to the fields and may choose to leave the fields blank. Example entries include {"Ethernet", "mgmt"} or {"infiniband", "data1"}.

──────────────── Advice to PMIx library implementers ────────────────

Note that the **pmix_coord_t** structure does not imply nor mandate any requirement on how the coordinate data is to be stored within the PMIx library. Implementers are free to store the coordinate in whatever format they choose.

A fabric coordinate is usually associated with a given fabric device - e.g., a particular NIC on a node. Thus, while the fabric coordinate of a device must be unique in a given view, the coordinate may be shared by multiple processes on a node. If the node contains multiple fabric devices, then either the device closest to the binding location of a process shall be used as its coordinate, or (if the process is unbound or its binding is not known) all devices on the node shall be reported as a **pmix_data_array_t** of **pmix_coord_t** structures.

Nodes with multiple fabric devices can also have those devices configured as multiple **fabric planes**. In such cases, a given process (even if bound to a specific location) may be associated with a coordinate on each plane. The resulting set of fabric coordinates shall be reported as a **pmix_data_array_t** of **pmix_coord_t** structures. The caller may request a coordinate from a specific fabric plane by passing the **PMIX_FABRIC_PLANE** attribute as a directive/qualifier to the **PMIx_Get** or **PMIx_Query_info_nb** call.

¹ ## 12.2.2 Fabric Coordinate Support Macros

² The following macros are provided to support the **`pmix_coord_t`** structure.

³ ### 12.2.2.1 Initialize the `pmix_coord_t` structure

⁴ Initialize the **`pmix_coord_t`** fields

PMIx v4.0 ──────────────── C ────────────────

⁵ **`PMIX_COORD_CONSTRUCT(m)`**

──────────────── C ────────────────

⁶ **IN m**
⁷ Pointer to the structure to be initialized (pointer to **`pmix_coord_t`**)

⁸ ### 12.2.2.2 Destruct the `pmix_coord_t` structure

⁹ Destruct the **`pmix_coord_t`** fields

PMIx v4.0 ──────────────── C ────────────────

¹⁰ **`PMIX_COORD_DESTRUCT(m)`**

──────────────── C ────────────────

¹¹ **IN m**
¹² Pointer to the structure to be destructed (pointer to **`pmix_coord_t`**)

¹³ ### 12.2.2.3 Create a `pmix_coord_t` array

¹⁴ Allocate and initialize a **`pmix_coord_t`** array

PMIx v4.0 ──────────────── C ────────────────

¹⁵ **`PMIX_COORD_CREATE(m, n)`**

──────────────── C ────────────────

¹⁶ **INOUT m**
¹⁷ Address where the pointer to the array of **`pmix_coord_t`** structures shall be stored (handle)
¹⁸ **IN n**
¹⁹ Number of structures to be allocated (**`size_t`**)

²⁰ ### 12.2.2.4 Release a `pmix_coord_t` array

²¹ Release an array of **`pmix_coord_t`** structures

PMIx v4.0 ──────────────── C ────────────────

²² **`PMIX_COORD_FREE(m, n)`**

──────────────── C ────────────────

²³ **IN m**
²⁴ Pointer to the array of **`pmix_coord_t`** structures (handle)
²⁵ **IN n**
²⁶ Number of structures in the array (**`size_t`**)

12.2.3 Fabric Coordinate Views

──────────────── C ────────────────

```
typedef uint8_t pmix_coord_view_t;
#define PMIX_COORD_VIEW_UNDEF        0x00
#define PMIX_COORD_LOGICAL_VIEW      0x01
#define PMIX_COORD_PHYSICAL_VIEW     0x02
```

──────────────── C ────────────────

Fabric coordinates can be reported based on different *views* according to user preference at the time of request. The following views have been defined:

PMIX_COORD_VIEW_UNDEF   The coordinate view has not been defined.
PMIX_COORD_LOGICAL_VIEW   The coordinates are provided in a *logical* view, typically given in Cartesian (x,y,z) dimensions, that describes the data flow in the fabric as defined by the arrangement of the hierarchical addressing scheme, fabric segmentation, routing domains, and other similar factors employed by that fabric.
PMIX_COORD_PHYSICAL_VIEW   The coordinates are provided in a *physical* view based on the actual wiring diagram of the fabric - i.e., values along each axis reflect the relative position of that interface on the specific fabric cabling.

──────────────── Advice to PMIx library implementers ────────────────

PMIx library implementers are advised to avoid declaring the above constants as actual **enum** values in order to allow host environments to add support for possibly proprietary coordinate views.

If the requester does not specify a view, coordinates shall default to the *logical* view.

12.2.4 Fabric Link State

The **pmix_link_state_t** is a **uint32_t** type for fabric link states.

──────────────── C ────────────────

```
typedef uint8_t pmix_link_state_t;
```

──────────────── C ────────────────

The following constants can be used to set a variable of the type **pmix_link_state_t** . All definitions were introduced in version 4 of the standard unless otherwise marked. Valid link state values start at zero.

PMIX_LINK_STATE_UNKNOWN   The port state is unknown or not applicable.
PMIX_LINK_DOWN   The port is inactive.
PMIX_LINK_UP   The port is active.

12.2.5 Fabric Operation Constants

PMIx v4.0 The `pmix_fabric_operation_t` structure is an enumerated type for specifying fabric operations used in the PMIx server module's `pmix_server_fabric_fn_t` API. All values were originally defined in version 4 of the standard unless otherwise marked.

PMIX_FABRIC_REQUEST_INFO Request information on a specific fabric - if the fabric isn't specified as per **PMIx_Fabric_register**, then return information on the system default fabric. Information to be returned is described in `pmix_fabric_t`.

PMIX_FABRIC_UPDATE_INFO Update information on a specific fabric - the index of the fabric (**PMIX_FABRIC_INDEX**) to be updated must be provided.

PMIX_FABRIC_GET_VERTEX_INFO Request information on a specific NIC within the identified fabric - the index of the device (**PMIX_FABRIC_DEVICE_INDEX**) and of the fabric (**PMIX_FABRIC_INDEX**) must be provided. If the NIC identifier is not specified, then return vertex info on all NICs in the fabric. Information to be included on each vertex is described in `pmix_fabric_t`.

———————————————————— Advice to users ————————————————————

Requesting information on every NIC in the fabric may be an expensive operation in terms of both memory footprint and time.

PMIX_FABRIC_GET_DEVICE_INDEX Request the fabric-wide index (returned as **PMIX_FABRIC_DEVICE_INDEX**) for a specific NIC within the identified fabric based on the provided vertex information. The index of the fabric must be provided.

12.2.6 Fabric registration structure

The `pmix_fabric_t` structure is used by a WLM to interact with fabric-related PMIx interfaces, and to provide information about the fabric for use in scheduling algorithms or other purposes.

PMIx v4.0 ————————————————————————— C —————————————————————————

```
typedef struct pmix_fabric_s {
    char *name;
    size_t index;
    pmix_info_t *info;
    size_t ninfo;
    void *module;
} pmix_fabric_t;;
```

Note that in this structure:

- *name* is an optional user-supplied string name identifying the fabric being referenced by this struct. If provided, the field must be a **NULL**-terminated string composed of standard alphanumeric values supported by common utilities such as *strcmp*.;

- *index* is a PMIx-provided number identifying this object;

- *info* is an array of **pmix_info_t** containing information (provided by the PMIx library) about the fabric;

- *ninfo* is the number of elements in the *info* array

- *module* points to an opaque object reserved for use by the PMIx server library.

Note that only the *name* field is provided by the user - all other fields are provided by the PMIx library and must not be modified by the user. The *info* array contains a varying amount of information depending upon both the PMIx implementation and information available from the fabric vendor. At a minimum, it must contain (ordering is arbitrary):

▼------------------ Required Attributes ------------------▼

PMIX_FABRIC_VENDOR "pmix.fab.vndr" (**string**)
Name of fabric vendor (e.g., Amazon, Mellanox, Cray, Intel)

PMIX_FABRIC_IDENTIFIER "pmix.fab.id" (**string**)
An identifier for the fabric (e.g., MgmtEthernet, Slingshot-11, OmniPath-1)

PMIX_FABRIC_NUM_VERTICES "pmix.fab.nverts" (**size_t**)
Total number of NICs in the system - corresponds to the number of vertices (i.e., rows and columns) in the cost matrix

▲--▲

and may optionally contain one or more of the following:

▼------------------ Optional Attributes ------------------▼

PMIX_FABRIC_COST_MATRIX "pmix.fab.cm" (**pointer**)
Pointer to a two-dimensional array of point-to-point relative communication costs expressed as **uint16_t** values

PMIX_FABRIC_GROUPS "pmix.fab.grps" (**string**)
A string delineating the group membership of nodes in the system, where each fabric group consists of the group number followed by a colon and a comma-delimited list of nodes in that group, with the groups delimited by semi-colons (e.g., 0:node000,node002,node004,node006;1:node001,node003,node005,node007)

PMIX_FABRIC_DIMS "pmix.fab.dims" (**uint32_t**)

Number of dimensions in the specified fabric plane/view. If no plane is specified in a
request, then the dimensions of all planes in the system will be returned as a
pmix_data_array_t containing an array of **uint32_t** values. Default is to provide
dimensions in *logical* view.

PMIX_FABRIC_PLANE   **"pmix.fab.plane"** (**char***)
    ID string of a fabric plane (e.g., CIDR for Ethernet). When used as a modifier in a request
    for information, specifies the plane whose information is to be returned. When used directly
    in a request, returns a **pmix_data_array_t** of string identifiers for all fabric planes in
    the system.

PMIX_FABRIC_SHAPE   **"pmix.fab.shape"** (**pmix_data_array_t***)
    The size of each dimension in the specified fabric plane/view, returned in a
    pmix_data_array_t containing an array of **uint32_t** values. The size is defined as
    the number of elements present in that dimension - e.g., the number of NICs in one
    dimension of a physical view of a fabric plane. If no plane is specified, then the shape of
    each plane in the system will be returned in an array of fabric shapes. Default is to provide
    the shape in *logical* view.

PMIX_FABRIC_SHAPE_STRING   **"pmix.fab.shapestr"** (**string**)
    Network shape expressed as a string (e.g., "10x12x2").

While unusual due to scaling issues, implementations may include an array of
PMIX_FABRIC_DEVICE elements describing the vertex information for each NIC in the system.
Each element shall contain a **pmix_data_array_t** of **pmix_info_t** values describing the
device. Each array may contain one or more of the following (ordering is arbitrary):

PMIX_FABRIC_DEVICE_NAME   **"pmix.fabdev.nm"** (**string**)
    The operating system name associated with the device. This may be a logical fabric interface
    name (e.g. eth0 or eno1) or an absolute filename.

PMIX_FABRIC_DEVICE_VENDOR   **"pmix.fabdev.vndr"** (**string**)
    Indicates the name of the vendor that distributes the NIC.

PMIX_FABRIC_DEVICE_ID   **"pmix.fabdev.devid"** (**string**)
    This is a vendor-provided identifier for the device or product.

PMIX_HOSTNAME   **"pmix.hname"** (**char***)
    Name of the host (e.g., where a specified process is running, or a given device is located).

PMIX_FABRIC_DEVICE_DRIVER   **"pmix.fabdev.driver"** (**string**)
    The name of the driver associated with the device

PMIX_FABRIC_DEVICE_FIRMWARE   **"pmix.fabdev.fmwr"** (**string**)
    The device's firmware version

PMIX_FABRIC_DEVICE_ADDRESS   **"pmix.fabdev.addr"** (**string**)
    The primary link-level address associated with the NIC, such as a Media Access
    Control (MAC) address. If multiple addresses are available, only one will be reported.

1 **PMIX_FABRIC_DEVICE_MTU** **"pmix.fabdev.mtu"** (**size_t**)
2   The maximum transfer unit of link level frames or packets, in bytes.

3 **PMIX_FABRIC_DEVICE_SPEED** **"pmix.fabdev.speed"** (**size_t**)
4   The active link data rate, given in bits per second.

5 **PMIX_FABRIC_DEVICE_STATE** **"pmix.fabdev.state"** (**pmix_link_state_t**)
6   The last available physical port state. Possible values are **PMIX_LINK_STATE_UNKNOWN** ,
7   **PMIX_LINK_DOWN** , and **PMIX_LINK_UP** , to indicate if the port state is unknown or not
8   applicable (unknown), inactive (down), or active (up).

9 **PMIX_FABRIC_DEVICE_TYPE** **"pmix.fabdev.type"** (**string**)
10   Specifies the type of fabric interface currently active on the device, such as Ethernet or
11   InfiniBand.

12 **PMIX_FABRIC_DEVICE_BUS_TYPE** **"pmix.fabdev.btyp"** (**string**)
13   The type of bus to which the device is attached (e.g., "PCI", "GEN-Z").

14 **PMIX_FABRIC_DEVICE_PCI_DEVID** **"pmix.fabdev.pcidevid"** (**string**)
15   A node-level unique identifier for a Peripheral Component Interconnect (PCI) device.
16   Provided only if the device is located on a PCI bus. The identifier is constructed as a
17   four-part tuple delimited by colons comprised of the PCI 16-bit domain, 8-bit bus, 8-bit
18   device, and 8-bit function IDs, each expressed in zero-extended hexadecimal form. Thus, an
19   example identifier might be "abc1:0f:23:01". The combination of node identifier (
20   **PMIX_HOSTNAME** or **PMIX_NODEID**) and **PMIX_FABRIC_DEVICE_PCI_DEVID**
21   shall be unique within the system.

▲- ▲

22 **12.3 Fabric Support Attributes**

23 The following attributes are used by the library supporting the system's WLM to either access or
24 return fabric-related information (e.g., as part of the **pmix_fabric_t** structure).

25 **PMIX_SERVER_SCHEDULER** **"pmix.srv.sched"** (**bool**)
26   Server requests access to WLM-supporting features - passed solely to the
27   **PMIx_server_init** API to indicate that the library is to be initialized for scheduler
28   support.
29 **PMIX_FABRIC_COST_MATRIX** **"pmix.fab.cm"** (**pointer**)
30   Pointer to a two-dimensional array of point-to-point relative communication costs expressed
31   as **uint16_t** values
32 **PMIX_FABRIC_GROUPS** **"pmix.fab.grps"** (**string**)
33   A string delineating the group membership of nodes in the system, where each fabric group
34   consists of the group number followed by a colon and a comma-delimited list of nodes in
35   that group, with the groups delimited by semi-colons (e.g.,
36   0:node000,node002,node004,node006;1:node001,node003,node005,node007)

The following attributes may be returned by calls to the scheduler-related APIs or in response to queries (e.g., **PMIx_Get** or **PMIx_Query_info**) made by processes or tools.

PMIX_FABRIC_VENDOR "pmix.fab.vndr" (**string**)
    Name of fabric vendor (e.g., Amazon, Mellanox, Cray, Intel)
PMIX_FABRIC_IDENTIFIER "pmix.fab.id" (**string**)
    An identifier for the fabric (e.g., MgmtEthernet, Slingshot-11, OmniPath-1)
PMIX_FABRIC_INDEX "pmix.fab.idx" (**size_t**)
    The index of the fabric as returned in **pmix_fabric_t**
PMIX_FABRIC_NUM_VERTICES "pmix.fab.nverts" (**size_t**)
    Total number of NICs in the system - corresponds to the number of vertices (i.e., rows and columns) in the cost matrix
PMIX_FABRIC_COORDINATE "pmix.fab.coord" (**pmix_data_array_t**)
    Fabric coordinate(s) of the specified process in the view and/or plane provided by the requester. If only one NIC has been assigned to the specified process, then the array will contain only one address. Otherwise, the array will contain the coordinates of all NICs available to the process in order of least to greatest distance from the process (NICs equally distant from the process will be listed in arbitrary order).
PMIX_FABRIC_VIEW "pmix.fab.view" (**pmix_coord_view_t**)
    Fabric coordinate view to be used for the requested coordinate - see **pmix_coord_view_t** for the list of accepted values.
PMIX_FABRIC_DIMS "pmix.fab.dims" (**uint32_t**)
    Number of dimensions in the specified fabric plane/view. If no plane is specified in a request, then the dimensions of all planes in the system will be returned as a **pmix_data_array_t** containing an array of **uint32_t** values. Default is to provide dimensions in *logical* view.
PMIX_FABRIC_PLANE "pmix.fab.plane" (**char***)
    ID string of a fabric plane (e.g., CIDR for Ethernet). When used as a modifier in a request for information, specifies the plane whose information is to be returned. When used directly in a request, returns a **pmix_data_array_t** of string identifiers for all fabric planes in the system.
PMIX_FABRIC_ENDPT "pmix.fab.endpt" (**pmix_data_array_t**)
    Fabric endpoints for a specified process. As multiple endpoints may be assigned to a given process (e.g., in the case where multiple NICs are associated with a package to which the process is bound), the returned values will be provided in a **pmix_data_array_t** - the returned data type of the individual values in the array varies by fabric provider.
PMIX_FABRIC_SHAPE "pmix.fab.shape" (**pmix_data_array_t***)
    The size of each dimension in the specified fabric plane/view, returned in a **pmix_data_array_t** containing an array of **uint32_t** values. The size is defined as the number of elements present in that dimension - e.g., the number of NICs in one dimension of a physical view of a fabric plane. If no plane is specified, then the shape of each plane in the system will be returned in an array of fabric shapes. Default is to provide the shape in *logical* view.
PMIX_FABRIC_SHAPE_STRING "pmix.fab.shapestr" (**string**)

1   Network shape expressed as a string (e.g., "10x12x2").
2   The following attributes are used to describe devices (a.k.a., NICs) attached to the fabric.

3   **PMIX_FABRIC_DEVICE "pmix.fabdev"** (**pmix_data_array_t**)
4   An array of **pmix_info_t** describing a particular fabric device (NIC).
5   **PMIX_FABRIC_DEVICE_INDEX "pmix.fabdev.idx"** (**uint32_t**)
6   System-unique index of a particular fabric device (NIC).
7   **PMIX_FABRIC_DEVICE_NAME "pmix.fabdev.nm"** (**string**)
8   The operating system name associated with the device. This may be a logical fabric interface
9   name (e.g. eth0 or eno1) or an absolute filename.
10 **PMIX_FABRIC_DEVICE_VENDOR "pmix.fabdev.vndr"** (**string**)
11   Indicates the name of the vendor that distributes the NIC.
12 **PMIX_FABRIC_DEVICE_BUS_TYPE "pmix.fabdev.btyp"** (**string**)
13   The type of bus to which the device is attached (e.g., "PCI", "GEN-Z").
14 **PMIX_FABRIC_DEVICE_ID "pmix.fabdev.devid"** (**string**)
15   This is a vendor-provided identifier for the device or product.
16 **PMIX_FABRIC_DEVICE_DRIVER "pmix.fabdev.driver"** (**string**)
17   The name of the driver associated with the device
18 **PMIX_FABRIC_DEVICE_FIRMWARE "pmix.fabdev.fmwr"** (**string**)
19   The device's firmware version
20 **PMIX_FABRIC_DEVICE_ADDRESS "pmix.fabdev.addr"** (**string**)
21   The primary link-level address associated with the NIC, such as a MAC address. If multiple
22   addresses are available, only one will be reported.
23 **PMIX_FABRIC_DEVICE_MTU "pmix.fabdev.mtu"** (**size_t**)
24   The maximum transfer unit of link level frames or packets, in bytes.
25 **PMIX_FABRIC_DEVICE_SPEED "pmix.fabdev.speed"** (**size_t**)
26   The active link data rate, given in bits per second.
27 **PMIX_FABRIC_DEVICE_STATE "pmix.fabdev.state"** (**pmix_link_state_t**)
28
29   The last available physical port state. Possible values are **PMIX_LINK_STATE_UNKNOWN** ,
30   **PMIX_LINK_DOWN** , and **PMIX_LINK_UP** , to indicate if the port state is unknown or not
31   applicable (unknown), inactive (down), or active (up).
32 **PMIX_FABRIC_DEVICE_TYPE "pmix.fabdev.type"** (**string**)
33   Specifies the type of fabric interface currently active on the device, such as Ethernet or
34   InfiniBand.
35 **PMIX_FABRIC_DEVICE_PCI_DEVID "pmix.fabdev.pcidevid"** (**string**)
36   A node-level unique identifier for a PCI device. Provided only if the device is located on a
37   PCI bus. The identifier is constructed as a four-part tuple delimited by colons comprised of
38   the PCI 16-bit domain, 8-bit bus, 8-bit device, and 8-bit function IDs, each expressed in
39   zero-extended hexadecimal form. Thus, an example identifier might be "abc1:0f:23:01". The
40   combination of node identifier (**PMIX_HOSTNAME** or **PMIX_NODEID**) and
41   **PMIX_FABRIC_DEVICE_PCI_DEVID** shall be unique within the system.

12.4 Fabric Support Functions

2 The following APIs allow the WLM to request specific services from the fabric subsystem via the
3 PMIx library.

————————————————— Advice to PMIx server hosts —————————————————

4 Due to their high cost in terms of execution, memory consumption, and interactions with other
5 SMS components (e.g., a fabric manager), it is strongly advised that the underlying implementation
6 of these APIs be restricted to a single PMIx server in a system that is supporting the SMS
7 component responsible for the scheduling of allocations (i.e., the system **scheduler**). The
8 **PMIX_SERVER_SCHEDULER** attribute can be used for this purpose to control the execution path.
9 Clients, tools, and other servers utilizing these functions are advised to have their requests
10 forwarded to the server supporting the scheduler using the **pmix_server_fabric_fn_t**
11 server module function, as needed.

12.4.1 `PMIx_Fabric_register`

13 **Summary**
14 Register for access to fabric-related information.

15 **Format**

PMIx v4.0 ————————————————————— C —————————————————————

```
16    pmix_status_t
17    PMIx_Fabric_register(pmix_fabric_t *fabric,
18                         const pmix_info_t directives[],
19                         size_t ndirs)
```

————————————————————— C —————————————————————

20 **IN fabric**
21   address of a **pmix_fabric_t** (backed by storage). User may populate the "name" field at
22   will - PMIx does not utilize this field (handle)
23 **IN directives**
24   an optional array of values indicating desired behaviors and/or fabric to be accessed. If **NULL**,
25   then the highest priority available fabric will be used (array of handles)
26 **IN ndirs**
27   Number of elements in the *directives* array (integer)

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

1 The following directives are required to be supported by all PMIx libraries to aid users in
2 identifying the fabric whose data is being sought:

3 **PMIX_FABRIC_PLANE** **"pmix.fab.plane"** (**char***)
4   ID string of a fabric plane (e.g., CIDR for Ethernet). When used as a modifier in a request
5   for information, specifies the plane whose information is to be returned. When used directly
6   in a request, returns a **pmix_data_array_t** of string identifiers for all fabric planes in
7   the system.

8 **PMIX_FABRIC_IDENTIFIER** **"pmix.fab.id"** (**string**)
9   An identifier for the fabric (e.g., MgmtEthernet, Slingshot-11, OmniPath-1)

10 **PMIX_FABRIC_VENDOR** **"pmix.fab.vndr"** (**string**)
11   Name of fabric vendor (e.g., Amazon, Mellanox, Cray, Intel)

▲--▲

12 **Description**
13 Register for access to fabric-related information, including the communication cost matrix. This
14 call must be made prior to requesting information from a fabric. The caller may request access to a
15 particular fabric using the vendor, type, or identifier, or to a specific **fabric plane** via the
16 **PMIX_FABRIC_PLANE** attribute - otherwise, the default fabric will be returned.

17 For performance reasons, the PMIx library does not provide thread protection for accessing the
18 information in the **pmix_fabric_t** structure. Instead, the PMIx implementation shall provide
19 two methods for coordinating updates to the provided fabric information:

20 • Users may periodically poll for updates using the **PMIx_Fabric_update** API

21 • Users may register for **PMIX_FABRIC_UPDATE_PENDING** events indicating that an update to
22   the cost matrix is pending. When received, users are required to terminate or pause any actions
23   involving access to the cost matrix before returning from the event. Completion of the
24   **PMIX_FABRIC_UPDATE_PENDING** event handler indicates to the PMIx library that the
25   fabric object's entries are available for updating. This may include releasing and re-allocating
26   memory as the number of vertices may have changed (e.g., due to addition or removal of one or
27   more NICs). When the update has been completed, the PMIx library will generate a
28   **PMIX_FABRIC_UPDATED** event indicating that it is safe to begin using the updated fabric
29   object(s).

30 There is no requirement that the caller exclusively use either one of these options. For example, the
31 user may choose to both register for fabric update events, but poll for an update prior to some
32 critical operation.

12.4.2 `PMIx_Fabric_update`

Summary

Update fabric-related information.

Format

PMIx v4.0

— C —

```
pmix_status_t
PMIx_Fabric_update(pmix_fabric_t *fabric)
```

— C —

IN   fabric
    address of a **pmix_fabric_t** (backed by storage) (handle)

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

Description

Update fabric-related information. This call can be made at any time to request an update of the fabric information contained in the provided **pmix_fabric_t** object. The caller is not allowed to access the provided **pmix_fabric_t** until the call has returned.

12.4.3 `PMIx_Fabric_deregister`

Summary

Deregister a fabric object.

Format

PMIx v4.0

— C —

```
pmix_status_t PMIx_Fabric_deregister(pmix_fabric_t *fabric)
```

— C —

IN   input
    address of a **pmix_fabric_t** (handle)

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

Description

Deregister a fabric object, providing an opportunity for the PMIx library to cleanup any information (e.g., cost matrix) associated with it. Contents of the provided **pmix_fabric_t** will be invalidated upon function return.

12.4.4 `PMIx_Fabric_get_vertex_info`

Summary

Given a communication cost matrix index for a specified fabric, return the corresponding vertex info.

Format

```
pmix_status_t
PMIx_Fabric_get_vertex_info(pmix_fabric_t *fabric, uint32_t index,
                            pmix_info_t **info, size_t *ninfo)
```

IN   fabric
    address of a **pmix_fabric_t** (handle)
IN   index
    vertex index (i.e., communication cost matrix row or column number) (integer)
INOUT info
    Address where a pointer to an array of **pmix_info_t** containing the results of the query
    can be returned (memory reference)
INOUT ninfo
    Address where the number of elements in *info* can be returned (handle)

Returns one of the following:

- **PMIX_SUCCESS** , indicating return of a valid value.

- **PMIX_ERR_BAD_PARAM** , indicating that the provided index is out of bounds.

- a PMIx error constant indicating either an error in the input or that the request failed.

Description

Query information about a specified vertex (fabric device, or NIC) in the system. The returned
status indicates if requested data was found or not. The returned array of **pmix_info_t** will
contain information on the specified vertex - the exact contents will depend on the PMIx
implementation and the fabric vendor. At a minimum, it must contain sufficient information to
uniquely identify the device within the system (ordering is arbitrary):

▼----------------- Required Attributes -----------------▼

PMIX_HOSTNAME **"pmix.hname"** (**char***)
    Name of the host (e.g., where a specified process is running, or a given device is located).
    The **PMIX_NODEID** may be returned in its place, or in addition to the hostname.

PMIX_FABRIC_DEVICE_NAME **"pmix.fabdev.nm"** (**string**)
    The operating system name associated with the device. This may be a logical fabric interface
    name (e.g. eth0 or eno1) or an absolute filename.

PMIX_FABRIC_DEVICE_VENDOR **"pmix.fabdev.vndr"** (**string**)
    Indicates the name of the vendor that distributes the NIC.

PMIX_FABRIC_DEVICE_BUS_TYPE **"pmix.fabdev.btyp"** (**string**)
    The type of bus to which the device is attached (e.g., "PCI", "GEN-Z").

PMIX_FABRIC_DEVICE_PCI_DEVID **"pmix.fabdev.pcidevid"** (**string**)

A node-level unique identifier for a PCI device. Provided only if the device is located on a PCI bus. The identifier is constructed as a four-part tuple delimited by colons comprised of the PCI 16-bit domain, 8-bit bus, 8-bit device, and 8-bit function IDs, each expressed in zero-extended hexadecimal form. Thus, an example identifier might be "abc1:0f:23:01". The combination of node identifier (**PMIX_HOSTNAME** or **PMIX_NODEID**) and **PMIX_FABRIC_DEVICE_PCI_DEVID** shall be unique within the system. This item should be included if the device bus type is PCI - the equivalent should be provided for any other bus type.

▲--▲

The returned array may optionally contain one or more of the following:

▼----------------- Optional Attributes -----------------▼

PMIX_FABRIC_DEVICE_ID **"pmix.fabdev.devid"** (**string**)
    This is a vendor-provided identifier for the device or product.

PMIX_FABRIC_DEVICE_DRIVER **"pmix.fabdev.driver"** (**string**)
    The name of the driver associated with the device

PMIX_FABRIC_DEVICE_FIRMWARE **"pmix.fabdev.fmwr"** (**string**)
    The device's firmware version

PMIX_FABRIC_DEVICE_ADDRESS **"pmix.fabdev.addr"** (**string**)
    The primary link-level address associated with the NIC, such as a MAC address. If multiple addresses are available, only one will be reported.

PMIX_FABRIC_DEVICE_MTU **"pmix.fabdev.mtu"** (**size_t**)
    The maximum transfer unit of link level frames or packets, in bytes.

PMIX_FABRIC_DEVICE_SPEED **"pmix.fabdev.speed"** (**size_t**)
    The active link data rate, given in bits per second.

PMIX_FABRIC_DEVICE_STATE **"pmix.fabdev.state"** (**pmix_link_state_t**)
    The last available physical port state. Possible values are **PMIX_LINK_STATE_UNKNOWN** , **PMIX_LINK_DOWN** , and **PMIX_LINK_UP** , to indicate if the port state is unknown or not applicable (unknown), inactive (down), or active (up).

PMIX_FABRIC_DEVICE_TYPE **"pmix.fabdev.type"** (**string**)
    Specifies the type of fabric interface currently active on the device, such as Ethernet or InfiniBand.

▲--▲

The caller is responsible for releasing the returned array.

₁ **12.4.5   `PMIx_Fabric_get_index`**

₂ **Summary**
₃ Given vertex info, return the corresponding communication cost matrix index.

₄ **Format**

PMIx v4.0
———————————————————————————— C ————————————————————————————
```
pmix_status_t
PMIx_Fabric_get_index(pmix_fabric_t *fabric,
                      const pmix_info_t vertex[], size_t ninfo,
                      uint32_t *index)
```
———————————————————————————— C ————————————————————————————

₉ **IN   fabric**
₁₀   address of a **`pmix_fabric_t`** (handle)
₁₁ **IN   vertex**
₁₂   array of **`pmix_info_t`** containing info describing the vertex whose index is being queried
₁₃   (handle)
₁₄ **IN   ninfo**
₁₅   number of elements in *vertex*
₁₆ **OUT index**
₁₇   pointer to the location where the index is to be returned (memory reference (handle))

₁₈ Returns one of the following:

₁₉ • **`PMIX_SUCCESS`** , indicating return of a valid value.

₂₀ • a PMIx error constant indicating either an error in the input or that the request failed.

₂₁ **Description**
₂₂ Query the index number of a vertex corresponding to the provided description. The description
₂₃ must provide adequate information to uniquely identify the target vertex. At a minimum, this must
₂₄ include identification of the node hosting the device using either the **`PMIX_HOSTNAME`** or
₂₅ **`PMIX_NODEID`** , plus a node-level unique identifier for the device (e.g., the
₂₆ **`PMIX_FABRIC_DEVICE_PCI_DEVID`** for a PCI device).

CHAPTER 13

Process Sets and Groups

---

1   PMIx supports two slightly related, but functionally different concepts known as *process sets* and
2   *process groups*. This chapter these two concepts and describes how they are utilized, along with
3   their corresponding APIs.

13.1   Process Sets

5   A PMIx *Process Set* is a user-provided label associated with a given set of application processes.
6   Definition of a PMIx process set typically occurs at time of application execution - e.g., on a
7   PRRTE command line:

<center>C</center>

8   ```
    $ prun -n 4 --pset ocean myoceanapp : -n 3 --pset ice myiceapp
    ```

<center>C</center>

9   In this example, the processes in the first application will be labeled with a **PMIX_PSET_NAME**
10 attribute of *ocean* while those in the second application will be labeled with an *ice* value. During
11 the execution, application processes could lookup the process set attribute for any other process
12 using **PMIx_Get** . Alternatively, other executing applications could utilize the
13 **PMIx_Query_info_nb** API to obtain the number of declared process sets in the system, a list
14 of their names, and other information about them. In other words, the *process set* identifier provides
15 a label by which an application can derive information about a process and its application - it does
16 *not*, however, confer any operational function.

17 Thus, process *sets* differ from process *groups* in several key ways:

18 • Process *sets* have no implied relationship between their members - i.e., a process in a process set
19   has no concept of a "pset rank" as it would in a process *group*

20 • Processes can only have one process *set* identifier, but can simultaneously belong to multiple
21   process *groups*

22 • Process *set* identifiers are considered job-level information set at launch. No PMIx API is
23   provided by which a user can change the process *set* value of a process on-the-fly. In contrast,
24   PMIx process *groups* can only be defined dynamically by the application.

1　• Process *groups* can be used in calls to PMIx operations. Members of process *groups* that are
2　　involved in an operation are translated by their PMIx server into their *native* identifier prior to the
3　　operation being passed to the host environment. For example, an application can define a process
4　　group to consist of ranks 0 and 1 from the host-assigned namespace of *210456*, identified by the
5　　group id of *foo*. If the application subsequently calls the **PMIx_Fence** API with a process
6　　identifier of {foo, PMIX_RANK_WILDCARD}, the PMIx server will replace that identifier
7　　with an array consisting of {210456, 0} and {210456, 1} - the host-assigned identifers of the
8　　participating processes - prior to passing the request up to the host environment

9　• Process *groups* can request that the host environment assign a unique **size_t** Process Group
10　　Context IDentifier (PGCID) to the group at time of group construction. An MPI library may, for
11　　example, use the PGCID as the MPI communicator identifier for the group.

12　The two concepts do, however, overlap in one specific area. Process *groups* are included in the
13　process *set* information returned by calls to **PMIx_Query_info_nb** . Thus, a *process group* can
14　effectively be considered an extended version of a *process set* that adds dynamic definition and
15　operational context to the *process set* concept.

───────────── Advice to PMIx library implementers ─────────────

16　PMIx implementations are required to include all active *group* identifiers in the returned list of
17　process *set* names provided in response to the appropriate **PMIx_Query_info_nb** call.

18 13.2 Process Groups

19　PMIx *Groups* are defined as a collection of processes desiring a common, unique identifier for
20　purposes such as passing events or participating in PMIx fence operations. As with processes that
21　assemble via **PMIx_Connect** , each member of the group is provided with both the job-level
22　information of any other namespace represented in the group, and the contact information for all
23　group members. However, *groups* differ from **PMIx_Connect** assemblages in the following key
24　areas:

25　• Relation to the host environment

26　– Calls to **PMIx_Connect** are relayed to the host environment. This means that the host RM
27　　should treat the failure of any process in the specified assemblage as a reportable event and
28　　take appropriate action. However, the environment is not required to define a new identifier for
29　　the connected assemblage or any of its member processes, nor does it define a new rank for
30　　each process within that assemblage. In addition, the PMIx server does not provide any
31　　tracking support for the assemblage. Thus, the caller is responsible for addressing members of
32　　the connected assemblage using their RM-provided identifiers.

- Calls to PMIx Group APIs are first processed within the local PMIx server. When constructed, the server creates a tracker that associates the specified processes with the user-provided group identifier, and assigns a new *group rank* based on their relative position in the array of processes provided in the call to `PMIx_Group_construct` . Members of the group can subsequently utilize the group identifier in PMIx function calls to address the group's members, using either `PMIX_RANK_WILDCARD` to refer to all of them or the group-level rank of specific members. The PMIx server will translate the specified processes into their RM-assigned identifiers prior to passing the request up to its host. Thus, the host environment has no visibility into the group's existence or membership.

—————————————————— Advice to users ——————————————————

User-provided group identifiers must be distinct from anything provided by the RM so as to avoid collisions between group identifiers and RM-assigned namespaces. This can usually be accomplished through the use of an application-specific prefix – e.g., "myapp-foo"

▲ ——— ▲

- Construction procedure

    - `PMIx_Connect` calls require that every process call the API before completing – i.e., it is modeled upon the bulk synchronous traditional MPI connect/accept methodology. Thus, a given application thread can only be involved in one connect/accept operation at a time, and is blocked in that operation until all specified processes participate. In addition, there is no provision for replacing processes in the assemblage due to failure to participate, nor a mechanism by which a process might decline participation.

    - PMIx Groups are designed to be more flexible in their construction procedure by relaxing these constraints. While a standard blocking form of constructing groups is provided, the event notification system is utilized to provide a designated *group leader* with the ability to replace participants that fail to participate within a given timeout period. This provides a mechanism by which the application can, if desired, replace members on-the-fly or allow the group to proceed with partial membership. In such cases, the final group membership is returned to all participants upon completion of the operation.

    Additionally, PMIx supports dynamic definition of group membership based on an invite/join model. A process can asynchronously initiate construction of a group of any processes via the `PMIx_Group_invite` function call. Invitations are delivered via a PMIx event (using the `PMIX_GROUP_INVITED` event) to the invited processes which can then either accept or decline the invitation using the `PMIx_Group_join` API. The initiating process tracks responses by registering for the events generated by the call to `PMIx_Group_join` , timeouts, or process terminations, optionally replacing processes that decline the invitation, fail to respond in time, or terminate without responding. Upon completion of the operation, the final list of participants is communicated to each member of the new group.

- Destruct procedure

1   – Processes that assemble via **PMIx_Connect** must all depart the assemblage together – i.e.,
2   no member can depart the assemblage while leaving the remaining members in it. Even the
3   non-blocking form of **PMIx_Disconnect** retains this requirement in that members remain
4   a part of the assemblage until all members have called **PMIx_Disconnect_nb**

5   – Members of a PMIx Group may depart the group at any time via the **PMIx_Group_leave**
6   API. Other members are notified of the departure via the **PMIX_GROUP_LEFT** event to
7   distinguish such events from those reporting process termination. This leaves the remaining
8   members free to continue group operations. The **PMIx_Group_destruct** operation offers
9   a collective method akin to **PMIx_Disconnect** for deconstructing the entire group.

10   Note that applications supporting dynamic group behaviors such as asynchronous departure
11   take responsibility for ensuring global consistency in the group definition prior to executing
12   group collective operations - i.e., it is the application's responsibility to either ensure that
13   knowledge of the current group membership is globally consistent across the participants, or to
14   register for appropriate events to deal with the lack of consistency during the operation.

15 In other words, members of PMIx Groups are *loosely coupled* as opposed to *tightly connected*
16 when constructed via **PMIx_Connect**. The relevant APIs are explained below.

———————————————————— Advice to users ————————————————————

17 The reliance on PMIx events in the PMIx Group concept dictates that processes utilizing these APIs
18 must register for the corresponding events. Failure to do so will likely lead to operational failures.
19 Users are recommended to utilize the **PMIX_TIMEOUT** directive (or retain an internal timer) on
20 calls to PMIx Group APIs (especially the blocking form of those functions) as processes that have
21 not registered for required events will never respond.

13.2.1 Group Operation Constants

23 *PMIx v4.0*   The **pmix_group_operation_t** structure is an enumerated type for specifying group
24 operations. All values were originally defined in version 4 of the standard unless otherwise marked.

25   **PMIX_GROUP_DECLINE**   Decline an invitation to join a PMIx group - provided for readability
26   of user code
27   **PMIX_GROUP_ACCEPT**   Accept an invitation to join a PMIx group - provided for readability
28   of user code
29   **PMIX_GROUP_CONSTRUCT**   Construct a group composed of the specified processes - used by
30   a PMIx server library to direct host operation
31   **PMIX_GROUP_DESTRUCT**   Destruct the specified group - used by a PMIx server library to
32   direct host operation

13.2.2 `PMIx_Group_construct`

Summary

Construct a PMIx process group

Format

PMIx v4.0

———————————————— C ————————————————

```
pmix_status_t
PMIx_Group_construct(const char grp[],
                     const pmix_proc_t procs[], size_t nprocs,
                     const pmix_info_t directives[], size_t ndirs,
                     pmix_info_t **results, size_t *nresults)
```

———————————————— C ————————————————

IN   grp
    NULL-terminated character array of maximum size **PMIX_MAX_NSLEN** containing the
    group identifier (string)

IN   procs
    Array of **pmix_proc_t** structures containing the PMIx identifiers of the member processes
    (array of handles)

IN   nprocs
    Number of elements in the *procs* array (**size_t**)

IN   directives
    Array of **pmix_info_t** structures (array of handles)

IN   ndirs
    Number of elements in the *directives* array (**size_t**)

INOUT results
    Pointer to a location where the array of **pmix_info_t** describing the results of the
    operation is to be returned (pointer to handle)

INOUT nresults
    Pointer to a **size_t** location where the number of elements in *results* is to be returned
    (memory reference)

Returns one of the following:

- **PMIX_SUCCESS** , indicating that the request has been successfully completed

- **PMIX_ERR_NOT_SUPPORTED** The PMIx library and/or the host RM does not support this
  operation

- a PMIx error constant indicating either an error in the input or that the request failed to be
  completed

1   The following attributes are *required* to be supported by all PMIx libraries that support this
2   operation:

3   **PMIX_GROUP_LEADER** **"pmix.grp.ldr"** (**bool**)
4   This process is the leader of the group

5   **PMIX_GROUP_OPTIONAL** **"pmix.grp.opt"** (**bool**)
6   Participation is optional - do not return an error if any of the specified processes terminate
7   without having joined. The default is false

8   **PMIX_GROUP_LOCAL_ONLY** **"pmix.grp.lcl"** (**bool**)
9   Group operation only involves local processes. PMIx implementations are *required* to
10   automatically scan an array of group members for local vs remote processes - if only local
11   processes are detected, the implementation need not execute a global collective for the
12   operation unless a context ID has been requested from the host environment. This can result
13   in significant time savings. This attribute can be used to optimize the operation by indicating
14   whether or not only local processes are represented, thus allowing the implementation to
15   bypass the scan. The default is false

16 Host environments that support this operation are *required* to provide the following attributes:

17 **PMIX_GROUP_ASSIGN_CONTEXT_ID** **"pmix.grp.actxid"** (**bool**)
18   Requests that the RM assign a new context identifier to the newly created group. The
19   identifier is an unsigned, **size_t** value that the RM guarantees to be unique across the range
20   specified in the request. Thus, the value serves as a means of identifying the group within
21   that range. If no range is specified, then the request defaults to **PMIX_RANGE_SESSION**.

22 **PMIX_GROUP_NOTIFY_TERMINATION** **"pmix.grp.notterm"** (**bool**)
23   Notify remaining members when another member terminates without first leaving the group.
24   The default is false

▲--▲

25 The following attributes are optional for host environments that support this operation:

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

▲--▲

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

Construct a new group composed of the specified processes and identified with the provided group identifier. The group identifier is a user-defined, **NULL**-terminated character array of length less than or equal to **PMIX_MAX_NSLEN** . Only characters accepted by standard string comparison functions (e.g., *strncmp*) are supported. Processes may engage in multiple simultaneous group construct operations so long as each is provided with a unique group ID. The *directives* array can be used to pass user-level directives regarding timeout constraints and other options available from the PMIx server.

If the **PMIX_GROUP_NOTIFY_TERMINATION** attribute is provided and has a value of **true**, then either the construct leader (if **PMIX_GROUP_LEADER** is provided) or all participants who register for the **PMIX_GROUP_MEMBER_FAILED** event will receive events whenever a process fails or terminates prior to calling **PMIx_Group_construct** – i.e. if a *group leader* is declared, *only* that process will receive the event. In the absence of a declared leader, *all* specified group members will receive the event.

The event will contain the identifier of the process that failed to join plus any other information that the host RM provided. This provides an opportunity for the leader or the collective members to react to the event – e.g., to decide to proceed with a smaller group or to abort the operation. The decision is communicated to the PMIx library in the results array at the end of the event handler. This allows PMIx to properly adjust accounting for procedure completion. When construct is complete, the participating PMIx servers will be alerted to any change in participants and each group member will receive an updated group membership (marked with the **PMIX_GROUP_MEMBERSHIP** attribute) as part of the *results* array returned by this API.

Failure of the declared leader at any time will cause a **PMIX_GROUP_LEADER_FAILED** event to be delivered to all participants so they can optionally declare a new leader. A new leader is identified by providing the **PMIX_GROUP_LEADER** attribute in the results array in the return of the event handler. Only one process is allowed to return that attribute, thereby declaring itself as the new leader. Results of the leader selection will be communicated to all participants via a **PMIX_GROUP_LEADER_SELECTED** event identifying the new leader. If no leader was selected, then the **pmix_info_t** provided to that event handler will include that information so the participants can take appropriate action.

Any participant that returns **PMIX_GROUP_CONSTRUCT_ABORT** from either the **PMIX_GROUP_MEMBER_FAILED** or the **PMIX_GROUP_LEADER_FAILED** event handler will

1 cause the construct process to abort, returning from the call with a
2 **PMIX_GROUP_CONSTRUCT_ABORT** status.

3 If the **PMIX_GROUP_NOTIFY_TERMINATION** attribute is not provided or has a value of
4 **false**, then the **PMIx_Group_construct** operation will simply return an error whenever a
5 proposed group member fails or terminates prior to calling **PMIx_Group_construct** .

6 Providing the **PMIX_GROUP_OPTIONAL** attribute with a value of **true** directs the PMIx library
7 to consider participation by any specified group member as non-required - thus, the operation will
8 return **PMIX_SUCCESS** if all members participate, or **PMIX_ERR_PARTIAL_SUCCESS** if
9 some members fail to participate. The *results* array will contain the final group membership in the
10 latter case. Note that this use-case can cause the operation to hang if the **PMIX_TIMEOUT**
11 attribute is not specified and one or more group members fail to call **PMIx_Group_construct**
12 while continuing to execute. Also, note that no leader or member failed events will be generated
13 during the operation.

14 Processes in a group under construction are not allowed to leave the group until group construction
15 is complete. Upon completion of the construct procedure, each group member will have access to
16 the job-level information of all namespaces represented in the group plus any information posted
17 via **PMIx_Put** (subject to the usual scoping directives) for every group member.

———————————— Advice to PMIx library implementers ————————————

18 At the conclusion of the construct operation, the PMIx library is *required* to ensure that job-related
19 information from each participating namespace plus any information posted by group members via
20 **PMIx_Put** (subject to scoping directives) is available to each member via calls to **PMIx_Get** .

———————————— Advice to PMIx server hosts ————————————

21 The collective nature of this API generally results in use of a fence-like operation by the backend
22 host environment. Host environments that utilize the array of process participants as a *signature* for
23 such operations may experience potential conflicts should both a **PMIx_Group_construct**
24 and a **PMIx_Fence** operation involving the same participants be simultaneously executed. As
25 PMIx allows for such use-cases, it is therefore the responsibility of the host environment to resolve
26 any potential conflicts.

27 13.2.3 **PMIx_Group_construct_nb**

28 **Summary**
29 Non-blocking form of **PMIx_Group_construct**

```
2    pmix_status_t
3    PMIx_Group_construct_nb(const char grp[],
4                            const pmix_proc_t procs[], size_t nprocs,
5                            const pmix_info_t directives[], size_t ndirs,
6                            pmix_info_cbfunc_t cbfunc, void *cbdata)
                                        C
```

7   **IN   grp**
8   **NULL**-terminated character array of maximum size **PMIX_MAX_NSLEN** containing the
9   group identifier (string)
10   **IN   procs**
11   Array of **pmix_proc_t** structures containing the PMIx identifiers of the member processes
12   (array of handles)
13   **IN   nprocs**
14   Number of elements in the *procs* array (**size_t**)
15   **IN   directives**
16   Array of **pmix_info_t** structures (array of handles)
17   **IN   ndirs**
18   Number of elements in the *directives* array (**size_t**)
19   **IN   cbfunc**
20   Callback function **pmix_info_cbfunc_t** (function reference)
21   **IN   cbdata**
22   Data to be passed to the callback function (memory reference)

23   Returns one of the following:

24   • **PMIX_SUCCESS** indicating that the request has been accepted for processing and the provided
25   callback function will be executed upon completion of the operation. Note that the library *must*
26   *not* invoke the callback function prior to returning from the API.

27   • **PMIX_OPERATION_SUCCEEDED** , indicating that the request was immediately processed and
28   returned *success* - the *cbfunc* will *not* be called

29   • **PMIX_ERR_NOT_SUPPORTED** The PMIx library does not support this operation - the *cbfunc*
30   will *not* be called

31   • a non-zero PMIx error constant indicating a reason for the request to have been rejected - the
32   *cbfunc* will *not* be called

33   If executed, the status returned in the provided callback function will be one of the following
34   constants:

35   • **PMIX_SUCCESS** The operation succeeded and all specified members participated.

1   • **PMIX_ERR_PARTIAL_SUCCESS** The operation succeeded but not all specified members
2   participated - the final group membership is included in the callback function

3   • **PMIX_ERR_NOT_SUPPORTED** While the PMIx server supports this operation, the host RM
4   does not.

5   • a non-zero PMIx error constant indicating a reason for the request's failure

▼------------------   Required Attributes   ------------------▼

6 PMIx libraries that choose not to support this operation *must* return
7 **PMIX_ERR_NOT_SUPPORTED** when the function is called.

8 The following attributes are *required* to be supported by all PMIx libraries that support this
9 operation:

10 **PMIX_GROUP_LEADER**   **"pmix.grp.ldr"** (**bool**)
11   This process is the leader of the group

12 **PMIX_GROUP_OPTIONAL**   **"pmix.grp.opt"** (**bool**)
13   Participation is optional - do not return an error if any of the specified processes terminate
14   without having joined. The default is false

15 **PMIX_GROUP_LOCAL_ONLY**   **"pmix.grp.lcl"** (**bool**)
16   Group operation only involves local processes. PMIx implementations are *required* to
17   automatically scan an array of group members for local vs remote processes - if only local
18   processes are detected, the implementation need not execute a global collective for the
19   operation unless a context ID has been requested from the host environment. This can result
20   in significant time savings. This attribute can be used to optimize the operation by indicating
21   whether or not only local processes are represented, thus allowing the implementation to
22   bypass the scan. The default is false

23 Host environments that support this operation are *required* to provide the following attributes:

24 **PMIX_GROUP_ASSIGN_CONTEXT_ID**   **"pmix.grp.actxid"** (**bool**)
25   Requests that the RM assign a new context identifier to the newly created group. The
26   identifier is an unsigned, **size_t** value that the RM guarantees to be unique across the range
27   specified in the request. Thus, the value serves as a means of identifying the group within
28   that range. If no range is specified, then the request defaults to **PMIX_RANGE_SESSION** .

29 **PMIX_GROUP_NOTIFY_TERMINATION**   **"pmix.grp.notterm"** (**bool**)
30   Notify remaining members when another member terminates without first leaving the group.
31   The default is false
▲--▲

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.

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 version of the **PMIx_Group_construct** operation. The callback function will
be called once all group members have called either **PMIx_Group_construct** or
PMIx_Group_construct_nb .

13.2.4 **PMIx_Group_destruct**

Summary
Destruct a PMIx process group

Format

```
pmix_status_t
PMIx_Group_destruct(const char grp[],
                    const pmix_info_t directives[], size_t ndirs)
```

C

IN   grp
  NULL-terminated character array of maximum size **PMIX_MAX_NSLEN** containing the
  identifier of the group to be destructed (string)
IN   directives
  Array of **pmix_info_t** structures (array of handles)
IN   ndirs
  Number of elements in the *directives* array (**size_t**)

Returns one of the following:

- **PMIX_SUCCESS** , indicating that the request has been successfully completed

- **PMIX_ERR_NOT_SUPPORTED** The PMIx library and/or the host RM does not support this
  operation

- a PMIx error constant indicating either an error in the input or that the request failed to be
  completed

▼------------------   Required Attributes   ------------------▼

For implementations and host environments that support the operation, there are no identified
required attributes for this API.

▲--▲

▼------------------   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.

▲--▲

1 We recommend that implementation of the **PMIX_TIMEOUT** attribute be left to the host
2 environment due to race condition considerations between completion of the operation versus
3 internal timeout in the PMIx server library. Implementers that choose to support **PMIX_TIMEOUT**
4 directly in the PMIx server library must take care to resolve the race condition and should avoid
5 passing **PMIX_TIMEOUT** to the host environment so that multiple competing timeouts are not
6 created.

7 **Description**
8 Destruct a group identified by the provided group identifier. Processes may engage in multiple
9 simultaneous group destruct operations so long as each involves a unique group ID. The *directives*
10 array can be used to pass user-level directives regarding timeout constraints and other options
11 available from the PMIx server.

12 The destruct API will return an error if any group process fails or terminates prior to calling
13 **PMIx_Group_destruct** or its non-blocking version unless the
14 **PMIX_GROUP_NOTIFY_TERMINATION** attribute was provided (with a value of **false**) at
15 time of group construction. If notification was requested, then the
16 **PMIX_GROUP_MEMBER_FAILED** event will be delivered for each process that fails to call
17 destruct and the destruct tracker updated to account for the lack of participation. The
18 **PMIx_Group_destruct** operation will subsequently return **PMIX_SUCCESS** when the
19 remaining processes have all called destruct – i.e., the event will serve in place of return of an error.

20 The collective nature of this API generally results in use of a fence-like operation by the backend
21 host environment. Host environments that utilize the array of process participants as a *signature* for
22 such operations may experience potential conflicts should both a **PMIx_Group_destruct** and
23 a **PMIx_Fence** operation involving the same participants be simultaneously executed. As PMIx
24 allows for such use-cases, it is therefore the responsibility of the host environment to resolve any
25 potential conflicts.

26 # 13.2.5 **PMIx_Group_destruct_nb**

27 **Summary**
28 Non-blocking form of **PMIx_Group_destruct**

Format

```
pmix_status_t
PMIx_Group_destruct_nb(const char grp[],
                       const pmix_info_t directives[], size_t ndirs,
                       pmix_op_cbfunc_t cbfunc, void *cbdata)
```

IN   **grp**
     NULL-terminated character array of maximum size **PMIX_MAX_NSLEN** containing the
     identifier of the group to be destructed (string)

IN   **directives**
     Array of **pmix_info_t** 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)

Returns one of the following:

- **PMIX_SUCCESS** , indicating that the request is being processed - 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

- **PMIX_ERR_NOT_SUPPORTED** The PMIx library does not support this operation - 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

If executed, the status returned in the provided callback function will be one of the following constants:

- **PMIX_SUCCESS** The operation was successfully completed

- **PMIX_ERR_NOT_SUPPORTED** While the PMIx server supports this operation, the host RM does not.

- a non-zero PMIx error constant indicating a reason for the request's failure

1   PMIx libraries that choose not to support this operation *must* return
2   **PMIX_ERR_NOT_SUPPORTED** when the function is called. For implementations and host
3   environments that support the operation, there are no identified required attributes for this API.

4   The following attributes are optional for host environments that support this operation:

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

9   We recommend that implementation of the **PMIX_TIMEOUT** attribute be left to the host
10   environment due to race condition considerations between completion of the operation versus
11   internal timeout in the PMIx server library. Implementers that choose to support **PMIX_TIMEOUT**
12   directly in the PMIx server library must take care to resolve the race condition and should avoid
13   passing **PMIX_TIMEOUT** to the host environment so that multiple competing timeouts are not
14   created.

Description

16   Non-blocking version of the **PMIx_Group_destruct** operation. The callback function will be
17   called once all members of the group have executed either **PMIx_Group_destruct** or
18   **PMIx_Group_destruct_nb**.

13.2.6 **PMIx_Group_invite**

Summary

21   Asynchronously construct a PMIx process group

Format

———————————————————————— C ————————————————————————

```
pmix_status_t
PMIx_Group_invite(const char grp[],
                  const pmix_proc_t procs[], size_t nprocs,
                  const pmix_info_t directives[], size_t ndirs,
                  pmix_info_t **results, size_t *nresult)
```

———————————————————————— C ————————————————————————

IN **grp**
   NULL-terminated character array of maximum size **PMIX_MAX_NSLEN** containing the
   group identifier (string)
IN **procs**
   Array of **pmix_proc_t** structures containing the PMIx identifiers of the processes to be
   invited (array of handles)
IN **nprocs**
   Number of elements in the *procs* array (**size_t**)
IN **directives**
   Array of **pmix_info_t** structures (array of handles)
IN **ndirs**
   Number of elements in the *directives* array (**size_t**)
INOUT results
   Pointer to a location where the array of **pmix_info_t** describing the results of the
   operation is to be returned (pointer to handle)
INOUT nresults
   Pointer to a **size_t** location where the number of elements in *results* is to be returned
   (memory reference)

Returns one of the following:

- **PMIX_SUCCESS** , indicating that the request has been successfully completed

- **PMIX_ERR_NOT_SUPPORTED** The PMIx library and/or the host RM does not support this
  operation

- a PMIx error constant indicating either an error in the input or that the request failed to be
  completed

▼----------------- Required Attributes -----------------▼

The following attributes are *required* to be supported by all PMIx libraries that support this
operation:

PMIX_GROUP_OPTIONAL **"pmix.grp.opt"** (**bool**)
   Participation is optional - do not return an error if any of the specified processes terminate
   without having joined. The default is false

Host environments that support this operation are *required* to provide the following attributes:

PMIX_GROUP_ASSIGN_CONTEXT_ID **"pmix.grp.actxid"** (**bool**)
> Requests that the RM assign a new context identifier to the newly created group. The identifier is an unsigned, **size_t** value that the RM guarantees to be unique across the range specified in the request. Thus, the value serves as a means of identifying the group within that range. If no range is specified, then the request defaults to **PMIX_RANGE_SESSION** .

PMIX_GROUP_NOTIFY_TERMINATION **"pmix.grp.notterm"** (**bool**)
> Notify remaining members when another member terminates without first leaving the group. The default is false

---------------------------------- 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

Explicitly invite the specified processes to join a group. The process making the
`PMIx_Group_invite` call is automatically declared to be the *group leader*. Each invited
process will be notified of the invitation via the `PMIX_GROUP_INVITED` event - the processes
being invited must therefore register for the `PMIX_GROUP_INVITED` event in order to be notified
of the invitation. Note that the PMIx event notification system caches events - thus, no ordering of
invite versus event registration is required.

The invitation event will include the identity of the inviting process plus the name of the group.
When ready to respond, each invited process provides a response using either the blocking or
non-blocking form of `PMIx_Group_join`. This will notify the inviting process that the
invitation was either accepted (via the `PMIX_GROUP_INVITE_ACCEPTED` event) or declined
(via the `PMIX_GROUP_INVITE_DECLINED` event). The `PMIX_GROUP_INVITE_ACCEPTED`
event is captured by the PMIx client library of the inviting process – i.e., the application itself does
not need to register for this event. The library will track the number of accepting processes and
alert the inviting process (by returning from the blocking form of `PMIx_Group_invite` or
calling the callback function of the non-blocking form) when group construction completes.

The inviting process should, however, register for the `PMIX_GROUP_INVITE_DECLINED` if the
application allows invited processes to decline the invitation. This provides an opportunity for the
application to either invite a replacement, declare "abort", or choose to remove the declining
process from the final group. The inviting process should also register to receive
`PMIX_GROUP_INVITE_FAILED` events whenever a process fails or terminates prior to
responding to the invitation. Actions taken by the inviting process in response to these events must
be communicated at the end of the event handler by returning the corresponding result so that the
PMIx library can adjust accordingly.

Upon completion of the operation, all members of the new group will receive access to the job-level
information of each other's namespaces plus any information posted via `PMIx_Put` by the other
members.

The inviting process is automatically considered the leader of the asynchronous group construction
procedure and will receive all failure or termination events for invited members prior to completion.
The inviting process is required to provide a `PMIX_GROUP_CONSTRUCT_COMPLETE` event
once the group has been fully assembled – this event is used by the PMIx library as a trigger to
release participants from their call to `PMIx_Group_join` and provides information (e.g., the
final group membership) to be returned in the *results* array.

———————————————— Advice to users ————————————————

Applications are not allowed to use the group in any operations until group construction is
complete. This is required in order to ensure consistent knowledge of group membership across all
participants.

1   Failure of the inviting process at any time will cause a **PMIX_GROUP_LEADER_FAILED** event to
2   be delivered to all participants so they can optionally declare a new leader. A new leader is
3   identified by providing the **PMIX_GROUP_LEADER** attribute in the results array in the return of
4   the event handler. Only one process is allowed to return that attribute, declaring itself as the new
5   leader. Results of the leader selection will be communicated to all participants via a
6   **PMIX_GROUP_LEADER_SELECTED** event identifying the new leader. If no leader was selected,
7   then the status code provided in the event handler will provide an error value so the participants can
8   take appropriate action.

9   13.2.7   **PMIx_Group_invite_nb**

10   **Summary**
11   Non-blocking form of **PMIx_Group_invite**

12   **Format**
     PMIx v4.0 ──────────────────────────────── C ────────────────────────────────

```
13   pmix_status_t
14   PMIx_Group_invite_nb(const char grp[],
15                        const pmix_proc_t procs[], size_t nprocs,
16                        const pmix_info_t directives[], size_t ndirs,
17                        pmix_info_cbfunc_t cbfunc, void *cbdata)
```
     ──────────────────────────────── C ────────────────────────────────

18   **IN   grp**
19   **NULL**-terminated character array of maximum size **PMIX_MAX_NSLEN** containing the
20   group identifier (string)
21   **IN   procs**
22   Array of **pmix_proc_t** structures containing the PMIx identifiers of the processes to be
23   invited (array of handles)
24   **IN   nprocs**
25   Number of elements in the *procs* array (**size_t**)
26   **IN   directives**
27   Array of **pmix_info_t** structures (array of handles)
28   **IN   ndirs**
29   Number of elements in the *directives* array (**size_t**)
30   **IN   cbfunc**
31   Callback function **pmix_info_cbfunc_t** (function reference)
32   **IN   cbdata**
33   Data to be passed to the callback function (memory reference)

34   Returns one of the following:

- **PMIX_SUCCESS** , indicating that the request is being processed - 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

- **PMIX_ERR_NOT_SUPPORTED** The PMIx library does not support this operation - 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

If executed, the status returned in the provided callback function will be one of the following constants:

- **PMIX_SUCCESS** The operation succeeded and all specified members participated.

- **PMIX_ERR_PARTIAL_SUCCESS** The operation succeeded but not all specified members participated - the final group membership is included in the callback function

- **PMIX_ERR_NOT_SUPPORTED** While the PMIx server supports this operation, the host RM does not.

- 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 that support this operation:

PMIX_GROUP_OPTIONAL **"pmix.grp.opt"** (**bool**)
> Participation is optional - do not return an error if any of the specified processes terminate without having joined. The default is false

Host environments that support this operation are *required* to provide the following attributes:

PMIX_GROUP_ASSIGN_CONTEXT_ID **"pmix.grp.actxid"** (**bool**)
> Requests that the RM assign a new context identifier to the newly created group. The identifier is an unsigned, **size_t** value that the RM guarantees to be unique across the range specified in the request. Thus, the value serves as a means of identifying the group within that range. If no range is specified, then the request defaults to **PMIX_RANGE_SESSION** .

PMIX_GROUP_NOTIFY_TERMINATION **"pmix.grp.notterm"** (**bool**)
> Notify remaining members when another member terminates without first leaving the group. The default is false

▲---▲

1   The following attributes are optional for host environments that support this operation:

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

6   We recommend that implementation of the **PMIX_TIMEOUT** attribute be left to the host
7   environment due to race condition considerations between completion of the operation versus
8   internal timeout in the PMIx server library. Implementers that choose to support **PMIX_TIMEOUT**
9   directly in the PMIx server library must take care to resolve the race condition and should avoid
10   passing **PMIX_TIMEOUT** to the host environment so that multiple competing timeouts are not
11   created.

12   **Description**
13   Non-blocking version of the **PMIx_Group_invite** operation. The callback function will be
14   called once all invited members of the group (or their substitutes) have executed either
15   **PMIx_Group_join** or **PMIx_Group_join_nb**.

16   ## 13.2.8 **PMIx_Group_join**

17   **Summary**
18   Accept an invitation to join a PMIx process group

Format

```
pmix_status_t
PMIx_Group_join(const char grp[],
                const pmix_proc_t *leader,
                pmix_group_operation_t opt,
                const pmix_info_t directives[], size_t ndirs,
                pmix_info_t **results, size_t *nresult)
```

IN   **grp**
  NULL-terminated character array of maximum size **PMIX_MAX_NSLEN** containing the
  group identifier (string)
IN   **leader**
  Process that generated the invitation (handle)
IN   **opt**
  Accept or decline flag (**pmix_group_operation_t**)
IN   **directives**
  Array of **pmix_info_t** structures (array of handles)
IN   **ndirs**
  Number of elements in the *directives* array (**size_t**)
INOUT **results**
  Pointer to a location where the array of **pmix_info_t** describing the results of the
  operation is to be returned (pointer to handle)
INOUT **nresults**
  Pointer to a **size_t** location where the number of elements in *results* is to be returned
  (memory reference)

Returns one of the following:

- **PMIX_SUCCESS** , indicating that the request has been successfully completed

- **PMIX_ERR_NOT_SUPPORTED** The PMIx library and/or the host RM does not support this
  operation

- a PMIx error constant indicating either an error in the input or that the request failed to be
  completed

------------------ Required Attributes ------------------

There are no identified required attributes for implementers.

1   The following attributes are optional for host environments that support this operation:

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

----------------- Advice to PMIx library implementers -----------------

6   We recommend that implementation of the **PMIX_TIMEOUT** attribute be left to the host
7   environment due to race condition considerations between completion of the operation versus
8   internal timeout in the PMIx server library. Implementers that choose to support **PMIX_TIMEOUT**
9   directly in the PMIx server library must take care to resolve the race condition and should avoid
10   passing **PMIX_TIMEOUT** to the host environment so that multiple competing timeouts are not
11   created.

12   **Description**
13   Respond to an invitation to join a group that is being asynchronously constructed. The process must
14   have registered for the **PMIX_GROUP_INVITED** event in order to be notified of the invitation.
15   When called, the event information will include the **pmix_proc_t** identifier of the process that
16   generated the invitation along with the identifier of the group being constructed. When ready to
17   respond, the process provides a response using either form of **PMIx_Group_join** .

----------------- Advice to users -----------------

18   Since the process is alerted to the invitation in a PMIx event handler, the process *must not* use the
19   blocking form of this call unless it first "thread shifts" out of the handler and into its own thread
20   context. Likewise, while it is safe to call the non-blocking form of the API from the event handler,
21   the process *must not* block in the handler while waiting for the callback function to be called.

1   Calling this function causes the inviting process (aka the *group leader*) to be notified that the
2   process has either accepted or declined the request. The blocking form of the API will return once
3   the group has been completely constructed or the group's construction has failed (as described
4   below) – likewise, the callback function of the non-blocking form will be executed upon the same
5   conditions.

6   Failure of the leader during the call to **PMIx_Group_join** will cause a
7   **PMIX_GROUP_LEADER_FAILED** event to be delivered to all invited participants so they can
8   optionally declare a new leader. A new leader is identified by providing the
9   **PMIX_GROUP_LEADER** attribute in the results array in the return of the event handler. Only one
10 process is allowed to return that attribute, declaring itself as the new leader. Results of the leader
11 selection will be communicated to all participants via a **PMIX_GROUP_LEADER_SELECTED**
12 event identifying the new leader. If no leader was selected, then the status code provided in the
13 event handler will provide an error value so the participants can take appropriate action.

14 Any participant that returns **PMIX_GROUP_CONSTRUCT_ABORT** from the leader failed event
15 handler will cause all participants to receive an event notifying them of that status. Similarly, the
16 leader may elect to abort the procedure by either returning **PMIX_GROUP_CONSTRUCT_ABORT**
17 from the handler assigned to the **PMIX_GROUP_INVITE_ACCEPTED** or
18 **PMIX_GROUP_INVITE_DECLINED** codes, or by generating an event for the abort code. Abort
19 events will be sent to all invited participants.

20 ## 13.2.9 **PMIx_Group_join_nb**

21 **Summary**
22 Non-blocking form of **PMIx_Group_join**

23 **Format**

PMIx v4.0

——————————————————— C ———————————————————

```
24  pmix_status_t
25  PMIx_Group_join_nb(const char grp[],
26                     const pmix_proc_t *leader,
27                     pmix_group_operation_t opt,
28                     const pmix_info_t directives[], size_t ndirs,
29                     pmix_info_cbfunc_t cbfunc, void *cbdata)
```

——————————————————— C ———————————————————

30 **IN   grp**
31   **NULL**-terminated character array of maximum size **PMIX_MAX_NSLEN** containing the
32   group identifier (string)
33 **IN   leader**
34   Process that generated the invitation (handle)

1	**IN** **opt**
2	Accept or decline flag (**`pmix_group_operation_t`**)
3	**IN** **directives**
4	Array of **`pmix_info_t`** structures (array of handles)
5	**IN** **ndirs**
6	Number of elements in the *directives* array (**`size_t`**)
7	**IN** **cbfunc**
8	Callback function **`pmix_info_cbfunc_t`** (function reference)
9	**IN** **cbdata**
10	Data to be passed to the callback function (memory reference)

11 Returns one of the following:

12 • **PMIX_SUCCESS** , indicating that the request is being processed - result will be returned in the
13   provided *cbfunc*. Note that the library *must not* invoke the callback function prior to returning
14   from the API.

15 • **PMIX_OPERATION_SUCCEEDED** , indicating that the request was immediately processed and
16   returned *success* - the *cbfunc* will *not* be called

17 • **PMIX_ERR_NOT_SUPPORTED** The PMIx library does not support this operation - the *cbfunc*
18   will *not* be called

19 • a PMIx error constant indicating either an error in the input or that the request was immediately
20   processed and failed - the *cbfunc* will *not* be called

21 If executed, the status returned in the provided callback function will be one of the following
22 constants:

23 • **PMIX_SUCCESS** The operation succeeded and group membership is in the callback function
24   parameters

25 • **PMIX_ERR_NOT_SUPPORTED** While the PMIx server supports this operation, the host RM
26   does not.

27 • a non-zero PMIx error constant indicating a reason for the request's failure

▼------------------   Required Attributes   ------------------▼

28 There are no identified required attributes for implementers.

▲--▲

▼------------------   Optional Attributes   ------------------▼

29 The following attributes are optional for host environments that support this operation:

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

1   We recommend that implementation of the **PMIX_TIMEOUT** attribute be left to the host
2   environment due to race condition considerations between completion of the operation versus
3   internal timeout in the PMIx server library. Implementers that choose to support **PMIX_TIMEOUT**
4   directly in the PMIx server library must take care to resolve the race condition and should avoid
5   passing **PMIX_TIMEOUT** to the host environment so that multiple competing timeouts are not
6   created.
----

7   **Description**
8   Non-blocking version of the **PMIx_Group_join** operation. The callback function will be called
9   once all invited members of the group (or their substitutes) have executed either
10 **PMIx_Group_join** or **PMIx_Group_join_nb** .

13.2.10 **PMIx_Group_leave**

12 **Summary**
13 Leave a PMIx process group

14 **Format**

PMIx v4.0 ————————————————————————— C —————————————————————————

```
15  pmix_status_t
16  PMIx_Group_leave(const char grp[],
17                   const pmix_info_t directives[], size_t ndirs)
```

————————————————————————— C —————————————————————————

18 **IN   grp**
19   **NULL**-terminated character array of maximum size **PMIX_MAX_NSLEN** containing the
20   group identifier (string)
21 **IN   directives**
22   Array of **pmix_info_t** structures (array of handles)
23 **IN   ndirs**
24   Number of elements in the *directives* array (**size_t**)

25 Returns one of the following:

26 • **PMIX_SUCCESS** , indicating that the request has been communicated to the local PMIx server

27 • **PMIX_ERR_NOT_SUPPORTED** The PMIx library and/or the host RM does not support this
28   operation

29 • a PMIx error constant indicating either an error in the input or that the request is unsupported

▼----------------- Required Attributes -----------------▼

30 There are no identified required attributes for implementers.
▲---▲

	Description
1	**Description**
2	Leave a PMIx Group. Calls to **PMIx_Group_leave** (or its non-blocking form) will cause a
3	**PMIX_GROUP_LEFT** event to be generated notifying all members of the group of the caller's
4	departure. The function will return (or the non-blocking function will execute the specified callback
5	function) once the event has been locally generated and is not indicative of remote receipt.

-- Advice to users --

6	The PMIx_Group_leave API is intended solely for asynchronous departures of individual processes
7	from a group as it is not a scalable operation – i.e., when a process determines it should no longer
8	be a part of a defined group, but the remainder of the group retains a valid reason to continue in
9	existence. Developers are advised to use PMIx_Group_destruct (or its non-blocking form) for all
10	other scenarios as it represents a more scalable operation.

13.2.11 `PMIx_Group_leave_nb`

12	**Summary**
13	Non-blocking form of **PMIx_Group_leave**

14	**Format**

PMIx v4.0 ──────────────────── C ────────────────────

```
pmix_status_t
PMIx_Group_leave_nb(const char grp[],
                    const pmix_info_t directives[], size_t ndirs,
                    pmix_op_cbfunc_t cbfunc, void *cbdata)
```

──────────────────── C ────────────────────

19	**IN grp**
20	**NULL**-terminated character array of maximum size **PMIX_MAX_NSLEN** containing the
21	group identifier (string)
22	**IN directives**
23	Array of **pmix_info_t** structures (array of handles)
24	**IN ndirs**
25	Number of elements in the *directives* array (**size_t**)
26	**IN cbfunc**
27	Callback function **pmix_op_cbfunc_t** (function reference)
28	**IN cbdata**
29	Data to be passed to the callback function (memory reference)

30	Returns one of the following:

31	• **PMIX_SUCCESS** , indicating that the request is being processed - result will be returned in the
32	provided *cbfunc*. Note that the library *must not* invoke the callback function prior to returning
33	from the API.

- **PMIX_OPERATION_SUCCEEDED** , indicating that the request was immediately processed and returned *success* - the *cbfunc* will *not* be called

- **PMIX_ERR_NOT_SUPPORTED** The PMIx library does not support this operation - 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

If executed, the status returned in the provided callback function will be one of the following constants:

- **PMIX_SUCCESS** The operation succeeded - i.e., the **PMIX_GROUP_LEFT** event was generated

- **PMIX_ERR_NOT_SUPPORTED** While the PMIx library supports this operation, the host RM does not.

- a non-zero PMIx error constant indicating a reason for the request's failure

▼------------------ Required Attributes ------------------▼

There are no identified required attributes for implementers.
▲--▲

Description

Non-blocking version of the **PMIx_Group_leave** operation. The callback function will be called once the event has been locally generated and is not indicative of remote receipt.

Data Structures and Types

1 This chapter defines PMIx standard data structures (along with macros for convenient use), types,
2 and constants. These apply to all consumers of the PMIx interface. Where necessary for
3 clarification, the description of, for example, an attribute may be copied from this chapter into a
4 section where it is used.

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

─────────────── Advice to PMIx library implementers ───────────────

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

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

─────────────────────── Advice to users ───────────────────────

13 Use of increment/decrement operations on indices inside PMIx macros is discouraged due to
14 unpredictable behavior. For example, the following sequence:

15 `PMIX_INFO_LOAD(&array[n++], "mykey", &mystring, PMIX_STRING);`
16 `PMIX_INFO_LOAD(&array[n++], "mykey2", &myint, PMIX_INT);`

17 will load the given key-values into incorrect locations if the macro is implemented as:

```
18 define PMIX_INFO_LOAD(m, k, v, t)                          \
19   do {                                                    \
20     if (NULL != (k)) {                                    \
21       pmix_strncpy((m)->key, (k), PMIX_MAX_KEYLEN);       \
22     }                                                     \
23     (m)->flags = 0;                                       \
24     pmix_value_load(&((m)->value), (v), (t));             \
25   } while (0)
```

26 since the index is cited more than once in the macro. The PMIx standard only governs the existence
27 and syntax of macros - it does not specify their implementation. Given the freedom of
28 implementation, a safer call sequence might be as follows:

```
1    PMIX_INFO_LOAD(&array[n], "mykey", &mystring, PMIX_STRING);
2    ++n;
3    PMIX_INFO_LOAD(&array[n], "mykey2", &myint, PMIX_INT);
4    ++n;
```

5   14.1 Constants

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

11   **PMIX_MAX_NSLEN**   Maximum namespace string length as an integer.

———————————— Advice to PMIx library implementers ————————————

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

15   **PMIX_MAX_KEYLEN**   Maximum key string length as an integer.
16   **PMIX_APP_WILDCARD**   A value to indicate that the user wants the data for the given key from
17   every application that posted that key, or that the given value applies to all applications within
18   the given nspace.

———————————— Advice to PMIx library implementers ————————————

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

14.1.1 PMIx 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`** . Values in the list that were deprecated in later standards are denoted as such. Values added to the list in this version of the standard are shown in **magenta**.

———————————— Advice to PMIx library implementers ————————————

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.

14.1.1.1 General Error Constants

These are general constants originally defined in versions 1 and 2 of the PMIx Standard.

`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
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.

1	**PMIX_MONITOR_HEARTBEAT_ALERT**	Job monitoring: Heartbeat alert
2	**PMIX_MONITOR_FILE_ALERT**	Job monitoring: File alert
3	**PMIX_PROC_TERMINATED**	Process terminated - can be either normal or abnormal
4	termination	
5	**PMIX_ERR_INVALID_TERMINATION**	Process terminated without calling
6	**PMIx_Finalize** , or was a member of an assemblage formed via **PMIx_Connect** and	
7	terminated or called **PMIx_Finalize** without first calling **PMIx_Disconnect** (or its	
8	non-blocking form) from that assemblage.	

9 14.1.1.2 Operational Error Constants

10	**PMIX_ERR_EVENT_REGISTRATION**	Error in event registration
11	**PMIX_ERR_JOB_TERMINATED**	Error job terminated
12	**PMIX_ERR_UPDATE_ENDPOINTS**	Error updating endpoints
13	**PMIX_MODEL_DECLARED**	Model declared
14	**PMIX_GDS_ACTION_COMPLETE**	The GDS action has completed
15	**PMIX_ERR_INVALID_OPERATION**	The requested operation is supported by the
16	implementation and host environment, but fails to meet a requirement (e.g., requesting to	
17	*disconnect* from processes without first *connecting* to them).	
18	**PMIX_PROC_HAS_CONNECTED**	A tool or client has connected to the PMIx server
19	**PMIX_CONNECT_REQUESTED**	Connection has been requested by a PMIx-based tool
20	**PMIX_MODEL_RESOURCES**	Resource usage by a programming model has changed
21	**PMIX_OPENMP_PARALLEL_ENTERED**	An OpenMP parallel code region has been entered
22	**PMIX_OPENMP_PARALLEL_EXITED**	An OpenMP parallel code region has completed
23	**PMIX_LAUNCH_DIRECTIVE**	Launcher directives have been received from a PMIx-enabled
24	tool	
25	**PMIX_LAUNCHER_READY**	Application launcher (e.g., mpiexec) is ready to receive directives
26	from a PMIx-enabled tool	
27	**PMIX_LAUNCH_COMPLETE**	A job has been launched - the nspace of the launched job will be
28	included in the notification	
29	**PMIX_OPERATION_IN_PROGRESS**	A requested operation is already in progress
30	**PMIX_OPERATION_SUCCEEDED**	The requested operation was performed atomically - no
31	callback function will be executed	
32	**PMIX_ERR_PARTIAL_SUCCESS**	The operation is considered successful but not all elements
33	of the operation were concluded (e.g., some members of a group construct operation chose	
34	not to participate)	
35	**PMIX_ERR_DUPLICATE_KEY**	The provided key has already been published on a different
36	data range	
37	**PMIX_ERR_INVALID_OPERATION**	The requested operation is not valid - this can possibly
38	indicate the inclusion of conflicting directives or a request to perform an operation that	
39	conflicts with an ongoing one.	
40	**PMIX_GROUP_INVITED**	The process has been invited to join a PMIx Group - the identifier of
41	the group and the ID's of other invited (or already joined) members will be included in the	
42	notification	

PMIX_GROUP_LEFT A process has asynchronously left a PMIx Group - the process identifier of the departing process will in included in the notification

PMIX_GROUP_MEMBER_FAILED A member of a PMIx Group has abnormally terminated (i.e., without formally leaving the group prior to termination) - the process identifier of the failed process will in included in the notification

PMIX_GROUP_INVITE_ACCEPTED A process has accepted an invitation to join a PMIx Group - the identifier of the group being joined will be included in the notification

PMIX_GROUP_INVITE_DECLINED A process has declined an invitation to join a PMIx Group - the identifier of the declined group will be included in the notification

PMIX_GROUP_INVITE_FAILED An invited process failed or terminated prior to responding to the invitation - the identifier of the failed process will be included in the notification.

PMIX_GROUP_MEMBERSHIP_UPDATE The membership of a PMIx group has changed - the identifiers of the revised membership will be included in the notification.

PMIX_GROUP_CONSTRUCT_ABORT Any participant in a PMIx group construct operation that returns **PMIX_GROUP_CONSTRUCT_ABORT** from the *leader failed* event handler will cause all participants to receive an event notifying them of that status. Similarly, the leader may elect to abort the procedure by either returning this error code from the handler assigned to the **PMIX_GROUP_INVITE_ACCEPTED** or **PMIX_GROUP_INVITE_DECLINED** codes, or by generating an event for the abort code. Abort events will be sent to all invited or existing members of the group.

PMIX_GROUP_CONSTRUCT_COMPLETE The group construct operation has completed - the final membership will be included in the notification.

PMIX_GROUP_LEADER_FAILED The current *leader* of a group including this process has abnormally terminated - the group identifier will be included in the notification.

PMIX_GROUP_LEADER_SELECTED A new *leader* of a group including this process has been selected - the identifier of the new leader will be included in the notification

PMIX_GROUP_CONTEXT_ID_ASSIGNED A new PGCID has been assigned by the host environment to a group that includes this process - the group identifier will be included in the notification.

PMIX_ERR_REPEAT_ATTR_REGISTRATION The attributes for an identical function have already been registered at the specified level (host, server, or client)

PMIX_ERR_IOF_FAILURE An IO forwarding operation failed - the affected channel will be included in the notification

PMIX_ERR_IOF_COMPLETE IO forwarding of the standard input for this process has completed - i.e., the stdin file descriptor has closed

PMIX_ERR_GET_MALLOC_REQD The data returned by **PMIx_Get** contains values that required dynamic memory allocations (i.e., "malloc"), despite a request for static pointers to the values in the key-value store. User is responsible for releasing the memory when done with the information.

14.1.1.3 System error constants

PMIX_ERR_SYS_BASE Mark the beginning of a dedicated range of constants for system event reporting.

PMIX_ERR_NODE_DOWN   A node has gone down - the identifier of the affected node will be
included in the notification

PMIX_ERR_NODE_OFFLINE   A node has been marked as *offline* - the identifier of the affected
node will be included in the notification

PMIX_ERR_SYS_OTHER   Mark the end of a dedicated range of constants for system event
reporting.

14.1.1.4 Event handler error constants

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

14.1.1.5 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.

14.1.2 Macros for use with PMIx constants

14.1.2.1 Detect system event constant

Test a given error constant to see if it falls within the dedicated range of constants for system event
reporting.

PMIx v2.2   ▼ ─────────────────── C ───────────────────── ▼

PMIX_SYSTEM_EVENT(a)

▲ ─────────────────── C ───────────────────── ▲

IN   a
Error constant to be checked (**pmix_status_t**)

Returns **true** if the provided values falls within the dedicated range of constants for system event
reporting

14.2 Data Types

This section defines various data types used by the PMIx APIs. The version of the standard in
which a particular data type was introduced is shown in the margin.

14.2.1 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 were 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.

▲ ─── ▲

14.2.1.1 Key support macro

Compare the key in a `pmix_info_t` to a given value

PMIx v3.0 ▼ ─────────────────────────── C ─────────────────────────── ▼

```
PMIX_CHECK_KEY(a, b)
```

▲ ─────────────────────────── C ─────────────────────────── ▲

IN a
    Pointer to the structure whose key is to be checked (pointer to `pmix_info_t`)
IN b
    String value to be compared against (`char*`)

Returns `true` if the key matches the given value

14.2.2 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 were 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.

▲ ─── ▲

14.2.2.1 Namespace support macro

Compare the string in a **pmix_nspace_t** to a given value

PMIx v3.0 ▼ ─────────────────────── C ───────────────────────── ▼

```
PMIX_CHECK_NSPACE(a, b)
```
▲ ─────────────────────── C ───────────────────────── ▲

IN   a
    Pointer to the structure whose value is to be checked (pointer to **pmix_nspace_t**)
IN   b
    String value to be compared against (**char***)

Returns **true** if the namespace matches the given value

14.2.3 Rank Structure

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

▼ ───────────────────── 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_RANK_LOCAL_NODE   Special rank value used to define groups of ranks. This constant defines the group of all ranks on a local node.

PMIX_RANK_LOCAL_PEERS   Special rank value used to define groups of rankss. This constant defines the group of all ranks on a local node within the same namespace as the current process.

PMIX_RANK_INVALID   An invalid rank value.

PMIX_RANK_VALID   Define an upper boundary for valid rank values.

14.2.4 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.

▼ ───────────────────── C ───────────────────── ▼

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

▲ ───────────────────── C ───────────────────── ▲

14.2.5 Process structure support macros

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

14.2.5.1 Initialize the `pmix_proc_t` structure

PMIX_PROC_CONSTRUCT

Initialize the `pmix_proc_t` fields

PMIx v1.0 ▼ ——————————————— C ——————————————— ▼

PMIX_PROC_CONSTRUCT(m)

▲ ——————————————— C ——————————————— ▲

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

14.2.5.2 Destruct the `pmix_proc_t` structure

There is nothing to release here as the fields in `pmix_proc_t` are all declared *static*. However, the macro is provided for symmetry in the code and for future-proofing should some allocated field be included some day.

14.2.5.3 Create a `pmix_proc_t` array

Allocate and initialize an array of `pmix_proc_t` structures

PMIx v1.0 ▼ ——————————————— 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`)

14.2.5.4 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`)

14.2.5.5 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`)

14.2.5.6 Compare identifiers

Compare two `pmix_proc_t` identifiers

PMIx v3.0 ▼─────────────────────── C ───────────────────────▼

`PMIX_CHECK_PROCID(a, b)`

▲─────────────────────── C ───────────────────────▲

IN　**a**
　　Pointer to a structure whose ID is to be compared (pointer to `pmix_proc_t`)
IN　**b**
　　Pointer to a structure whose ID is to be compared (pointer to `pmix_proc_t`)

Returns **true** if the two structures contain matching namespaces and:

- the ranks are the same value

- one of the ranks is `PMIX_RANK_WILDCARD`

14.2.6 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.

1	**PMIX_PROC_STATE_UNDEF**	Undefined process state
2	**PMIX_PROC_STATE_PREPPED**	Process is ready to be launched
3	**PMIX_PROC_STATE_LAUNCH_UNDERWAY**	Process launch is underway
4	**PMIX_PROC_STATE_RESTART**	Process is ready for restart
5	**PMIX_PROC_STATE_TERMINATE**	Process is marked for termination
6	**PMIX_PROC_STATE_RUNNING**	Process has been locally **fork**'ed by the RM
7	**PMIX_PROC_STATE_CONNECTED**	Process has connected to PMIx server

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

12   **PMIX_PROC_STATE_TERMINATED**   Process has terminated and is no longer running

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

16	**PMIX_PROC_STATE_KILLED_BY_CMD**	Process was killed by a command
17	**PMIX_PROC_STATE_ABORTED**	Process was aborted by a call to **PMIx_Abort**
18	**PMIX_PROC_STATE_FAILED_TO_START**	Process failed to start
19	**PMIX_PROC_STATE_ABORTED_BY_SIG**	Process aborted by a signal
20	**PMIX_PROC_STATE_TERM_WO_SYNC**	Process exited without calling **PMIx_Finalize**
21	**PMIX_PROC_STATE_COMM_FAILED**	Process communication has failed
22	**PMIX_PROC_STATE_CALLED_ABORT**	Process called **PMIx_Abort**

23   **PMIX_PROC_STATE_MIGRATING**   Process failed and is waiting for resources before
24   restarting

25	**PMIX_PROC_STATE_CANNOT_RESTART**	Process failed and cannot be restarted
26	**PMIX_PROC_STATE_TERM_NON_ZERO**	Process exited with a non-zero status
27	**PMIX_PROC_STATE_FAILED_TO_LAUNCH**	Unable to launch process

14.2.7   Process Information Structure

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

PMIx v2.0

```
1     typedef struct pmix_proc_info {
2         /** Process structure */
3         pmix_proc_t proc;
4         /** Hostname where process resides */
5         char *hostname;
6         /** Name of the executable */
7         char *executable_name;
8         /** Process ID on the host */
9         pid_t pid;
10        /** Exit code of the process. Default: 0 */
11        int exit_code;
12        /** Current state of the process */
13        pmix_proc_state_t state;
14    } pmix_proc_info_t;
```

C

14.2.8 Process Information Structure support macros

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

14.2.8.1 Initialize the **pmix_proc_info_t** structure

Initialize the **pmix_proc_info_t** fields

PMIx v2.0 ▼ ———————————————— C ————————————————— ▼

```
PMIX_PROC_INFO_CONSTRUCT(m)
```

▲ ———————————————— C ————————————————— ▲

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

14.2.8.2 Destruct the **pmix_proc_info_t** structure

Destruct the **pmix_proc_info_t** fields

PMIx v2.0 ▼ ———————————————— C ————————————————— ▼

```
PMIX_PROC_INFO_DESTRUCT(m)
```

▲ ———————————————— C ————————————————— ▲

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

14.2.8.3 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`)

14.2.8.4 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`)

14.2.9 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.

14.2.10 Job State Structure

PMIx v4.0 The **pmix_job_state_t** structure is a **uint8_t** type for job state values. The following constants can be used to set a variable of the type **pmix_job_state_t** . All values were originally defined in version 4 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 job state values during the job lifetime. This is particularly true of states in the launch process, and for short-lived jobs.

PMIX_JOB_STATE_UNDEF   Undefined job state
PMIX_JOB_STATE_PREPPED   Job is ready to be launched
PMIX_JOB_STATE_LAUNCH_UNDERWAY   Job launch is underway
PMIX_JOB_STATE_RUNNING   All processes in the job have been spawned
PMIX_JOB_STATE_SUSPENDED   All processes in the job have been suspended
PMIX_JOB_STATE_CONNECTED   All processes in the job have connected to their PMIx
       server
PMIX_JOB_STATE_UNTERMINATED   Define a "boundary" between the terminated states
       and **PMIX_JOB_STATE_TERMINATED** so users can easily and quickly determine if a job
       is still running or not. Any value less than this constant means that the job has not terminated.
PMIX_JOB_STATE_TERMINATED   All processes in the job have terminated and are no
       longer running - typically will be accompanied by the job exit status in response to a query
PMIX_JOB_STATE_TERMINATED_WITH_ERROR   Define a boundary so users can easily
       and quickly determine if a job abnormally terminated - typically will be accompanied by a
       job-related error code in response to a query Any value above this constant means that the job
       terminated abnormally.

14.2.11 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.

| 1 | **PMIX_RANGE_PROC_LOCAL** | Data is only available to this process. |
| 2 | **PMIX_RANGE_INVALID** | Invalid value |

— Advice to users —

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

14.2.12 Data Persistence Structure

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

10	**PMIX_PERSIST_INDEF**	Retain data until specifically deleted.
11	**PMIX_PERSIST_FIRST_READ**	Retain data until the first access, then the data is deleted.
12	**PMIX_PERSIST_PROC**	Retain data until the publishing process terminates.
13	**PMIX_PERSIST_APP**	Retain data until the application terminates.
14	**PMIX_PERSIST_SESSION**	Retain data until the session/allocation terminates.
15	**PMIX_PERSIST_INVALID**	Invalid value

14.2.13 Data Array Structure

PMIx v2.0 — C —

```
17      typedef struct pmix_data_array
18          pmix_data_type_t type;
19          size_t size;
20          void *array;
21       pmix_data_array_t;
```

— C —

22 The **pmix_data_array_t** structure is used to pass arrays of related values. Any PMIx data
23 type (including complex structures) can be included in the array.

14.2.14 Data array structure support macros

25 The following macros are provided to support the **pmix_data_array_t** structure.

14.2.14.1 Initialize the `pmix_data_array_t` structure

Initialize the `pmix_data_array_t` fields, allocating memory for the array itself.

▽ ─────── C ─────── ▽

```
PMIX_DATA_ARRAY_CONSTRUCT(m, n, t)
```

△ ─────── C ─────── △

IN `m`
    Pointer to the structure to be initialized (pointer to `pmix_data_array_t`)
IN `n`
    Number of elements in the array (`size_t`)
IN `t`
    PMIx data type for the array elements (`pmix_data_type_t`)

14.2.14.2 Destruct the `pmix_data_array_t` structure

Destruct the `pmix_data_array_t` fields, releasing the array's memory.

▽ ─────── C ─────── ▽

```
PMIX_DATA_ARRAY_DESTRUCT(m)
```

△ ─────── C ─────── △

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

14.2.14.3 Create and initialize a `pmix_data_array_t` object

Allocate and initialize a `pmix_data_array_t` structure and initialize it, allocating memory for
the array itself as well.

▽ ─────── C ─────── ▽

```
PMIX_DATA_ARRAY_CREATE(m, n, t)
```

△ ─────── C ─────── △

INOUT `m`
    Address where the pointer to the `pmix_data_array_t` structure shall be stored (handle)
IN `n`
    Number of elements in the array (`size_t`)
IN `t`
    PMIx data type for the array elements (`pmix_data_type_t`)

1 ### 14.2.14.4 Free a `pmix_data_array_t` object

2 Release a `pmix_data_array_t` structure, including releasing the array's memory.

PMIx v2.2 ───────────────────────────── C ─────────────────────────────

3 **PMIX_DATA_ARRAY_FREE(m)**
   ───────────────────────────── C ─────────────────────────────

4 **IN   m**
5   Pointer to the `pmix_data_array_t` structure (handle)

14.2.15 Value Structure

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

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

PMIx v1.0 ───────────────────────────── C ─────────────────────────────

```
13  typedef struct pmix_value {
14      pmix_data_type_t type;
15      union {
16          bool flag;
17          uint8_t byte;
18          char *string;
19          size_t size;
20          pid_t pid;
21          int integer;
22          int8_t int8;
23          int16_t int16;
24          int32_t int32;
25          int64_t int64;
26          unsigned int uint;
27          uint8_t uint8;
28          uint16_t uint16;
29          uint32_t uint32;
30          uint64_t uint64;
31          float fval;
32          double dval;
33          struct timeval tv;
34          time_t time;                    // version 2.0
35          pmix_status_t status;           // version 2.0
```

```
1              pmix_rank_t rank;               // version 2.0
2              pmix_proc_t *proc;              // version 2.0
3              pmix_byte_object_t bo;
4              pmix_persistence_t persist;     // version 2.0
5              pmix_scope_t scope;             // version 2.0
6              pmix_data_range_t range;        // version 2.0
7              pmix_proc_state_t state;        // version 2.0
8              pmix_proc_info_t *pinfo;        // version 2.0
9              pmix_data_array_t *darray;      // version 2.0
10             void *ptr;                      // version 2.0
11             pmix_alloc_directive_t adir;    // version 2.0
12         } data;
13     } pmix_value_t;
```
— C —

14.2.16 Value structure support macros

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

14.2.16.1 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**)

14.2.16.2 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**)

14.2.16.3 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`)

14.2.16.4 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`)

14.2.16.5 Load a value 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.

14.2.16.6 Unload a **pmix_value_t** structure

Summary

Unload data from a **pmix_value_t** structure.

PMIx v2.2 ───────────────────────────────── C ─────────────────────────────────

```
PMIX_VALUE_UNLOAD(r, v, d, t);
```

───────────────────────────────── C ─────────────────────────────────

OUT r
    Status code indicating result of the operation **pmix_status_t**
IN v
    The **pmix_value_t** from which the data is to be unloaded (pointer to **pmix_value_t**)
INOUT d
    Pointer to the location where the data value is to be returned (handle)
INOUT t
    Pointer to return the data type of the unloaded value (handle)

Description

This macro simplifies the unloading of data from a **pmix_value_t** .

───────────────────────────────── Advice to users ─────────────────────────────────

Memory will be allocated and the data will be in the **pmix_value_t** returned - the source **pmix_value_t** will not be altered.

1 ## 14.2.16.7 Transfer data between `pmix_value_t` structures

2 **Summary**

3 Transfer the data value between two `pmix_value_t` structures.

PMIx v2.0 ──────────────────── C ────────────────────

4 ```
PMIX_VALUE_XFER(r, d, s);
```
──────────────────── C ────────────────────

5 **OUT** `r`
6     Status code indicating success or failure of the transfer ( `pmix_status_t` )
7 **IN** `d`
8     Pointer to the `pmix_value_t` destination (handle)
9 **IN** `s`
10     Pointer to the `pmix_value_t` source (handle)

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

───────────────────── Advice to users ─────────────────────

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

16 ## 14.2.16.8 Retrieve a numerical value from a `pmix_value_t`

17 Retrieve a numerical value from a `pmix_value_t` structure

*PMIx v3.0* ──────────────────── C ────────────────────

18 ```
PMIX_VALUE_GET_NUMBER(s, m, n, t)
```
──────────────────── C ────────────────────

19 **OUT** `s`
20   Status code for the request (`pmix_status_t`)
21 **IN** `m`
22   Pointer to the `pmix_value_t` structure (handle)
23 **OUT** `n`
24   Variable to be set to the value (match expected type)
25 **IN** `t`
26   Type of number expected in *m* (`pmix_data_type_t`)

27 Sets the provided variable equal to the numerical value contained in the given `pmix_value_t` ,
28 returning success if the data type of the value matches the expected type and
29 `PMIX_ERR_BAD_PARAM` if it doesn't

14.2.17 Info Structure

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 ─────────────────────────

```
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 ─────────────────────────

14.2.18 Info structure support macros

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

14.2.18.1 Initialize the **pmix_info_t** structure

Initialize the **pmix_info_t** fields

PMIx v1.0 ───────────────────────── C ─────────────────────────

```
PMIX_INFO_CONSTRUCT(m)
```

───────────────────────── C ─────────────────────────

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

14.2.18.2 Destruct the **pmix_info_t** structure

Destruct the **pmix_info_t** fields

PMIx v1.0 ───────────────────────── C ─────────────────────────

```
PMIX_INFO_DESTRUCT(m)
```

───────────────────────── C ─────────────────────────

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

14.2.18.3 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**)

14.2.18.4 Free a `pmix_info_t` array

2   Release an array of `pmix_info_t` structures

PMIx v1.0   ▼ ─────────────── C ─────────────── ▼

3   `PMIX_INFO_FREE(m, n)`

    ▲ ─────────────── C ─────────────── ▲

4   **IN   m**
5   Pointer to the array of `pmix_info_t` structures (handle)
6   **IN   n**
7   Number of structures in the array (`size_t`)

8 **14.2.18.5 Load key and value data into a `pmix_info_t`**

PMIx v1.0   ▼ ─────────────── C ─────────────── ▼

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

    ▲ ─────────────── C ─────────────── ▲

10   **IN   v**
11   Pointer to the `pmix_info_t` into which the key and data are to be loaded (pointer to
12   `pmix_info_t`)
13   **IN   k**
14   String key to be loaded - must be less than or equal to `PMIX_MAX_KEYLEN` in length
15   (handle)
16   **IN   d**
17   Pointer to the data value to be loaded (handle)
18   **IN   t**
19   Type of the provided data value (`pmix_data_type_t`)

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

──────────────────── Advice to users ────────────────────

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

    ▲ ── ▲

14.2.18.6 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.

14.2.18.7 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**

14.2.19 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.
PMIX_INFO_ARRAY_END   Mark that this **pmix_info_t** struct is at the end of an array
   created by the **PMIX_INFO_CREATE** macro. 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.

14.2.20 Info Directive support macros

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

14.2.20.1 Mark an info structure as required

Summary

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

PMIx v2.0 ▼——————————— 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.

14.2.20.2 Mark an info structure as optional

Summary

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

PMIx v2.0 ▼——————————— C ———————————▼

```
PMIX_INFO_OPTIONAL(info);
```

▲——————————— C ———————————▲

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

This macro simplifies marking a **pmix_info_t** structure as *optional*.

14.2.20.3 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 ———————————▼

```
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.

14.2.20.4 Test an info structure for *optional* directive

Summary

Test a `pmix_info_t` structure, returning **true** if the structure is *optional*.

PMIx v2.0

```
PMIX_INFO_IS_OPTIONAL(info);
```

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

Test the `PMIX_INFO_REQD` flag in a `pmix_info_t` structure, returning **true** if the flag is *not* set.

14.2.20.5 Test an info structure for *end of array* directive

Summary

Test a `pmix_info_t` structure, returning **true** if the structure is at the end of an array created by the `PMIX_INFO_CREATE` macro.

PMIx v2.2

```
PMIX_INFO_IS_END(info);
```

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

This macro simplifies the testing of the end-of-array flag in `pmix_info_t` structures.

14.2.21 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.

14.2.22 IO Forwarding Channels

The **pmix_iof_channel_t** structure is a **uint16_t** type that defines a set of bit-mask flags for specifying IO forwarding channels. These can be bitwise OR'd together to reference multiple channels.

PMIX_FWD_NO_CHANNELS   Forward no channels
PMIX_FWD_STDIN_CHANNEL   Forward stdin
PMIX_FWD_STDOUT_CHANNEL   Forward stdout
PMIX_FWD_STDERR_CHANNEL   Forward stderr
PMIX_FWD_STDDIAG_CHANNEL   Forward stddiag, if available
PMIX_FWD_ALL_CHANNELS   Forward all available channels

14.2.23 Environmental Variable Structure

Define a structure for specifying environment variable modifications. Standard environment variables (e.g., **PATH**, **LD_LIBRARY_PATH**, and **LD_PRELOAD**) take multiple arguments separated by delimiters. Unfortunately, the delimiters depend upon the variable itself - some use semi-colons, some colons, etc. Thus, the operation requires not only the name of the variable to be modified and the value to be inserted, but also the separator to be used when composing the aggregate value.

───────────────── C ─────────────────

```
typedef struct
    char *envar;
    char *value;
    char separator;
 pmix_envar_t;
```

───────────────── C ─────────────────

14.2.24 Environmental variable support macros

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

14.2.24.1 Initialize the **pmix_envar_t** structure

Initialize the **pmix_envar_t** fields

───────────────── C ─────────────────

PMIX_ENVAR_CONSTRUCT(m)

───────────────── C ─────────────────

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

14.2.24.2 Destruct the `pmix_envar_t` structure

Clear the `pmix_envar_t` fields

▼ ——————————————— C ——————————————— ▼

```
PMIX_ENVAR_DESTRUCT(m)
```

▲ ——————————————— C ——————————————— ▲

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

14.2.24.3 Create a `pmix_envar_t` array

Allocate and initialize an array of `pmix_envar_t` structures

▼ ——————————————— C ——————————————— ▼

```
PMIX_ENVAR_CREATE(m, n)
```

▲ ——————————————— C ——————————————— ▲

INOUT m
    Address where the pointer to the array of `pmix_envar_t` structures shall be stored (handle)
IN   n
    Number of structures to be allocated (`size_t`)

14.2.24.4 Free a `pmix_envar_t` array

Release an array of `pmix_envar_t` structures

▼ ——————————————— C ——————————————— ▼

```
PMIX_ENVAR_FREE(m, n)
```

▲ ——————————————— C ——————————————— ▲

IN   m
    Pointer to the array of `pmix_envar_t` structures (handle)
IN   n
    Number of structures in the array (`size_t`)

14.2.24.5 Load a `pmix_envar_t` structure

Load values into a `pmix_envar_t`

▼ ──────────────── C ──────────────── ▼

PMIX_ENVAR_LOAD(m, e, v, s)

▲ ──────────────── C ──────────────── ▲

IN m
Pointer to the structure to be loaded (pointer to `pmix_envar_t`)
IN e
Environmental variable name (**char***)
IN v
Value of variable (**char***)
IN v
Separator character (**char**)

14.2.25 Lookup Returned Data Structure

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

▼ ──────────────── C ──────────────── ▼

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

▲ ──────────────── C ──────────────── ▲

14.2.26 Lookup data structure support macros

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

14.2.26.1 Initialize the `pmix_pdata_t` structure

Initialize the `pmix_pdata_t` fields

▼ ──────────────── C ──────────────── ▼

PMIX_PDATA_CONSTRUCT(m)

▲ ──────────────── C ──────────────── ▲

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

14.2.26.2 Destruct the `pmix_pdata_t` structure

Destruct the `pmix_pdata_t` fields

▼ ─────────────── C ─────────────── ▼

```
PMIX_PDATA_DESTRUCT(m)
```

▲ ─────────────── C ─────────────── ▲

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

14.2.26.3 Create a `pmix_pdata_t` array

Allocate and initialize an array of `pmix_pdata_t` structures

▼ ─────────────── 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`)

14.2.26.4 Free a `pmix_pdata_t` array

Release an array of `pmix_pdata_t` structures

▼ ─────────────── 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`)

14.2.26.5 Load a lookup data structure

Summary

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

PMIx v1.0 ──────────────────────────── 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.

1 ### 14.2.26.6 Transfer a lookup data structure

2 **Summary**

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

PMIx v2.0 ▼ ───────────────────── C ───────────────────── ▼

4
```
PMIX_PDATA_XFER(d, s);
```
▲ ───────────────────── C ───────────────────── ▲

5 **IN d**
6   Pointer to the destination **pmix_pdata_t** (pointer to **pmix_pdata_t**)
7 **IN s**
8   Pointer to the source **pmix_pdata_t** (pointer to **pmix_pdata_t**)

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

▼ ──────────────────── Advice to users ────────────────────

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

▲ ──

13 ## 14.2.27 Application Structure

14 The **pmix_app_t** structure describes the application context for the **PMIx_Spawn** and
15 **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 ───────────────────── ▲

CHAPTER 14. DATA STRUCTURES AND TYPES   **321**

1 14.2.28 App structure support macros

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

3 14.2.28.1 Initialize the `pmix_app_t` structure

4   Initialize the **`pmix_app_t`** fields

PMIx v1.0 ▼ ——————————————— C ———————————————— ▼

5   **`PMIX_APP_CONSTRUCT(m)`**

▲ ——————————————— C ———————————————— ▲

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

8 14.2.28.2 Destruct the `pmix_app_t` structure

9   Destruct the **`pmix_app_t`** fields

PMIx v1.0 ▼ ——————————————— C ———————————————— ▼

10   **`PMIX_APP_DESTRUCT(m)`**

▲ ——————————————— C ———————————————— ▲

11   **IN   m**
12   Pointer to the structure to be destructed (pointer to **`pmix_app_t`**)

13 14.2.28.3 Create a `pmix_app_t` array

14   Allocate and initialize an array of **`pmix_app_t`** structures

PMIx v1.0 ▼ ——————————————— C ———————————————— ▼

15   **`PMIX_APP_CREATE(m, n)`**

▲ ——————————————— C ———————————————— ▲

16   **INOUT   m**
17   Address where the pointer to the array of **`pmix_app_t`** structures shall be stored (handle)
18   **IN   n**
19   Number of structures to be allocated (**`size_t`**)

20 14.2.28.4 Free a `pmix_app_t` array

21   Release an array of **`pmix_app_t`** structures

PMIx v1.0 ▼ ——————————————— C ———————————————— ▼

22   **`PMIX_APP_FREE(m, n)`**

▲ ——————————————— C ———————————————— ▲

23   **IN   m**
24   Pointer to the array of **`pmix_app_t`** structures (handle)
25   **IN   n**
26   Number of structures in the array (**`size_t`**)

14.2.28.5 Create the `pmix_info_t` array of application directives

Create an array of `pmix_info_t` structures for passing application-level directives, updating the *ninfo* field of the `pmix_app_t` structure.

PMIx v2.2 ──────────────────────── C ────────────────────────

```
PMIX_APP_INFO_CREATE(m, n)
```

──────────────────────── C ────────────────────────

IN   m
    Pointer to the `pmix_app_t` structure (handle)
IN   n
    Number of directives to be allocated (`size_t`)

14.2.29 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 ────────────────────────

14.2.30 Query structure support macros

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

14.2.30.1 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`)

14.2.30.2 Destruct the `pmix_query_t` structure

Destruct the `pmix_query_t` fields

▼ ──────────────── C ──────────────── ▼

3   `PMIX_QUERY_DESTRUCT(m)`

▲ ──────────────── C ──────────────── ▲

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

14.2.30.3 Create a `pmix_query_t` array

Allocate and initialize an array of `pmix_query_t` structures

▼ ──────────────── C ──────────────── ▼

8   `PMIX_QUERY_CREATE(m, n)`

▲ ──────────────── C ──────────────── ▲

9   **INOUT m**
10   Address where the pointer to the array of `pmix_query_t` structures shall be stored (handle)
11   **IN   n**
12   Number of structures to be allocated (`size_t`)

14.2.30.4 Free a `pmix_query_t` array

Release an array of `pmix_query_t` structures

▼ ──────────────── C ──────────────── ▼

15   `PMIX_QUERY_FREE(m, n)`

▲ ──────────────── C ──────────────── ▲

16   **IN   m**
17   Pointer to the array of `pmix_query_t` structures (handle)
18   **IN   n**
19   Number of structures in the array (`size_t`)

14.2.30.5 Create the `pmix_info_t` array of query qualifiers

Create an array of `pmix_info_t` structures for passing query qualifiers, updating the *nqual* field
of the `pmix_query_t` structure.

▼ ──────────────── C ──────────────── ▼

23   `PMIX_QUERY_QUALIFIERS_CREATE(m, n)`

▲ ──────────────── C ──────────────── ▲

24   **IN   m**
25   Pointer to the `pmix_query_t` structure (handle)
26   **IN   n**
27   Number of qualifiers to be allocated (`size_t`)

14.2.31 Attribute registration structure

The `pmix_regattr_t` structure is used to register attribute support for a PMIx function.

PMIx v4.0 ▼ ——————————————— C ——————————————— ▼

```
typedef struct pmix_regattr {
    char *name;
    pmix_key_t *string;
    pmix_data_type_t type;
    pmix_info_t *info;
    size_t ninfo;
    char **description;
} pmix_regattr_t;;
```

▲ ——————————————— C ——————————————— ▲

Note that in this structure:

- the *name* is the actual name of the attribute - e.g., "PMIX_MAX_PROCS"; and

- the *string* is the literal string value of the attribute - e.g., "pmix.max.size" for the `PMIX_MAX_PROCS` attribute

- *type* must be a PMIx data type identifying the type of data associated with this attribute.

- the *info* array contains machine-usable information regarding the range of accepted values. This may include entries for `PMIX_MIN_VALUE` , `PMIX_MAX_VALUE` , `PMIX_ENUM_VALUE` , or a combination of them. For example, an attribute that supports all positive integers might delineate it by including a `pmix_info_t` with a key of `PMIX_MIN_VALUE` , type of `PMIX_INT` , and value of zero. The lack of an entry for `PMIX_MAX_VALUE` indicates that there is no ceiling to the range of accepted values.

- *ninfo* indicates the number of elements in the *info* array

- The *description* field consists of a **NULL**-terminated array of strings describing the attribute, optionally including a human-readable description of the range of accepted values - e.g., "ALL POSITIVE INTEGERS", or a comma-delimited list of enum value names. No correlation between the number of entries in the *description* and the number of elements in the *info* array is implied or required.

The attribute *name* and *string* fields must be **NULL**-terminated strings composed of standard alphanumeric values supported by common utilities such as *strcmp*.

▼ ——————————— Advice to PMIx library implementers ——————————— ▼

Although not strictly required, PMIx library implementers are strongly encouraged to provide both human-readable and machine-parsable descriptions of supported attributes.

▲ —— ▲

1 Although not strictly required, host environments are strongly encouraged to provide both
2 human-readable and machine-parsable descriptions of supported attributes when registering them.

3 14.2.32 Attribute registration structure support macros

4 The following macros are provided to support the **`pmix_regattr_t`** structure.

5 14.2.32.1 Initialize the **`pmix_regattr_t`** structure

6 Initialize the **`pmix_regattr_t`** fields

PMIx v4.0   C

7   **`PMIX_REGATTR_CONSTRUCT(m)`**
                                                    C

8 **IN   m**
9   Pointer to the structure to be initialized (pointer to **`pmix_regattr_t`**)

10 14.2.32.2 Destruct the **`pmix_regattr_t`** structure

11 Destruct the **`pmix_regattr_t`** fields, releasing all strings.

PMIx v4.0   C

12   **`PMIX_REGATTR_DESTRUCT(m)`**
                                                    C

13 **IN   m**
14   Pointer to the structure to be destructed (pointer to **`pmix_regattr_t`**)

15 14.2.32.3 Create a **`pmix_regattr_t`** array

16 Allocate and initialize an array of **`pmix_regattr_t`** structures

PMIx v4.0   C

17   **`PMIX_REGATTR_CREATE(m, n)`**
                                                    C

18 **INOUT m**
19   Address where the pointer to the array of **`pmix_regattr_t`** structures shall be stored
20   (handle)
21 **IN   n**
22   Number of structures to be allocated (**`size_t`**)

14.2.32.4 Free a `pmix_regattr_t` array

Release an array of `pmix_regattr_t` structures

───────────────────── C ─────────────────────

```
PMIX_REGATTR_FREE(m, n)
```
───────────────────── C ─────────────────────

INOUT m
  Pointer to the array of `pmix_regattr_t` structures (handle)
IN n
  Number of structures in the array (`size_t`)

14.2.32.5 Load a `pmix_regattr_t` structure

Load values into a `pmix_regattr_t` structure. The macro can be called multiple times to add as many strings as desired to the same structure by passing the same address and a `NULL` key to the macro. Note that the *t* type value must be given each time.

───────────────────── C ─────────────────────

```
PMIX_REGATTR_LOAD(a, n, k, t, ni, v)
```
───────────────────── C ─────────────────────

IN a
  Pointer to the structure to be loaded (pointer to `pmix_proc_t`)
IN n
  String name of the attribute (string)
IN k
  Key value to be loaded (`pmix_key_t`)
IN t
  Type of data associated with the provided key (`pmix_data_type_t`)
IN ni
  Number of `pmix_info_t` elements to be allocated in *info* (`size_t`)
IN v
  One-line description to be loaded (more can be added separately) (string)

14.2.32.6 Transfer a `pmix_regattr_t` to another `pmix_regattr_t`

Non-destructively transfer the contents of a `pmix_regattr_t` structure to another one.

───────────────────── C ─────────────────────

```
PMIX_REGATTR_XFER(m, n)
```
───────────────────── C ─────────────────────

INOUT m
  Pointer to the destination `pmix_regattr_t` structure (handle)
IN m
  Pointer to the source `pmix_regattr_t` structure (handle)

14.2.33   PMIx Group Directives

PMIx v4.0   The **pmix_group_opt_t** type is an enumerated type used with the **PMIx_Group_join** API
to indicate *accept* or *decline* of the invitation - these are provided for readability of user code:

PMIX_GROUP_DECLINE   Decline the invitation
PMIX_GROUP_ACCEPT   Accept the invitation.

14.2.34   Byte Object Type

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

PMIx v1.0 ▼ ———————————————— C ———————————————— ▼

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

▲ ———————————————— C ———————————————— ▲

14.2.35   Byte object support macros

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

14.2.35.1   Initialize the **pmix_byte_object_t** structure

Initialize the **pmix_byte_object_t** fields

PMIx v2.0 ▼ ———————————————— C ———————————————— ▼

PMIX_BYTE_OBJECT_CONSTRUCT(m)

▲ ———————————————— C ———————————————— ▲

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

14.2.35.2   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**)

14.2.35.3 Create a `pmix_byte_object_t` structure

Allocate and intitialize an array of `pmix_byte_object_t` structures

———————————————————————— 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`)

14.2.35.4 Free a `pmix_byte_object_t` array

Release an array of `pmix_byte_object_t` structures

———————————————————————— 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`)

14.2.35.5 Load a `pmix_byte_object_t` structure

Load values into a `pmix_byte_object_t`

———————————————————————— C ————————————————————————————

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

———————————————————————— C ————————————————————————————

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`)

14.2.36 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 ——————————————— ▲

14.2.37 Data array support macros

The following macros support the `pmix_data_array_t` structure.

14.2.37.1 Initialize a `pmix_data_array_t` structure

Initialize the `pmix_data_array_t` fields, allocating memory for the array of the indicated type.

PMIx v2.2 ▼ ——————————————— C ——————————————— ▼

```
PMIX_DATA_ARRAY_CONSTRUCT(m, n, t)
```

▲ ——————————————— C ——————————————— ▲

IN   m
    Pointer to the structure to be initialized (pointer to `pmix_data_array_t`)
IN   n
    Number of elements in the array (`size_t`)
IN   t
    PMIx data type of the array elements (`pmix_data_type_t`)

14.2.37.2 Destruct a `pmix_data_array_t` structure

Destruct the `pmix_data_array_t` , releasing the memory in the array.

PMIx v2.2 ▼ ——————————————— C ——————————————— ▼

```
PMIX_DATA_ARRAY_CONSTRUCT(m)
```

▲ ——————————————— C ——————————————— ▲

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

14.2.37.3 Create a `pmix_data_array_t` structure

Allocate memory for the `pmix_data_array_t` object itself, and then allocate memory for the array of the indicated type.

———————————————————————— C ————————————————————————

```
PMIX_DATA_ARRAY_CREATE(m, n, t)
```

———————————————————————— C ————————————————————————

INOUT m
    Variable to be set to the address of the structure (pointer to `pmix_data_array_t`)
IN n
    Number of elements in the array (`size_t`)
IN t
    PMIx data type of the array elements (`pmix_data_type_t`)

14.2.37.4 Free a `pmix_data_array_t` structure

Release the memory in the array, and then release the `pmix_data_array_t` object itself.

———————————————————————— C ————————————————————————

```
PMIX_DATA_ARRAY_FREE(m)
```

———————————————————————— C ————————————————————————

IN m
    Pointer to the structure to be released (pointer to `pmix_data_array_t`)

14.2.38 Argument Array Macros

The following macros support the construction and release of **NULL**-terminated argv arrays of strings.

14.2.38.1 Argument array extension

Summary

Append a string to a NULL-terminated, argv-style array of strings.

———————————————————————— C ————————————————————————

```
PMIX_ARGV_APPEND(r, a, b);
```

———————————————————————— C ————————————————————————

OUT r
    Status code indicating success or failure of the operation (`pmix_status_t`)
INOUT a
    Argument list (pointer to NULL-terminated array of strings)
IN b
    Argument to append to the list (string)

Description

This function helps the caller build the **argv** portion of **pmix_app_t** structure, arrays of keys
for querying, or other places where argv-style string arrays are required in the way that the PRI
expects it to be constructed.

———————————————————— Advice to users ————————————————————

The provided argument is copied into the destination array - thus, the source string can be free'd
without affecting the array once the macro has completed.

14.2.38.2 Argument array extension - unique

Summary

Append a string to a NULL-terminated, argv-style array of strings, but only if the provided
argument doesn't already exist somewhere in the array.

———————————————————————— C ————————————————————————

```
PMIX_ARGV_APPEND_UNIQUE(r, a, b);
```
———————————————————————— C ————————————————————————

OUT r
    Status code indicating success or failure of the operation (**pmix_status_t**)
INOUT a
    Argument list (pointer to NULL-terminated array of strings)
IN   b
    Argument to append to the list (string)

Description

This function helps the caller build the **argv** portion of **pmix_app_t** structure, arrays of keys
for querying, or other places where argv-style string arrays are required in the way that the PRI
expects it to be constructed.

———————————————————— Advice to users ————————————————————

The provided argument is copied into the destination array - thus, the source string can be free'd
without affecting the array once the macro has completed.

14.2.38.3 Argument array release

Summary

Free an argv-style array and all of the strings that it contains

— C —

```
PMIX_ARGV_FREE(a);
```

— C —

IN a
  Argument list (pointer to NULL-terminated array of strings)

Description

This function releases the array and all of the strings it contains.

14.2.38.4 Argument array split

Summary

Split a string into a NULL-terminated argv array.

— C —

```
PMIX_ARGV_SPLIT(a, b, c);
```

— C —

OUT a
  Resulting argv-style array (**char****)
IN b
  String to be split (**char***)
IN c
  Delimiter character (**char**)

Description

Split an input string into a NULL-terminated argv array. Do not include empty strings in the resulting array.

— Advice to users —

All strings are inserted into the argv array by value; the newly-allocated array makes no references to the src_string argument (i.e., it can be freed after calling this function without invalidating the output argv array)

14.2.38.5 Argument array join

Summary

Join all the elements of an argv array into a single newly-allocated string.

———————————————————— C ————————————————————

```
PMIX_ARGV_JOIN(a, b, c);
```

———————————————————— C ————————————————————

OUT **a**
    Resulting string (**char***)
IN **b**
    Argv-style array to be joined (**char****)
IN **c**
    Delimiter character (**char**)

Description

Join all the elements of an argv array into a single newly-allocated string.

14.2.38.6 Argument array count

Summary

Return the length of a NULL-terminated argv array.

———————————————————— C ————————————————————

```
PMIX_ARGV_COUNT(r, a);
```

———————————————————— C ————————————————————

OUT **r**
    Number of strings in the array (integer)
IN **a**
    Argv-style array (**char****)

Description

Count the number of elements in an argv array

14.2.38.7 Argument array copy

Summary

Copy an argv array, including copying all off its strings.

———————————————————— C ————————————————————

```
PMIX_ARGV_COPY(a, b);
```

———————————————————— C ————————————————————

OUT **a**
    New argv-style array (**char****)
IN **b**
    Argv-style array (**char****)

Description

Copy an argv array, including copying all off its strings.

14.2.39 Set Environment Variable

Summary

Set an environment variable in a **NULL**-terminated, env-style array

━━━━━━━━━━━━━━━━━━━━━━ C ━━━━━━━━━━━━━━━━━━━━━━

```
PMIX_SETENV(r, name, value, env);
```

━━━━━━━━━━━━━━━━━━━━━━ C ━━━━━━━━━━━━━━━━━━━━━━

OUT r
    Status code indicating success or failure of the operation (**pmix_status_t**)
IN   name
    Argument name (string)
IN   value
    Argument value (string)
INOUT env
    Environment array to update (pointer to array of strings)

Description

Similar to **setenv** from the C API, this allows the caller to set an environment variable in the specified **env** array, which could then be passed to the **pmix_app_t** structure or any other destination.

━━━━━━━━━━━━━━━━ Advice to users ━━━━━━━━━━━━━━━━

The provided name and value are copied into the destination environment array - thus, the source strings can be free'd without affecting the array once the macro has completed.

14.3 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. New data type values introduced in this version of the Standard are shown in **magenta**.

————————— 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.

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 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_STATUS   Status code **pmix_status_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

1	**PMIX_PERSIST**	Persistance (**pmix_persistence_t**)
2	**PMIX_POINTER**	Pointer to an object (**void***)
3	**PMIX_SCOPE**	Scope (**pmix_scope_t**)
4	**PMIX_DATA_RANGE**	Range for data (**pmix_data_range_t**)
5	**PMIX_COMMAND**	PMIx command code (used internally)
6	**PMIX_INFO_DIRECTIVES**	Directives flag for **pmix_info_t** (
7	**pmix_info_directives_t**)	
8	**PMIX_DATA_TYPE**	Data type code (**pmix_data_type_t**)
9	**PMIX_PROC_STATE**	Process state (**pmix_proc_state_t**)
10	**PMIX_PROC_INFO**	Process information (**pmix_proc_info_t**)
11	**PMIX_DATA_ARRAY**	Data array (**pmix_data_array_t**)
12	**PMIX_PROC_RANK**	Process rank (**pmix_rank_t**)
13	**PMIX_QUERY**	Query structure (**pmix_query_t**)
14	**PMIX_COMPRESSED_STRING**	String compressed with zlib (**char***)
15	**PMIX_ALLOC_DIRECTIVE**	Allocation directive (**pmix_alloc_directive_t**)
16	**PMIX_IOF_CHANNEL**	Input/output forwarding channel (**pmix_iof_channel_t**)
17	**PMIX_ENVAR**	Environmental variable structure (**pmix_envar_t**)
18	**PMIX_COORD**	Structure containing fabric coordinates (**pmix_coord_t**)
19	**PMIX_REGATTR**	Structure supporting attribute registrations (**pmix_regattr_t**)
20	**PMIX_REGEX**	Regular expressions - can be a valid NULL-terminated string or an arbitrary
21		array of bytes

22 **14.4 Reserved attributes**

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

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

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

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

36   **PMIX_ATTR_UNDEF NULL** (**NULL**)
37   Constant representing an undefined attribute.

14.4.1   Initialization attributes

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

4   **PMIX_EVENT_BASE "pmix.evbase" (struct event_base *)**
5   Pointer to libevent[1] **event_base** to use in place of the internal progress thread.
6   **PMIX_SERVER_TOOL_SUPPORT "pmix.srvr.tool" (bool)**
7   The host RM wants to declare itself as willing to accept tool connection requests.
8   **PMIX_SERVER_REMOTE_CONNECTIONS "pmix.srvr.remote" (bool)**
9   Allow connections from remote tools. Forces the PMIx server to not exclusively use
10   loopback device.
11 **PMIX_SERVER_SYSTEM_SUPPORT "pmix.srvr.sys" (bool)**
12   The host RM wants to declare itself as being the local system server for PMIx connection
13   requests.
14 **PMIX_SERVER_TMPDIR "pmix.srvr.tmpdir" (char*)**
15   Top-level temporary directory for all client processes connected to this server, and where the
16   PMIx server will place its tool rendezvous point and contact information.
17 **PMIX_SYSTEM_TMPDIR "pmix.sys.tmpdir" (char*)**
18   Temporary directory for this system, and where a PMIx server that declares itself to be a
19   system-level server will place a tool rendezvous point and contact information.
20 **PMIX_SERVER_ENABLE_MONITORING "pmix.srv.monitor" (bool)**
21   Enable PMIx internal monitoring by the PMIx server.
22 **PMIX_SERVER_NSPACE "pmix.srv.nspace" (char*)**
23   Name of the namespace to use for this PMIx server.
24 **PMIX_SERVER_RANK "pmix.srv.rank" (pmix_rank_t)**
25   Rank of this PMIx server
26 **PMIX_SERVER_GATEWAY "pmix.srv.gway" (bool)**
27   Server is acting as a gateway for PMIx requests that cannot be serviced on backend nodes
28   (e.g., logging to email)

14.4.2   Tool-related attributes

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

33 **PMIX_TOOL_NSPACE "pmix.tool.nspace" (char*)**
34   Name of the namespace to use for this tool.
35 **PMIX_TOOL_RANK "pmix.tool.rank" (uint32_t)**
36   Rank of this tool.
37 **PMIX_SERVER_PIDINFO "pmix.srvr.pidinfo" (pid_t)**
38   PID of the target PMIx server for a tool.
39 **PMIX_CONNECT_TO_SYSTEM "pmix.cnct.sys" (bool)**

---

[1]http://libevent.org/

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***)

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.

PMIX_RECONNECT_SERVER "pmix.tool.recon" (**bool**)

Tool is requesting to change server connections

PMIX_LAUNCHER "pmix.tool.launcher" (**bool**)

Tool is a launcher and needs rendezvous files created

14.4.3 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_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.

PMIX_PSET_NAME "pmix.pset.nm" (**char***)

User-assigned name for the process set containing the given process.

14.4.4 Programming model attributes

These attributes are associated with programming models.

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_MODEL_NUM_THREADS "pmix.mdl.nthrds" (**uint64_t**)
    Number of active threads being used by the model
PMIX_MODEL_NUM_CPUS "pmix.mdl.ncpu" (**uint64_t**)
    Number of cpus being used by the model
PMIX_MODEL_CPU_TYPE "pmix.mdl.cputype" (**char***)
    Granularity - "hwthread", "core", etc.
PMIX_MODEL_PHASE_NAME "pmix.mdl.phase" (**char***)
    User-assigned name for a phase in the application execution (e.g., "cfd reduction")
PMIX_MODEL_PHASE_TYPE "pmix.mdl.ptype" (**char***)
    Type of phase being executed (e.g., "matrix multiply")
PMIX_MODEL_AFFINITY_POLICY "pmix.mdl.tap" (**char***)
    Thread affinity policy - e.g.: "master" (thread co-located with master thread), "close" (thread
    located on cpu close to master thread), "spread" (threads load-balanced across available cpus)

14.4.5 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.

14.4.6 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 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.

14.4.7 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.

14.4.8 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**)

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

1   **true** if this process resulted from a call to **PMIx_Spawn** .
2   **PMIX_ARCH "pmix.arch"** (**uint32_t**)
3   Architecture flag.

14.4.9 Scratch directory attributes

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

7   **PMIX_TMPDIR "pmix.tmpdir"** (**char***)
8   Full path to the top-level temporary directory assigned to the session.
9   **PMIX_NSDIR "pmix.nsdir"** (**char***)
10   Full path to the temporary directory assigned to the namespace, under **PMIX_TMPDIR** .
11   **PMIX_PROCDIR "pmix.pdir"** (**char***)
12   Full path to the subdirectory under **PMIX_NSDIR** assigned to the process.
13   **PMIX_TDIR_RMCLEAN "pmix.tdir.rmclean"** (**bool**)
14   Resource Manager will clean session directories

14.4.10 Relative Rank Descriptive Attributes

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

18   **PMIX_CLUSTER_ID "pmix.clid"** (**char***)
19   A string name for the cluster this proc is executing on
20   **PMIX_PROCID "pmix.procid"** (**pmix_proc_t**)
21   Process identifier
22   **PMIX_NSPACE "pmix.nspace"** (**char***)
23   Namespace of the job.
24   **PMIX_JOBID "pmix.jobid"** (**char***)
25   Job identifier assigned by the scheduler.
26   **PMIX_APPNUM "pmix.appnum"** (**uint32_t**)
27   Application number within the job.
28   **PMIX_RANK "pmix.rank"** (**pmix_rank_t**)
29   Process rank within the job.
30   **PMIX_GLOBAL_RANK "pmix.grank"** (**pmix_rank_t**)
31   Process rank spanning across all jobs in this session.
32   **PMIX_APP_RANK "pmix.apprank"** (**pmix_rank_t**)
33   Process rank within this application.
34   **PMIX_NPROC_OFFSET "pmix.offset"** (**pmix_rank_t**)
35   Starting global rank of this job - referenced using **PMIX_RANK_WILDCARD** .
36   **PMIX_LOCAL_RANK "pmix.lrank"** (**uint16_t**)
37   Local rank on this node within this job.
38   **PMIX_NODE_RANK "pmix.nrank"** (**uint16_t**)
39   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 (e.g., where a specified process is running, or a given device is located).
PMIX_NODEID "pmix.nodeid" (**uint32_t**)
　　Node identifier expressed as the node's index (beginning at zero) in an array of nodes within
　　the active session. The value must be unique and directly correlate to the **PMIX_HOSTNAME**
　　of the node - i.e., users can interchangeably reference the same location using either the
　　PMIX_HOSTNAME or corresponding **PMIX_NODEID** .
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.
PMIX_EXIT_CODE "pmix.exit.code" (**int**)
　　Exit code returned when process terminated

14.4.11 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.

14.4.12 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 11.2.3.1 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. The **PMIX_SESSION_ID** attribute of the **session** containing the **job** is required to be included in the array whenever the PMIx server library may host multiple sessions (e.g., when executing with a host RM daemon). As information is registered one job (aka namespace) at a time via the **PMIx_server_register_nspace** API, there is no requirement that the array contain either the **PMIX_NSPACE** or **PMIX_JOBID** attributes when used in that context (though either or both of them may be included). At least one of the job identifiers must be provided in all other contexts where the job being referenced is ambiguous.

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 application, plus its **PMIX_APPNUM** attribute, are must to be included in the array when the array is *not* included as part of a call to **PMIx_server_register_nspace** - i.e., when the job containing the application is ambiguous. The job identification is otherwise optional.

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.

Note that these assemblages can be used hierarchically:

- a **PMIX_JOB_INFO_ARRAY** might contain multiple **PMIX_APP_INFO_ARRAY** elements, each describing values for a specific application within the job

- a **PMIX_JOB_INFO_ARRAY** could contain a **PMIX_NODE_INFO_ARRAY** for each node hosting processes from that job, each array describing job-level values for that node

- a **PMIX_SESSION_INFO_ARRAY** might contain multiple **PMIX_JOB_INFO_ARRAY** elements, each describing a job executing within the session. Each job array could, in turn, contain both application and node arrays, thus providing a complete picture of the active operations within the allocation

—————————————— Advice to PMIx library implementers ——————————————

PMIx implementations must be capable of properly parsing and storing any hierarchical depth of information arrays. The resulting stored values are must to be accessible via both **PMIx_Get** and **PMIx_Query_info_nb** APIs, assuming appropriate directives are provided by the caller.

14.4.13 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_MAX_PROCS** 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. Note that this value can be different from **PMIX_MAX_PROCS** . For example, users may choose to subdivide an allocation (running several jobs in parallel within it), and dynamic programming models may support adding and removing processes from a running **job** on-they-fly. In the latter case, PMIx events must be used to notify processes within the job that the job size has changed.

1	**PMIX_JOB_NUM_APPS "pmix.job.napps"** (**uint32_t**)
2	Number of applications in this job.
3	**PMIX_APP_SIZE "pmix.app.size"** (**uint32_t**)
4	Number of processes in this application.
5	**PMIX_LOCAL_SIZE "pmix.local.size"** (**uint32_t**)
6	Number of processes in this job or application on this node.
7	**PMIX_NODE_SIZE "pmix.node.size"** (**uint32_t**)
8	Number of processes across all jobs on this node.
9	**PMIX_MAX_PROCS "pmix.max.size"** (**uint32_t**)
10	Maximum number of processes that can be executed in this context (session, namespace,
11	application, or node). Typically, this is a constraint imposed by a scheduler or by user
12	settings in a hostfile or other resource description.
13	**PMIX_NUM_SLOTS "pmix.num.slots"** (**uint32_t**)
14	Number of slots allocated in this context (session, namespace, application, or node). Note
15	that this attribute is the equivalent to **PMIX_MAX_PROCS** used in the corresponding
16	context - it is included in the Standard for historical reasons.
17	**PMIX_NUM_NODES "pmix.num.nodes"** (**uint32_t**)
18	Number of nodes in this session, or that are currently executing processes from the
19	associated namespace or application.

14.4.14 Memory information attributes

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

23	**PMIX_AVAIL_PHYS_MEMORY "pmix.pmem"** (**uint64_t**)
24	Total available physical memory on this node.
25	**PMIX_DAEMON_MEMORY "pmix.dmn.mem"** (**float**)
26	Megabytes of memory currently used by the RM daemon.
27	**PMIX_CLIENT_AVG_MEMORY "pmix.cl.mem.avg"** (**float**)
28	Average Megabytes of memory used by client processes.

14.4.15 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.

32	**PMIX_LOCAL_TOPO "pmix.ltopo"** (**char***)
33	XML representation of local node topology.
34	**PMIX_TOPOLOGY "pmix.topo"** (**hwloc_topology_t**)
35	Pointer to the PMIx client's internal hwloc topology object.
36	**PMIX_TOPOLOGY_XML "pmix.topo.xml"** (**char***)
37	XML-based description of topology
38	**PMIX_TOPOLOGY_FILE "pmix.topo.file"** (**char***)
39	Full path to file containing XML topology description

```
1    PMIX_TOPOLOGY_SIGNATURE "pmix.toposig" (char*)
2         Topology signature string.
3    PMIX_LOCALITY_STRING "pmix.locstr" (char*)
4         String describing a process's bound location - referenced using the process's rank. The string
5         is of the form:
6         NM%s:SK%s:L3%s:L2%s:L1%s:CR%s:HT%s
7         Where the %s is replaced with an integer index or inclusive range for hwloc. NM identifies
8         the numa node(s). SK identifies the socket(s). L3 identifies the L3 cache(s). L2 identifies the
9         L2 cache(s). L1 identifies the L1 cache(s). CR identifies the cores(s). HT identifies the
10        hardware thread(s). If your architecture does not have the specified hardware designation
11        then it can be omitted from the signature.
12        For example: NM0:SK0:L30-4:L20-4:L10-4:CR0-4:HT0-39.
13        This means numa node 0, socket 0, L3 caches 0,1,2,3,4, L2 caches 0-4, L1 caches
14        0-4, cores 0,1,2,3,4, and hardware threads 0-39.
15   PMIX_HWLOC_SHMEM_ADDR "pmix.hwlocaddr" (size_t)
16        Address of the HWLOC shared memory segment.
17   PMIX_HWLOC_SHMEM_SIZE "pmix.hwlocsize" (size_t)
18        Size of the HWLOC shared memory segment.
19   PMIX_HWLOC_SHMEM_FILE "pmix.hwlocfile" (char*)
20        Path to the HWLOC shared memory file.
21   PMIX_HWLOC_XML_V1 "pmix.hwlocxml1" (char*)
22        XML representation of local topology using HWLOC's v1.x format.
23   PMIX_HWLOC_XML_V2 "pmix.hwlocxml2" (char*)
24        XML representation of local topology using HWLOC's v2.x format.
25   PMIX_HWLOC_SHARE_TOPO "pmix.hwlocsh" (bool)
26        Share the HWLOC topology via shared memory
27   PMIX_HWLOC_HOLE_KIND "pmix.hwlocholek" (char*)
28        Kind of VM "hole" HWLOC should use for shared memory
```

14.4.16 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.

```
32   PMIX_COLLECT_DATA "pmix.collect" (bool)
33        Collect data and return it at the end of the operation.
34   PMIX_TIMEOUT "pmix.timeout" (int)
35        Time in seconds before the specified operation should time out (0 indicating infinite) in
36        error. The timeout parameter can help avoid "hangs" due to programming errors that prevent
37        the target process from ever exposing its data.
38   PMIX_IMMEDIATE "pmix.immediate" (bool)
39        Specified operation should immediately return an error from the PMIx server if the requested
40        data cannot be found - do not request it from the host RM.
41   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_GET_STATIC_VALUES **"pmix.get.static"** (**bool**)
Request that any pointers in the returned value point directly to values in the key-value store

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 returned by job upon its termination. The status will be communicated as part of a PMIx event payload provided by the host environment upon termination of a job. Note that generation of the **PMIX_ERR_JOB_TERMINATED** event is optional and host environments may choose to provide it only upon request.

PMIX_PROC_STATE_STATUS **"pmix.proc.state"** (**pmix_proc_state_t**)
State of the specified process as of the last report - may not be the actual current state based on update rate.

PMIX_PROC_TERM_STATUS **"pmix.proc.term.status"** (**pmix_status_t**)
Status returned by a process upon its termination. The status will be communicated as part of a PMIx event payload provided by the host environment upon termination of a process. Note that generation of the **PMIX_PROC_TERMINATED** event is optional and host environments may choose to provide it only upon request.

14.4.17 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 11.2.3.1 for an explanation of its generation.

PMIX_PROC_MAP "pmix.pmap" (**char***)
    Regular expression describing processes on each node - see 11.2.3.1 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.

14.4.18 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.

14.4.19 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.

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.

PMIX_EVENT_PROXY "pmix.evproxy" (pmix_proc_t*)
　　PMIx server that sourced the event

PMIX_EVENT_TEXT_MESSAGE "pmix.evtext" (char*)
　　Text message suitable for output by recipient - e.g., describing the cause of the event

14.4.20 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.

1 **PMIX_EVENT_ACTION_TIMEOUT "pmix.evtimeout"** (**int**)
2   The time in seconds before the RM will execute error response.
3 **PMIX_EVENT_NO_TERMINATION "pmix.evnoterm"** (**bool**)
4   Indicates that the handler has satisfactorily handled the event and believes termination of the
5   application is not required.
6 **PMIX_EVENT_WANT_TERMINATION "pmix.evterm"** (**bool**)
7   Indicates that the handler has determined that the application should be terminated

8 14.4.21 Spawn attributes

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

16 **PMIX_PERSONALITY "pmix.pers"** (**char***)
17   Name of personality to use.
18 **PMIX_HOST "pmix.host"** (**char***)
19   Comma-delimited list of hosts to use for spawned processes.
20 **PMIX_HOSTFILE "pmix.hostfile"** (**char***)
21   Hostfile to use for spawned processes.
22 **PMIX_ADD_HOST "pmix.addhost"** (**char***)
23   Comma-delimited list of hosts to add to the allocation.
24 **PMIX_ADD_HOSTFILE "pmix.addhostfile"** (**char***)
25   Hostfile listing hosts to add to existing allocation.
26 **PMIX_PREFIX "pmix.prefix"** (**char***)
27   Prefix to use for starting spawned processes.
28 **PMIX_WDIR "pmix.wdir"** (**char***)
29   Working directory for spawned processes.
30 **PMIX_MAPPER "pmix.mapper"** (**char***)
31   Mapping mechanism to use for placing spawned processes - when accessed using
32   **PMIx_Get**, use the **PMIX_RANK_WILDCARD** value for the rank to discover the mapping
33   mechanism used for the provided namespace.
34 **PMIX_DISPLAY_MAP "pmix.dispmap"** (**bool**)
35   Display process mapping upon spawn.
36 **PMIX_PPR "pmix.ppr"** (**char***)
37   Number of processes to spawn on each identified resource.
38 **PMIX_MAPBY "pmix.mapby"** (**char***)
39   Process mapping policy - when accessed using **PMIx_Get**, use the
40   **PMIX_RANK_WILDCARD** value for the rank to discover the mapping policy used for the
41   provided namespace

```
1    PMIX_RANKBY  "pmix.rankby" (char*)
2         Process ranking policy - when accessed using PMIx_Get , use the
3         PMIX_RANK_WILDCARD  value for the rank to discover the ranking algorithm used for the
4         provided namespace
5    PMIX_BINDTO  "pmix.bindto" (char*)
6         Process binding policy - when accessed using PMIx_Get , use the
7         PMIX_RANK_WILDCARD  value for the rank to discover the binding policy used for the
8         provided namespace
9    PMIX_PRELOAD_BIN  "pmix.preloadbin" (bool)
10        Preload binaries onto nodes.
11   PMIX_PRELOAD_FILES  "pmix.preloadfiles" (char*)
12        Comma-delimited list of files to pre-position on nodes.
13   PMIX_NON_PMI  "pmix.nonpmi" (bool)
14        Spawned processes will not call PMIx_Init .
15   PMIX_STDIN_TGT  "pmix.stdin" (uint32_t)
16        Spawned process rank that is to receive stdin.
17   PMIX_FWD_STDIN  "pmix.fwd.stdin" (bool)
18        Forward this process's stdin to the designated process.
19   PMIX_FWD_STDOUT  "pmix.fwd.stdout" (bool)
20        Forward stdout from spawned processes to this process.
21   PMIX_FWD_STDERR  "pmix.fwd.stderr" (bool)
22        Forward stderr from spawned processes to this process.
23   PMIX_FWD_STDDIAG  "pmix.fwd.stddiag" (bool)
24        If a diagnostic channel exists, forward any output on it from the spawned processes to this
25        process (typically used by a tool)
26   PMIX_DEBUGGER_DAEMONS  "pmix.debugger" (bool)
27        Spawned application consists of debugger daemons.
28   PMIX_COSPAWN_APP  "pmix.cospawn" (bool)
29        Designated application is to be spawned as a disconnected job. Meaning that it is not part of
30        the "comm_world" of the parent process.
31   PMIX_SET_SESSION_CWD  "pmix.ssncwd" (bool)
32        Set the application's current working directory to the session working directory assigned by
33        the RM - when accessed using PMIx_Get , use the PMIX_RANK_WILDCARD  value for
34        the rank to discover the session working directory assigned to the provided namespace
35   PMIX_TAG_OUTPUT  "pmix.tagout" (bool)
36        Tag application output with the identity of the source process.
37   PMIX_TIMESTAMP_OUTPUT  "pmix.tsout" (bool)
38        Timestamp output from applications.
39   PMIX_MERGE_STDERR_STDOUT  "pmix.mergeerrout" (bool)
40        Merge stdout and stderr streams from application processes.
41   PMIX_OUTPUT_TO_FILE  "pmix.outfile" (char*)
42        Output application output to the specified file.
43   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_SPAWN_TOOL "pmix.spwn.tool" (**bool**)
Indicate that the job being spawned is a tool

14.4.22 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** .

1 **PMIX_QUERY_LOCAL_PROC_TABLE "pmix.qry.lptable"** (**char***)
2   Input namespace of the job whose information is being requested returns (
3   **pmix_data_array_t**) an array of **pmix_proc_info_t** for processes in job on same
4   node.
5 **PMIX_QUERY_AUTHORIZATIONS "pmix.qry.auths"** (**bool**)
6   Return operations the PMIx tool is authorized to perform.
7 **PMIX_QUERY_SPAWN_SUPPORT "pmix.qry.spawn"** (**bool**)
8   Return a comma-delimited list of supported spawn attributes.
9 **PMIX_QUERY_DEBUG_SUPPORT "pmix.qry.debug"** (**bool**)
10   Return a comma-delimited list of supported debug attributes.
11 **PMIX_QUERY_MEMORY_USAGE "pmix.qry.mem"** (**bool**)
12   Return information on memory usage for the processes indicated in the qualifiers.
13 **PMIX_QUERY_LOCAL_ONLY "pmix.qry.local"** (**bool**)
14   Constrain the query to local information only.
15 **PMIX_QUERY_REPORT_AVG "pmix.qry.avg"** (**bool**)
16   Report only average values for sampled information.
17 **PMIX_QUERY_REPORT_MINMAX "pmix.qry.minmax"** (**bool**)
18   Report minimum and maximum values.
19 **PMIX_QUERY_ALLOC_STATUS "pmix.query.alloc"** (**char***)
20   String identifier of the allocation whose status is being requested.
21 **PMIX_TIME_REMAINING "pmix.time.remaining"** (**char***)
22   Query number of seconds (**uint32_t**) remaining in allocation for the specified namespace.
23 **PMIX_QUERY_ATTRIBUTE_SUPPORT "pmix.qry.attrs"** (**bool**)
24   Query list of supported attributes for specified APIs
25 **PMIX_QUERY_NUM_PSETS "pmix.qry.psetnum"** (**size_t**)
26   Return the number of psets defined in the specified range (defaults to session).
27 **PMIX_QUERY_PSET_NAMES "pmix.qry.psets"** (**char***)
28   Return a comma-delimited list of the names of the psets defined in the specified range
29   (defaults to session).

14.4.23 Log attributes

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

33 **PMIX_LOG_SOURCE "pmix.log.source"** (**pmix_proc_t***)
34   ID of source of the log request
35 **PMIX_LOG_STDERR "pmix.log.stderr"** (**char***)
36   Log string to **stderr**.
37 **PMIX_LOG_STDOUT "pmix.log.stdout"** (**char***)
38   Log string to **stdout**.
39 **PMIX_LOG_SYSLOG "pmix.log.syslog"** (**char***)
40   Log data to syslog. Defaults to **ERROR** priority. Will log to global syslog if available,
41   otherwise to local syslog

PMIX_LOG_LOCAL_SYSLOG "pmix.log.lsys" (**char***)
    Log data to local syslog. Defaults to **ERROR** priority.
PMIX_LOG_GLOBAL_SYSLOG "pmix.log.gsys" (**char***)
    Forward data to system "gateway" and log msg to that syslog Defaults to **ERROR** priority.
PMIX_LOG_SYSLOG_PRI "pmix.log.syspri" (**int**)
    Syslog priority level
PMIX_LOG_TIMESTAMP "pmix.log.tstmp" (**time_t**)
    Timestamp for log report
PMIX_LOG_GENERATE_TIMESTAMP "pmix.log.gtstmp" (**bool**)
    Generate timestamp for log
PMIX_LOG_TAG_OUTPUT "pmix.log.tag" (**bool**)
    Label the output stream with the channel name (e.g., "stdout")
PMIX_LOG_TIMESTAMP_OUTPUT "pmix.log.tsout" (**bool**)
    Print timestamp in output string
PMIX_LOG_XML_OUTPUT "pmix.log.xml" (**bool**)
    Print the output stream in XML format
PMIX_LOG_ONCE "pmix.log.once" (**bool**)
    Only log this once with whichever channel can first support it, taking the channels in priority
    order
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_SENDER_ADDR "pmix.log.emfaddr" (**char***)
    Return email address of sender
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.
PMIX_LOG_EMAIL_SERVER "pmix.log.esrvr" (**char***)
    Hostname (or IP address) of estmp server
PMIX_LOG_EMAIL_SRVR_PORT "pmix.log.esrvrprt" (**int32_t**)
    Port the email server is listening to
PMIX_LOG_GLOBAL_DATASTORE "pmix.log.gstore" (**bool**)
    Store the log data in a global data store (e.g., database)
PMIX_LOG_JOB_RECORD "pmix.log.jrec" (**bool**)
    Log the provided information to the host environment's job record

14.4.24 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.

PMIX_DEBUG_JOB_DIRECTIVES "pmix.dbg.jdirs" (**pmix_data_array_t***)
> Array of job-level directives

PMIX_DEBUG_APP_DIRECTIVES "pmix.dbg.adirs" (**pmix_data_array_t***)
> Array of app-level directives

14.4.25 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.

14.4.26 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.envar.set" (pmix_envar_t*)
    Set the envar to the given value, overwriting any pre-existing one
PMIX_UNSET_ENVAR "pmix.envar.unset" (char*)
    Unset the environment variable specified in the string.
PMIX_ADD_ENVAR "pmix.envar.add" (pmix_envar_t*)
    Add the environment variable, but do not overwrite any pre-existing one
PMIX_PREPEND_ENVAR "pmix.envar.prepnd" (pmix_envar_t*)
    Prepend the given value to the specified environmental value using the given separator
    character, creating the variable if it doesn't already exist
PMIX_APPEND_ENVAR "pmix.envar.appnd" (pmix_envar_t*)
    Append the given value to the specified environmental value using the given separator
    character, creating the variable if it doesn't already exist

14.4.27 Job Allocation attributes

Attributes used to describe the job allocation - these are values passed to and/or returned by the **PMIx_Allocation_request_nb** and **PMIx_Allocation_request** APIs and are not accessed using the **PMIx_Get** API

PMIX_ALLOC_REQ_ID "pmix.alloc.reqid" (char*)
    User-provided string identifier for this allocation request which can later be used to query
    status of the request.
PMIX_ALLOC_ID "pmix.alloc.id" (char*)
    A string identifier (provided by the host environment) for the resulting allocation which can
    later be used to reference the allocated resources in, for example, a call to **PMIx_Spawn** .
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)
    Changed to **PMIX_ALLOC_FABRIC**
PMIX_ALLOC_FABRIC "pmix.alloc.net" (array)

Array of **pmix_info_t** describing requested fabric resources. This must include at least:
PMIX_ALLOC_FABRIC_ID , **PMIX_ALLOC_FABRIC_TYPE** , and
PMIX_ALLOC_FABRIC_ENDPTS , plus whatever other descriptors are desired.

PMIX_ALLOC_NETWORK_ID "pmix.alloc.netid" (**char***)
  Changed to **PMIX_ALLOC_FABRIC_ID**

PMIX_ALLOC_FABRIC_ID "pmix.alloc.netid" (**char***)
  The key to be used when accessing this requested fabric allocation. The allocation will be
  returned/stored as a **pmix_data_array_t** of **pmix_info_t** indexed by this key and
  containing at least one entry with the same key and the allocated resource description. The
  type of the included value depends upon the fabric support. For example, a TCP allocation
  might consist of a comma-delimited string of socket ranges such as
  "32000-32100,33005,38123-38146". Additional entries will consist of any provided
  resource request directives, along with their assigned values. Examples include:
  PMIX_ALLOC_FABRIC_TYPE - the type of resources provided;
  PMIX_ALLOC_FABRIC_PLANE - if applicable, what plane the resources were assigned
  from; **PMIX_ALLOC_FABRIC_QOS** - the assigned QoS; **PMIX_ALLOC_BANDWIDTH** -
  the allocated bandwidth; **PMIX_ALLOC_FABRIC_SEC_KEY** - a security key for the
  requested fabric allocation. NOTE: the assigned values may differ from those requested,
  especially if **PMIX_INFO_REQD** was not set in the request.

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

PMIX_ALLOC_NETWORK_QOS "pmix.alloc.netqos" (**char***)
  Changed to **PMIX_ALLOC_FABRIC_QOS**

PMIX_ALLOC_FABRIC_QOS "pmix.alloc.netqos" (**char***)
  Quality of service level.

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

PMIX_ALLOC_NETWORK_TYPE "pmix.alloc.nettype" (**char***)
  Changed to **PMIX_ALLOC_FABRIC_TYPE**

PMIX_ALLOC_FABRIC_TYPE "pmix.alloc.nettype" (**char***)
  Type of desired transport (e.g., *"tcp"*, *"udp"*)

PMIX_ALLOC_NETWORK_PLANE "pmix.alloc.netplane" (**char***)
  Changed to **PMIX_ALLOC_FABRIC_PLANE**

PMIX_ALLOC_FABRIC_PLANE "pmix.alloc.netplane" (**char***)
  ID string for the NIC (aka *plane*) to be used for this allocation (e.g., CIDR for Ethernet)

PMIX_ALLOC_NETWORK_ENDPTS "pmix.alloc.endpts" (**size_t**)
  Changed to **PMIX_ALLOC_FABRIC_ENDPTS**

PMIX_ALLOC_FABRIC_ENDPTS "pmix.alloc.endpts" (**size_t**)
  Number of endpoints to allocate per process

PMIX_ALLOC_NETWORK_ENDPTS_NODE "pmix.alloc.endpts.nd" (**size_t**)
  Changed to **PMIX_ALLOC_FABRIC_ENDPTS_NODE**

PMIX_ALLOC_FABRIC_ENDPTS_NODE "pmix.alloc.endpts.nd" (**size_t**)
  Number of endpoints to allocate per node

1   **PMIX_ALLOC_NETWORK_SEC_KEY** **"pmix.alloc.nsec"** (**pmix_byte_object_t**)
2   Changed to **PMIX_ALLOC_FABRIC_SEC_KEY**
3   **PMIX_ALLOC_FABRIC_SEC_KEY** **"pmix.alloc.nsec"** (**pmix_byte_object_t**)
4   Fabric security key

14.4.28   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. The user can provide an identifier for the
        requested operation, thus allowing them to later request status of the operation or to
        terminate it. The host, therefore, shall track it with the request for future reference.
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 - the provided request ID must match the
        PMIX_JOB_CTRL_ID provided to a previous call to **PMIx_Job_control** . An ID of
        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**)

1	Indicate that the job can be pre-empted.
2	**PMIX_JOB_CTRL_TERMINATE "pmix.jctrl.term"** (**bool**)
3	Politely terminate the specified processes.
4	**PMIX_REGISTER_CLEANUP "pmix.reg.cleanup"** (**char***)
5	Comma-delimited list of files to be removed upon process termination
6	**PMIX_REGISTER_CLEANUP_DIR "pmix.reg.cleanupdir"** (**char***)
7	Comma-delimited list of directories to be removed upon process termination
8	**PMIX_CLEANUP_RECURSIVE "pmix.clnup.recurse"** (**bool**)
9	Recursively cleanup all subdirectories under the specified one(s)
10	**PMIX_CLEANUP_EMPTY "pmix.clnup.empty"** (**bool**)
11	Only remove empty subdirectories
12	**PMIX_CLEANUP_IGNORE "pmix.clnup.ignore"** (**char***)
13	Comma-delimited list of filenames that are not to be removed
14	**PMIX_CLEANUP_LEAVE_TOPDIR "pmix.clnup.lvtop"** (**bool**)
15	When recursively cleaning subdirectories, do not remove the top-level directory (the one
16	given in the cleanup request)

14.4.29 Monitoring attributes

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

20	**PMIX_MONITOR_ID "pmix.monitor.id"** (**char***)
21	Provide a string identifier for this request.
22	**PMIX_MONITOR_CANCEL "pmix.monitor.cancel"** (**char***)
23	Identifier to be canceled (**NULL** means cancel all monitoring for this process).
24	**PMIX_MONITOR_APP_CONTROL "pmix.monitor.appctrl"** (**bool**)
25	The application desires to control the response to a monitoring event.
26	**PMIX_MONITOR_HEARTBEAT "pmix.monitor.mbeat"** (**void**)
27	Register to have the PMIx server monitor the requestor for heartbeats.
28	**PMIX_SEND_HEARTBEAT "pmix.monitor.beat"** (**void**)
29	Send heartbeat to local PMIx server.
30	**PMIX_MONITOR_HEARTBEAT_TIME "pmix.monitor.btime"** (**uint32_t**)
31	Time in seconds before declaring heartbeat missed.
32	**PMIX_MONITOR_HEARTBEAT_DROPS "pmix.monitor.bdrop"** (**uint32_t**)
33	Number of heartbeats that can be missed before generating the event.
34	**PMIX_MONITOR_FILE "pmix.monitor.fmon"** (**char***)
35	Register to monitor file for signs of life.
36	**PMIX_MONITOR_FILE_SIZE "pmix.monitor.fsize"** (**bool**)
37	Monitor size of given file is growing to determine if the application is running.
38	**PMIX_MONITOR_FILE_ACCESS "pmix.monitor.faccess"** (**char***)
39	Monitor time since last access of given file to determine if the application is running.
40	**PMIX_MONITOR_FILE_MODIFY "pmix.monitor.fmod"** (**char***)
41	Monitor time since last modified of given file to determine if the application is running.

1	**PMIX_MONITOR_FILE_CHECK_TIME "pmix.monitor.ftime"** (**uint32_t**)
2	Time in seconds between checking the file.
3	**PMIX_MONITOR_FILE_DROPS "pmix.monitor.fdrop"** (**uint32_t**)
4	Number of file checks that can be missed before generating the event.

14.4.30 Security attributes

6 *PMIx v3.0*   Attributes for managing security credentials

7	**PMIX_CRED_TYPE "pmix.sec.ctype"** (**char***)
8	When passed in **PMIx_Get_credential** , a prioritized, comma-delimited list of desired
9	credential types for use in environments where multiple authentication mechanisms may be
10	available. When returned in a callback function, a string identifier of the credential type.
11	**PMIX_CRYPTO_KEY "pmix.sec.key"** (**pmix_byte_object_t**)
12	Blob containing crypto key

14.4.31 IO Forwarding attributes

14 *PMIx v3.0*   Attributes used to control IO forwarding behavior

15	**PMIX_IOF_CACHE_SIZE "pmix.iof.csize"** (**uint32_t**)
16	The requested size of the server cache in bytes for each specified channel. By default, the
17	server is allowed (but not required) to drop all bytes received beyond the max size.
18	**PMIX_IOF_DROP_OLDEST "pmix.iof.old"** (**bool**)
19	In an overflow situation, drop the oldest bytes to make room in the cache.
20	**PMIX_IOF_DROP_NEWEST "pmix.iof.new"** (**bool**)
21	In an overflow situation, drop any new bytes received until room becomes available in the
22	cache (default).
23	**PMIX_IOF_BUFFERING_SIZE "pmix.iof.bsize"** (**uint32_t**)
24	Controls grouping of IO on the specified channel(s) to avoid being called every time a bit of
25	IO arrives. The library will execute the callback whenever the specified number of bytes
26	becomes available. Any remaining buffered data will be "flushed" upon call to deregister the
27	respective channel.
28	**PMIX_IOF_BUFFERING_TIME "pmix.iof.btime"** (**uint32_t**)
29	Max time in seconds to buffer IO before delivering it. Used in conjunction with buffering
30	size, this prevents IO from being held indefinitely while waiting for another payload to arrive.
31	**PMIX_IOF_COMPLETE "pmix.iof.cmp"** (**bool**)
32	Indicates whether or not the specified IO channel has been closed by the source.
33	**PMIX_IOF_TAG_OUTPUT "pmix.iof.tag"** (**bool**)
34	Tag output with the channel it comes from.
35	**PMIX_IOF_TIMESTAMP_OUTPUT "pmix.iof.ts"** (**bool**)
36	Timestamp output
37	**PMIX_IOF_XML_OUTPUT "pmix.iof.xml"** (**bool**)
38	Format output in XML

14.4.32 Application setup attributes

Attributes for controlling contents of application setup data

PMIX_SETUP_APP_ENVARS "pmix.setup.env" (**bool**)
      Harvest and include relevant environmental variables
PMIX_SETUP_APP_NONENVARS ""pmix.setup.nenv" (**bool**)
      Include all relevant data other than environmental variables
PMIX_SETUP_APP_ALL "pmix.setup.all" (**bool**)
      Include all relevant data

14.4.33 Attribute support level attributes

PMIX_CLIENT_FUNCTIONS "pmix.client.fns" (**bool**)
      Request a list of functions supported by the PMIx client library
PMIX_CLIENT_ATTRIBUTES "pmix.client.attrs" (**bool**)
      Request attributes supported by the PMIx client library
PMIX_SERVER_FUNCTIONS "pmix.srvr.fns" (**bool**)
      Request a list of functions supported by the PMIx server library
PMIX_SERVER_ATTRIBUTES "pmix.srvr.attrs" (**bool**)
      Request attributes supported by the PMIx server library
PMIX_HOST_FUNCTIONS "pmix.srvr.fns" (**bool**)
      Request a list of functions supported by the host environment
PMIX_HOST_ATTRIBUTES "pmix.host.attrs" (**bool**)
      Request attributes supported by the host environment
PMIX_TOOL_FUNCTIONS "pmix.tool.fns" (**bool**)
      Request a list of functions supported by the PMIx tool library
PMIX_TOOL_ATTRIBUTES "pmix.setup.env" (**bool**)
      Request attributes supported by the PMIx tool library functions

14.4.34 Descriptive attributes

PMIX_MAX_VALUE "pmix.descr.maxval" (**varies**)
      Used in **pmix_regattr_t** to describe the maximum valid value for the associated
      attribute.
PMIX_MIN_VALUE "pmix.descr.minval" (**varies**)
      Used in **pmix_regattr_t** to describe the minimum valid value for the associated
      attribute.
PMIX_ENUM_VALUE "pmix.descr.enum" (**char***)
      Used in **pmix_regattr_t** to describe accepted values for the associated attribute.
      Numerical values shall be presented in a form convertible to the attribute's declared data
      type. Named values (i.e., values defined by constant names via a typical C-language enum
      declaration) must be provided as their numerical equivalent.

14.4.35 Process group attributes

PMIx v4.0 Attributes for controlling the PMIx Group APIs

PMIX_GROUP_ID **"pmix.grp.id"** (**char***)
    User-provided group identifier
PMIX_GROUP_LEADER **"pmix.grp.ldr"** (**bool**)
    This process is the leader of the group
PMIX_GROUP_OPTIONAL **"pmix.grp.opt"** (**bool**)
    Participation is optional - do not return an error if any of the specified processes terminate
    without having joined. The default is false
PMIX_GROUP_NOTIFY_TERMINATION **"pmix.grp.notterm"** (**bool**)
    Notify remaining members when another member terminates without first leaving the group.
    The default is false
PMIX_GROUP_INVITE_DECLINE **"pmix.grp.decline"** (**bool**)
    Notify the inviting process that this process does not wish to participate in the proposed
    group The default is true
PMIX_GROUP_MEMBERSHIP **"pmix.grp.mbrs"** (**pmix_data_array_t***)
    Array of group member ID's
PMIX_GROUP_ASSIGN_CONTEXT_ID **"pmix.grp.actxid"** (**bool**)
    Requests that the RM assign a new context identifier to the newly created group. The
    identifier is an unsigned, **size_t** value that the RM guarantees to be unique across the range
    specified in the request. Thus, the value serves as a means of identifying the group within
    that range. If no range is specified, then the request defaults to **PMIX_RANGE_SESSION** .
PMIX_GROUP_CONTEXT_ID **"pmix.grp.ctxid"** (**size_t**)
    Context identifier assigned to the group by the host RM.
PMIX_GROUP_LOCAL_ONLY **"pmix.grp.lcl"** (**bool**)
    Group operation only involves local processes. PMIx implementations are *required* to
    automatically scan an array of group members for local vs remote processes - if only local
    processes are detected, the implementation need not execute a global collective for the
    operation unless a context ID has been requested from the host environment. This can result
    in significant time savings. This attribute can be used to optimize the operation by indicating
    whether or not only local processes are represented, thus allowing the implementation to
    bypass the scan. The default is false
PMIX_GROUP_ENDPT_DATA **"pmix.grp.endpt"** (**pmix_byte_object_t**)
    Data collected to be shared during group construction

14.5 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.

14.5.1   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.

14.5.2   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 business card
exchange (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**
1		
2		Callback for releasing *data* (function pointer)
3	IN	**release_cbdata**
4		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.

14.5.3 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**
16		
17		Status associated with the operation (handle)
18	IN	**nspace**
19		Namespace string (**pmix_nspace_t**)
20	IN	**cbdata**
21		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.

1## 14.5.4 Op Callback Function

2**Summary**

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

PMIx v1.0 ────────────────────── C ──────────────────────

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

────────────────────── C ──────────────────────

6**IN   status**
7   Status associated with the operation (handle)
8**IN   cbdata**
9   Callback data passed to original API call (memory reference)

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

14.5.5 Lookup Callback Function

15**Summary**

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

PMIx v1.0 ────────────────────── C ──────────────────────

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

────────────────────── C ──────────────────────

21**IN   status**
22   Status associated with the operation (handle)
23**IN   data**
24   Array of data returned (**pmix_pdata_t**)
25**IN   ndata**
26   Number of elements in the *data* array (**size_t**)
27**IN   cbdata**
28   Callback data passed to original API call (memory reference)

 366   PMIx Standard – Version 4.0 (Draft) – *Created on June 22, 2020*

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.

14.5.6 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.

14.5.7 Info Callback Function

Summary

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

PMIx v2.0 ──────────────── 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);
```

1	**IN status**
2	Status associated with the operation (**pmix_status_t**)
3	**IN info**
4	Array of **pmix_info_t** returned by the operation (pointer)
5	**IN ninfo**
6	Number of elements in the *info* array (**size_t**)
7	**IN cbdata**
8	Callback data passed to original API call (memory reference)
9	**IN release_fn**
10	Function to be called when done with the *info* data (function pointer)
11	**IN release_cbdata**
12	Callback data to be passed to *release_fn* (memory reference)

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

16   14.5.8   Event Handler Registration Callback Function

17   The **pmix_evhdlr_reg_cbfunc_t** callback function.

---------------------------------- Advice to users ----------------------------------

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

PMIx v2.0 ─────────────────────────── C ───────────────────────────

```
22      typedef void (*pmix_evhdlr_reg_cbfunc_t)
23          (pmix_status_t status,
24           size_t evhdlr_ref,
25           void *cbdata)
```

─────────────────────────── C ───────────────────────────

26	**IN status**
27	Status indicates if the request was successful or not (**pmix_status_t**)
28	**IN evhdlr_ref**
29	Reference assigned to the event handler by PMIx — this reference * must be used to
30	deregister the err handler (**size_t**)
31	**IN cbdata**
32	Callback data passed to original API call (memory reference)

| 1 | **Description** |
| 2 | Define a callback function for calls to **PMIx_Register_event_handler** |

14.5.9 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 ─────────────────

```
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_EVENT_ACTION_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.

14.5.10 Notification Function

2 **Summary**
3 The **pmix_notification_fn_t** is called by PMIx to deliver notification of an event.

——————————————— Advice to users ———————————————

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

PMIx v2.0 ——————————————— C ———————————————

```
7  typedef void (*pmix_notification_fn_t)
8      (size_t evhdlr_registration_id,
9       pmix_status_t status,
10      const pmix_proc_t *source,
11      pmix_info_t info[], size_t ninfo,
12      pmix_info_t results[], size_t nresults,
13      pmix_event_notification_cbfunc_fn_t cbfunc,
14      void *cbdata);
```

——————————————— C ———————————————

15 **IN   evhdlr_registration_id**
16   Registration number of the handler being called (**size_t**)
17 **IN   status**
18   Status associated with the operation (**pmix_status_t**)
19 **IN   source**
20   Identifier of the process that generated the event (**pmix_proc_t**). If the source is the SMS,
21   then the nspace will be empty and the rank will be PMIX_RANK_UNDEF
22 **IN   info**
23   Information describing the event (**pmix_info_t**). This argument will be NULL if no
24   additional information was provided by the event generator.
25 **IN   ninfo**
26   Number of elements in the info array (**size_t**)
27 **IN   results**
28   Aggregated results from prior event handlers servicing this event (**pmix_info_t**). This
29   argument will be **NULL** if this is the first handler servicing the event, or if no prior handlers
30   provided results.
31 **IN   nresults**
32   Number of elements in the results array (**size_t**)
33 **IN   cbfunc**
34   **pmix_event_notification_cbfunc_fn_t** callback function to be executed upon
35   completion of the handler's operation and prior to handler return (function reference).

1 **IN   cbdata**
2   Callback data to be passed to cbfunc (memory reference)

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

───────────────────────────── Advice to users ─────────────────────────────

8 Possible uses of the *info* array include:

9 • for the host RM to alert the process as to planned actions, such as aborting the session, in
10   response to the reported event

11 • provide a timeout for alternative action to occur, such as for the application to request an
12   alternate response to the event

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

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

───────────────────────── Advice to PMIx server hosts ─────────────────────────

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

14.5.11 Server Setup Application Callback Function

25 The **PMIx_server_setup_application** callback function.

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

Format

──────────────────────── 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.

14.5.12 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

──────────────────────── C ────────────────────────

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

1   **IN**   **status**
2   Returned status of the request (**pmix_status_t**)
3   **IN**   **data**
4   Pointer to a data "blob" containing the requested information (handle)
5   **IN**   **sz**
6   Number of bytes in the *data* blob (integer)
7   **IN**   **cbdata**
8   Data passed into the initial call to **PMIx_server_dmodex_request** (memory reference)

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

14.5.13   PMIx Client Connection Callback Function

15 **Summary**
16 Callback function for incoming connection request from a local client

17 **Format**

PMIx v1.0

18 **typedef void (*pmix_connection_cbfunc_t)(**
19   **int incoming_sd, void *cbdata)**

20   **IN**   **incoming_sd**
21   (integer)
22   **IN**   **cbdata**
23   (memory reference)

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

14.5.14   PMIx Tool Connection Callback Function

28 **Summary**
29 Callback function for incoming tool connections.

Format

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

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.

14.5.15 Credential callback function

Summary

Callback function to return a requested security credential

Format

```
typedef void (*pmix_credential_cbfunc_t)(
                                pmix_status_t status,
                                pmix_byte_object_t *credential,
                                pmix_info_t info[], size_t ninfo,
                                void *cbdata)
```

IN status
   pmix_status_t value (handle)
IN credential
   pmix_byte_object_t structure containing the security credential (handle)
IN info
   Array of provided by the system to pass any additional information about the credential - e.g.,
   the identity of the issuing agent. (handle)
IN ninfo
   Number of elements in *info* (**size_t**)
IN cbdata
   Object passed in original request (memory reference)

Description
Define a callback function to return a requested security credential. Information provided by the
issuing agent can subsequently be used by the application for a variety of purposes. Examples
include:

- checking identified authorizations to determine what requests/operations are feasible as a means
  to steering **workflows**

- compare the credential type to that of the local SMS for compatibility

———————————————————— Advice to users ————————————————————

The credential is opaque and therefore understandable only by a service compatible with the issuer.
The *info* array is owned by the PMIx library and is not to be released or altered by the receiving
party.

14.5.16 Credential validation callback function

Summary
Callback function for security credential validation

Format

```
C ─────────────────────
typedef void (*pmix_validation_cbfunc_t)(
                                pmix_status_t status,
                                pmix_info_t info[], size_t ninfo,
                                void *cbdata);
C ─────────────────────
```

6   **IN   status**
7   **pmix_status_t** value (handle)
8   **IN   info**
9   Array of **pmix_info_t** provided by the system to pass any additional information about
10   the authentication - e.g., the effective userid and group id of the certificate holder, and any
11   related authorizations (handle)
12 **IN   ninfo**
13   Number of elements in *info* (**size_t**)
14 **IN   cbdata**
15   Object passed in original request (memory reference)

Description
16 Define a validation callback function to indicate if a provided credential is valid, and any
17 corresponding information regarding authorizations and other security matters.
18

──────────────── Advice to users ────────────────

19 The precise contents of the array will depend on the host environment and its associated security
20 system. At the minimum, it is expected (but not required) that the array will contain entries for the
21 **PMIX_USERID** and **PMIX_GRPID** of the client described in the credential. The *info* array is
22 owned by the PMIx library and is not to be released or altered by the receiving party.

14.5.17   IOF delivery function

Summary
Callback function for delivering forwarded IO to a process

```
typedef void (*pmix_iof_cbfunc_t)(
                              size_t iofhdlr, pmix_iof_channel_t channel,
                              pmix_proc_t *source, char *payload,
                              pmix_info_t info[], size_t ninfo);
```

6 **IN   iofhdlr**
7 Registration number of the handler being invoked (**size_t**)
8 **IN   channel**
9 bitmask identifying the channel the data arrived on (**pmix_iof_channel_t**)
10 **IN   source**
11 Pointer to a **pmix_proc_t** identifying the namespace/rank of the process that generated the
12 data (**char***)
13 **IN   payload**
14 Pointer to character array containing the data.
15 **IN   info**
16 Array of **pmix_info_t** provided by the source containing metadata about the payload.
17 This could include **PMIX_IOF_COMPLETE** (handle)
18 **IN   ninfo**
19 Number of elements in *info* (**size_t**)

20 **Description**
21 Define a callback function for delivering forwarded IO to a process. This function will be called
22 whenever data becomes available, or a specified buffering size and/or time has been met.

— Advice to users —

23 Multiple strings may be included in a given *payload*, and the *payload* may *not* be **NULL** terminated.
24 The user is responsible for releasing the *payload* memory. The *info* array is owned by the PMIx
25 library and is not to be released or altered by the receiving party.

14.5.18   IOF and Event registration function

27 **Summary**
28 Callback function for calls to register handlers, e.g., event notification and IOF requests.

Format

```
2    typedef void (*pmix_hdlr_reg_cbfunc_t)(pmix_status_t status,
3                                           size_t refid,
4                                           void *cbdata);
```

5   **IN   status**
6   **PMIX_SUCCESS** or an appropriate error constant (**pmix_status_t**)
7   **IN   refid**
8   reference identifier assigned to the handler by PMIx, used to deregister the handler (**size_t**)
9   **IN   cbdata**
10   object provided to the registration call (pointer)

11   **Description**
12   Callback function for calls to register handlers, e.g., event notification and IOF requests.

14.6 Constant String Functions

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

16   **Summary**
17   String representation of a **pmix_status_t** .

```
18   const char*
19   PMIx_Error_string(pmix_status_t status);
```

20   **Summary**
21   String representation of a **pmix_proc_state_t** .

```
22   const char*
23   PMIx_Proc_state_string(pmix_proc_state_t state);
```

1 **Summary**
2 String representation of a **pmix_scope_t** .

PMIx v2.0 ▼ ─────────────────── C ─────────────────── ▼

```
3 const char*
4 PMIx_Scope_string(pmix_scope_t scope);
```

▲ ─────────────────── C ─────────────────── ▲

5 **Summary**
6 String representation of a **pmix_persistence_t** .

PMIx v2.0 ▼ ─────────────────── C ─────────────────── ▼

```
7 const char*
8 PMIx_Persistence_string(pmix_persistence_t persist);
```

▲ ─────────────────── C ─────────────────── ▲

9 **Summary**
10 String representation of a **pmix_data_range_t** .

PMIx v2.0 ▼ ─────────────────── C ─────────────────── ▼

```
11 const char*
12 PMIx_Data_range_string(pmix_data_range_t range);
```

▲ ─────────────────── C ─────────────────── ▲

13 **Summary**
14 String representation of a **pmix_info_directives_t** .

PMIx v2.0 ▼ ─────────────────── C ─────────────────── ▼

```
15 const char*
16 PMIx_Info_directives_string(pmix_info_directives_t directives);
```

▲ ─────────────────── C ─────────────────── ▲

17 **Summary**
18 String representation of a **pmix_data_type_t** .

PMIx v2.0 ▼ ─────────────────── C ─────────────────── ▼

```
19 const char*
20 PMIx_Data_type_string(pmix_data_type_t type);
```

▲ ─────────────────── C ─────────────────── ▲

Summary

String representation of a **pmix_alloc_directive_t** .

PMIx v2.0 ▼ ─────────────────────── C ─────────────────────── ▼

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

▲ ─────────────────────── C ─────────────────────── ▲

Summary

String representation of a **pmix_iof_channel_t** .

PMIx v3.0 ▼ ─────────────────────── C ─────────────────────── ▼

```
const char*
PMIx_IOF_channel_string(pmix_iof_channel_t channel);
```

▲ ─────────────────────── C ─────────────────────── ▲

Summary

String representation of a **pmix_job_state_t** .

PMIx v4.0 ▼ ─────────────────────── C ─────────────────────── ▼

```
const char*
PMIx_Job_state_string(pmix_job_state_t state);
```

▲ ─────────────────────── C ─────────────────────── ▲

Summary

String representation of a PMIx attribute

PMIx v4.0 ▼ ─────────────────────── C ─────────────────────── ▼

```
const char*
PMIx_Get_attribute_string(char *attributename);
```

▲ ─────────────────────── C ─────────────────────── ▲

Summary

Return the PMIx attribute name corresponding to the given attribute string

PMIx v4.0 ▼ ─────────────────────── C ─────────────────────── ▼

```
const char*
PMIx_Get_attribute_name(char *attributestring);
```

▲ ─────────────────────── C ─────────────────────── ▲

1   **Summary**
2   String representation of a **pmix_link_state_t**

PMIx v4.0 ───────────────────── C ─────────────────────

3   `const char*`
4   `PMIx_Link_state_string(pmix_link_state_t state);`

───────────────────── C ─────────────────────

Python Bindings

<div>
1   While the PMIx Standard is defined in terms of C-based APIs, there is no intent to limit the use of
2   PMIx to that specific language. Support for other languages is captured in the Standard by
3   describing their equivalent syntax for the PMIx APIs and native forms for the PMIx datatypes. This
4   Appendix specifically deals with Python interfaces, beginning with a review of the PMIx datatypes.
5   Support is restricted to Python 3 and above - i.e., the Python bindings do not support Python 2.
</div>

6 Note: the PMIx APIs have been loosely collected into three Python classes based on their PMIx
7 "class" (i.e., client, server, and tool). All processes have access to a basic set of the APIs, and
8 therefore those have been included in the "client" class. Servers can utilize any of those functions
9 plus a set focused on operations not commonly executed by an application process. Finally, tools
10 can also act as servers but have their own initialization function.

A.1 Design Considerations

12 Several issues arose during design of the Python bindings:

A.1.1 Error Codes vs Python Exceptions

14 The C programming language reports errors through the return of the corresponding integer status
15 codes. PMIx has defined a range of negative values for this purpose. However, Python has the
16 option of raising *exceptions* that effectively operate as interrupts that can be trapped if the program
17 appropriately tests for them. The PMIx Python bindings opted to follow the C-based standard and
18 return PMIx status codes in lieu of raising exceptions as this method was considered more
19 consistent for those working in both domains.

A.1.2 Representation of Structured Data

21 PMIx utilizes a number of C-language structures to efficiently bundle related information. For
22 example, the PMIx process identifier is represented as a struct containing a character array for the
23 namespace and a 32-bit unsigned integer for the process rank. There are several options for
24 translating such objects to Python – e.g., the PMIx process identifier could be represented as a
25 two-element tuple (nspace, rank) or as a dictionary 'nspace': name, 'rank': 0. Exploration found no
26 discernible benefit to either representation, nor was any clearly identifiable rationale developed that
27 would lead a user to expect one versus the other for a given PMIx data type. Consistency in the
28 translation (i.e., exclusively using tuple or dictionary) appeared to be the most important criterion.
29 Hence, the decision was made to express all complex datatypes as Python dictionaries.

A.2 Datatype Definitions

PMIx defines a number of datatypes comprised of fixed-size character arrays, restricted range integers (e.g., uint32_t), and structures. Each datatype is represented by a named unsigned 16-bit integer (**uint16_t**) constant. Users are advised to use the named PMIx constants for indicating datatypes instead of integer values to ensure compatibility with future PMIx versions.

With only a few exceptions, the C-based PMIx datatypes defined in Chapter 14 on page 287 directly translate to Python. However, Python lacks the size-specific value definitions of C (e.g., **uint8_t**) and thus some care must be taken to protect against overflow/underflow situations when moving between the languages. Python bindings that accept values including PMIx datatypes shall therefore have the datatype and associated value checked for compatibility with their PMIx-defined equivalents, returning an error if:

- datatypes not defined by PMIx are encountered

- provided values fall outside the range of the C-equivalent definition - e.g., if a value identified as **PMIX_UINT8** lies outside the **uint8_t** range

Note that explicit labeling of PMIx datatype, even when Python itself doesn't care, is often required for the Python bindings to know how to properly interpret and label the provided value when passing it to the PMIx library.

Table A.1 lists the correspondence between datatypes in the two languages.

Table A.1.: C-to-Python Datatype Correspondence

C-Definition	PMIx Name	Python Definition	Notes
`bool`	PMIX_BOOL	boolean	
`byte`	PMIX_BYTE	A single element byte array (i.e., a byte array of length one)	
`char*`	PMIX_STRING	string	
`size_t`	PMIX_SIZE	integer	
`pid_t`	PMIX_PID	integer	value shall be limited to the `uint32_t` range
`int, int8_t, int16_t, int32_t, int64_t`	PMIX_INT, PMIX_INT8, PMIX_INT16, PMIX_INT32, PMIX_INT64	integer	value shall be limited to its corresponding range
`uint, uint8_t, uint16_t, uint32_t, uint64_t`	PMIX_UINT, PMIX_UINT8, PMIX_UINT16, PMIX_UINT32, PMIX_UINT64	integer	value shall be limited to its corresponding range
`float, double`	PMIX_FLOAT, PMIX_DOUBLE	float	value shall be limited to its corresponding range
`struct timeval`	PMIX_TIMEVAL	{'sec': sec, 'usec': microsec}	each field is an integer value
`time_t`	PMIX_TIME	integer	limited to positive values
`pmix_data_type_t`	PMIX_DATA_TYPE	integer	value shall be limited to the `uint16_t` range
`pmix_status_t`	PMIX_STATUS	integer	
`pmix_key_t`	N/A	string	The string's length shall be limited to one less than the size of the `pmix_key_t` array (to reserve space for the terminating `NULL`)
`pmix_nspace_t`	N/A	string	The string's length shall be limited to one less than the size of the `pmix_nspace_t` array (to reserve space for the terminating `NULL`)

Table A.1.: C-to-Python Datatype Correspondence

C-Definition	PMIx Name	Python Definition	Notes
`pmix_rank_t`	PMIX_PROC_RANK	integer	value shall be limited to the `uint32_t` range excepting the reserved values near `UINT32_MAX`
`pmix_proc_t`	PMIX_PROC	{'nspace': nspace, 'rank': rank}	*nspace* is a Python string and *rank* is an integer value. The *nspace* string's length shall be limited to one less than the size of the `pmix_nspace_t` array (to reserve space for the terminating `NULL`), and the *rank* value shall conform to the constraints associated with `pmix_rank_t`
`pmix_byte_object_t`	PMIX_BYTE_OBJECT	{'bytes': bytes, 'size': size}	*bytes* is a Python byte array and *size* is the integer number of bytes in that array.
`pmix_persistence_t`	PMIX_PERSISTENCE	integer	value shall be limited to the `uint8_t` range
`pmix_scope_t`	PMIX_SCOPE	integer	value shall be limited to the `uint8_t` range
`pmix_data_range_t`	PMIX_RANGE	integer	value shall be limited to the `uint8_t` range
`pmix_proc_state_t`	PMIX_PROC_STATE	integer	value shall be limited to the `uint8_t` range
`pmix_proc_info_t`	PMIX_PROC_INFO	{'proc': {'nspace': nspace, 'rank': rank}, 'hostname': hostname, 'executable': executable, 'pid': pid, 'exitcode': exitcode, 'state': state}	*proc* is a Python `proc` dictionary; *hostname* and *executable* are Python strings; and *pid*, *exitcode*, and *state* are Python integers

Table A.1.: C-to-Python Datatype Correspondence

C-Definition	PMIx Name	Python Definition	Notes
pmix_data_array_t	PMIX_DATA_ARRAY	{'type': type, 'array': array}	*type* is the PMIx type of object in the array and *array* is a Python *list* containing the individual array elements. Note that *array* can consist of *any* PMIx types, including (for example) a Python **info** object that itself contains an **array** value
pmix_info_directives_t	PMIX_INFO_DIRECTIVES	integer	value shall be limited to the **uint32_t** range
pmix_alloc_directive_t	PMIX_ALLOC_DIRECTIVE	integer	value shall be limited to the **uint8_t** range
pmix_iof_channel_t	PMIX_IOF_CHANNEL	integer	value shall be limited to the **uint16_t** range
pmix_envar_t	PMIX_ENVAR	{'envar': envar, 'value': value, 'separator': separator}	*envar* and *value* are Python strings, and *separator* a single-character Python string
pmix_value_t	PMIX_VALUE	{'value': value, 'val_type': type}	*type* is the PMIx datatype of *value*, and *value* is the associated value expressed in the appropriate Python form for the specified datatype
pmix_info_t	PMIX_INFO	{'key': key, 'flags': flags, value': value, 'val_type': type}	*key* is a Python string **key** , *flags* is a bitmask of **info directives** , *type* is the PMIx datatype of *value*, and *value* is the associated value expressed in the appropriate Python form for the specified datatype
pmix_pdata_t	PMIX_PDATA	{'proc': {'nspace': nspace, 'rank': rank}, 'key': key, 'value': value, 'val_type': type}	*proc* is a Python **proc** dictionary; *key* is a Python string **key** ; *type* is the PMIx datatype of *value*; and *value* is the associated value expressed in the appropriate Python form for the specified datatype

Table A.1.: C-to-Python Datatype Correspondence

C-Definition	PMIx Name	Python Definition	Notes
pmix_app_t	PMIX_APP	{'cmd': cmd, 'argv': [argv], 'env': [env], 'maxprocs': maxprocs, 'info': [info]}	*cmd* is a Python string; *argv* and *env* are Python *lists* containing Python strings; *maxprocs* is an integer; and *info* is a Python *list* of **info** values
pmix_query_t	PMIX_QUERY	{'keys': [keys], 'qualifiers': [info]}	*keys* is a Python *list* of Python strings, and *qualifiers* is a Python *list* of **info** values
pmix_regattr_t	PMIX_REGATTR	{'name': name, 'key': key, 'type': type, 'info': [info], 'description': [desc]}	*name* and *string* are Python strings; *type* is the PMIx datatype for the attribute's value; *info* is a Python *list* of **info** values; and *description* is a list of Python strings describing the attribute
pmix_link_state_t	PMIX_LINK_STATE	integer	value shall be limited to the **uint8_t** range

A.2.1 Example

Converting a C-based program to its Python equivalent requires translation of the relevant datatypes as well as use of the appropriate API form. An example small program may help illustrate the changes. Consider the following C-based program snippet:

```
#include <pmix.h>
...

pmix_info_t info[2];

PMIX_INFO_LOAD(&info[0], PMIX_PROGRAMMING_MODEL, "TEST", PMIX_STRING)
PMIX_INFO_LOAD(&info[1], PMIX_MODEL_LIBRARY_NAME, "PMIX", PMIX_STRING)

rc = PMIx_Init(&myproc, info, 2);

PMIX_INFO_DESTRUCT(&info[0]);  // free the copied string
PMIX_INFO_DESTRUCT(&info[1]);  // free the copied string
```

Moving to the Python version requires that the **pmix_info_t** be translated to the Python **info** equivalent, and that the returned information be captured in the return parameters as opposed to a pointer parameter in the function call, as shown below:

```
import pmix
...

myclient = PMIxClient()
info = [{'key':PMIX_PROGRAMMING_MODEL,
            'value':'TEST', 'val_type':PMIX_STRING},
         {'key':PMIX_MODEL_LIBRARY_NAME,
            'value':'PMIX', 'val_type':PMIX_STRING}]
(rc,myproc) = myclient.init(info)
```

Note the use of the **PMIX_STRING** identifier to ensure the Python bindings interpret the provided string value as a PMIx "string" and not an array of bytes.

A.3 Callback Function Definitions

A.3.1 IOF Delivery Function

Summary
Callback function for delivering forwarded IO to a process

Format

```
def iofcbfunc(iofhdlr:integer, channel:integer,
              source:dict, payload:dict, info:list)
```

IN   iofhdlr
   Registration number of the handler being invoked (integer)
IN   channel
   Python **channel** bitmask identifying the channel the data arrived on (integer)
IN   source
   Python **proc** identifying the namespace/rank of the process that generated the data (dict)
IN   payload
   Python **byteobject** containing the data (dict)
IN   info
   List of Python **info** provided by the source containing metadata about the payload. This
   could include **PMIX_IOF_COMPLETE** (list)

Returns: nothing

See **pmix_iof_cbfunc_t** for details

A.3.2 Event Handler

Summary
Callback function for event handlers

Format

```
def evhandler(evhdlr:integer, status:integer,
              source:dict, info:list, results:list)
```

IN   iofhdlr
   Registration number of the handler being invoked (integer)
IN   status
   Status associated with the operation (integer)
IN   source
   Python **proc** identifying the namespace/rank of the process that generated the event (dict)
IN   info
   List of Python **info** provided by the source containing metadata about the event (list)
IN   results
   List of Python **info** containing the aggregated results of all prior evhandlers (list)

Returns:

1   • *rc* - Status returned by the event handler's operation (integer)

2   • *results* - List of Python **info** containing results from this event handler's operation on the event
3   (list)

4   See **pmix_notification_fn_t** for details

5 A.3.3 Server Module Functions

6   The following definitions represent functions that may be provided to the PMIx server library at
7   time of initialization for servicing of client requests. Module functions that are not provided default
8   to returning "not supported" to the caller.

9 A.3.3.1 Client Connected

10 Summary
11   Notify the host server that a client connected to this server.

12 Format
PMIx v4.0 ──────────────────────────── Python ────────────────────────────▼

13   **def clientconnected(proc:dict is not None)**
     ──────────────────────────── Python ────────────────────────────▲

14   **IN   proc**
15   Python **proc** identifying the namespace/rank of the process that connected (dict)

16   Returns:

17   • *rc* - **PMIX_SUCCESS** or a PMIx error code indicating the connection should be rejected
18   (integer)

19   See **pmix_server_client_connected_fn_t** for details

20 A.3.3.2 Client Finalized

21 Summary
22   Notify the host environment that a client called **PMIx_Finalize**.

23 Format
PMIx v4.0 ──────────────────────────── Python ────────────────────────────▼

24   **def clientfinalized(proc:dict is not None):**
     ──────────────────────────── Python ────────────────────────────▲

25   **IN   proc**
26   Python **proc** identifying the namespace/rank of the process that finalized (dict)

27   Returns: nothing

28   See **pmix_server_client_finalized_fn_t** for details

A.3.3.3 Client Aborted

Summary

Notify the host environment that a local client called **`PMIx_Abort`** .

Format

———————————————————— Python ————————————————————

```
def clientaborted(args:dict is not None)
```

———————————————————— Python ————————————————————

IN   args
Python dictionary containing:

- 'caller': Python **`proc`** identifying the namespace/rank of the process calling abort (dict)

- 'status': PMIx status to be returned on exit (integer)

- 'msg': Optional string message to be printed (string)

- 'targets': Optional list of Python **`proc`** identifying the namespace/rank of the processes to
  be aborted (list)

Returns:

- *rc* - **`PMIX_SUCCESS`** or a PMIx error code indicating the operation failed (integer)

See **`pmix_server_abort_fn_t`** for details

A.3.3.4 Fence

Summary

At least one client called either **`PMIx_Fence`** or **`PMIx_Fence_nb`**

Format

———————————————————— Python ————————————————————

```
def fence(args:dict is not None)
```

———————————————————— Python ————————————————————

IN   args
Python dictionary containing:

- 'procs': List of Python **`proc`** identifying the namespace/rank of the participating
  processes (list)

- 'directives': Optional list of Python **`info`** containing directives controlling the operation
  (list)

- 'data': Optional Python bytearray of data to be circulated during fence operation (bytearray)

Returns:

- *rc* - **PMIX_SUCCESS** or a PMIx error code indicating the operation failed (integer)

- *data* - Python bytearray containing the aggregated data from all participants (bytearray)

See **pmix_server_fencenb_fn_t** for details

A.3.3.5 Direct Modex

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 v4.0

──────────────── Python ────────────────▼

```
def dmodex(args:dict is not None)
```

──────────────── Python ────────────────▲

IN args
    Python dictionary containing:

- 'proc': Python **proc** of process whose data is being requested (dict)

- 'directives': Optional list of Python **info** containing directives controlling the operation (list)

Returns:

- *rc* - **PMIX_SUCCESS** or a PMIx error code indicating the operation failed (integer)

- *data* - Python bytearray containing the data for the specified process (bytearray)

See **pmix_server_dmodex_req_fn_t** for details

A.3.3.6 Publish

Summary

Publish data per the PMIx API specification.

Format

PMIx v4.0

──────────────── Python ────────────────▼

```
def publish(args:dict is not None)
```

──────────────── Python ────────────────▲

IN args
    Python dictionary containing:

- 'proc': Python **proc** dictionary of process publishing the data (dict)

- 'directives': List of Python **info** containing data and directives (list)

Returns:

- *rc* - **PMIX_SUCCESS** or a PMIx error code indicating the operation failed (integer)

See **pmix_server_publish_fn_t** for details

A.3.3.7 Lookup

Summary

Lookup published data.

Format

———————————————— Python ————————————————

```
def lookup(args:dict is not None)
```

———————————————— Python ————————————————

IN args

Python dictionary containing:

- 'proc': Python **proc** of process seeking the data (dict)

- 'keys': List of Python strings (list)

- 'directives': Optional list of Python **info** containing directives (list)

Returns:

- *rc* - **PMIX_SUCCESS** or a PMIx error code indicating the operation failed (integer)

- *pdata* - List of **pdata** containing the returned results (list)

See **pmix_server_lookup_fn_t** for details

A.3.3.8 Unpublish

Summary

Delete data from the data store.

Format

———————————————— Python ————————————————

```
def unpublish(args:dict is not None)
```

———————————————— Python ————————————————

IN args

Python dictionary containing:

- 'proc': Python **proc** of process unpublishing data (dict)

- 'keys': List of Python strings (list)

- 'directives': Optional list of Python **info** containing directives (list)

Returns:

- *rc* - **PMIX_SUCCESS** or a PMIx error code indicating the operation failed (integer)

See **pmix_server_unpublish_fn_t** for details

A.3.3.9   Spawn

2   **Summary**
3   Spawn a set of applications/processes as per the **PMIx_Spawn** API.

4   **Format**

PMIx v4.0 ──────────── Python ────────────

5   ```
def spawn(args:dict is not None)
```
──────────── Python ────────────

6   **IN    args**
7       Python dictionary containing:

8       • 'proc': Python **proc** of process making the request (dict)

9       • 'jobinfo': Optional list of Python **info** job-level directives and information (list)

10      • 'apps': List of Python **app** describing applications to be spawned (list)

11  Returns:

12  • *rc* - **PMIX_SUCCESS** or a PMIx error code indicating the operation failed (integer)

13  • *nspace* - Python string containing namespace of the spawned job (str)

14  See **pmix_server_spawn_fn_t** for details

15  **A.3.3.10   Connect**

16  **Summary**
17  Record the specified processes as *connected*.

18  **Format**

*PMIx v4.0*  ──────────── Python ────────────

19  ```
def connect(args:dict is not None)
```
──────────── Python ────────────

20 **IN   args**
21   Python dictionary containing:

22   • 'procs': List of Python **proc** identifying the namespace/rank of the participating
23   processes (list)

24   • 'directives': Optional list of Python **info** containing directives controlling the operation
25   (list)

26 Returns:

27 • *rc* - **PMIX_SUCCESS** or a PMIx error code indicating the operation failed (integer)

28 See **pmix_server_connect_fn_t** for details

A.3.3.11 Disconnect

Summary

Disconnect a previously connected set of processes.

Format

PMIx v4.0

———————————————— Python ————————————————

```
def disconnect(args:dict is not None)
```

———————————————— Python ————————————————

IN   args

Python dictionary containing:

- 'procs': List of Python **proc** identifying the namespace/rank of the participating processes (list)

- 'directives': Optional list of Python **info** containing directives controlling the operation (list)

Returns:

- *rc* - **PMIX_SUCCESS** or a PMIx error code indicating the operation failed (integer)

See **pmix_server_disconnect_fn_t** for details

A.3.3.12 Register Events

Summary

Register to receive notifications for the specified events.

Format

PMIx v4.0

———————————————— Python ————————————————

```
def register_events(args:dict is not None)
```

———————————————— Python ————————————————

IN   args

Python dictionary containing:

- 'codes': List of Python integers (list)

- 'directives': Optional list of Python **info** containing directives controlling the operation (list)

Returns:

- *rc* - **PMIX_SUCCESS** or a PMIx error code indicating the operation failed (integer)

See **pmix_server_register_events_fn_t** for details

A.3.3.13 Deregister Events

Summary

Deregister to receive notifications for the specified events.

Format

———————————————————————— Python ————————————————————————

```
def deregister_events(args:dict is not None)
```
———————————————————————— Python ————————————————————————

IN   args
    Python dictionary containing:

- 'codes': List of Python integers (list)

Returns:

- *rc* - **PMIX_SUCCESS** or a PMIx error code indicating the operation failed (integer)

See **pmix_server_deregister_events_fn_t** for details

A.3.3.14 Notify Event

Summary

Notify the specified range of processes of an event.

Format

———————————————————————— Python ————————————————————————

```
def notify_event(args:dict is not None)
```
———————————————————————— Python ————————————————————————

IN   args
    Python dictionary containing:

- 'code': Python integer **pmix_status_t** (integer)

- 'source': Python **proc** of process that generated the event (dict)

- 'range': Python **range** in which the event is to be reported (integer)

- 'directives': Optional list of Python **info** directives (list)

Returns:

- *rc* - **PMIX_SUCCESS** or a PMIx error code indicating the operation failed (integer)

See **pmix_server_notify_event_fn_t** for details

A.3.3.15 Query

Summary

Query information from the resource manager.

Format

—————————————————————————— Python ——————————————————————————————

```
def query(args:dict is not None)
```

—————————————————————————— Python ——————————————————————————————

IN   args
    Python dictionary containing:

- 'source': Python **proc** of requesting process (dict)

- 'queries': List of Python **query** directives (list)

Returns:

- *rc* - **PMIX_SUCCESS** or a PMIx error code indicating the operation failed (integer)

- *info* - List of Python **info** containing the returned results (list)

See **pmix_server_query_fn_t** for details

A.3.3.16 Tool Connected

Summary
Register that a tool has connected to the server.

Format

—————————————————————————— Python ——————————————————————————————

```
def tool_connected(args:dict is not None)
```

—————————————————————————— Python ——————————————————————————————

IN   args
    Python dictionary containing:

- 'directives': Optional list of Python **info** info on the connecting tool (list)

Returns:

- *rc* - **PMIX_SUCCESS** or a PMIx error code indicating the operation failed (integer)

- *proc* - Python **proc** containing the assigned namespace:rank for the tool (dict)

See **pmix_server_tool_connection_fn_t** for details

A.3.3.17 Log

Summary
Log data on behalf of a client.

1   **Format**

PMIx v4.0 ───────────────────────── Python ─────────────────────────

2   ```
     def log(args:dict is not None)
     ```
     ───────────────────────── Python ─────────────────────────

3    **IN    args**
4           Python dictionary containing:

5           • 'source': Python **proc** of requesting process (dict)

6           • 'data': Optional list of Python **info** containing data to be logged (list)

7           • 'directives': Optional list of Python **info** containing directives (list)

8    Returns:

9    • *rc* - **PMIX_SUCCESS** or a PMIx error code indicating the operation failed (integer)

10   See **pmix_server_log_fn_t** for details

### A.3.3.18   Allocate Resources

12   **Summary**
13   Request allocation operations on behalf of a client.

14   **Format**

*PMIx v4.0*  ───────────────────────── Python ─────────────────────────

15   ```
     def allocate(args:dict is not None)
     ```
     ───────────────────────── Python ─────────────────────────

16   **IN   args**
17   Python dictionary containing:

18   • 'source': Python **proc** of requesting process (dict)

19   • 'action': Python **allocdir** specifying requested action (integer)

20   • 'directives': Optional list of Python **info** containing directives (list)

21   Returns:

22   • *rc* - **PMIX_SUCCESS** or a PMIx error code indicating the operation failed (integer)

23   • refarginfo - List of Python **info** containing results of requested operation (list)

24   See **pmix_server_alloc_fn_t** for details

A.3.3.19   Job Control

26   **Summary**
27   Execute a job control action on behalf of a client.

Format

──────────────────── Python ────────────────────

```
def job_control(args:dict is not None)
```

──────────────────── Python ────────────────────

IN   args
Python dictionary containing:

- 'source': Python **proc** of requesting process (dict)

- 'targets': List of Python **proc** specifying target processes (list)

- 'directives': Optional list of Python **info** containing directives (list)

Returns:

- *rc* - **PMIX_SUCCESS** or a PMIx error code indicating the operation failed (integer)

See **pmix_server_job_control_fn_t** for details

A.3.3.20   Monitor

Summary
Request that a client be monitored for activity.

Format

──────────────────── Python ────────────────────

```
def monitor(args:dict is not None)
```

──────────────────── Python ────────────────────

IN   args
Python dictionary containing:

- 'source': Python **proc** of requesting process (dict)

- 'monitor': Python **info** attribute indicating the type of monitor being requested (dict)

- 'error': Status code to be used when generating an event notification (integer) alerting that
  the monitor has been triggered.

- 'directives': Optional list of Python **info** containing directives (list)

Returns:

- *rc* - **PMIX_SUCCESS** or a PMIx error code indicating the operation failed (integer)

See **pmix_server_monitor_fn_t** for details

A.3.3.21   Get Credential

Summary
Request a credential from the host environment

2   ```
def get_credential(args:dict is not None)
```
▲ ─────────────────────── Python ─────────────────────── ▲

3    **IN    args**
4         Python dictionary containing:

5         • 'source': Python **proc** of requesting process (dict)

6         • 'directives': Optional list of Python **info** containing directives (list)

7    Returns:

8    • *rc* - **PMIX_SUCCESS** or a PMIx error code indicating the operation failed (integer)

9    • *cred* - Python **byteobject** containing returned credential (dict)

10   • *info* - List of Python **info** containing any additional info about the credential (list)

11   See **pmix_server_get_cred_fn_t** for details

## A.3.3.22   Validate Credential

13   **Summary**
14   Request validation of a credential

15   **Format**
*PMIx v4.0*   ▼ ─────────────────────── Python ─────────────────────── ▼

16   ```
def validate_credential(args:dict is not None)
```
▲ ─────────────────────── Python ─────────────────────── ▲

17   **IN   args**
18   Python dictionary containing:

19   • 'source': Python **proc** of requesting process (dict)

20   • 'credential': Python **byteobject** containing credential (dict)

21   • 'directives': Optional list of Python **info** containing directives (list)

22   Returns:

23   • *rc* - **PMIX_SUCCESS** or a PMIx error code indicating the operation failed (integer)

24   • *info* - List of Python **info** containing any additional info from the credential (list)

25   See **pmix_server_validate_cred_fn_t** for details

A.3.3.23   IO Forward

27   **Summary**
28   Request the specified IO channels be forwarded from the given array of processes.

Format

― Python ―

2   `def iof_pull(args:dict is not None)`

― Python ―

3   **IN   args**
4   Python dictionary containing:

5   • 'sources': List of Python **proc** of processes whose IO is being requested (list)

6   • 'channels': Bitmask of Python **channel** identifying IO channels to be forwarded (integer)

7   • 'directives': Optional list of Python **info** containing directives (list)

8   Returns:

9   • *rc* - **PMIX_SUCCESS** or a PMIx error code indicating the operation failed (integer)

10   See **pmix_server_iof_fn_t** for details

11   A.3.3.24   IO Push

12   **Summary**
13   Pass standard input data to the host environment for transmission to specified recipients.

14   **Format**

― Python ―

15   `def iof_push(args:dict is not None)`

― Python ―

16   **IN   args**
17   Python dictionary containing:

18   • 'source': Python **proc** of process whose input is being forwarded (dict)

19   • 'payload': Python **byteobject** containing input bytes (dict)

20   • 'targets': List of **proc** of processes that are to receive the payload (list)

21   • 'directives': Optional list of Python **info** containing directives (list)

22   Returns:

23   • *rc* - **PMIX_SUCCESS** or a PMIx error code indicating the operation failed (integer)

24   See **pmix_server_stdin_fn_t** for details

25   A.3.3.25   Group Operations

26   **Summary**
27   Request group operations (construct, destruct, etc.) on behalf of a set of processes.

Format

————————————————— Python —————————————————

```
def group(args:dict is not None)
```
————————————————— Python —————————————————

IN   args
    Python dictionary containing:

- 'op': Operation host is to perform on the specified group (integer)

- 'group': String identifier of target group (str)

- 'procs': List of Python **proc** of participating processes (dict)

- 'directives': Optional list of Python **info** containing directives (list)

Returns:

- *rc* - **PMIX_SUCCESS** or a PMIx error code indicating the operation failed (integer)

- refarginfo - List of Python **info** containing results of requested operation (list)

See **pmix_server_grp_fn_t** for details

A.3.3.26   Fabric Operations

Summary
Request fabric-related operations (e.g., information on a fabric) on behalf of a tool or other process.

Format

————————————————— Python —————————————————

```
def fabric(args:dict is not None)
```
————————————————— Python —————————————————

IN   args
    Python dictionary containing:

- 'source': Python **proc** of requesting process (dict)

- 'op': Operation host is to perform on the specified fabric (integer)

- 'directives': Optional list of Python **info** containing directives (list)

Returns:

- *rc* - **PMIX_SUCCESS** or a PMIx error code indicating the operation failed (integer)

- refarginfo - List of Python **info** containing results of requested operation (list)

See **pmix_server_fabric_fn_t** for details

1 A.4 PMIxClient

2 The client Python class is by far the richest in terms of APIs as it houses all the APIs that an
3 application might utilize. Due to the datatype translation requirements of the C-Python interface,
4 only the blocking form of each API is supported – providing a Python callback function directly to
5 the C interface underlying the bindings was not a supportable option.

6 A.4.1 Client.init

7 Summary
8 Initialize the PMIx client library after obtaining a new PMIxClient object

9 Format
PMIx v4.0 ──────────── Python ────────────

```
10  rc, proc = myclient.init(info:list)
```
──────────── Python ────────────

11 **IN   info**
12   List of Python **info** dictionaries (list)

13 Returns:

14 • *rc* - **PMIX_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)

15 • *proc* - a Python **proc** dictionary (dict)

16 See **PMIx_Init** for description of all relevant attributes and behaviors

17 A.4.2 Client.initialized

18 Format
PMIx v4.0 ──────────── Python ────────────

```
19  rc = myclient.initialized()
```
──────────── Python ────────────

20 Returns:

21 • *rc* - a value of **1** (true) will be returned if the PMIx library has been initialized, and **0** (false)
22   otherwise (integer)

23 See **PMIx_Initialized** for description of all relevant attributes and behaviors

A.4.3 Client.get_version

Format

PMIx v4.0

———————————————————— Python ————————————————————————

```
vers = myclient.get_version()
```

———————————————————— Python ————————————————————————

Returns:

- *vers* - Python string containing the version of the PMIx library (e.g., "3.1.4") (integer)

See **PMIx_Get_version** for description of all relevant attributes and behaviors

A.4.4 Client.finalize

Summary

Finalize the PMIx client library.

Format

PMIx v4.0

———————————————————— Python ————————————————————————

```
rc = myclient.finalize(info:list)
```

———————————————————— Python ————————————————————————

IN   info
    List of Python **info** dictionaries (list)

Returns:

- *rc* - **PMIX_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)

See **PMIx_Finalize** for description of all relevant attributes and behaviors

A.4.5 Client.abort

Summary

Request that the provided list of procs be aborted

Format

──────────────────────────── Python ────────────────────────────▼

2   `rc = myclient.abort(status:integer, msg:str, targets:list)`

▲──────────────────────────── Python ────────────────────────────▲

3   **IN**   `status`
4   PMIx status to be returned on exit (integer)
5   **IN**   `msg`
6   String message to be printed (string)
7   **IN**   `targets`
8   List of Python **`proc`** dictionaries (list)

9   Returns:

10   • *rc* - **`PMIX_SUCCESS`** or a negative value corresponding to a PMIx error constant (integer)

11   See **`PMIx_Abort`** for description of all relevant attributes and behaviors

12   A.4.6   Client.store_internal

13   **Summary**
14   Store some data locally for retrieval by other areas of the process

15   **Format**

──────────────────────────── Python ────────────────────────────▼

16   `rc = myclient.store_internal(proc:dict, key:str, value:dict)`

▲──────────────────────────── Python ────────────────────────────▲

17   **IN**   `proc`
18   Python **`proc`** dictionary of the process being referenced (dict)
19   **IN**   `key`
20   String key of the data (string)
21   **IN**   `value`
22   Python **`value`** dictionary (dict)

23   Returns:

24   • *rc* - **`PMIX_SUCCESS`** or a negative value corresponding to a PMIx error constant (integer)

25   See **`PMIx_Store_internal`** for details

26   A.4.7   Client.put

27   **Summary**
28   Push a key/value pair into the client's namespace.

1 **Format**
   PMIx v4.0 ▼ ──────────────── Python ──────────────── ▼

2 ```
   rc = myclient.put(scope:integer, key:str, value:dict)
   ```
   ▲ ──────────────── Python ──────────────── ▲

3  **IN    scope**
4  Scope of the data being posted (integer)
5  **IN    key**
6  String key of the data (string)
7  **IN    value**
8  Python **value** dictionary (dict)

9  Returns:

10  • *rc* - **PMIX_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)

11  See **PMIx_Put** for description of all relevant attributes and behaviors

## 12  A.4.8  Client.commit

13  **Summary**
14  Push all previously **PMIxClient.put** values to the local PMIx server.

15  **Format**
   *PMIx v4.0* ▼ ──────────────── Python ──────────────── ▼

16  ```
   rc = myclient.commit()
   ```
   ▲ ──────────────── Python ──────────────── ▲

17 Returns:

18 • *rc* - **PMIX_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)

19 See **PMIx_Commit** for description of all relevant attributes and behaviors

20 A.4.9 Client.fence

21 **Summary**
22 Execute a blocking barrier across the processes identified in the specified list

1   **Format**

──────────── Python ────────────────────▼

2   ```
rc = myclient.fence(peers:list, directives:list)
```
     ▲──────────── Python ────────────────────▲

3    **IN    peers**
4        List of Python **proc** dictionaries (list)
5    **IN    directives**
6        List of Python **info** dictionaries (list)

7    Returns:

8    • *rc* - **PMIX_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)

9    See **PMIx_Fence** for description of all relevant attributes and behaviors

## 10    A.4.10    Client.get

11    **Summary**
12    Retrieve a key/value pair

13    **Format**

──────────── Python ────────────────────▼

14    ```
rc, val = myclient.get(proc:dict, key:str, directives:list)
```
     ▲──────────── Python ────────────────────▲

15   **IN   proc**
16   Python **proc** whose data is being requested (dict)
17   **IN   key**
18   Python string key of the data to be returned (str)
19   **IN   directives**
20   List of Python **info** dictionaries (list)

21   Returns:

22   • *rc* - **PMIX_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)

23   • *val* - Python **value** containing the returned data (dict)

24   See **PMIx_Get** for description of all relevant attributes and behaviors

25   A.4.11   Client.publish

26   **Summary**
27   Publish data for later access via **PMIx_Lookup** .

1 **Format**

PMIx v4.0 ──────────────────── Python ────────────────────▼

2 ```
rc = myclient.publish(directives:list)
```
──────────────────── Python ────────────────────▲

3 **IN    directives**
4     List of Python **info** dictionaries containing data to be published and directives (list)

5 Returns:

6 • *rc* - **PMIX_SUCCESS**  or a negative value corresponding to a PMIx error constant (integer)

7 See **PMIx_Publish**  for description of all relevant attributes and behaviors

## 8 A.4.12   Client.lookup

9 **Summary**
10 Lookup information published by this or another process with **PMIx_Publish** .

11 **Format**

*PMIx v4.0* ──────────────────── Python ────────────────────▼

12 ```
rc,info = myclient.lookup(pdata:list, directives:list)
```
──────────────────── Python ────────────────────▲

13 **IN   pdata**
14   List of Python **pdata** dictionaries identifying data to be retrieved (list)
15 **IN   directives**
16   List of Python **info** dictionaries (list)

17 Returns:

18 • *rc* - **PMIX_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)

19 • *info* - Python list of **info** containing the returned data (list)

20 See **PMIx_Lookup** for description of all relevant attributes and behaviors

21 A.4.13   Client.unpublish

22 **Summary**
23 Delete data published by this process with **PMIx_Publish** .

1 **Format**

▼ ——————————————— Python ————————————————————— ▼

2 ```
rc = myclient.unpublish(keys:list, directives:list)
```
▲ ——————————————— Python ————————————————————— ▲

3 **IN    keys**
4     List of Python string keys identifying data to be deleted (list)
5 **IN    directives**
6     List of Python **info** dictionaries (list)

7 Returns:

8 • *rc* - **PMIX_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)

9 See **PMIx_Unpublish** for description of all relevant attributes and behaviors

## 10 A.4.14    Client.spawn

11 **Summary**
12 Spawn a new job.

13 **Format**

▼ ——————————————— Python ————————————————————— ▼

14 ```
rc,nspace = myclient.spawn(jobinfo:list, apps:list)
```
▲ ——————————————— Python ————————————————————— ▲

15 **IN   jobinfo**
16   List of Python **info** dictionaries (list)
17 **IN   apps**
18   List of Python **app** dictionaries (list)

19 Returns:

20 • *rc* - **PMIX_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)

21 • *nspace* - Python **nspace** of the new job (dict)

22 See **PMIx_Spawn** for description of all relevant attributes and behaviors

23 A.4.15   Client.connect

24 **Summary**
25 Connect namespaces.

1   **Format**

PMIx v4.0 ──────────────────── Python ────────────────────

2   `rc = myclient.connect(peers:list, directives:list)`
     ──────────────────── Python ────────────────────

3   **IN   peers**
4   List of Python **proc** dictionaries (list)
5   **IN   directives**
6   List of Python **info** dictionaries (list)

7   Returns:

8   • *rc* - **PMIX_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)

9   See **PMIx_Connect** for description of all relevant attributes and behaviors

10   A.4.16   Client.disconnect

11   **Summary**
12   Disconnect namespaces.

13   **Format**

PMIx v4.0 ──────────────────── Python ────────────────────

14   `rc = myclient.disconnect(peers:list, directives:list)`
     ──────────────────── Python ────────────────────

15   **IN   peers**
16   List of Python **proc** dictionaries (list)
17   **IN   directives**
18   List of Python **info** dictionaries (list)

19   Returns:

20   • *rc* - **PMIX_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)

21   See **PMIx_Disconnect** for description of all relevant attributes and behaviors

22   A.4.17   Client.resolve_peers

23   **Summary**
24   Return list of processes within the specified **nspace** on the given node.

Format

—————————————————— Python ——————————————————

2   `rc,procs = myclient.resolve_peers(node:str, nspace:str)`

—————————————————— Python ——————————————————

3   **IN   node**
4   Name of node whose processes are being requested (str)
5   **IN   nspace**
6   Python **nspace** whose processes are to be returned (str)

7   Returns:

8   • *rc* - **PMIX_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)

9   • *procs* - List of Python **proc** dictionaries (list)

10   See **PMIx_Resolve_peers** for description of all relevant attributes and behaviors

11   A.4.18   Client.resolve_nodes

12   **Summary**
13   Return list of nodes hosting processes within the specified **nspace** .

14   **Format**

—————————————————— Python ——————————————————

15   `rc,nodes = myclient.resolve_nodes(nspace:str)`

—————————————————— Python ——————————————————

16   **IN   nspace**
17   Python **nspace** (str)

18   Returns:

19   • *rc* - **PMIX_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)

20   • *nodes* - List of Python string node names (list)

21   See **PMIx_Resolve_nodes** for description of all relevant attributes and behaviors

22   A.4.19   Client.query

23   **Summary**
24   Query information about the system in general

1   **Format**
     PMIx v4.0 ──────────────────── Python ────────────────────────

2   ```
     rc,info = myclient.query(queries:list)
     ```
     ──────────────────── Python ────────────────────────

3    **IN    queries**
4        List of Python **query** dictionaries (list)

5    Returns:

6    • *rc* - **PMIX_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)

7    • *info* - List of Python **info** containing results of the query (list)

8    See **PMIx_Query_info_nb** for description of all relevant attributes and behaviors

## 9    A.4.20    Client.log

10   **Summary**
11   Log data to a central data service/store

12   **Format**
     *PMIx v4.0* ──────────────────── Python ────────────────────────

13   ```
     rc = myclient.log(data:list, directives:list)
     ```
     ──────────────────── Python ────────────────────────

14   **IN   data**
15   List of Python **info** (list)
16   **IN   directives**
17   Optional list of Python **info** (list)

18   Returns:

19   • *rc* - **PMIX_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)

20   See **PMIx_Log** for description of all relevant attributes and behaviors

21   A.4.21   Client.allocate

22   **Summary**
23   Request an allocation operation from the host resource manager.

1 **Format**

PMIx v4.0 ▼——————————————— Python ———————————————▼

2 ```
rc,info = myclient.allocate(request:integer, directives:list)
```
▲——————————————— Python ———————————————▲

3 **IN    request**
4    Python **allocdir** specifying requested operation (integer)
5 **IN    directives**
6    List of Python **info** describing request (list)

7 Returns:

8 • *rc* - **PMIX_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)

9 • *info* - List of Python **info** containing results of the request (list)

10 See **PMIx_Allocation_request_nb** for description of all relevant attributes and behaviors

## 11 A.4.22    Client.job_ctrl

12 **Summary**
13 Request a job control action

14 **Format**

*PMIx v4.0*  ▼——————————————— Python ———————————————▼

15 ```
rc,info = myclient.job_ctrl(targets:list, directives:list)
```
▲——————————————— Python ———————————————▲

16 **IN   targets**
17   List of Python **proc** specifying targets of requested operation (integer)
18 **IN   directives**
19   List of Python **info** describing operation to be performed (list)

20 Returns:

21 • *rc* - **PMIX_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)

22 • *info* - List of Python **info** containing results of the request (list)

23 See **PMIx_Job_control_nb** for description of all relevant attributes and behaviors

24 A.4.23   Client.monitor

25 **Summary**
26 Request that something be monitored

PMIx v4.0 ▼─────────────────── **Python** ───────────────────▼

2   `rc,info = myclient.monitor(monitor:dict, error_code:integer, directives:list`

▲─────────────────── **Python** ───────────────────▲

3   **IN**   **monitor**
4   Python **info** specifying specifying the type of monitor being requested (dict)
5   **IN**   **error_code**
6   Status code to be used when generating an event notification alerting that the monitor has
7   been triggered (integer)
8   **IN**   **directives**
9   List of Python **info** describing request (list)

10   Returns:

11   • *rc* - **PMIX_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)

12   • *info* - List of Python **info** containing results of the request (list)

13   See **PMIx_Process_monitor_nb** for description of all relevant attributes and behaviors

14 A.4.24 Client.get_credential

15   **Summary**
16   Request a credential from the PMIx server/SMS

17   **Format**

PMIx v4.0 ▼─────────────────── **Python** ───────────────────▼

18   `rc,cred = myclient.get_credential(directives:list)`

▲─────────────────── **Python** ───────────────────▲

19   **IN**   **directives**
20   Optional list of Python **info** describing request (list)

21   Returns:

22   • *rc* - **PMIX_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)

23   • *cred* - Python **byteobject** containing returned credential (dict)

24   See **PMIx_Get_credential** for description of all relevant attributes and behaviors

25 A.4.25 Client.validate_credential

26   **Summary**
27   Request validation of a credential by the PMIx server/SMS

1 **Format**

PMIx v4.0 ───────────────── Python ─────────────────▼

2 `rc,info = myclient.validate_credential(cred:dict, directives:list)`
   ───────────────── Python ─────────────────▲

3 **IN   cred**
4   Python **byteobject** containing credential (dict)
5 **IN   directives**
6   Optional list of Python **info** describing request (list)

7 Returns:

8 • *rc* - **PMIX_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)

9 • *info* - List of Python **info** containing additional results of the request (list)

10 See **PMIx_Validate_credential** for description of all relevant attributes and behaviors

11 A.4.26 Client.group_construct

12 **Summary**
13 Construct a new group composed of the specified processes and identified with the provided group
14 identifier

15 **Format**

PMIx v4.0 ───────────────── Python ─────────────────▼

16 `rc,info = myclient.construct_group(grp:string, members:list, directives:list`
   ───────────────── Python ─────────────────▲

17 **IN   grp**
18   Python string identifier for the group (str)
19 **IN   members**
20   List of Python **proc** dictionaries identifying group members (list)
21 **IN   directives**
22   Optional list of Python **info** describing request (list)

23 Returns:

24 • *rc* - **PMIX_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)

25 • *info* - List of Python **info** containing results of the request (list)

26 See **PMIx_Group_construct** for description of all relevant attributes and behaviors

27 A.4.27 Client.group_invite

28 **Summary**
29 Explicitly invite specified processes to join a group

1	**Format**
PMIx v4.0	──────────────── Python ────────────────

2   `rc,info = myclient.group_invite(grp:string, members:list, directives:list)`
──────────────── Python ────────────────

3	**IN grp**
4	Python string identifier for the group (str)
5	**IN members**
6	List of Python **proc** dictionaries identifying processes to be invited (list)
7	**IN directives**
8	Optional list of Python **info** describing request (list)

9   Returns:

10   • *rc* - **PMIX_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)

11   • *info* - List of Python **info** containing results of the request (list)

12   See **PMIx_Group_invite** for description of all relevant attributes and behaviors

13   A.4.28   Client.group_join

14	**Summary**
15	Respond to an invitation to join a group that is being asynchronously constructed

16	**Format**
PMIx v4.0	──────────────── Python ────────────────

17   `rc,info = myclient.group_join(grp:string, leader:dict, opt:integer, directiv`
──────────────── Python ────────────────

18	**IN grp**
19	Python string identifier for the group (str)
20	**IN leader**
21	Python **proc** dictionary identifying process leading the group (dict)
22	**IN opt**
23	One of the **pmix_group_opt_t** values indicating decline/accept (integer)
24	**IN directives**
25	Optional list of Python **info** describing request (list)

26   Returns:

27   • *rc* - **PMIX_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)

28   • *info* - List of Python **info** containing results of the request (list)

29   See **PMIx_Group_join** for description of all relevant attributes and behaviors

A.4.29 Client.group_leave

Summary

Leave a PMIx Group

Format

PMIx v4.0 ──────────────── Python ────────────────

```
rc = myclient.group_leave(grp:string, directives:list)
```

──────────────── Python ────────────────

IN   grp
  Python string identifier for the group (str)
IN   directives
  Optional list of Python **info** describing request (list)

Returns:

- *rc* - **PMIX_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)

See **PMIx_Group_leave** for description of all relevant attributes and behaviors

A.4.30 Client.group_destruct

Summary

Destruct a PMIx Group

Format

PMIx v4.0 ──────────────── Python ────────────────

```
rc = myclient.group_destruct(grp:string, directives:list)
```

──────────────── Python ────────────────

IN   grp
  Python string identifier for the group (str)
IN   directives
  Optional list of Python **info** describing request (list)

Returns:

- *rc* - **PMIX_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)

See **PMIx_Group_destruct** for description of all relevant attributes and behaviors

A.4.31 Client.register_event_handler

Summary

Register an event handler to report events.

1   **Format**

PMIx v4.0
            ▼   Python   ▼

2   ```
rc,id = myclient.register_event_handler(codes:list, directives:list, cbfunc)
```
            ▲                 Python                 ▲

3    **IN    codes**
4        List of Python integer status codes that should be reported to this handler (llist)
5    **IN    directives**
6        Optional list of Python **info** describing request (list)
7    **IN    cbfunc**
8        Python **evhandler** to be called when event is received (func)

9    Returns:

10   • *rc* - **PMIX_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)

11   • *id* - PMIx reference identifier for handler (integer)

12   See **PMIx_Register_event_handler** for description of all relevant attributes and behaviors

# 13    A.4.32    Client.deregister_event_handler

14   **Summary**
15   Deregister an event handler

16   **Format**

*PMIx v4.0*
             ▼                   Python                   ▼

17   ```
myclient.deregister_event_handler(id:integer)
```
            ▲   Python   ▲

18   **IN   id**
19   PMIx reference identifier for handler (integer)

20   Returns: None

21   See **PMIx_Deregister_event_handler** for description of all relevant attributes and
22   behaviors

23   A.4.33   Client.notify_event

24   **Summary**
25   Report an event for notification via any registered handler.

1 **Format**

————————————————— Python —————————————————

2   `rc = myclient.notify_event(status:integer, source:dict,`
3   `range:integer, directives:list)`

————————————————— Python —————————————————

4   **IN**   `status`
5   PMIx status code indicating the event being reported (integer)
6   **IN**   `source`
7   Python **proc** of the process that generated the event (dict)
8   **IN**   `range`
9   Python **range** in which the event is to be reported (integer)
10   **IN**   `directives`
11   Optional list of Python **info** dictionaries describing the event (list)

12 Returns:

13 • *rc* - **PMIX_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)

14 See **PMIx_Notify_event** for description of all relevant attributes and behaviors

A.4.34 Client.fabric_register

16 **Summary**
17 Register for access to fabric-related information, including communication cost matrix.

18 **Format**

————————————————— Python —————————————————

19   `rc,fabricinfo = myserver.fabric_register(directives:list)`

————————————————— Python —————————————————

20   **IN**   `directives`
21   Optional list of Python **info** containing directives (list)

22 Returns:

23 • *rc* - **PMIX_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)

24 • *fabricinfo* - List of Python **info** containing fabric info (list)

25 See **PMIx_Fabric_register** for details

A.4.35 Client.fabric_update

27 **Summary**
28 Update fabric-related information, including communication cost matrix.

Format
──────────────────────────── Python ────────────────────────────▼

2   `rc,fabricinfo = myserver.fabric_update()`
──────────────────────────── Python ────────────────────────────▲

3   Returns:

4   • *rc* - **PMIX_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)

5   • *fabricinfo* - List of Python **info** containing updated fabric info (list)

6   See **PMIx_Fabric_update** for details

7 A.4.36 Client.fabric_deregister

8   **Summary**
9   Deregister fabric

10   **Format**
PMIx v4.0 ──────────────────────────── Python ────────────────────────────▼

11   `rc = myserver.fabric_deregister()`
──────────────────────────── Python ────────────────────────────▲

12   Returns:

13   • *rc* - **PMIX_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)

14   See **PMIx_Fabric_deregister** for details

15 A.4.37 Client.fabric_get_vertex_info

16   **Summary**
17   Given a communication cost matrix index for a specified fabric, return an array of information
18   describing the corresponding NIC.

19   **Format**
PMIx v4.0 ──────────────────────────── Python ────────────────────────────▼

20   `rc,nicinfo = myserver.fabric_get_vertex_info(index:integer)`
──────────────────────────── Python ────────────────────────────▲

21   Returns:

22   • *rc* - **PMIX_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)

23   • *nicinfo* - List of Python **info** describing the referenced NIC (list)

24   See **PMIx_Fabric_get_vertex_info** for details

A.4.38 Client.fabric_get_index

Summary

Given info describing a given vertex, return the corresponding communication cost matrix index

Format

PMIx v4.0

———————————————————————— Python ————————————————————————

```
rc,index = myserver.fabric_get_index(info:list)
```

———————————————————————— Python ————————————————————————

IN   info

   List of Python **info** containing vertex description (list)

Returns:

- *rc* - **PMIX_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)

- *index* - Index of corresponding NIC (integer)

See **PMIx_Fabric_get_index** for details

A.4.39 Client.error_string

Summary

Pretty-print string representation of **pmix_status_t**.

Format

PMIx v4.0

———————————————————————— Python ————————————————————————

```
rep = myclient.error_string(status:integer)
```

———————————————————————— Python ————————————————————————

IN   status

   PMIx status code (integer)

Returns:

- *rep* - String representation of the provided status code (str)

See **PMIx_Error_string** for further details

A.4.40 Client.proc_state_string

Summary

Pretty-print string representation of **pmix_proc_state_t**.

Format

Python ▼

```
rep = myclient.proc_state_string(state:integer)
```
Python ▲

3 **IN   state**
4   PMIx process state code (integer)

5 Returns:

6 • *rep* - String representation of the provided process state (str)

7 See **PMIx_Proc_state_string** for further details

8 A.4.41 Client.scope_string

9 **Summary**
10 Pretty-print string representation of **pmix_scope_t** .

11 **Format**

Python ▼

```
rep = myclient.scope_string(scope:integer)
```
Python ▲

13 **IN   scope**
14   PMIx scope value (integer)

15 Returns:

16 • *rep* - String representation of the provided scope (str)

17 See **PMIx_Scope_string** for further details

18 A.4.42 Client.persistence_string

19 **Summary**
20 Pretty-print string representation of **pmix_persistence_t** .

21 **Format**

Python ▼

```
rep = myclient.persistence_string(persistence:integer)
```
Python ▲

23 **IN   persistence**
24   PMIx persistence value (integer)

25 Returns:

26 • *rep* - String representation of the provided persistence (str)

27 See **PMIx_Persistence_string** for further details

A.4.43 Client.data_range_string

2 **Summary**
3 Pretty-print string representation of **pmix_data_range_t** .

4 **Format**

PMIx v4.0 ───────────────── Python ─────────────────▼

5 ```
rep = myclient.data_range_string(range:integer)
```
────────────────── Python ──────────────────

6 **IN    range**
7     PMIx data range value (integer)

8 Returns:

9 • *rep* - String representation of the provided data range (str)

10 See **PMIx_Data_range_string**  for further details

# A.4.44  Client.info_directives_string

12 **Summary**
13 Pretty-print string representation of **pmix_info_directives_t** .

14 **Format**

*PMIx v4.0* ───────────────── Python ─────────────────▼

15 ```
rep = myclient.info_directives_string(directives:integer)
```
────────────────── Python ──────────────────

16 **IN   directives**
17   PMIx info directives value (integer)

18 Returns:

19 • *rep* - String representation of the provided info directives (str)

20 See **PMIx_Info_directives_string** for further details

A.4.45 Client.data_type_string

22 **Summary**
23 Pretty-print string representation of **pmix_data_type_t** .

1 **Format**
— Python —

2 `rep = myclient.data_type_string(dtype:integer)`
— Python —

3 **IN dtype**
4   PMIx datatype value (integer)

5 Returns:

6 • *rep* - String representation of the provided datatype (str)

7 See **PMIx_Data_type_string** for further details

8 A.4.46   Client.alloc_directive_string

9 **Summary**
10 Pretty-print string representation of **pmix_alloc_directive_t** .

11 **Format**
— Python —

12 `rep = myclient.alloc_directive_string(adir:integer)`
— Python —

13 **IN adir**
14   PMIx allocation directive value (integer)

15 Returns:

16 • *rep* - String representation of the provided allocation directive (str)

17 See **PMIx_Alloc_directive_string** for further details

18 A.4.47   Client.iof_channel_string

19 **Summary**
20 Pretty-print string representation of **pmix_iof_channel_t** .

21 **Format**
— Python —

22 `rep = myclient.iof_channel_string(channel:integer)`
— Python —

23 **IN channel**
24   PMIx IOF channel value (integer)

25 Returns:

26 • *rep* - String representation of the provided IOF channel (str)

27 See **PMIx_IOF_channel_string** for further details

A.4.48 Client.job_state_string

Summary

Pretty-print string representation of **pmix_job_state_t** .

Format

PMIx v4.0

——————————————— Python ———————————————

```
rep = myclient.job_state_string(state:integer)
```

——————————————— Python ———————————————

IN   state
    PMIx job state value (integer)

Returns:

- *rep* - String representation of the provided job state (str)

See **PMIx_Job_state_string** for further details

A.4.49 Client.get_attribute_string

Summary

Pretty-print string representation of a PMIx attribute.

Format

PMIx v4.0

——————————————— Python ———————————————

```
rep = myclient.get_attribute_string(attribute:str)
```

——————————————— Python ———————————————

IN   attribute
    PMIx attribute name (string)

Returns:

- *rep* - String representation of the provided attribute (str)

See **PMIx_Get_attribute_string** for further details

A.4.50 Client.get_attribute_name

Summary

Pretty-print name of a PMIx attribute corresponding to the provided string

Format

PMIx v4.0 ▽ ——————————————— Python ——————————————— ▽

2 `rep = myclient.get_attribute_name(attribute:str)`
▲ ——————————————— Python ——————————————— ▲

3 **IN   attributestring**
4   Attribute string (string)

5 Returns:

6 • *rep* - Attribute name corresponding to the provided string (str)

7 See **PMIx_Get_attribute_name** for further details

8 A.4.51   Client.link_state_string

9 **Summary**
10 Pretty-print string representation of **pmix_link_state_t** .

11 **Format**

PMIx v4.0 ▽ ——————————————— Python ——————————————— ▽

12 `rep = myclient.link_state_string(state:integer)`
▲ ——————————————— Python ——————————————— ▲

13 **IN   state**
14   PMIx link state value (integer)

15 Returns:

16 • *rep* - String representation of the provided link state (str)

17 See **PMIx_Link_state_string** for further details

18 A.5   PMIxServer

19 The server Python class inherits the Python "client" class as its parent. Thus, it includes all client
20 functions in addition to the ones defined in this section.

21 A.5.1   Server.init

22 **Summary**
23 Initialize the PMIx server library after obtaining a new PMIxServer object

1 **Format**

PMIx v4.0 ──────────────────── Python ────────────────────▼

2 ```
rc = myserver.init(directives:list, map:dict)
```
▲──────────────────── Python ────────────────────▲

3 **IN    directives**
4     List of Python **info** dictionaries (list)
5 **IN    map**
6     Python dictionary key-function pairs that map **server module** callback functions to
7     provided implementations (dict)

8 Returns:

9 • *rc* - **PMIX_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)

10 See **PMIx_server_init** for description of all relevant attributes and behaviors

## 11 A.5.2 Server.finalize

12 **Summary**
13 Finalize the PMIx server library

14 **Format**

*PMIx v4.0* ──────────────────── Python ────────────────────▼

15 ```
rc = myserver.finalize()
```
▲──────────────────── Python ────────────────────▲

16 Returns:

17 • *rc* - **PMIX_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)

18 See **PMIx_server_finalize** for details

19 A.5.3 Server.generate_regex

20 **Summary**
21 Generate a regular expression representation of the input strings.

Format

────────────────────── Python ──────────────────────

2   `rc,regex = myserver.generate_regex(input:list)`
────────────────────── Python ──────────────────────

3   **IN**   `input`
4   List of Python strings (e.g., node names) (list)

5   Returns:

6   • *rc* - **PMIX_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)

7   • *regex* - Python **bytearray** containing regular expression representation of the input list
8   (**bytearray**)

9   See **PMIx_generate_regex** for details

10 A.5.4 Server.generate_ppn

11   **Summary**
12   Generate a regular expression representation of the input strings.

13   **Format**

────────────────────── Python ──────────────────────

14   `rc,regex = myserver.generate_ppn(input:list)`
────────────────────── Python ──────────────────────

15   **IN**   `input`
16   List of Python strings, each string consisting of a comma-delimited list of ranks on each node,
17   with the strings being in the same order as the node names provided to "generate_regex" (list)

18   Returns:

19   • *rc* - **PMIX_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)

20   • *regex* - Python **bytearray** containing regular expression representation of the input list
21   (**bytearray**)

22   See **PMIx_generate_ppn** for details

23 A.5.5 Server.register_nspace

24   **Summary**
25   Setup the data about a particular namespace.

1 **Format**

PMIx v4.0 ▼——————————————— Python ————————————————▼

```
2  rc = myserver.register_nspace(nspace:str,
3                                    nlocalprocs:integer,
4                                    directives:list)
```

▲——————————————— Python ————————————————▲

5 **IN   nspace**
6   Python string containing the namespace (str)
7 **IN   nlocalprocs**
8   Number of local processes (integer)
9 **IN   directives**
10   List of Python **info** dictionaries (list)

11 Returns:

12 • *rc* - **PMIX_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)

13 See **PMIx_server_register_nspace** for description of all relevant attributes and behaviors

14 A.5.6 Server.deregister_nspace

15 **Summary**
16 Deregister a namespace.

17 **Format**

PMIx v4.0 ▼——————————————— Python ————————————————▼

```
18  myserver.deregister_nspace(nspace:str)
```

▲——————————————— Python ————————————————▲

19 **IN   nspace**
20   Python string containing the namespace (str)

21 Returns: None

22 See **PMIx_server_deregister_nspace** for details

23 A.5.7 Server.register_client

24 **Summary**
25 Register a client process with the PMIx server library.

1 **Format**

Python

2 ```
rc = myserver.register_client(proc:dict, uid:integer, gid:integer)
```

Python

3 **IN    proc**
4     Python **proc** dictionary identifying the client process (dict)
5 **IN    uid**
6     Linux uid value for user executing client process (integer)
7 **IN    gid**
8     Linux gid value for user executing client process (integer)

9 Returns:

10 • *rc* - **PMIX_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)

11 See **PMIx_server_register_client** for details

## 12 **A.5.8    Server.deregister_client**

13 **Summary**
14 Dergister a client process and purge all data relating to it

15 **Format**

Python

16 ```
myserver.deregister_client(proc:dict)
```

Python

17 **IN   proc**
18   Python **proc** dictionary identifying the client process (dict)

19 Returns: None

20 See **PMIx_server_deregister_client** for details

21 **A.5.9   Server.setup_fork**

22 **Summary**
23 Setup the environment of a child process that is to be forked by the host

1 **Format**

PMIx v4.0 ───────────────────── Python ─────────────────────▼

2 ```
rc = myserver.setup_fork(proc:dict, envin:dict)
```
▲───────────────────── Python ─────────────────────

3 **IN    proc**
4     Python **proc** dictionary identifying the client process (dict)
5 **INOUT  envin**
6     Python dictionary containing the environment to be passed to the client (dict)

7 Returns:

8 • *rc* - **PMIX_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)

9 See **PMIx_server_setup_fork** for details

## 10 A.5.10 Server.dmodex_request

11 **Summary**
12 Function by which the host server can request modex data from the local PMIx server.

13 **Format**

*PMIx v4.0* ───────────────────── Python ─────────────────────▼

14 ```
rc,data = myserver.dmodex_request(proc:dict)
```
▲───────────────────── Python ─────────────────────

15 **IN   proc**
16   Python **proc** dictionary identifying the process whose data is requested (dict)

17 Returns:

18 • *rc* - **PMIX_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)

19 • *data* - Python **byteobject** containing the returned data (dict)

20 See **PMIx_server_dmodex_request** for details

21 A.5.11 Server.setup_application

22 **Summary**
23 Function by which the resource manager can request application-specific setup data prior to launch
24 of a **job** .

PMIx v4.0 ──────────────────── Python ────────────────────▼

2   ```
rc,info = myserver.setup_application(nspace:str, directives:list)
```
▲ ──────────────────── Python ────────────────────

3   **IN    nspace**
4       Namespace whose setup information is being requested (str)
5   **IN    directives**
6       Python list of **info** directives

7   Returns:

8   • *rc* - **PMIX_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)

9   • *info* - Python list of **info** dictionaries containing the returned data (list)

10  See **PMIx_server_setup_application** for details

## 11   A.5.12   Server.register_attributes

12  **Summary**
13  Register host environment attribute support for a function.

14  **Format**

*PMIx v4.0*  ▼──────────────────── Python ────────────────────

15  ```
rc = myserver.register_attributes(function:str, attrs:list)
```
▲ ──────────────────── Python ────────────────────

16 **IN   function**
17   Name of the function (str)
18 **IN   attrs**
19   Python list of **regattr** describing the supported attributes

20 Returns:

21 • *rc* - **PMIX_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)

22 See **PMIx_Register_attributes** for details

23   A.5.13   Server.setup_local_support

24 **Summary**
25 Function by which the local PMIx server can perform any application-specific operations prior to
26 spawning local clients of a given application

Format

▼ ──────────────────── Python ──────────────────── ▼

2   `rc = myserver.setup_local_support(nspace:str, info:list)`

▲ ──────────────────── Python ──────────────────── ▲

3   **IN   nspace**
4   Namespace whose setup information is being requested (str)
5   **IN   info**
6   Python list of **info** containing the setup data (list)

7   Returns:

8   • *rc* - **PMIX_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)

9   See **PMIx_server_setup_local_support** for details

10 A.5.14 Server.iof_deliver

11   **Summary**
12   Function by which the host environment can pass forwarded IO to the PMIx server library for
13   distribution to its clients.

14   **Format**

PMIx v4.0 ▼ ──────────────────── Python ──────────────────── ▼

15   `rc = myserver.iof_deliver(source:dict, channel:integer,`
16   `data:dict, directives:list)`

▲ ──────────────────── Python ──────────────────── ▲

17   **IN   source**
18   Python **proc** dictionary identifying the process who generated the data (dict)
19   **IN   channel**
20   Python **channel** bitmask identifying IO channel of the provided data (integer)
21   **IN   data**
22   Python **byteobject** containing the data (dict)
23   **IN   directives**
24   Python list of **info** containing directives (list)

25   Returns:

26   • *rc* - **PMIX_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)

27   See **PMIx_server_IOF_deliver** for details

28 A.5.15 Server.collect_inventory

29   **Summary**
30   Collect inventory of resources on a node

1   **Format**

───────────────── Python ─────────────────

2   `rc,info = myserver.collect_inventory(directives:list)`
                                    ───────────── Python ─────────────

3   **IN   directives**
4   Optional Python list of **info** containing directives (list)

5   Returns:

6   • *rc* - **PMIX_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)

7   • *info* - Python list of **info** containing the returned data (list)

8   See **PMIx_server_collect_inventory** for details

9   A.5.16   Server.deliver_inventory

10   **Summary**
11   Pass collected inventory to the PMIx server library for storage

12   **Format**

───────────────── Python ─────────────────

13   `rc = myserver.deliver_inventory(info:list, directives:list)`
                                    ───────────── Python ─────────────

14   **IN   info**
15   - Python list of **info** dictionaries containing the inventory data (list)
16   **IN   directives**
17   Python list of **info** dictionaries containing directives (list)

18   Returns:

19   • *rc* - **PMIX_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)

20   See **PMIx_server_deliver_inventory** for details

21   A.6   PMIxTool

22   The tool Python class inherits the Python "server" class as its parent. Thus, it includes all client and
23   server functions in addition to the ones defined in this section.

24   A.6.1   Tool.init

25   **Summary**
26   Initialize the PMIx tool library after obtaining a new PMIxTool object

1 **Format**

──────────────── Python ────────────────▼

2 `rc,proc = mytool.init(info:list)`
──────────────── Python ────────────────▲

3 **IN   info**
4   List of Python **info** directives (list)

5 Returns:

6 • *rc* - **PMIX_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)

7 • *proc* - a Python **proc** (dict)

8 See **PMIx_tool_init** for description of all relevant attributes and behaviors

9 A.6.2 Tool.finalize

10 **Summary**
11 Finalize the PMIx tool library, closing the connection to the server.

12 **Format**

──────────────── Python ────────────────▼

13 `rc = mytool.finalize()`
──────────────── Python ────────────────▲

14 Returns:

15 • *rc* - **PMIX_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)

16 See **PMIx_tool_finalize** for description of all relevant attributes and behaviors

17 A.6.3 Tool.connect_to_server

18 **Summary**
19 Switch connection from the current PMIx server to another one, or initialize a connection to a
20 specified server.

1 **Format**

PMIx v4.0 ────────────────────── Python ──────────────────────

2 `rc,proc = mytool.connect_to_server(info:list)`
   ────────────────────── Python ──────────────────────

3 **IN   info**
4   List of Python **info** dictionaries (list)

5 Returns:

6 • *rc* - **PMIX_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)

7 • *proc* - a Python **proc** (dict)

8 See **PMIx_tool_connect_to_server** for description of all relevant attributes and behaviors

A.6.4 Tool.iof_pull

10 **Summary**
11 Register to receive output forwarded from a remote process.

12 **Format**

PMIx v4.0 ────────────────────── Python ──────────────────────

13 `rc,id = mytool.iof_pull(sources:list, channel:integer, directives:list, cbfu`
   ────────────────────── Python ──────────────────────

14 **IN   sources**
15   List of Python **proc** dictionaries of processes whose IO is being requested (list)
16 **IN   channel**
17   Python **channel** bitmask identifying IO channels to be forwarded (integer)
18 **IN   directives**
19   List of Python **info** dictionaries describing request (list)
20 **IN   cbfunc**
21   Python **iofcbfunc** to receive IO payloads (func)

22 Returns:

23 • *rc* - **PMIX_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)

24 • *id* - PMIx reference identifier for request (integer)

25 See **PMIx_IOF_pull** for description of all relevant attributes and behaviors

A.6.5 Tool.iof_deregister

27 **Summary**
28 Deregister from output forwarded from a remote process.

1 **Format**

▼ ——————————————— Python ——————————————— ▼

2 ```
rc = mytool.iof_deregister(id:integer, directives:list)
```
▲ ——————————————— Python ——————————————— ▲

3  **IN  id**
4      PMIx reference identifier returned by pull request (list)
5  **IN  directives**
6      List of Python **info** dictionaries describing request (list)

7  Returns:

8  • *rc* - **PMIX_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)

9  See **PMIx_IOF_deregister** for description of all relevant attributes and behaviors

## 10  A.6.6  Tool.iof_push

11  **Summary**
12  Push data collected locally (typically from stdin) to stdin of target recipients

13  **Format**

▼ ——————————————— Python ——————————————— ▼

14  ```
rc = mytool.iof_push(targets:list, data:dict, directives:list)
```
▲ ——————————————— Python ——————————————— ▲

15 **IN   sources**
16   List of Python **proc** of target processes (list)
17 **IN   data**
18   Python **byteobject** containing data to be delivered (dict)
19 **IN   directives**
20   Optional list of Python **info** describing request (list)

21 Returns:

22 • *rc* - **PMIX_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)

23 See **PMIx_IOF_push** for description of all relevant attributes and behaviors

24 A.7 Example Usage

25 The following examples are provided to illustrate the use of the Python bindings.

A.7.1 Python Client

The following example contains a client program that illustrates a fairly common usage pattern. The program instantiates and initializes the PMIxClient class, posts some data that is to be shared across all processes in the job, executes a "fence" that circulates the data, and then retrieves a value posted by one of its peers. Note that the example has been formatted to fit the document layout.

———————————————— Python ————————————————

```python
from pmix import *


def main():
    # Instantiate a client object
    myclient = PMIxClient()
    print("Testing PMIx ", myclient.get_version())

    # Initialize the PMIx client library, declaring the programming model
    # as "TEST" and the library name as "PMIX", just for the example
    info = ['key':PMIX_PROGRAMMING_MODEL,
                'value':'TEST', 'val_type':PMIX_STRING,
               'key':PMIX_MODEL_LIBRARY_NAME,
                'value':'PMIX', 'val_type':PMIX_STRING]
    rc,myname = myclient.init(info)
    if PMIX_SUCCESS != rc:
        print("FAILED TO INIT WITH ERROR", myclient.error_string(rc))
        exit(1)

    # try posting a value
    rc = myclient.put(PMIX_GLOBAL, "mykey",
                          'value':1, 'val_type':PMIX_INT32)
    if PMIX_SUCCESS != rc:
        print("PMIx_Put FAILED WITH ERROR", myclient.error_string(rc))
        # cleanly finalize
        myclient.finalize()
        exit(1)

    # commit it
    rc = myclient.commit()
    if PMIX_SUCCESS != rc:
        print("PMIx_Commit FAILED WITH ERROR",
                myclient.error_string(rc))
        # cleanly finalize
        myclient.finalize()
        exit(1)
```

```
1                  # execute fence across all processes in my job
2                  procs = []
3                  info = []
4                  rc = myclient.fence(procs, info)
5                  if PMIX_SUCCESS != rc:
6                      print("PMIx_Fence FAILED WITH ERROR", myclient.error_string(rc))
7                      # cleanly finalize
8                      myclient.finalize()
9                      exit(1)
10
11                 # Get a value from a peer
12                 if 0 != myname['rank']:
13                     info = []
14                     rc, get_val = myclient.get('nspace':"testnspace", 'rank': 0,
15                                                "mykey", info)
16                     if PMIX_SUCCESS != rc:
17                         print("PMIx_Commit FAILED WITH ERROR",
18                               myclient.error_string(rc))
19                         # cleanly finalize
20                         myclient.finalize()
21                         exit(1)
22                     print("Get value returned: ", get_val)
23
24                 # test a fence that should return not_supported because
25                 # we pass a required attribute that the server is known
26                 # not to support
27                 procs = []
28                 info = ['key': 'ARBIT', 'flags': PMIX_INFO_REQD,
29                         'value':10, 'val_type':PMIX_INT]
30                 rc = myclient.fence(procs, info)
31                 if PMIX_SUCCESS == rc:
32                     print("PMIx_Fence SUCCEEDED BUT SHOULD HAVE FAILED")
33                     # cleanly finalize
34                     myclient.finalize()
35                     exit(1)
36
37                 # Publish something
38                 info = ['key': 'ARBITRARY', 'value':10, 'val_type':PMIX_INT]
39                 rc = myclient.publish(info)
40                 if PMIX_SUCCESS != rc:
41                     print("PMIx_Publish FAILED WITH ERROR",
42                           myclient.error_string(rc))
43                     # cleanly finalize
```

```
1              myclient.finalize()
2              exit(1)
3
4          # finalize
5          info = []
6          myclient.finalize(info)
7          print("Client finalize complete")
8
9      # Python main program entry point
10     if __name__ == '__main__':
11         main()
```

12 A.7.2 Python Server

13 The following example contains a minimum-level server host program that instantiates and
14 initializes the PMIxServer class. The program illustrates passing several server module functions to
15 the bindings and includes code to setup and spawn a simple client application, waiting until the
16 spawned client terminates before finalizing and exiting itself. Note that the example has been
17 formatted to fit the document layout.

Python

```
18     from pmix import *
19     import signal, time
20     import os
21     import select
22     import subprocess
23
24     def clientconnected(proc:tuple is not None):
25         print("CLIENT CONNECTED", proc)
26         return PMIX_OPERATION_SUCCEEDED
27
28     def clientfinalized(proc:tuple is not None):
29         print("CLIENT FINALIZED", proc)
30         return PMIX_OPERATION_SUCCEEDED
31
32     def clientfence(procs:list, directives:list, data:bytearray):
33         # check directives
34         if directives is not None:
35             for d in directives:
36                 # these are each an info dict
37                 if "pmix" not in d['key']:
38                     # we do not support such directives - see if
```

```
1                          # it is required
2                          try:
3                              if d['flags'] & PMIX_INFO_REQD:
4                                  # return an error
5                                  return PMIX_ERR_NOT_SUPPORTED
6                          except:
7                              #it can be ignored
8                              pass
9           return PMIX_OPERATION_SUCCEEDED
10
11      def main():
12          try:
13              myserver = PMIxServer()
14          except:
15              print("FAILED TO CREATE SERVER")
16              exit(1)
17          print("Testing server version ", myserver.get_version())
18
19          args = ['key':PMIX_SERVER_SCHEDULER,
20                     'value':'T', 'val_type':PMIX_BOOL]
21          map = 'clientconnected': clientconnected,
22                  'clientfinalized': clientfinalized,
23                  'fencenb': clientfence
24          my_result = myserver.init(args, map)
25
26          # get our environment as a base
27          env = os.environ.copy()
28
29          # register an nspace for the client app
30          (rc, regex) = myserver.generate_regex("test000,test001,test002")
31          (rc, ppn) = myserver.generate_ppn("0")
32          kvals = ['key':PMIX_NODE_MAP,
33                      'value':regex, 'val_type':PMIX_STRING,
34                     'key':PMIX_PROC_MAP,
35                      'value':ppn, 'val_type':PMIX_STRING,
36                     'key':PMIX_UNIV_SIZE,
37                      'value':1, 'val_type':PMIX_UINT32,
38                     'key':PMIX_JOB_SIZE,
39                      'value':1, 'val_type':PMIX_UINT32]
40          rc = foo.register_nspace("testnspace", 1, kvals)
41          print("RegNspace ", rc)
42
43          # register a client
```

```
1          uid = os.getuid()
2          gid = os.getgid()
3          rc = myserver.register_client('nspace':"testnspace", 'rank':0,
4                                           uid, gid)
5          print("RegClient ", rc)
6          # setup the fork
7          rc = myserver.setup_fork('nspace':"testnspace", 'rank':0, env)
8          print("SetupFrk", rc)
9
10         # setup the client argv
11         args = ["./client.py"]
12         # open a subprocess with stdout and stderr
13         # as distinct pipes so we can capture their
14         # output as the process runs
15         p = subprocess.Popen(args, env=env,
16             stdout=subprocess.PIPE, stderr=subprocess.PIPE)
17         # define storage to catch the output
18         stdout = []
19         stderr = []
20         # loop until the pipes close
21         while True:
22             reads = [p.stdout.fileno(), p.stderr.fileno()]
23             ret = select.select(reads, [], [])
24
25             stdout_done = True
26             stderr_done = True
27
28             for fd in ret[0]:
29                 # if the data
30                 if fd == p.stdout.fileno():
31                     read = p.stdout.readline()
32                     if read:
33                         read = read.decode('utf-8').rstrip()
34                         print('stdout: ' + read)
35                         stdout_done = False
36                 elif fd == p.stderr.fileno():
37                     read = p.stderr.readline()
38                     if read:
39                         read = read.decode('utf-8').rstrip()
40                         print('stderr: ' + read)
41                         stderr_done = False
42
43             if stdout_done and stderr_done:
```

```
1                      break
2            print("FINALIZING")
3            myserver.finalize()
4
5
6        if __name__ == '__main__':
7            main()
```

————————————————— Python —————————————————

444

B.2   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

B.3 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

General terms and other items not induced in the other indices.

Index of APIs

Index of Support Macros

Index of Data Structures

Index of Constants

Index of Attributes